NAG Fortran Library Manual
Mark 18
Volume 1
Contents – C06

NAG®

NAG Fortran Library Manual, Mark 18

Printed and produced by NAG

1st Edition – September 1997 ISBN 1-85206-147-2

NAG Ltd
Wilkinson House
Jordan Hill Road
OXFORD
United Kingdom OX2 8DR

Tel: +44 (0)1865 511245
Fax: +44 (0)1865 310139

NAG GmbH
Schleißheimerstraße 5
D-85748 Garching
Deutschland

Tel: +49 (0)89 3207395
Fax: +49 (0)89 3207396

NAG Inc
1400 Opus Place, Suite 200
Downers Grove, IL 60515-5702
USA

Tel: +1 630 971 2337
Fax: +1 630 971 2706

NAG also has a number of distributors throughout the world. Please contact NAG for further details.

[*NP3086/18*]

Contents of the NAG Fortran Library Manual, Mark 18

Foreword to the NAG Fortran Library Manual

The following Foreword was contributed by the late Professor Fox and the late Dr Wilkinson to the NAG Fortran Library Manual which was released in 1975.

Those who have organised computing services are well aware of the two main problems which face the users of computing machines in scientific computation. First, considerable experience is needed before the user can transform a given algorithm into a very efficient program, and there are many examples in which relatively small amendments to a few instructions can transform a modest program into one considerably more economical in time and storage space. Second, our user needs knowledge of the principles and techniques of numerical analysis, however efficient he might be at program construction, before he can reasonably guarantee to have an efficient algorithm which is as free as possible from numerical instability and which gives good results in economic time. Both the cost of computation and the ever-present desire for quick results make obligatory at least a partial solution to these two problems.

Many computing laboratories and computing services have made some attempts at solution by constructing libraries of computer programs, but only in the last few years has it been possible to develop really comprehensive schemes based on two or more decades of research into methods and their error analysis by numerical mathematicians, and on the development of a new breed of expert in 'numerical software'. This NAG Fortran Library was in fact initiated by a small mixed university band of numerical analysts and their software counterparts, but has increasingly received encouragement, support and material from many 'extramural' organisations.

The compilers of this library have used, as main criteria for the selection of their programs, the concepts of (i) usefulness, (ii) robustness, (iii) numerical stability, (iv) accuracy and (v) speed. But within these criteria several rather difficult decisions have to be made. First, how many different routines are needed in each particular subject area, such as linear equations, optimization, ordinary differential equations, partial differential equations and so on? What is relevant here is the number of 'parameters' of the particular subject area. With linear equations, for example, the matrix might be 'dense' or have some particular 'sparse' structure, it might be symmetric and, if so, possibly positive definite, it might be too large for the high-speed store of some particular computer, it might be one for which an iterative method is known to converge, or the problem might involve the same matrix but have many different right-hand sides, and so on. Each of these sub-groups may require quite different routines for best efficiency, but within each sub-group there may also be several computing techniques requiring a further selection decision.

A second question which has to be answered is the nature and amount of material to be provided for the 'answer' to problems. If the data of the problem are exact, and if the problem has a unique solution, then it is meaningful to ask for results accurate to a specified number of figures. Whether one can get them easily, say with single-precision arithmetic, will depend on the sensitivity of the answers to small changes in the data. For even the storage of exact numbers cannot usually be performed exactly, so that from the outset our problem differs slightly from the one we hoped to solve. Moreover inevitable computer rounding errors will produce solutions which are the exact solutions of a perturbation of the original problem, the amount of the perturbation depending on the degree of stability of the numerical method. With so-called 'ill-conditioned' problems small perturbations from any of those sources produce large changes in the answers, so that 'exact' or very accurate solutions can be difficult to obtain even if they are meaningful.

But the data may not be known exactly. Some of them may be measured by physical apparatus or involve physical constants known with certainty only to a few figures. In that case the answers are meaningful only to a few figures and perhaps even to no figures, and whether the precision of the answers is larger or smaller than that of the data again depends on the degree of ill-conditioning of the problem. How much of this sort of information should the routines provide?

A third decision is the amount of explanation to be included with the programs. It is clearly desirable to include elements of 'why' something is done as well as 'what' is done, but the desirable amount of such information is rather delicate. If there is too much the expert may be too bored to read all of it and may therefore miss something important, while the amateur may find the discussion rather involved, appearing to him rather like an introductory text in numerical analysis, and again may skip most of it

but now on the grounds of indigestibility. Too little, on the other hand, may detract from the value of the routines by giving the amateur too little guidance in the choice which he also always has to make.

This NAG Fortran Library deals with these problems about as well as could be expected in the present state of knowledge of numerical analysts, software and library compilers, and the majority of the users. With regard to the number of routines to be provided it usually gives just the best available within each sub-group, and selects the particular sub-groups which at present seem to be the most needed and for which good techniques are available.

With regard to sensitivity and accuracy it achieves rather less, but this is a problem so far not well treated even by numerical analysts. Information is provided in a fairly economical way for the solution of linear equations, in which the so-called 'iterative refinement' involving a little double precision arithmetic gives valuable information on the sensitivity and a more accurate answer when this is meaningful. For many other problems the user can only obtain this sort of information by his own efforts, for example by deliberately introducing small perturbations and observing their effects on his solutions. This whole area is one in which one hopes for continual improvements in the library routines when better ways to implement them are discovered.

With regard to annotation, the routines do include a fair but not prohibitive amount of 'why' as well as 'what', and there is no doubt that a mastery of this material will enable the user not only to increase the value he gets from this library but also to improve his performance in the inevitable writing of his own routines for problems not directly treated here.

Two other topics are worth mentioning. First, the routines which appear in this library are the result of years of detailed study by numerical analysts and software experts, and it is dangerous in varying degrees to tamper with them and to try to modify them for 'local needs'. In the solution of linear equations, for example, one could without great peril omit the iterative refinement and still get useful results. One loses here just the extra but often extremely valuable knowledge about the 'condition' of the problem which iterative refinement gives comparatively economically. A far greater danger would arise from an attempt to 'speed-up' the routine by, for example, omitting the row interchanges. which are essentially unnecessary with exact arithmetic. Computer arithmetic is not exact, and this fact could cause complete rubbish in the solutions obtained by neglecting interchanges, which in this context ruins the stability of the numerical method.

Second, the library cannot help the user in the proper formulation of his problem. Given, for example, the problem of computing

$$I_r = e^{-1} \int_0^1 e^x x^r dx, \quad \text{for } r = 0, 1, 2, \ldots, 20$$

the library will have routines for evaluating this integral by numerical quadrature, to whatever accuracy is required, for each value of r. But nothing in the library can tell the user that a very much faster method would use the recurrence relation (in the 'backwards direction')

$$I_{r-1} = \frac{1 - I_r}{r}, \quad \text{with } I_N = 0,$$

where N (> 20) depends on the accuracy required but is determinable by simple and very rapid numerical experiment (and even, in this simple case, by elementary analysis). Nor could the library tell him that the perhaps more obvious use of the forward recurrence

$$I_r = 1 - rI_{r-1}, \quad \text{with } I_0 = 1 - e^{-1},$$

would fail to produce accurate results beyond the first few values of r with only single-precision arithmetic: that this formulation, in fact, gives a very ill-conditioned problem.

In summary, then, this NAG Fortran Library represents a timely and very important aid to the computer user in scientific computation. Here, and in future extensions, it provides the best available routines for a wide variety of numerical subject areas, backed by a non-prohibitive amount of sensible explanation of both what is being done and why it is being done. But the user must realise that the library can provide no more than it claims in its annotation, that it cannot except where explicitly stated determine for him the degree of ill-conditioning of his problem, nor help him in general to cast his problem into a better form. For such information he should study some numerical analysis or ask the advice of a colleague reasonably experienced in this field. It may happen that in future editions of the library it will be possible

to give more assistance of this kind to the general user, and it is our hope, in welcoming warmly this edition, that future productions will have some useful expansions of this kind, in addition to the obvious need for new routines in the subject areas which in this first venture are not touched upon or treated only sparsely. The research involved will be both exciting and fruitful!

Professor L Fox (Oxford University)

Dr J H Wilkinson, FRS (National Physical Laboratory, England)

Essential Introduction to the NAG Fortran Library

This document is essential reading for any prospective user of the Library.

Contents

1 The Library and its Documentation

1.1 Structure of the Library

The NAG Fortran Library is a comprehensive collection of Fortran **routines** for the solution of numerical and statistical problems. The word 'routine' is used to denote 'subroutine' or 'function'.

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.

D01 – Quadrature

Exceptionally, two chapters (H and S) have one-character names. (The chapters and their names are based on the ACM modified SHARE classification index [1].)

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 always appears as 'F' in the documentation, but may be changed to 'E' in some single-precision implementations (see Section 1.6).

The F06 Chapter (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. F06AAF (SROTG/DROTG). The names in brackets are the equivalent single and double precision BLAS names respectively. The F07 Chapter (Linear Equations (LAPACK)) and the F08 Chapter (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 (SGETRF/DGETRF). Details regarding these alternate names can be found in the relevant Chapter Introductions.

1.2 Structure of the Documentation

The **NAG Fortran Library Manual** is the principal printed form of documentation for the NAG Fortran Library. 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 and indexes are placed at the beginning of the Manual.

Each chapter consists of the following documents:

Chapter Contents, e.g. Contents – D01;

Chapter Introduction, e.g. Introduction – D01;

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. Exceptionally, some chapters (F06, X01, X02) do not have individual routine documents; instead, all the routines are described together in the Chapter Introduction.

In addition to the full printed Manual, NAG produces a printed **Introductory Guide**, which contains all the introductory material from the Manual, together with all the Chapter Contents and Chapter Introductions.

1.3 Alternative Forms of Documentation

NAG also provides machine-based documentation. The delivery mechanisms are specifically designed to provide information in a convenient form on a wide range of machines and operating systems.

NAG Fortran Library TextWare is a hypertext version of the full Manual, available on a variety of UNIX and Windows platforms.

The NAG Fortran On-line Information Supplement provides a system to assist with routine selection (NAG-Help) and also to display information derived from the NAG Fortran Library Manual. It is available in several forms, the principal ones being a generic UNIX version (suitable for use on a wide range of machines running UNIX) and Open VMS versions for Digital VAX and Digital Alpha.

Information from the On-line Information Supplement has also been made available in a Windows Help system and as a PC DOS browser form, and is included with the PC implementations.

1.4 Marks of the Library

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

At each Mark, the documentation of the Library is updated. You must make sure that your documentation has been updated to the same Mark as the Library software that you are using.

Marks are numbered, e.g. 15, 16, 17. The current Mark is 18.

The Library software may be updated between Marks to an intermediate maintenance level, in order to incorporate corrections. Maintenance levels are indicated by a letter following the Mark number, e.g. 18A, 18B, and so on (Mark 18 documentation supports all these maintenance levels).

1.5 Implementations of the Library

The NAG Fortran Library is available on many different computer systems. For each distinct system, an **implementation** of the Library is prepared by NAG, e.g. the Cray C-90 Unicos 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 DEC VAX range); it may also be specific to a particular operating system, Fortran compiler, or compiler option (e.g. scalar or vector mode).

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 3.4).

1.6 Precision of the Library

The NAG Fortran Library is developed in both **single precision** and **double precision** versions. REAL variables and arrays in the single precision version are replaced by DOUBLE PRECISION variables and arrays in the double precision version.

On most systems only one precision of the Library is available; the precision chosen is that which is considered most suitable in general for numerical computation (double precision on most systems).

On some systems both precisions are provided: in this case, the double precision routines have names ending in 'F' (as in the documentation), and the single precision routines have names ending in 'E'. Thus in DEC VAX/VMS implementations:

D01AJF is a routine in the double precision implementation;

D01AJE is the corresponding routine in the single precision implementation.

Whatever the precision, INTEGER variables (and elements of arrays) always occupy one numeric storage unit, that is the Fortran Library is **not** implemented using non-standard [7] integer storage, e.g. INTEGER*2.

1.7 Library Identification

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 (or A00AAE in some single precision implementations). This routine has no parameters; it simply outputs text to the NAG Library advisory message unit (see Section 2.4). An example of the output is:

```
*** Start of NAG Library implementation details ***
Implementation title: Sun(SPARC) Solaris
           Precision: double
        Product Code: FLSOL18D
                Mark: 18
*** End of NAG Library implementation details ***
```

(The product code can be ignored, except possibly when communicating with NAG; see Section 4.)

1.8 Fortran Language Standards

All routines in the Library conform to the ISO Fortran 90 Standard [8], except for the use of a double precision complex data type (usually COMPLEX*16) in some routines in Fortran 77 compiled double precision implementations of the Library – there is no provision for this data type in the old ANSI Standard Fortran 77 [7].

Many of the routines in the Library were originally written to conform to the earlier Fortran 66 standard [6], and their calling sequences contain a few parameters which are not strictly necessary in Fortran 77.

2 Using the Library

2.1 General Advice

A NAG Fortran 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 2 is with (b).

2.2 Programming Advice

The NAG Fortran Library and its documentation are designed on the assumption that users know how to write a calling program in Fortran.

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 3.3 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 a double precision implementation of the Library from a single precision program, or vice versa.

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?

Is your program in the same precision as the NAG Library routines to which your program is being linked?

Have NAG routine names been modified – if necessary – as described in Section 1.6 and Section 2.5?

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.

2.3 Error Handling and the Parameter IFAIL

NAG Fortran Library routines may detect various kinds of error, failure or warning conditions. Such conditions are handled in a systematic way by the Library. They fall roughly into three classes:

(i) an invalid value of a parameter on entry to a routine;
(ii) a numerical failure during computation (e.g. approximate singularity of a matrix, failure of an iteration to converge);
(iii) a warning that although the computation has been completed, the results cannot be guaranteed to be completely reliable.

All three classes are handled in the same way by the Library, and are all referred to here simply as 'errors'.

The error-handling mechanism uses the parameter IFAIL, which occurs as the last parameter in the calling sequence of most NAG Library routines. IFAIL serves two purposes:

(i) it allows users to specify what action a Library routine should take if it detects an error;
(ii) it reports the outcome of a call to a Library routine, either 'success' (IFAIL = 0) or 'failure' (IFAIL $\neq$ 0, with different values indicating different reasons for the failure, as explained in Section 6 of the routine document).

For the first purpose IFAIL **must** be assigned a value before calling the routine; since IFAIL is reset by the routine, it **must** be passed as a variable, not as an integer constant. Allowed values on entry are:

IFAIL = 0: an error message is output, and execution is terminated ('hard failure');

IFAIL = +1: execution continues without any error message;

IFAIL = −1: an error message is output, and execution continues.

The settings IFAIL = $\pm$1 are referred to as 'soft failure'.

The safest choice is to set IFAIL to 0, but this is not always convenient: some routines return useful results even though a failure (in some cases merely a warning) is indicated. However, if IFAIL is set to $\pm$1 on entry, it is **essential** for the program to test its value on exit from the routine, and to take appropriate action.

The specification of IFAIL in Section 5 of a routine document suggests a suitable setting of IFAIL for that routine.

For a full description of the error-handling mechanism, see Chapter P01.

Routines in Chapter F07 and Chapter F08 do **not** use the usual error handling mechanism; in order to preserve complete compatibility with LAPACK software, they have a diagnostic output parameter INFO which need not be set before entry. See the F07 or F08 Chapter Introduction for further details.

2.4 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 Chapter E04 and Chapter 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.

2.5 Auxiliary Routines

In addition to those Library routines which are documented and are intended to be called by users, the Library also contains many auxiliary routines. Details of all the auxiliary routines which are called directly or indirectly by any documented NAG Library routine are supplied to sites in machine-readable form with the Library software.

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.

In a single precision implementation in which the names of documented routines end in 'E', the names of auxiliary routines have their first three and last three characters interchanged, e.g.

BAZD01 is an auxiliary routine (corresponding to D01BAZ) called by D01BAE.

3 Using the Documentation

3.1 Using the Manual

The Manual is designed to serve the following functions:

- 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. The following may help you to find the chapter, and possibly the routine, which you need to solve your problem:

Contents Summary Mark 18	–	a structured list of routines in the Library, by chapter;
KWIC Index	–	a keyword index to chapters and routines;
GAMS 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 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.

3.2 Structure of Routine Documents

Note that at Mark 17 a new typesetting scheme was used to generate documentation. If you have a Manual which contains pre-Mark 17 routine documents, you will find that it contains older documents which differ in appearance, although the structure is the same.

Note also that at Mark 14 some changes were made to the style and appearance of routine documents. If you have a Manual which contains pre-Mark 14 routine documents, you will find that it contains older documents which differ in style, although they contain essentially the same information. Section 3.2, Section 3.3 and Section 3.5 of this Essential Introduction describe the **new-style** routine documents. Section 3.7 gives some details about the old-style 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 3.3 below)
6. **Error Indicators and Warnings**
7. **Accuracy**
8. **Further Comments**
9. **Example** (see Section 3.5 below)

In a few documents there are a further three sections:

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

3.3 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.

3.3.1 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 on or before entry to the routine; the routine may assign values to them.

Input/Output: you must assign values to these parameters on or 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.

External Procedure: a subroutine or function 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 2.5, 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.)

3.3.2 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 non-zero value of IFAIL (see Section 2.3).

In newer routine documents, constraints on parameters of type CHARACTER only list uppercase alphabetic characters, e.g.

Constraint: STRING = 'A' or 'B'.

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

The phrase '*Suggested Value:*' 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.

3.3.3 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 *Input*

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 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 only uses 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)
```

Fortran requires that the dimensions which occur in array declarations must be greater than zero. Many NAG routines are designed so that they can be called with a parameter like N in the above example set to 0 (in which case they would usually exit immediately without doing anything). If so, the declarations in the Library routine would use the 'assumed size' array dimension, and would be given as:

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

However, the original declaration of an array in your calling program must always have constant dimensions, greater than or equal to 1.

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

3.4 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 are the following; their interpretation depends on whether the implementation is in single precision or double precision:

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

Another important bold italicised term is ***machine precision*** which denotes the relative precision to which ***real*** 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 function X02AJF. Other functions 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 ***blocksize*** is used only in Chapter F07 and Chapter 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.

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 X02 routines;
- the default unit numbers for output (see Section 2.4);
- details of name changes for Library routines (see Section 1.6 and Section 2.5).

In Chapter F06, Chapter F07 and Chapter F08 where alternate routine names are available for BLAS and LAPACK derived routines the alternate name appears in bold italics – for example, ***sgetrf*** which should be interpreted as either SGETRF (in single precision) or DGETRF (in double precision) in the case of F07ADF, which handles real matrices. Similarly, F07ARF for complex matrices uses ***cgetrf*** which should be interpreted as either CGETRF (in single precision) or ZGETRF (in double precision).

3.5 Example Programs and Results

The **example program** in Section 9 of each routine document illustrates a simple call of the routine. The programs are designed so that they can fairly easily be modified, and so serve as the basis for a simple program to solve a user's own problem.

Bold italicised terms are used in the printed text of the example program, to denote precision-dependent features in the code. The correct Fortran code must therefore be substituted before the program can be run. In addition to the terms ***real*** and ***complex*** which were explained in Section 3.4, the following terms are used in the example programs:

Intrinsic Functions:	***real***	means	REAL	or	DBLE	(see Note below)
	imag	means	AIMAG	or	DIMAG	
	cmplx	means	CMPLX	or	DCMPLX	
	conjg	means	CONJG	or	DCONJG	
Edit Descriptor:	***e***	means	E	or	D	(in FORMAT statements)
Exponent Letter:	***e***	means	E	or	D	(in constants)

Note that in some implementations, the intrinsic function ***real*** with a ***complex*** argument must be interpreted as DREAL rather than DBLE.

The examples in Chapter F07 and Chapter F08 use the precision-dependent LAPACK routine names as mentioned in Section 3.4.

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 in this form to users.

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

The Users' Note for your implementation will mention any special changes which need to be made to the example programs, and any significant differences in the results.

3.6 Summary for New Users

If you are unfamiliar with the NAG Library and are thinking of using a routine from it, please follow these instructions:

(a) read the whole of the **Essential Introduction**;
(b) consult the **Library Contents** to choose an appropriate chapter or routine;
(c) or search through the **KWIC Index**, **GAMS Index** or via an on-line search facility;
(d) read the relevant **Chapter Introduction**;
(e) choose a routine, and read the **routine document**. If the routine does not after all meet your needs, return to steps (b) or (c);
(f) read the **Users' Note** for your implementation;
(g) consult local documentation, which should be provided by your local support staff, about access to the NAG Library on your computing system.

You should now be in a position to include a call to the routine in a program, and to attempt to 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 (f) can be omitted, but it is always essential to:

- be familiar with the Chapter Introduction;
- read the routine document;
- be aware of the Users' Note for your implementation.

3.7 Pre-Mark 14 Routine Documents

You need only read this section if you have an updated Manual which contains pre-Mark 14 documents.

You will find that older routine documents appear in a somewhat different style, or even several styles if your Manual dates back to Mark 7, say. The following are the most important differences between the earlier styles and the new style introduced at Mark 14:

- before Mark 12, routine documents had 13 sections: the extra sections have either been dropped or merged with the present Section 8 (Further Comments);
- in Section 5, parameters were not classified as *Input*, *Output* and so on; the phrase 'Unchanged on exit' was used to indicate an input parameter;

- the example programs were revised at Mark 12 and again at Mark 14, to take advantage of features of Fortran 77: the programs printed in older documents do not correspond exactly with those which are now distributed to sites in machine-readable form;
- before Mark 12, the printed example programs did not use bold italicised terms; they were written in standard single precision Fortran;
- before Mark 9, the printed example results were generated on an ICL 1906A (with approximately 11 digits of precision), and between Marks 9 and 12 they were generated on an ICL 2900 (with approximately 16 digits of precision);
- before Mark 13, documents referred to 'the appropriate implementation document'; this means the same as 'the Users' Note for your implementation'.

4 Contact between Users and NAG

For further advice or communication about the NAG Library, you should first turn to the staff of your local computer installation. This covers such matters as:

- obtaining a copy of the Users' Note for your implementation;
- obtaining information about local access to the Library;
- seeking advice about using the Library;
- reporting suspected errors in routines or documents;
- making suggestions for new routines or features;
- purchasing NAG documentation.

Your installation may have advisory and/or information services to handle such enquiries. In addition NAG asks each installation mounting the Library to nominate a NAG **site representative**, who may be approached directly in the absence of an advisory service. Site representatives receive information from NAG about confirmed errors, the imminence of updates, and so on, and will forward users' enquiries to the appropriate person in the NAG organisation if they cannot be dealt with locally. If you are unable to make contact with your local site representative please contact NAG directly. Full details of the NAG Response Centres are given in the Users' Note.

5 Further Information

General information about the NAG project is given in the document 'Development of NAG'. References [2], [3], [4], and [5] discuss various aspects of the design and development of the NAG Library, and NAG's technical policies and organisation.

Details of other NAG products and services, including other numerical subroutine libraries, are available via the NAG Response Centres or via the NAG Website.

6 References

[1] (1960–1976) Collected algorithms from ACM index by subject to algorithms

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

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

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

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

[6] (1966) USA standard Fortran *Publication X3.9* American National Standards Institute

[7] (1978) American National Standard Fortran *Publication X3.9* American National Standards Institute

[8] ISO Fortran 90 programming language (ISO 1539:1991)

Mark 15 News

1. New Features of Mark 15

Mark 15 represents a further considerable expansion of the NAG Fortran Library. It contains a total of 1045 documented routines, of which 167 are new at this Mark. Two new chapters have been introduced:

F07 – Linear Equations (LAPACK)
G12 – Survival Analysis

Out of 167 new routines, 98 are in the new chapter F07. This new chapter includes routines to compute factorizations, solutions, error bounds and condition numbers for real and complex linear systems; symmetric, Hermitian, positive-definite, general, band and triangular systems are provided for.

35 of the new routines are in the Statistics chapters. They include facilities (in the stated chapters) for:

- further statistical distribution functions (G01)
- factor analysis and discriminant analysis (G03)
- improved random number generators (G05)
- computation of confidence intervals; maximum likelihood estimates (G07)
- generation of a multivariate time series (G05)
- forecasting from a vector autoregressive moving average model, allowing for: transforming and/or differencing of a multivariate time series; testing for stationarity and invertibility (G13)

New routines have been introduced in other chapters of the Library for:

- solution of parabolic partial differential equations with coupled differential algebraic systems (D03)
- solution of symmetric positive-definite Toeplitz systems (F04)
- computation of matrix norms for different types of matrix (F06)
- mixed integer LP (H)
- special functions (S)
- machine specific numbers (X02)

For a number of existing routines (mainly in the linear algebra chapters) the underlying code has been replaced by calls to the new F07 routines or their auxiliaries. The modified versions can achieve much better performance on some machines.

2. New Routines

For details, please refer to the relevant chapter introductions and routine documents. Routines in the F06 and X02 chapters are described in the F06 and X02 Chapter Introductions; they do not have individual routine documents. A concise summary of the purpose of all documented routines in the Library is given in the document 'Contents Summary, Mark 15' (with the exception of routines which have been superseded).

The following 167 new routines are included in the NAG Fortran Library at Mark 15:

D03PCF	F07AHF	F07GWF	F07QRF	G01JDF
D03PDF	F07AJF	F07HDF	F07QSF	G01MBF
D03PHF	F07ARF	F07HEF	F07QUF	G02FCF
D03PJF	F07ASF	F07HGF	F07QVF	G03BAF
D03PYF	F07AUF	F07HHF	F07QWF	G03BCF
D03PZF	F07AVF	F07HRF	F07TEF	G03CAF
F04FEF	F07AWF	F07HSF	F07TGF	G03CCF
F04FFF	F07BDF	F07HUF	F07THF	G03DAF
F04MEF	F07BEF	F07HVF	F07TJF	G03DBF
F04MFF	F07BGF	F07MDF	F07TSF	G03DCF
F06RAF	F07BHF	F07MEF	F07TUF	G03ZAF
F06RBF	F07BRF	F07MGF	F07TVF	G05DRF
F06RCF	F07BSF	F07MHF	F07TWF	G05FEF
F06RDF	F07BUF	F07MJF	F07UEF	G05FFF
F06REF	F07BVF	F07MRF	F07UGF	G05HDF
F06RJF	F07FDF	F07MSF	F07UHF	G07AAF
F06RKF	F07FEF	F07MUF	F07UJF	G07ABF
F06RLF	F07FGF	F07MVF	F07USF	G07BBF
F06RMF	F07FHF	F07MWF	F07UUF	G07BEF
F06UAF	F07FJF	F07NRF	F07UVF	G07CAF
F06UBF	F07FRF	F07NSF	F07UWF	G08ALF
F06UCF	F07FSF	F07NUF	F07VEF	G12AAF
F06UDF	F07FUF	F07NVF	F07VGF	G13DJF
F06UEF	F07FVF	F07NWF	F07VHF	G13DKF
F06UFF	F07FWF	F07PDF	F07VSF	G13DLF
F06UGF	F07GDF	F07PEF	F07VUF	G13DMF
F06UHF	F07GEF	F07PGF	F07VVF	G13DNF
F06UJF	F07GGF	F07PHF	G01DHF	G13DXF
F06UKF	F07GHF	F07PJF	G01EAF	H02BZF
F06ULF	F07GJF	F07PRF	G01EMF	S21CAF
F06UMF	F07GRF	F07PSF	G01EPF	X02ANF
F07ADF	F07GSF	F07PUF	G01FAF	
F07AEF	F07GUF	F07PVF	G01FMF	
F07AGF	F07GVF	F07PWF	G01HBF	

3. Withdrawn Routines

The following routines have been withdrawn from the NAG Fortran Library at Mark 15. Warning of their withdrawal was included in the Mark 14 Library Manual, together with advice on which routines to use instead. The relevant Chapter Introduction documents give more detailed guidance.

Withdrawn Routine	Recommended Replacement
C02ADF	C02AFF
E01ACF	E01DAF and E02DEF
F01CDF	F01CTF
F01CEF	F01CTF
F01CGF	F01CTF
F01CHF	F01CTF
F01LZF	F02SWF and F02SXF
F01QAF	F01QCF
F01QBF	F01QJF
F02SZF	F02SYF
H02BAF	H02BBF

4. Routines Scheduled for Withdrawal

The routines listed below are scheduled for withdrawal from the NAG Fortran Library, because improved routines have now been included in the Library. Users are advised to stop using routines which are scheduled for withdrawal immediately and to use recommended replacement routines instead. The relevant chapter introduction documents give further guidance, including detailed advice on how to change a call to the old routine into a call to the new routine.

The following routines will be withdrawn at Mark 16:

Routine scheduled for withdrawal	Recommended Replacement
C02AEF	C02AGF
D03PAF	D03PCF
D03PBF	D03PCF
D03PGF	D03PCF
E02DBF	E02DEF
E04HBF	not needed except with E04JBF
E04JBF	E04UCF
E04KBF	E04UCF
F01ACF	F01ABF
F01BQF	F07GDF (SPPTRF/DPPTRF) or F07PDF (SPPTRF/DSPTRF)
F01CLF	F06YAF (SGEMM/DGEMM)
F02WAF	F02WEF
F04AQF	F07GEF (SPPTRS/DPPTRS) or F07PEF (SSPTRS/DSPTRS)
F06QGF	F06RAF, F06RCF and F06RJF
F06VGF	F06UAF, F06UCF and F06UJF
G01BAF	G01EBF
G01BBF	G01EDF
G01BCF	G01ECF
G01BDF	G01EEF
G01CAF	G01FBF
G01CBF	G01FDF
G01CCF	G01FCF
G01CDF	G01FEF
G02CJF	G02DAF and G02DGF
G05DGF	G05FFF
G05DLF	G05FEF
G05DMF	G05FEF
G08ABF	G08AGF
G08ADF	G08AHF, G08AKF and G08AJF
G08CAF	G08CBF
M01AJF	M01DAF, M01ZAF and M01CAF
M01AKF	M01DAF, M01ZAF and M01CAF
M01APF	M01CAF
X02AAF	X02AJF
X02ABF	X02AKF
X02ACF	X02ALF
X02AGF	X02AMF

The following routines have been superseded, but will not be withdrawn from the Library until Mark 17 at the earliest. They are being retained at Marks 15 and 16 because of their length of life in the Library, and to give users a longer time to make the transition to the new routines.

Superseded routine	Recommended Replacement
F01AAF	F07ADF (SGETRF/DGETRF) and F07AJF (SGETRI/DGETRI)
F01BNF	F07FRF (CPOTRF/ZPOTRF)
F01BPF	F07FRF (CPOTRF/ZPOTRF) and

	F07FWF (CPOTRI/ZPOTRI)
F01BTF	F07ADF (SGETRF/DGETRF)
F01BXF	F07FDF (SPOTRF/DPOTRF)
F01LBF	F07BDF (SGBTRF/DGBTRF)
F01NAF	F07BRF (CGBTRF/ZGBTRF)
F03AGF	F07HDF (SPBTRF/DPBTRF)
F03AHF	F07ARF (CGETRF/ZGETRF)
F03AMF	see F03 Chapter Introduction
F04AKF	F07ASF (CGETRS/ZGETRS)
F04ALF	F07HEF (SPBTRS/DPBTRS)
F04AWF	F07FSF (CPOTRS/ZPOTRS)
F04AYF	F07AEF (SGETRS/DGETRS)
F04AZF	F07FEF (SPOTRS/DPOTRS)
F04LDF	F07BEF (SGBTRS/DGBTRS)
F04NAF	F07BSF (CGBTRS/ZGBTRS)
G13DAF	G13DMF
X02CAF	not needed except with F01BTF and F01BXF

5. Routines Revised at Mark 15

5.1. Linear Algebra Routines

The following routines have had underlying code replaced to call new F07 routines or their auxiliaries. This has resulted in the addition of new error exits from these routines.

F01ADF	F03AAF	F03AFF	F04ADF	F04ATF
F01AEF	F03ABF	F03AHF	F04AEF	
F01BNF	F03ADF	F04AAF	F04ARF	
F01BPF	F03AEF	F04ABF	F04ASF	

5.2. Fast Fourier Transforms

Some minor revisions have been made to routines for fast Fourier transforms in Chapter C06, in order to improve their efficiency.

5.3. Ordinary Differential Equations

Some minor revisions have been made to auxiliaries of routines solving boundary value problems by shooting methods, namely D02HAF, D02HBF and D02SAF, in order to improve reliability.

5.4. Time Series Analysis

Some modifications have been made to G13 routines. An algorithmic improvement has been incorporated into G13DCF. The matrices computed in the argument C of G13DAF and R of G13DSF now return the correct results; previously they contained the transpose of the results.

NAG Fortran Library, Mark 16

FLSU416D

Sun 4 (SPARC) Double Precision

Users' Note

[*NP2660/UN*]

December 1993

Printed and produced by NAG®

NAG Ltd
Wilkinson House
Jordan Hill Road
OXFORD
United Kingdom OX2 8DR

Tel: +44 (0)865 511245
Fax: +44 (0)865 310139

NAG Ltd Response Centre
Tel: +44 (0)865 311744
Fax: +44 (0)865 311755
Email: infodesk@nag.co.uk

NAG Inc Response Center
Tel: +1 708 971 2345
Fax: +1 708 971 2346
Email: infodesk@nag.com

NAG GmbH
Schleißheimerstraße 5
D-85748 Garching
Deutschland

Tel: +49 (0)89 3207395
Fax: +49 (0)89 3207396

NAG Inc
1400 Opus Place, Suite 200
Downers Grove, IL 60515-5702
USA

Tel: +1 708 971 2337
Fax: +1 708 971 2706

NAG also has a number of distributors throughout the world. Please contact NAG for further details.

NAG Fortran Library, Mark 16

FLSU416D

Sun 4 (SPARC) Double Precision

Users' Note

Contents

1. Introduction

This document is essential reading for every user of the NAG Fortran Library Implementation specified in the title. It provides implementation-specific detail that augments the information provided in the NAG Fortran Library Manual and Introductory Guide. Wherever those manuals refer to the "Users' Note for your implementation", the user should consult this note. A tabbed divider has been provided near the beginning of Volume 1 of the NAG Fortran Library Manual, indicating a suitable place to store the document, should you so wish.

NAG recommends that users read the following minimum reference material before calling any library routine:

(a) Essential Introduction
(b) Chapter Introduction
(c) Routine Document
(d) Implementation-specific Users' Note

Items (a), (b) and (c) are included in the NAG Fortran Library Manual; items (a) and (b) are also included in the NAG Fortran Library Introductory Guide; item (d) is this document. Items (a) and (d) are also supplied on the distribution medium. Each NAG Fortran Library Service site is supplied with at least one copy of each of the above. Please ask your NAG Site Contact for further information.

2. Availability of Routines

All routines listed in the chapter contents documents of the NAG Fortran Library Manual, Mark 16 are available in this implementation. At Mark 16, 126 new primary ('user-callable') routines have been introduced, and 34 deleted. Please consult the document FORTRAN MARK 16 NEWS in the manual for lists of these routines and for a list of routines scheduled for withdrawal at Mark 17 or later. The file news supplied to your site on the library distribution medium (see Section 3.5) also gives an outline of the new material available.

3. General Information

3.1. Accessing the Library

Assuming that libnag.a has been installed in /usr/local/lib or /usr/lib, then a user may link to the NAG Fortran Library in the following manner:

```
f77 driver.f -lnag
```

where driver.f is the user's application program.

3.2. Interpretation of Bold Italicised Terms

For this double precision implementation, the bold italicised terms used in the NAG Fortran Library Manual (and shown here as //.....///) should be interpreted as:

```
//real//                  - DOUBLE PRECISION (REAL*8)
//basic precision//       - double precision
//complex//               - COMPLEX*16
//additional precision// - quadruple precision (REAL*16)
//machine precision//     - the machine precision, see the value
                            returned by X02AJF in Section 4, below.
```

Thus a parameter described as //real// should be declared as DOUBLE PRECISION in the user's program. If a routine accumulates an inner product in //additional precision//, it is using software to simulate quadruple precision.

In routine documents that have been newly typeset since Mark 12 additional bold italicised terms are used in the published example programs and they must be interpreted as follows:

```
//real// as an intrinsic function name - DBLE
//imag//                                - DIMAG
//cmplx//                               - DCMPLX
//conjg//                               - DCONJG
//e// in constants, e.g. 1.0//e//-4     - D, e.g. 1.0D-4
//e// in formats, e.g. //e//12.4        - D, e.g. D12.4
```

All references to routines in Chapter F07 – Linear Equations (LAPACK) and Chapter F08 – Least-squares and Eigenvalue Problems (LAPACK) use the LAPACK name, not the NAG F07/F08 name. The LAPACK name is precision dependent, and hence the name appears in a bold italicised typeface.

The typeset examples use the single precision form of the LAPACK name. To convert this name to its double precision form, change the first character either from S to D or C to Z as appropriate. For example:

```
//sgetrf// refers to the LAPACK routine name - DGETRF
//cpotrs//                                   - ZPOTRS
```

3.3. Example Programs

In the NAG Fortran Library Manual, routine documents that have been typeset since Mark 12 present the example programs in a generalised form, using bold italicised terms as described in Section 3.2.

In other routine documents, the example programs are in single precision and require modification for use with double precision routines. This conversion can entail:

- Changing REAL or COMPLEX type specifications to REAL*8 or COMPLEX*16;
- Changing certain intrinsic function references, e.g. REAL or FLOAT to DBLE, ALOG to DLOG, CMPLX to DCMPLX, and so on;
- Changing real constants to double precision form, e.g. 0.1 or 0.1E0 to 0.1D0.

The example programs supplied to a site in machine-readable form have been modified as necessary so that they are suitable for immediate execution. Note that all the distributed

example programs have been revised and do not correspond exactly with the programs published in the manual, unless the documents have been recently typeset. The distributed example programs should be used in preference wherever possible.

3.4. Explicit Output from NAG Routines

Certain routines produce explicit error messages and advisory messages via output units which either have default values or can be reset by using X04AAF for error messages and X04ABF for advisory messages. (The default values are given in Section 4). The maximum record lengths of error messages and advisory messages (including carriage control characters) are 80 characters, except where otherwise specified.

3.5. User Documentation

The following machine-readable information files are provided by NAG on the library distribution medium. Please consult your local advisory service or NAG Site Contact for further details:

`un`	- Users' Note (this document)
`essint`	- the Essential Introduction to the NAG Fortran Library
`summary`	- a brief summary of the routines
`news`	- an outline of the new and enhanced routines available at Mark 16
`replaced`	- a list of routines available at earlier Marks of the Library but since withdrawn, together with recommended replacements
`calls`	- a list of routines called directly or indirectly by each routine in the library, and by each example program
`called`	- for each routine in the library (including auxiliaries), a list of routines and example programs which call it directly or indirectly

See Section 5 for additional documentation available from NAG.

4. Routine-specific Information

Any further information which applies to one or more routines in this implementation is listed below, chapter by chapter.

(a) F01, F04

The following routines call the function X02CAF for an estimate of the amount of actual (as opposed to virtual) store available, in order to reduce the amount of paging when solving large problems:

`F01BTF` `F01BXF` `F04AYF` `F04AZF`

See X02CAF below for further information.

(b) G02

The value of ACC, the machine-dependent constant mentioned in several documents in the chapter is approximately 1.0D-13.

(c) P01

On hard failure, P01ABF writes the error message to the error message unit specified by X04AAF, and then terminates execution.

(d) S07 – S21

The constants referred to in the NAG Fortran Library Manual have the following values in this implementation:

```
S07AAF  F(1)    = 1.0D+13
        F(2)    = 1.0D-14

S10AAF  E(1)    = 18.50
S10ABF  E(1)    = 708.0
S10ACF  E(1)    = 708.0

S13AAF  x(hi)   = 708.3
S13ACF  x(hi)   = 1.0D+16
S13ADF  x(hi)   = 1.0D+17

S14AAF  IFAIL   = 1 if X > 170.0
        IFAIL   = 2 if X < -170.0
        IFAIL   = 3 if abs(X) < 2.23D-308
S14ABF  IFAIL   = 2 if X > 2.55D+305

S15ADF  x(hi)   = 26.6
        x(low)  = -6.25
S15AEF  x(hi)   = 6.25

S17ACF  IFAIL   = 1 if X > 1.0D+16
S17ADF  IFAIL   = 1 if X > 1.0D+16
        IFAIL   = 3 if 0.0 < X <= 2.23D-308
S17AEF  IFAIL   = 1 if abs(X) > 1.0D+16
S17AFF  IFAIL   = 1 if abs(X) > 1.0D+16
S17AGF  IFAIL   = 1 if X > 103.8
        IFAIL   = 2 if X < -5.6D+10
S17AHF  IFAIL   = 1 if X > 104.1
        IFAIL   = 2 if X < -5.6D+10
S17AJF  IFAIL   = 1 if X > 104.1
        IFAIL   = 2 if X < -1.8D+09
S17AKF  IFAIL   = 1 if X > 104.1
        IFAIL   = 2 if X < -1.8D+09
S17DCF  IFAIL   = 2 if abs (Z) < 3.93D-305
        IFAIL   = 4 if abs (Z) or FNU+N-1 > 3.27D+04
        IFAIL   = 5 if abs (Z) or FNU+N-1 > 1.07D+09
S17DEF  IFAIL   = 2 if imag (Z) > 700.0
        IFAIL   = 3 if abs (Z) or FNU+N-1 > 3.27D+04
        IFAIL   = 4 if abs (Z) or FNU+N-1 > 1.07D+09
S17DGF  IFAIL   = 3 if abs (Z) > 1.02D+03
        IFAIL   = 4 if abs (Z) > 1.04D+06
```

```
S17DHF  IFAIL  = 3 if abs (Z) > 1.02D+03
        IFAIL  = 4 if abs (Z) > 1.04D+06
S17DLF  IFAIL  = 2 if abs (Z) < 3.93D-305
        IFAIL  = 4 if abs (Z) or FNU+N-1 > 3.27D+04
        IFAIL  = 5 if abs (Z) or FNU+N-1 > 1.07D+09

S18ADF  IFAIL  = 2 if 0.0 < X <= 2.23D-308
S18AEF  IFAIL  = 1 if abs(X) > 711.6
S18AFF  IFAIL  = 1 if abs(X) > 711.6
S18CDF  IFAIL  = 2 if 0.0 < X <= 2.23D-308
S18DCF  IFAIL  = 2 if abs (Z) < 3.93D-305
        IFAIL  = 4 if abs (Z) or FNU+N-1 > 3.27D+04
        IFAIL  = 5 if abs (Z) or FNU+N-1 > 1.07D+09
S18DEF  IFAIL  = 2 if real (Z) > 700.0
        IFAIL  = 3 if abs (Z) or FNU+N-1 > 3.27D+04
        IFAIL  = 4 if abs (Z) or FNU+N-1 > 1.07D+09

S19AAF  IFAIL  = 1 if abs(x) >= 49.50
S19ABF  IFAIL  = 1 if abs(x) >= 49.50
S19ACF  IFAIL  = 1 if X > 997.26
S19ADF  IFAIL  = 1 if X > 997.26

S21BCF  IFAIL  = 3 if an argument < 1.579D-205
        IFAIL  = 4 if an argument >= 3.774D+202
S21BDF  IFAIL  = 3 if an argument < 2.820D-103
        IFAIL  = 4 if an argument >= 1.404D+102
```

(e) X01

The values of the mathematical constants are:

```
X01AAF (PI)    = 3.1415926535897932
X01ABF (GAMMA) = 0.5772156649015329
```

(f) X02

The values of the machine constants are:

The basic parameters of the model

```
X02BHF = 2
X02BJF = 53
X02BKF = -1021
X02BLF = 1024
X02DJF = .TRUE.
```

Derived parameters of floating-point arithmetic

```
X02AJF = 1.11022302462516D-16
X02AKF = 2.22507385850721D-308
X02ALF = 1.79769313486231D+308
X02AMF = 2.22507385850721D-308
X02ANF = 2.22507385850721D-308
```

Parameters of other aspects of the computing environment

```
X02AHF = 1.42724769270596D+45
X02BBF = 2147483647
X02BEF = 15
X02CAF = 50000
X02DAF = .FALSE.
```

X02CAF

This function is called by F01BTF, F01BXF, F04AYF and F04AZF. It does not affect the numerical results returned by those routines, but merely the amount of paging performed when using these routines to operate on large matrices when running on a paged system. The constant value returned by the version of X02CAF in the compiled library should give a performance close to the optimum in almost all circumstances, but users who wish to override this value (e.g. if their jobs are run in a special session with an untypically large amount of actual store available) may do so by supplying their own function of the same name, to be compiled and loaded with their main program. Such a function should have the form:

```
INTEGER FUNCTION X02CAF(X)
DOUBLE PRECISION X
X02CAF = new value
RETURN
END
```

(g) X04

The default output units for error and advisory messages for those routines which can produce explicit output are Fortran Units 0 and 6, respectively.

(h) X05

The finest granularity of wall-clock time available on this system is one second, so the seventh element of the integer array passed as a parameter to X05AAF will always be returned with the value 0.

5. Additional Services from NAG

(a) Printed Manuals

Where a manual has been provided as part of the contract issue, this manual is updated automatically at each new release of the software, by the supply of a manual update set or a complete new manual. If additional manuals have been ordered in the past then updates to these manuals may be ordered separately. They are NOT sent automatically. Additional complete manuals and the manual updates (where relevant) are available at prices published in the NAG documentation order form.

(b) On-line Information Supplement

To complement the manuals NAG produces an On-line Information Supplement which fulfils two roles:

- it gives key-word-driven guidance on the selection of the appropriate NAG routine
- it gives abbreviated on-line documentation of the NAG routines, to enable the user to call the routines and investigate any IFAIL messages without recourse to the manual.

6. Support from NAG

(a) Contact with NAG

Queries concerning this document or the implementation generally should be directed initially to your local Advisory Service. If you have difficulty in making contact locally, you can write to NAG directly, at one of the supplied addresses. Users subscribing to the support service are encouraged to contact one of the NAG Response Centres (see below).

(b) NAG Response Centres

The NAG Response Centres are available for general enquiries from all users and also for technical queries from sites with an annually licensed product or support service.

The Response Centres are open during office hours, but contact is possible by fax, email and phone (answering machine) at all times.

When contacting a Response Centre please quote your NAG user reference and NAG product code.

(c) Network

Network, NAG's newsletter, is produced quarterly and sent free of charge to sites with a supported product or service.

(d) NAG Bulletin Board

The NAG Bulletin Board 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; new features are added and special interest groups are continually developing.

7. NAG Users Association

NAGUA, the NAG Users Association, is a self-financing, non-profitmaking body. It exists to promote communications between NAG and users of its products and services. It provides information to NAG on the requirements of users, who are in turn kept informed of developments in services. Membership is available to any institution or individual which holds a licence for any NAG product or service.

Members receive discounts on the registration fees at conferences and workshops. Members also receive 'NAGUA News', NAGUA's own newsletter.

For an information pack and membership application form, please contact NAGUA at the supplied address.

Mark 17 News

1 New Features of Mark 17

A variety of new facilities and improvements in areas of exisiting coverage are the main features of Mark 17 of the NAG Fortran Library. It contains a total of 1152 documented routines, of which 43 are new at this Mark. One new chapter has been introduced:

F11 – Sparse Linear Algebra

A new 3-D complex discrete Fourier transform routine is included in the C06 chapter.

Coverage in the differential equations chapters has been extended with (in the stated chapters):

- new collocation methods for the boundary value problem in ordinary differential equations (D02)
- an upwind scheme using a numerical flux function based on a Riemann solver for first order partial differential equations in conservative form in one spatial dimension (with remeshing facilities and coupled differential algebraic systems) (D03)

An enhanced routine for computing the minimum of a sum of squares is included in the E04 chapter.

There are eight new Black Box routines, which exploit existing F08 routines, for computing (in the stated chapters):

- selected eigenvalues and eigenvectors of real nonsymmetric, real symmetric, complex nonsymmetric and complex Hermitian matrices (F02)
- real and complex equality-constrained linear least-squares, and real and complex Gauss-Markov linear models (including weighted least-squares) (F04)

Ten routines are in the new chapter F11. Its current coverage addresses the solution of real sparse symmetric linear systems using iterative techniques. The chapter will be extended to treat nonsymmetric systems at the next Mark. It includes conjugate gradient and Lanczos algorithms, incomplete Cholseky, SSOR and Jacobi preconditioning, and support routines.

Thirteen of the new routines are in the Statistics chapters. They include facilities (in the stated chapters) for:

- partial corrleations (G02)
- scaling (G03)
- analysis of variance (G04)
- multiway table analysis (G11)
- Cox's proportional hazard (G12)
- Kalman filter (G13)

2 New Routines

For details, please refer to the relevant chapter introductions and routine documents. A concise summary of the purpose of all documented routines in the Library is given in the document Contents Summary, Mark 17 (with the exception of routines which have been superseded).

The following 43 new routines are included in the NAG Fortran Library at Mark 17:

C06FXF	D02TKF	D02TVF	D02TXF	D02TYF	D02TZF
D03PFF	D03PLF	D03PSF	D03PUF	D03PVF	E04UNF
F02ECF	F02FCF	F02GCF	F02HCF	F04JLF	F04JMF
F04KLF	F04KMF	F11GAF	F11GBF	F11GCF	F11JAF
F11JBF	F11JCF	F11JDF	F11JEF	F11XEF	F11ZBF
G02BYF	G03FAF	G03FCF	G04BCF	G04DAF	G04DBF
G04EAF	G11BAF	G11BBF	G11BCF	G12BAF	G13EAF
G13EBF					

3 Withdrawn Routines

The following routines have been withdrawn from the NAG Fortran Library at Mark 17. Warning of their withdrawal was included in the Mark 16 Library Manual, together with advice on which routines to use instead. See the document Advice on Replacement Calls for Superseded/Withdrawn Routines for more detailed guidance.

Withdrawn Routine	**Recommended Replacement**
D02QDF	D02NBF or D02NCF
D02QQF	not needed except with D02QDF
D03PAF	D03PCF
D03PBF	D03PCF
D03PGF	D03PCF
E04VCF	E04UCF
E04VDF	E04UCF
F01AAF	F07ADF (SGETRF/DGETRF) and F07AJF (SGETRI/DGETRI)
F01BNF	F07FRF (CPOTRF/ZPOTRF)
F01BPF	F07FRF (CPOTRF/ZPOTRF) and F07FWF (CPOTRI/ZPOTRI)
F01BXF	F07FDF (SPOTRF/DPOTRF)
F01NAF	F07BRF (CGBTRF/ZGBTRF)
F03AGF	F07HDF (SPBTRF/DPBTRF)
F03AHF	F07ARF (CGETRF/ZGETRF)
F03AMF	none – see Chapter Introduction
F04AKF	F07ASF (CGETRS/ZGETRS)
F04ALF	F07HEF (SPBTRS/DPBTRS)
F04AWF	F07FSF (CPOTRS/ZPOTRS)
F04AZF	F07FEF (SPOTRS/DPOTRS)
F04NAF	F07BSF (CGBTRS/ZGBTRS)
G04ADF	G04BBF
G04AEF	G04BBF
G04AFF	G04CAF
G13DAF	G13DMF
X02CAF	not needed except with F01BTF and F01BXF

4 Routines Scheduled for Withdrawal

The routines listed below are scheduled for withdrawal from the NAG Fortran Library, because improved routines have now been included in the Library. Users are advised to stop using routines which are scheduled for withdrawal immediately and to use recommended replacement routines instead. See the document Advice on Replacement Calls for Superseded/Withdrawn Routines for more detailed guidance, including detailed advice on how to change a call to the old routine into a call to the new routine.

The following routines will be withdrawn at Mark 18:

Routine Scheduled for Withdrawal	**Recommended Replacement**
D02BAF	D02PCF and associated D02P routines
D02BBF	D02PCF and associated D02P routines
D02BDF	D02PCF and associated D02P routines
D02CAF	D02CJF
D02CBF	D02CJF
D02CGF	D02CJF
D02CHF	D02CJF
D02EAF	D02EJF
D02EBF	D02EJF
D02EGF	D02EJF
D02EHF	D02EJF
D02PAF	D02PDF and associated D02P routines
D02XAF	D02PXF and associated D02P routines

D02XBF	D02PXF and associated D02P routines
D02YAF	D02PDF and associated D02P routines
E04MBF	E04MFF
E04NAF	E04NFF
F01AEF	F07FDF (SPOTRF/DPOTRF) and F08SEF (SSYGST/DSYGST)
F01AFF	F06YJF (STRSM/DTRSM)
F01AGF	F08FEF (SSYTRD/DSYTRD)
F01AHF	F08FGF (SORMTR/DORMTR)
F01AJF	F08FEF (SSYTRD/DSYTRD) and F08FFF (SORGTR/DORGTR)
F01AKF	F08NEF (SGEHRD/DGEHRD)
F01ALF	F08NGF (SORMHR/DORMHR)
F01AMF	F08NSF (CGEHRD/ZGEHRD)
F01ANF	F08NTF (CUNMHR/ZUNMHR)
F01APF	F08NFF (SORGHR/DORGHR)
F01ATF	F08NHF (SGEBAL/DGEBAL)
F01AUF	F08NJF (SGEBAK/DGEBAK)
F01AVF	F08NVF (CGEBAL/ZGEBAL)
F01AWF	F08NWF (CGEBAK/ZGEBAK)
F01AXF	F08BEF (SGEQPF/CGEQPF)
F01AYF	F08GEF (SSPTRD/DSPTRD)
F01AZF	F08GGF (SOPMTR/DOPMTR)
F01BCF	F08FSF (CHETRD/ZHETRD) and F08FTF (CUNGTR/ZUNGTR)
F01BDF	F07FDF (SPOTRF/DPOTRF) and F08SEF (SSYGST/DSYGST)
F01BEF	F06YFF (STRMM/DTRMM)
F01BTF	F07ADF (SGETRF/DGETRF)
F01BWF	F08HEF (SSBTRD/DSBTRD)
F01LBF	F07BDF (SGBTRF/DGBTRF)
F01QCF	F08AEF (SGEQRF/DGEQRF)
F01QDF	F08AGF (SORMQR/DORMQR)
F01QEF	F08AFF (SORGQR/DORGQR)
F01QFF	F08BEF (SGEQPF/DGEQPF)
F01RCF	F08ASF (CGEQRF/ZGEQRF)
F01RDF	F08AUF (CUNMQR/ZUNMQR)
F01REF	F08ATF (CUNGQR/ZUNGQR)
F01RFF	F08BSF (CGEQPF/ZGEQPF)
F02AAF	F02FAF
F02ABF	F02FAF
F02ADF	F02FDF
F02AEF	F02FDF
F02AFF	F02EBF
F02AGF	F02EBF
F02AJF	F02GBF
F02AKF	F02GBF
F02AMF	F08JEF (SSTEQR/DSTEQR)
F02ANF	F08PSF (CHSEQR/ZHSEQR)
F02APF	F08PEF (SHSEQR/DHSEQR)
F02AQF	F08PEF (SHSEQR/DHSEQR) and F08QKF (STREVC/DTREVC)
F02ARF	F08PSF (CHSEQR/ZHSEQR) and F08QXF (CTREVC/ZTREVC)
F02AVF	F08JFF (SSTERF/DSTERF)
F02AWF	F02HAF
F02AXF	F02HAF
F02AYF	F08JSF (CSTEQR/ZSTEQR)
F02BEF	F08JJF (SSTEBZ/DSTEBZ) and F08JKF (SSTEIN/DSTEIN)
F02BFF	F08JJF (SSTEBZ/DSTEBZ)
F02BKF	F08PKF (SHSEIN/DHSEIN)
F02BLF	F08PXF (CHSEIN/ZHSEIN)
F02SWF	F08KEF (SGEBRD/DGEBRD)
F02SXF	F08KFF (SORGBR/DORGBR) or F08KGF (SORMBR/DORMBR)

F02SYF	F08MEF (SBDSQR/DBDSQR)
F02UWF	F08KSF (CGEBRD/ZGEBRD)
F02UXF	F08KTF (CUNGBR/ZUNGBR) or F08KUF (CUNMBR/ZUNMBR)
F02UYF	F08MSF (CBDSQR/ZBDSQR)
F04ANF	F08AGF (SORMQR/DORMQR) and F06PJF (STRSV/DTRSV)
F04AYF	F07AEF (SGETRS/DGETRS)
F04LDF	F07BEF (SGBTRS/DGBTRS)
G01CEF	G01FAF

The following routines have been superseded, but will not be withdrawn from the Library until Mark 19 at the earliest. They are being retained at Marks 17 and 18 because of their length of life in the Library, and to give users a longer time to make the transition to the new routines.

Superseded routine	**Recommended Replacement**
E04UPF	E04UNF
F01MAF	F11JAF
F02BBF	F02FCF
F02BCF	F02ECF
F02BDF	F02GCF
F04MAF	F11JCF
F04MBF	F11GAF, F11GBF and F11GCF (or F11JCF or F11JEF)

5 Routines Revised at Mark 17

5.1 Optimization

Improvements to the diagnositic facilties of E04JAF and E04KAF have been made.

5.2 Linear Algebra

A minor correction has been made to F06ZPF (CHERK/ZHERK) and F06ZRF(CHERK2/ZHERK2).

Algorithmic improvements have been made to F08MEF (SBDSQR/DBDSQR) and F08MSF (CBDSQR/ZBDSQR) to improve the efficiency of the computation of singular values and to increase the relative accuracy obtained for small singular values.

5.3 Statistics

The options available in G02BXF and G03AAF have been extended.

Algorithmic improvements to G02DAF have been made to improve the efficiency of the computation of leverages.

In G04BBF the computation of treatment means for non-orthogonal designs has been adjusted to correspond to least-squares means.

In G08AKF a correction has been made to the way that the required probability is calculated from the computed lower tail probability.

Mark 18 News

1 New Features of Mark 18

At Mark 18 of the Fortran Library a variety of new facilities have been introduced and improvements made in existing areas. The Library now contains a total of 1108 documented routines, of which 35 are new at this Mark. These extend the areas of ordinary differential equations (ODEs), partial differential equations (PDEs), interpolation, optimization, sparse linear algebra and operations research (OR).

Coverage in the differential equations chapters (Chapter D02 and Chapter D03) has been extended as follows:

- simple driver interface for the integration of a system of first order ODEs using a fixed order Runge–Kutta method until a user-specified function is zero (Chapter D02)
- new approximate and exact Riemann solvers for 1-D Euler equations (Chapter D03)
- solution of time dependent second order PDEs in 2-D using adaptive mesh refinement (Chapter D03)

New routines for generating and evaluating interpolants to 2-D and 3-D scattered data sets (using a modified Shepard interpolant) are included in the interpolation chapter (Chapter E01).

The most significant additions to the optimization chapter (Chapter E04) are as follows:

- new routines to solve sparse LP and QP problems (including MPSX data-handling capabilities)
- new routines for unconstrained minimization with an extended parameter list to replace existing routines with reserved names
- a new reverse communication routine for constrained minimization using a sequential quadratic programming method

Coverage in the sparse linear algebra chapter (Chapter F11) has been extended to provide iterative methods and preconditioners for real nonsymmetric linear systems of equations.

A new routine for finding the shortest path through a network is included in the operations research chapter (Chapter H).

2 New Routines

For details, please refer to the relevant chapter introductions and routine documents. A concise summary of the purpose of all documented routines in the Library (with the exception of routines which have been superseded) is given in the document 'Library Contents, Mark 18'.

The following 35 new user-callable routines are included in the NAG Fortran Library at Mark 18:

D02BJF	D03PWF	D03PXF	D03RAF	D03RBF	D03RYF
D03RZF	E01SGF	E01SHF	E01TGF	E01THF	E04FYF
E04GYF	E04GZF	E04HYF	E04JYF	E04KYF	E04KZF
E04LYF	E04MZF	E04NKF	E04NLF	E04NMF	E04UFF
F11BAF	F11BBF	F11BCF	F11DAF	F11DBF	F11DCF
F11DDF	F11DEF	F11XAF	F11ZAF	H03ADF	

3 Withdrawn Routines

The following routines have been withdrawn from the NAG Fortran Library at Mark 18. Warning of their withdrawal was included in the Mark 17 Library Manual, together with advice on which routines to use instead. See the document 'Advice on Replacement Calls for Superseded/Withdrawn Routines' for more detailed guidance.

Withdrawn Routine	Recommended Replacement
D02BAF	D02PCF and associated D02P routines
D02BBF	D02PCF and associated D02P routines
D02BDF	D02PCF and associated D02P routines
D02CAF	D02CJF
D02CBF	D02CJF
D02CGF	D02CJF
D02CHF	D02CJF
D02EAF	D02EJF
D02EBF	D02EJF
D02EGF	D02EJF
D02EHF	D02EJF
D02PAF	D02PDF and associated D02P routines
D02XAF	D02PXF and associated D02P routines
D02XBF	D02PXF and associated D02P routines
D02YAF	D02PDF and associated D02P routines
E04MBF	E04MFF
E04NAF	E04NFF
F01AEF	F07FDF (SPOTRF/DPOTRF) and F08SEF (SSYGST/DSYGST)
F01AFF	F06YJF (STRSM/DTRSM)
F01AGF	F08FEF (SSYTRD/DSYTRD)
F01AHF	F08FGF (SORMTR/DORMTR)
F01AJF	F08FEF (SSYTRD/DSYTRD) and F08FFF (SORGTR/DORGTR)
F01AKF	F08NEF (SGEHRD/DGEHRD)
F01ALF	F08NGF (SORMHR/DORMHR)
F01AMF	F08NSF (CGEHRD/ZGEHRD)
F01ANF	F08NTF (CUNMHR/ZUNMHR)
F01APF	F08NFF (SORGHR/DORGHR)
F01ATF	F08NHF (SGEBAL/DGEBAL)
F01AUF	F08NJF (SGEBAK/DGEBAK)
F01AVF	F08NVF (CGEBAL/ZGEBAL)
F01AWF	F08NWF (CGEBAK/ZGEBAK)
F01AXF	F08BEF (SGEQPF/CGEQPF)
F01AYF	F08GEF (SSPTRD/DSPTRD)
F01AZF	F08GGF (SOPMTR/DOPMTR)
F01BCF	F08FSF (CHETRD/ZHETRD) and F08FTF (CUNGTR/ZUNGTR)
F01BDF	F07FDF (SPOTRF/DPOTRF) and F08SEF (SSYGST/DSYGST)
F01BEF	F06YFF (STRMM/DTRMM)
F01BTF	F07ADF (SGETRF/DGETRF)
F01BWF	F08HEF (SSBTRD/DSBTRD)
F01LBF	F07BDF (SGBTRF/DGBTRF)
F01QCF	F08AEF (SGEQRF/DGEQRF)
F01QDF	F08AGF (SORMQR/DORMQR)
F01QEF	F08AFF (SORGQR/DORGQR)
F01QFF	F08BEF (SGEQPF/DGEQPF)
F01RCF	F08ASF (CGEQRF/ZGEQRF)
F01RDF	F08AUF (CUNMQR/ZUNMQR)
F01REF	F08ATF (CUNGQR/ZUNGQR)
F01RFF	F08BSF (CGEQPF/ZGEQPF)
F02AAF	F02FAF
F02ABF	F02FAF
F02ADF	F02FDF
F02AEF	F02FDF
F02AFF	F02EBF
F02AGF	F02EBF
F02AJF	F02GBF
F02AKF	F02GBF
F02AMF	F08JEF (SSTEQR/DSTEQR)

F02ANF	F08PSF (CHSEQR/ZHSEQR)
F02APF	F08PEF (SHSEQR/DHSEQR)
F02AQF	F08PEF (SHSEQR/DHSEQR) and F08QKF (STREVC/DTREVC)
F02ARF	F08PSF (CHSEQR/ZHSEQR) and F08QXF (CTREVC/ZTREVC)
F02AVF	F08JFF (SSTERF/DSTERF)
F02AWF	F02HAF
F02AXF	F02HAF
F02AYF	F08JSF (CSTEQR/ZSTEQR)
F02BEF	F08JJF (SSTEBZ/DSTEBZ) and F08JKF (SSTEIN/DSTEIN)
F02BFF	F08JJF (SSTEBZ/DSTEBZ)
F02BKF	F08PKF (SHSEIN/DHSEIN)
F02BLF	F08PXF (CHSEIN/ZHSEIN)
F02SWF	F08KEF (SGEBRD/DGEBRD)
F02SXF	F08KFF (SORGBR/DORGBR) or F08KGF (SORMBR/DORMBR)
F02SYF	F08MEF (SBDSQR/DBDSQR)
F02UWF	F08KSF (CGEBRD/ZGEBRD)
F02UXF	F08KTF (CUNGBR/ZUNGBR) or F08KUF (CUNMBR/ZUNMBR)
F02UYF	F08MSF (CBDSQR/ZBDSQR)
F04ANF	F08AGF (SORMQR/DORMQR) and F06PJF (STRSV/DTRSV)
F04AYF	F07AEF (SGETRS/DGETRS)
F04LDF	F07BEF (SGBTRS/DGBTRS)
G01CEF	G01FAF

4 Routines Scheduled for Withdrawal

The routines listed below are scheduled for withdrawal from the NAG Fortran Library, because improved routines have now been included in the Library. Users are advised to stop using routines which are scheduled for withdrawal immediately and to use recommended replacement routines instead. See the document 'Advice on Replacement Calls for Superseded/Withdrawn Routines' for more detailed guidance, including detailed advice on how to change a call to the old routine into a call to the new routine.

The following routines will be withdrawn at Mark 19:

Routine Scheduled for Withdrawal	**Recommended Replacement**
E04FDF	E04FYF
E04GCF	E04GYF
E04GEF	E04GZF
E04HFF	E04HYF
E04JAF	E04JYF
E04KAF	E04KYF
E04KCF	E04KZF
E04LAF	E04LYF
E04UPF	E04UNF
F01MAF	F11JAF
F02BBF	F02FCF
F02BCF	F02ECF
F02BDF	F02GCF
F04MAF	F11JCF
F04MBF	F11GAF, F11GBF and F11GCF (or F11JCF or F11JEF)

The following routines have been superseded, but will not be withdrawn from the Library until Mark 20 at the earliest. They are being retained at Marks 18 and 19 because of their length of life in the Library, and to give users a longer time to make the transition to the new routines.

Superseded routine	**Recommended Replacement**
E01SEF	E01SGF
E01SFF	E01SHF

Thread-safety

International standards are now making it practicable for developers to write portable multi-threaded applications. Consequently there is an increasing demand for Library developers to produce software that is thread safe.

In a Fortran77 context the constructs that prohibit thread-safety are, potentially, DATA, SAVE and COMMON. At Mark 18 of the NAG Fortran Library the use of some of these constructs has been eliminated making the majority of routines in the Library thread safe. Furthermore NAG plans to remove the use of these constructs totally at Mark 19.

Users are advised to refer to the Users' Note for details of whether the Library has been compiled in a manner that facilitates the use of threads. Please note however that, at Mark 18, the routines listed in the following table are not thread safe in any implementations.

C02AFF	C02AGF	C02AHF	C02AJF	C05NDF	C05PDF
D01AHF	D01EAF	D01FDF	D01GBF	D01GCF	D01GDF
D01JAF	D02BJF	D02CAF	D02CBF	D02CGF	D02CHF
D02CJF	D02EAF	D02EBF	D02EGF	D02EHF	D02EJF
D02GAF	D02GBF	D02HAF	D02HBF	D02JAF	D02JBF
D02KAF	D02KDF	D02KEF	D02LAF	D02LXF	D02LYF
D02LZF	D02MVF	D02MZF	D02NBF	D02NCF	D02NDF
D02NGF	D02NHF	D02NJF	D02NMF	D02NNF	D02NRF
D02NSF	D02NTF	D02NUF	D02NVF	D02NWF	D02NXF
D02NYF	D02NZF	D02PAF	D02PCF	D02PDF	D02PVF
D02PWF	D02PXF	D02PYF	D02PZF	D02QFF	D02QGF
D02RAF	D02SAF	D02XJF	D02XKF	D02ZAF	D03PCF
D03PDF	D03PEF	D03PFF	D03PHF	D03PJF	D03PKF
D03PLF	D03PPF	D03PRF	D03PSF	D03PUF	D03PWF
D03PXF	D03PVF	D03PZF	D03RAF	D03RBF	D05BDF
D05BEF	E02GBF	E04DGF	E04DJF	E04DKF	E04MBF
E04MFF	E04MGF	E04MHF	E04MZF	E04NAF	E04NCF
E04NDF	E04NEF	E04NFF	E04NGF	E04NHF	E04NKF
E04NLF	E04NMF	E04UCF	E04UDF	E04UEF	E04UFF
E04UNF	E04UPF	E04UQF	E04URF	E04XAF	F02FCF
F02FJF	F02HCF	F04YCF	F04ZCF	F08JKF	F08JXF
F11BAF	F11BBF	F11BCF	F11DCF	F11DEF	F11GAF
F11GBF	F11GCF	F11JCF	F11JEF	G01DCF	G01DHF
G01EMF	G01HBF	G01JDF	G03FAF	G05CAF	G05CBF
G05CCF	G05CFF	G05CGF	G05DAF	G05DBF	G05DCF
G05DDF	G05DEF	G05DFF	G05DHF	G05DJF	G05DKF
G05DPF	G05DRF	G05DYF	G05DZF	G05EGF	G05EHF
G05EJF	G05EWF	G05EYF	G05EZF	G05FAF	G05FBF
G05FDF	G05FEF	G05FFF	G05FSF	G05GAF	G05GBF
G05HDF	G07AAF	G07BEF	G07EAF	G07EBF	G08EAF
G08EBF	G08ECF	G08EDF	G10BAF	G13DCF	H02BBF
H02BFF	H02BVF	X04AAF	X04ABF		

NAG Fortran Library, Mark 18 Library Contents

Chapter A00 – Library Identification

A00AAF Prints details of the NAG Fortran Library implementation

Chapter A02 – Complex Arithmetic

A02AAF Square root of a complex number
A02ABF Modulus of a complex number
A02ACF Quotient of two complex numbers

Chapter C02 – Zeros of Polynomials

C02AFF All zeros of complex polynomial, modified Laguerre method
C02AGF All zeros of real polynomial, modified Laguerre method
C02AHF All zeros of complex quadratic
C02AJF All zeros of real quadratic

Chapter C05 – Roots of One or More Transcendental Equations

C05ADF Zero of continuous function in given interval, Bus and Dekker algorithm
C05AGF Zero of continuous function, Bus and Dekker algorithm, from given starting value, binary search for interval
C05AJF Zero of continuous function, continuation method, from a given starting value
C05AVF Binary search for interval containing zero of continuous function (reverse communication)
C05AXF Zero of continuous function by continuation method, from given starting value (reverse communication)
C05AZF Zero in given interval of continuous function by Bus and Dekker algorithm (reverse communication)
C05NBF Solution of system of nonlinear equations using function values only (easy-to-use)
C05NCF Solution of system of nonlinear equations using function values only (comprehensive)
C05NDF Solution of systems of nonlinear equations using function values only (reverse communication)
C05PBF Solution of system of nonlinear equations using 1st derivatives (easy-to-use)
C05PCF Solution of system of nonlinear equations using 1st derivatives (comprehensive)
C05PDF Solution of systems of nonlinear equations using 1st derivatives (reverse communication)
C05ZAF Check user's routine for calculating 1st derivatives

Chapter C06 – Summation of Series

C06BAF Acceleration of convergence of sequence, Shanks' transformation and epsilon algorithm
C06DBF Sum of a Chebyshev series
C06EAF Single 1-D real discrete Fourier transform, no extra workspace
C06EBF Single 1-D Hermitian discrete Fourier transform, no extra workspace
C06ECF Single 1-D complex discrete Fourier transform, no extra workspace
C06EKF Circular convolution or correlation of two real vectors, no extra workspace
C06FAF Single 1-D real discrete Fourier transform, extra workspace for greater speed
C06FBF Single 1-D Hermitian discrete Fourier transform, extra workspace for greater speed
C06FCF Single 1-D complex discrete Fourier transform, extra workspace for greater speed
C06FFF 1-D complex discrete Fourier transform of multi-dimensional data
C06FJF Multi-dimensional complex discrete Fourier transform of multi-dimensional data
C06FKF Circular convolution or correlation of two real vectors, extra workspace for greater speed
C06FPF Multiple 1-D real discrete Fourier transforms
C06FQF Multiple 1-D Hermitian discrete Fourier transforms
C06FRF Multiple 1-D complex discrete Fourier transforms
C06FUF 2-D complex discrete Fourier transform
C06FXF 3-D complex discrete Fourier transform
C06GBF Complex conjugate of Hermitian sequence

C06GCF Complex conjugate of complex sequence
C06GQF Complex conjugate of multiple Hermitian sequences
C06GSF Convert Hermitian sequences to general complex sequences
C06HAF Discrete sine transform
C06HBF Discrete cosine transform
C06HCF Discrete quarter-wave sine transform
C06HDF Discrete quarter-wave cosine transform
C06LAF Inverse Laplace transform, Crump's method
C06LBF Inverse Laplace transform, modified Weeks' method
C06LCF Evaluate inverse Laplace transform as computed by C06LBF

Chapter D01 – Quadrature

D01AHF 1-D quadrature, adaptive, finite interval, strategy due to Patterson, suitable for well-behaved integrands
D01AJF 1-D quadrature, adaptive, finite interval, strategy due to Piessens and de Doncker, allowing for badly-behaved integrands
D01AKF 1-D quadrature, adaptive, finite interval, method suitable for oscillating functions
D01ALF 1-D quadrature, adaptive, finite interval, allowing for singularities at user-specified break-points
D01AMF 1-D quadrature, adaptive, infinite or semi-infinite interval
D01ANF 1-D quadrature, adaptive, finite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01APF 1-D quadrature, adaptive, finite interval, weight function with end-point singularities of algebraico-logarithmic type
D01AQF 1-D quadrature, adaptive, finite interval, weight function $1/(x-c)$, Cauchy principal value (Hilbert transform)
D01ARF 1-D quadrature, non-adaptive, finite interval with provision for indefinite integrals
D01ASF 1-D quadrature, adaptive, semi-infinite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01ATF 1-D quadrature, adaptive, finite interval, variant of D01AJF efficient on vector machines
D01AUF 1-D quadrature, adaptive, finite interval, variant of D01AKF efficient on vector machines
D01BAF 1-D Gaussian quadrature
D01BBF Pre-computed weights and abscissae for Gaussian quadrature rules, restricted choice of rule
D01BCF Calculation of weights and abscissae for Gaussian quadrature rules, general choice of rule
D01BDF 1-D quadrature, non-adaptive, finite interval
D01DAF 2-D quadrature, finite region
D01EAF Multi-dimensional adaptive quadrature over hyper-rectangle, multiple integrands
D01FBF Multi-dimensional Gaussian quadrature over hyper-rectangle
D01FCF Multi-dimensional adaptive quadrature over hyper-rectangle
D01FDF Multi-dimensional quadrature, Sag–Szekeres method, general product region or n-sphere
D01GAF 1-D quadrature, integration of function defined by data values, Gill–Miller method
D01GBF Multi-dimensional quadrature over hyper-rectangle, Monte Carlo method
D01GCF Multi-dimensional quadrature, general product region, number-theoretic method
D01GDF Multi-dimensional quadrature, general product region, number-theoretic method, variant of D01GCF efficient on vector machines
D01GYF Korobov optimal coefficients for use in D01GCF or D01GDF, when number of points is prime
D01GZF Korobov optimal coefficients for use in D01GCF or D01GDF, when number of points is product of two primes
D01JAF Multi-dimensional quadrature over an n-sphere, allowing for badly-behaved integrands
D01PAF Multi-dimensional quadrature over an n-simplex

Chapter D02 – Ordinary Differential Equations

D02AGF ODEs, boundary value problem, shooting and matching technique, allowing interior matching point, general parameters to be determined
D02BGF ODEs, IVP, Runge–Kutta–Merson method, until a component attains given value (simple driver)
D02BHF ODEs, IVP, Runge–Kutta–Merson method, until function of solution is zero (simple driver)
D02BJF ODEs, IVP, Runge–Kutta method, until function of solution is zero, integration over range with intermediate output (simple driver)

D02CJF ODEs, IVP, Adams method, until function of solution is zero, intermediate output (simple driver)
D02EJF ODEs, stiff IVP, BDF method, until function of solution is zero, intermediate output (simple driver)
D02GAF ODEs, boundary value problem, finite difference technique with deferred correction, simple nonlinear problem
D02GBF ODEs, boundary value problem, finite difference technique with deferred correction, general linear problem
D02HAF ODEs, boundary value problem, shooting and matching, boundary values to be determined
D02HBF ODEs, boundary value problem, shooting and matching, general parameters to be determined
D02JAF ODEs, boundary value problem, collocation and least-squares, single nth order linear equation
D02JBF ODEs, boundary value problem, collocation and least-squares, system of 1st order linear equations
D02KAF 2nd order Sturm–Liouville problem, regular system, finite range, eigenvalue only
D02KDF 2nd order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue only, user-specified break-points
D02KEF 2nd order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue and eigenfunction, user-specified break-points
D02LAF 2nd order ODEs, IVP, Runge–Kutta–Nystrom method
D02LXF 2nd order ODEs, IVP, set-up for D02LAF
D02LYF 2nd order ODEs, IVP, diagnostics for D02LAF
D02LZF 2nd order ODEs, IVP, interpolation for D02LAF
D02MVF ODEs, IVP, DASSL method, set-up for D02M-N routines
D02MZF ODEs, IVP, interpolation for D02M-N routines, natural interpolant
D02NBF Explicit ODEs, stiff IVP, full Jacobian (comprehensive)
D02NCF Explicit ODEs, stiff IVP, banded Jacobian (comprehensive)
D02NDF Explicit ODEs, stiff IVP, sparse Jacobian (comprehensive)
D02NGF Implicit/algebraic ODEs, stiff IVP, full Jacobian (comprehensive)
D02NHF Implicit/algebraic ODEs, stiff IVP, banded Jacobian (comprehensive)
D02NJF Implicit/algebraic ODEs, stiff IVP, sparse Jacobian (comprehensive)
D02NMF Explicit ODEs, stiff IVP (reverse communication, comprehensive)
D02NNF Implicit/algebraic ODEs, stiff IVP (reverse communication, comprehensive)
D02NRF ODEs, IVP, for use with D02M-N routines, sparse Jacobian, enquiry routine
D02NSF ODEs, IVP, for use with D02M-N routines, full Jacobian, linear algebra set-up
D02NTF ODEs, IVP, for use with D02M-N routines, banded Jacobian, linear algebra set-up
D02NUF ODEs, IVP, for use with D02M-N routines, sparse Jacobian, linear algebra set-up
D02NVF ODEs, IVP, BDF method, set-up for D02M-N routines
D02NWF ODEs, IVP, Blend method, set-up for D02M-N routines
D02NXF ODEs, IVP, sparse Jacobian, linear algebra diagnostics, for use with D02M-N routines
D02NYF ODEs, IVP, integrator diagnostics, for use with D02M-N routines
D02NZF ODEs, IVP, set-up for continuation calls to integrator, for use with D02M-N routines
D02PCF ODEs, IVP, Runge–Kutta method, integration over range with output
D02PDF ODEs, IVP, Runge–Kutta method, integration over one step
D02PVF ODEs, IVP, set-up for D02PCF and D02PDF
D02PWF ODEs, IVP, resets end of range for D02PDF
D02PXF ODEs, IVP, interpolation for D02PDF
D02PYF ODEs, IVP, integration diagnostics for D02PCF and D02PDF
D02PZF ODEs, IVP, error assessment diagnostics for D02PCF and D02PDF
D02QFF ODEs, IVP, Adams method with root-finding (forward communication, comprehensive)
D02QGF ODEs, IVP, Adams method with root-finding (reverse communication, comprehensive)
D02QWF ODEs, IVP, set-up for D02QFF and D02QGF
D02QXF ODEs, IVP, diagnostics for D02QFF and D02QGF
D02QYF ODEs, IVP, root-finding diagnostics for D02QFF and D02QGF
D02QZF ODEs, IVP, interpolation for D02QFF or D02QGF
D02RAF ODEs, general nonlinear boundary value problem, finite difference technique with deferred correction, continuation facility
D02SAF ODEs, boundary value problem, shooting and matching technique, subject to extra algebraic equations, general parameters to be determined

D02TGF nth order linear ODEs, boundary value problem, collocation and least-squares
D02TKF ODEs, general nonlinear boundary value problem, collocation technique
D02TVF ODEs, general nonlinear boundary value problem, set-up for D02TKF
D02TXF ODEs, general nonlinear boundary value problem, continuation facility for D02TKF
D02TYF ODEs, general nonlinear boundary value problem, interpolation for D02TKF
D02TZF ODEs, general nonlinear boundary value problem, diagnostics for D02TKF
D02XJF ODEs, IVP, interpolation for D02M-N routines, natural interpolant
D02XKF ODEs, IVP, interpolation for D02M-N routines, C_1 interpolant
D02ZAF ODEs, IVP, weighted norm of local error estimate for D02M-N routines

Chapter D03 – Partial Differential Equations

D03EAF Elliptic PDE, Laplace's equation, 2-D arbitrary domain
D03EBF Elliptic PDE, solution of finite difference equations by SIP, five-point 2-D molecule, iterate to convergence
D03ECF Elliptic PDE, solution of finite difference equations by SIP for seven-point 3-D molecule, iterate to convergence
D03EDF Elliptic PDE, solution of finite difference equations by a multigrid technique
D03EEF Discretize a second order elliptic PDE on a rectangle
D03FAF Elliptic PDE, Helmholtz equation, 3-D Cartesian co-ordinates
D03MAF Triangulation of a plane region
D03PCF General system of parabolic PDEs, method of lines, finite differences, one space variable
D03PDF General system of parabolic PDEs, method of lines, Chebyshev C^0 collocation, one space variable
D03PEF General system of first order PDEs, method of lines, Keller box discretisation, one space variable
D03PFF General system of convection-diffusion PDEs with source terms in conservative form, method of lines, upwind scheme using numerical flux function based on Riemann solver, one space variable
D03PHF General system of parabolic PDEs, coupled DAEs, method of lines, finite differences, one space variable
D03PJF General system of parabolic PDEs, coupled DAEs, method of lines, Chebyshev C^0 collocation, one space variable
D03PKF General system of first order PDEs, coupled DAEs, method of lines, Keller box discretisation, one space variable
D03PLF General system of convection-diffusion PDEs with source terms in conservative form, coupled DAEs, method of lines, upwind scheme using numerical flux function based on Riemann solver, one space variable
D03PPF General system of parabolic PDEs, coupled DAEs, method of lines, finite differences, remeshing, one space variable
D03PRF General system of first order PDEs, coupled DAEs, method of lines, Keller box discretisation, remeshing, one space variable
D03PSF General system of convection-diffusion PDEs with source terms in conservative form, coupled DAEs, method of lines, upwind scheme using numerical flux function based on Riemann solver, remeshing, one space variable
D03PUF Roe's approximate Riemann solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PVF Osher's approximate Riemann solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PWF Modified HLL Riemann solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PXF Exact Riemann Solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PYF PDEs, spatial interpolation with D03PDF or D03PJF
D03PZF PDEs, spatial interpolation with D03PCF, D03PEF, D03PFF, D03PHF, D03PKF, D03PLF, D03PPF, D03PRF or D03PSF
D03RAF General system of second order PDEs, method of lines, finite differences, remeshing, two space variables, rectangular region
D03RBF General system of second order PDEs, method of lines, finite differences, remeshing, two space variables, rectilinear region

D03RYF Check initial grid data in D03RBF
D03RZF Extract grid data from D03RBF
D03UAF Elliptic PDE, solution of finite difference equations by SIP, five-point 2-D molecule, one iteration
D03UBF Elliptic PDE, solution of finite difference equations by SIP, seven-point 3-D molecule, one iteration

Chapter D04 – Numerical Differentiation

D04AAF Numerical differentiation, derivatives up to order 14, function of one real variable

Chapter D05 – Integral Equations

D05AAF Linear non-singular Fredholm integral equation, 2nd kind, split kernel
D05ABF Linear non-singular Fredholm integral equation, 2nd kind, smooth kernel
D05BAF Nonlinear Volterra convolution equation, 2nd kind
D05BDF Nonlinear convolution Volterra–Abel equation, 2nd kind, weakly singular
D05BEF Nonlinear convolution Volterra–Abel equation, 1st kind, weakly singular
D05BWF Generate weights for use in solving Volterra equations
D05BYF Generate weights for use in solving weakly singular Abel type equations

Chapter E01 – Interpolation

E01AAF Interpolated values, Aitken's technique, unequally spaced data, one variable
E01ABF Interpolated values, Everett's formula, equally spaced data, one variable
E01AEF Interpolating functions, polynomial interpolant, data may include derivative values, one variable
E01BAF Interpolating functions, cubic spline interpolant, one variable
E01BEF Interpolating functions, monotonicity-preserving, piecewise cubic Hermite, one variable
E01BFF Interpolated values, interpolant computed by E01BEF, function only, one variable,
E01BGF Interpolated values, interpolant computed by E01BEF, function and 1st derivative, one variable
E01BHF Interpolated values, interpolant computed by E01BEF, definite integral, one variable
E01DAF Interpolating functions, fitting bicubic spline, data on rectangular grid
E01RAF Interpolating functions, rational interpolant, one variable
E01RBF Interpolated values, evaluate rational interpolant computed by E01RAF, one variable
E01SAF Interpolating functions, method of Renka and Cline, two variables
E01SBF Interpolated values, evaluate interpolant computed by E01SAF, two variables
E01SEF Interpolating functions, modified Shepard's method, two variables
E01SFF Interpolated values, evaluate interpolant computed by E01SEF, two variables
E01SGF Interpolating functions, modified Shepard's method, two variables
E01SHF Interpolated values, evaluate interpolant computed by E01SGF, function and first derivatives, two variables
E01TGF Interpolating functions, modified Shepard's method, three variables
E01THF Interpolated values, evaluate interpolant computed by E01TGF, function and first derivatives, three variables

Chapter E02 – Curve and Surface Fitting

E02ACF Minimax curve fit by polynomials
E02ADF Least-squares curve fit, by polynomials, arbitrary data points
E02AEF Evaluation of fitted polynomial in one variable from Chebyshev series form (simplified parameter list)
E02AFF Least-squares polynomial fit, special data points (including interpolation)
E02AGF Least-squares polynomial fit, values and derivatives may be constrained, arbitrary data points,
E02AHF Derivative of fitted polynomial in Chebyshev series form
E02AJF Integral of fitted polynomial in Chebyshev series form
E02AKF Evaluation of fitted polynomial in one variable, from Chebyshev series form
E02BAF Least-squares curve cubic spline fit (including interpolation)
E02BBF Evaluation of fitted cubic spline, function only
E02BCF Evaluation of fitted cubic spline, function and derivatives
E02BDF Evaluation of fitted cubic spline, definite integral

E02BEF Least-squares cubic spline curve fit, automatic knot placement
E02CAF Least-squares surface fit by polynomials, data on lines
E02CBF Evaluation of fitted polynomial in two variables
E02DAF Least-squares surface fit, bicubic splines
E02DCF Least-squares surface fit by bicubic splines with automatic knot placement, data on rectangular grid
E02DDF Least-squares surface fit by bicubic splines with automatic knot placement, scattered data
E02DEF Evaluation of a fitted bicubic spline at a vector of points
E02DFF Evaluation of a fitted bicubic spline at a mesh of points
E02GAF L_1-approximation by general linear function
E02GBF L_1-approximation by general linear function subject to linear inequality constraints
E02GCF L_∞-approximation by general linear function
E02RAF Padé-approximants
E02RBF Evaluation of fitted rational function as computed by E02RAF
E02ZAF Sort 2-D data into panels for fitting bicubic splines

Chapter E04 – Minimizing or Maximizing a Function

E04ABF Minimum, function of one variable using function values only
E04BBF Minimum, function of one variable, using 1st derivative
E04CCF Unconstrained minimum, simplex algorithm, function of several variables using function values only (comprehensive)
E04DGF Unconstrained minimum, pre-conditioned conjugate gradient algorithm, function of several variables using 1st derivatives (comprehensive)
E04DJF Read optional parameter values for E04DGF from external file
E04DKF Supply optional parameter values to E04DGF
E04FCF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using function values only (comprehensive)
E04FYF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using function values only (easy-to-use)
E04GBF Unconstrained minimum of a sum of squares, combined Gauss–Newton and quasi-Newton algorithm using 1st derivatives (comprehensive)
E04GDF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using 1st derivatives (comprehensive)
E04GEF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using 1st derivatives (easy-to-use)
E04GYF Unconstrained minimum of a sum of squares, combined Gauss–Newton and quasi-Newton algorithm, using 1st derivatives (easy-to-use)
E04GZF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using 1st derivatives (easy-to-use)
E04HCF Check user's routine for calculating 1st derivatives of function
E04HDF Check user's routine for calculating 2nd derivatives of function
E04HEF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm, using 2nd derivatives (comprehensive)
E04HFF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm, using 2nd derivatives (easy-to-use)
E04HYF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm, using 2nd derivatives (easy-to-use)
E04JYF Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using function values only (easy-to-use)
E04KAF Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using 1st derivatives (easy-to-use)
E04KCF Minimum, function of several variables, modified Newton algorithm, simple bounds, using 1st derivatives (easy-to-use)
E04KDF Minimum, function of several variables, modified Newton algorithm, simple bounds, using 1st derivatives (comprehensive)
E04KYF Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using 1st derivatives (easy-to-use)

E04KZF Minimum, function of several variables, modified Newton algorithm, simple bounds, using 1st derivatives (easy-to-use)
E04LAF Minimum, function of several variables, modified Newton algorithm, simple bounds, using 1st and 2nd derivatives (easy-to-use)
E04LBF Minimum, function of several variables, modified Newton algorithm, simple bounds, using 1st and 2nd derivatives (comprehensive)
E04LYF Minimum, function of several variables, modified Newton algorithm, simple bounds, using 1st and 2nd derivatives (easy-to-use)
E04MFF LP problem
E04MGF Read optional parameter values for E04MFF from external file
E04MHF Supply optional parameter values to E04MFF
E04MZF Converts MPSX data file defining LP or QP problem to format required by E04NKF
E04NCF Convex QP problem or linearly-constrained linear least-squares problem (dense)
E04NDF Read optional parameter values for E04NCF from external file
E04NEF Supply optional parameter values to E04NCF
E04NFF QP problem (dense)
E04NGF Read optional parameter values for E04NFF from external file
E04NHF Supply optional parameter values to E04NFF
E04NKF LP or QP problem (sparse)
E04NLF Read optional parameter values for E04NKF from external file
E04NMF Supply optional parameter values to E04NKF
E04UCF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally 1st derivatives (forward communication, comprehensive)
E04UDF Read optional parameter values for E04UCF or E04UFF from external file
E04UEF Supply optional parameter values to E04UCF or E04UFF
E04UFF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally 1st derivatives (reverse communication, comprehensive)
E04UNF Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally 1st derivatives (comprehensive)
E04UPF Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally 1st derivatives (comprehensive)
E04UQF Read optional parameter values for E04UNF or E04UPF from external file
E04URF Supply optional parameter values to E04UNF or E04UPF
E04XAF Estimate (using numerical differentiation) gradient and/or Hessian of a function
E04YAF Check user's routine for calculating Jacobian of 1st derivatives
E04YBF Check user's routine for calculating Hessian of a sum of squares
E04YCF Covariance matrix for nonlinear least-squares problem (unconstrained)
E04ZCF Check user's routines for calculating 1st derivatives of function and constraints

Chapter F01 – Matrix Factorizations

F01ABF Inverse of real symmetric positive-definite matrix using iterative refinement
F01ADF Inverse of real symmetric positive-definite matrix
F01BLF Pseudo-inverse and rank of a real m by n matrix ($m \geq n$)
F01BRF LU factorization of real sparse matrix
F01BSF LU factorization of real sparse matrix with known sparsity pattern
F01BUF $ULDL^TU^T$ factorization of real symmetric positive-definite band matrix
F01BVF Reduction to standard form, generalized real symmetric-definite banded eigenproblem
F01CKF Matrix multiplication
F01CRF Matrix transposition
F01CTF Sum or difference of two real matrices, optional scaling and transposition
F01CWF Sum or difference of two complex matrices, optional scaling and transposition
F01LEF LU factorization of real tridiagonal matrix
F01LHF LU factorization of real almost block diagonal matrix
F01MAF LL^T factorization of real sparse symmetric positive-definite matrix
F01MCF LDL^T factorization of real symmetric positive-definite variable-bandwidth matrix
F01QGF RQ factorization of real m by n upper trapezoidal matrix ($m \leq n$)
F01QJF RQ factorization of real m by n matrix ($m \leq n$)

F01QKF Operations with orthogonal matrices, form rows of Q, after RQ factorization by F01QJF
F01RGF RQ factorization of complex m by n upper trapezoidal matrix ($m \leq n$)
F01RJF RQ factorization of complex m by n matrix ($m \leq n$)
F01RKF Operations with unitary matrices, form rows of Q, after RQ factorization by F01RJF
F01ZAF Convert real matrix between packed triangular and square storage schemes
F01ZBF Convert complex matrix between packed triangular and square storage schemes
F01ZCF Convert real matrix between packed banded and rectangular storage schemes
F01ZDF Convert complex matrix between packed banded and rectangular storage schemes

Chapter F02 – Eigenvalues and Eigenvectors

F02BBF Selected eigenvalues and eigenvectors of real symmetric matrix (Black Box)
F02BCF Selected eigenvalues and eigenvectors of real matrix (Black Box)
F02BDF Selected eigenvalues and eigenvectors of complex matrix (Black Box)
F02BJF All eigenvalues and optionally eigenvectors of generalized eigenproblem by QZ algorithm, real matrices (Black Box)
F02EAF All eigenvalues and Schur factorization of real general matrix (Black Box)
F02EBF All eigenvalues and eigenvectors of real general matrix (Black Box)
F02ECF Selected eigenvalues and eigenvectors of real nonsymmetric matrix (Black Box)
F02FAF All eigenvalues and eigenvectors of real symmetric matrix (Black Box)
F02FCF Selected eigenvalues and eigenvectors of real symmetric matrix (Black Box)
F02FDF All eigenvalues and eigenvectors of real symmetric-definite generalized problem (Black Box)
F02FHF All eigenvalues of generalized banded real symmetric-definite eigenproblem (Black Box)
F02FJF Selected eigenvalues and eigenvectors of sparse symmetric eigenproblem (Black Box)
F02GAF All eigenvalues and Schur factorization of complex general matrix (Black Box)
F02GBF All eigenvalues and eigenvectors of complex general matrix (Black Box)
F02GCF Selected eigenvalues and eigenvectors of complex nonsymmetric matrix (Black Box)
F02GJF All eigenvalues and optionally eigenvectors of generalized complex eigenproblem by QZ algorithm (Black Box)
F02HAF All eigenvalues and eigenvectors of complex Hermitian matrix (Black Box)
F02HCF Selected eigenvalues and eigenvectors of complex Hermitian matrix (Black Box)
F02HDF All eigenvalues and eigenvectors of complex Hermitian-definite generalized problem (Black Box)
F02SDF Eigenvector of generalized real banded eigenproblem by inverse iteration
F02WDF QR factorization, possibly followed by SVD
F02WEF SVD of real matrix (Black Box)
F02WUF SVD of a real upper triangular matrix (Black Box)
F02XEF SVD of complex matrix (Black Box)
F02XUF SVD of complex upper triangular matrix (Black Box)

Chapter F03 – Determinants

F03AAF Determinant of real matrix (Black Box)
F03ABF Determinant of real symmetric positive-definite matrix (Black Box)
F03ACF Determinant of real symmetric positive-definite band matrix (Black Box)
F03ADF Determinant of complex matrix (Black Box)
F03AEF LL^T factorization and determinant of real symmetric positive-definite matrix
F03AFF LU factorization and determinant of real matrix

Chapter F04 – Simultaneous Linear Equations

F04AAF Solution of real simultaneous linear equations with multiple right-hand sides (Black Box)
F04ABF Solution of real symmetric positive-definite simultaneous linear equations with multiple right-hand sides using iterative refinement (Black Box)
F04ACF Solution of real symmetric positive-definite banded simultaneous linear equations with multiple right-hand sides (Black Box)
F04ADF Solution of complex simultaneous linear equations with multiple right-hand sides (Black Box)
F04AEF Solution of real simultaneous linear equations with multiple right-hand sides using iterative refinement (Black Box)
F04AFF Solution of real symmetric positive-definite simultaneous linear equations using iterative refinement (coefficient matrix already factorized by F03AEF)

F04AGF Solution of real symmetric positive-definite simultaneous linear equations (coefficient matrix already factorized by F03AEF)
F04AHF Solution of real simultaneous linear equations using iterative refinement (coefficient matrix already factorized by F03AFF)
F04AJF Solution of real simultaneous linear equations (coefficient matrix already factorized by F03AFF)
F04AMF Least-squares solution of m real equations in n unknowns, rank $= n$, $m \geq n$ using iterative refinement (Black Box)
F04ARF Solution of real simultaneous linear equations, one right-hand side (Black Box)
F04ASF Solution of real symmetric positive-definite simultaneous linear equations, one right-hand side using iterative refinement (Black Box)
F04ATF Solution of real simultaneous linear equations, one right-hand side using iterative refinement (Black Box)
F04AXF Solution of real sparse simultaneous linear equations (coefficient matrix already factorized)
F04EAF Solution of real tridiagonal simultaneous linear equations, one right-hand side (Black Box)
F04FAF Solution of real symmetric positive-definite tridiagonal simultaneous linear equations, one right-hand side (Black Box)
F04FEF Solution of the Yule–Walker equations for a real symmetric positive-definite Toeplitz matrix, one right-hand side
F04FFF Solution of real symmetric positive-definite Toeplitz system, one right-hand side
F04JAF Minimal least-squares solution of m real equations in n unknowns, rank$\leq n$, $m \geq n$
F04JDF Minimal least-squares solution of m real equations in n unknowns, rank$\leq n$, $m \geq n$
F04JGF Least-squares (if rank=n) or minimal least-squares (if rank$< n$) solution of m real equations in n unknowns, rank$\leq n$, $m \geq n$
F04JLF Real general Gauss–Markov linear model (including weighted least-squares)
F04JMF Equality-constrained real linear least squares problem
F04KLF Complex general Gauss–Markov linear model (including weighted least-squares)
F04KMF Equality-constrained complex linear least-squares
F04LEF Solution of real tridiagonal simultaneous linear equations (coefficient matrix already factorized by F01LEF)
F04LHF Solution of real almost block diagonal simultaneous linear equations (coefficient matrix already factorized by F01LHF)
F04MAF Real sparse symmetric positive-definite simultaneous linear equations (coefficient matrix already factorized by F01MAF)
F04MBF Real sparse symmetric simultaneous linear equations
F04MCF Solution of real symmetric positive-definite variable-bandwidth simultaneous linear equations (coefficient matrix already factorized by F01MCF)
F04MEF Update solution of the Yule–Walker equations for a real symmetric positive-definite Toeplitz matrix
F04MFF Update solution of real symmetric positive-definite Toeplitz system
F04QAF Sparse linear least-squares problem, m real equations in n unknowns
F04YAF Covariance matrix for linear least-squares problems, m real equations in n unknowns
F04YCF Norm estimation (for use in condition estimation), real matrix
F04ZCF Norm estimation (for use in condition estimation), complex matrix

Chapter F05 – Orthogonalisation

F05AAF Gram–Schmidt orthogonalisation of n vectors of order m

Chapter F06 – Linear Algebra Support Routines

F06AAF Generate real plane rotation (SROTG/DROTG)
F06BAF Generate real plane rotation, storing tangent
F06BCF Recover cosine and sine from given real tangent
F06BEF Generate real Jacobi plane rotation
F06BHF Apply real similarity rotation to 2 by 2 symmetric matrix
F06BLF Compute quotient of two real scalars, with overflow flag
F06BMF Compute Euclidean norm from scaled form
F06BNF Compute square root of $(a^2 + b^2)$, real a and b

F06BPF	Compute eigenvalue of 2 by 2 real symmetric matrix
F06CAF	Generate complex plane rotation, storing tangent, real cosine
F06CBF	Generate complex plane rotation, storing tangent, real sine
F06CCF	Recover cosine and sine from given complex tangent, real cosine
F06CDF	Recover cosine and sine from given complex tangent, real sine
F06CHF	Apply complex similarity rotation to 2 by 2 Hermitian matrix
F06CLF	Compute quotient of two complex scalars, with overflow flag
F06DBF	Broadcast scalar into integer vector
F06DFF	Copy integer vector
F06EAF	Dot product of two real vectors (SDOT/DDOT)
F06ECF	Add scalar times real vector to real vector (SAXPY/DAXPY)
F06EDF	Multiply real vector by scalar (SSCAL/DSCAL)
F06EFF	Copy real vector (SCOPY/DCOPY)
F06EGF	Swap two real vectors (SSWAP/DSWAP)
F06EJF	Compute Euclidean norm of real vector (SNRM2/DNRM2)
F06EKF	Sum the absolute values of real vector elements (SASUM/DASUM)
F06EPF	Apply real plane rotation (SROT/DROT)
F06ERF	Dot product of two real sparse vectors (SDOTI/DDOTI)
F06ETF	Add a scalar times a real sparse vector to another real sparse vector (SAXPYI/DAXPYI)
F06EUF	Gather a real sparse vector (SGTHR/DGTHR)
F06EVF	Gather and set to zero a real sparse vector (SGTHRZ/DGTHRZ)
F06EWF	Scatter a real sparse vector (SSCTR/DSCTR)
F06EXF	Apply plane rotation to two real sparse vectors (SROTI/DROTI)
F06FAF	Compute cosine of angle between two real vectors
F06FBF	Broadcast scalar into real vector
F06FCF	Multiply real vector by diagonal matrix
F06FDF	Multiply real vector by scalar, preserving input vector
F06FGF	Negate real vector
F06FJF	Update Euclidean norm of real vector in scaled form
F06FKF	Compute weighted Euclidean norm of real vector
F06FLF	Elements of real vector with largest and smallest absolute value
F06FPF	Apply real symmetric plane rotation to two vectors
F06FQF	Generate sequence of real plane rotations
F06FRF	Generate real elementary reflection, NAG style
F06FSF	Generate real elementary reflection, LINPACK style
F06FTF	Apply real elementary reflection, NAG style
F06FUF	Apply real elementary reflection, LINPACK style
F06GAF	Dot product of two complex vectors, unconjugated (CDOTU/ZDOTU)
F06GBF	Dot product of two complex vectors, conjugated (CDOTC/ZDOTC)
F06GCF	Add scalar times complex vector to complex vector (CAXPY/ZAXPY)
F06GDF	Multiply complex vector by complex scalar (CSCAL/ZSCAL)
F06GFF	Copy complex vector (CCOPY/ZCOPY)
F06GGF	Swap two complex vectors (CSWAP/ZSWAP)
F06GRF	Dot product of two complex sparse vector, unconjugated (CDOTUI/ZDOTUI)
F06GSF	Dot product of two complex sparse vector, conjugated (CDOTCI/ZDOTCI)
F06GTF	Add a scalar times a complex sparse vector to another complex sparse vector (CAXPYI/ZAXPYI)
F06GUF	Gather a complex sparse vector (CGTHR/ZGTHR)
F06GVF	Gather and set to zero a complex sparse vector (CGTHRZ/ZGTHRZ)
F06GWF	Scatter a complex sparse vector (CSCTR/ZSCTR)
F06HBF	Broadcast scalar into complex vector
F06HCF	Multiply complex vector by complex diagonal matrix
F06HDF	Multiply complex vector by complex scalar, preserving input vector
F06HGF	Negate complex vector
F06HPF	Apply complex plane rotation
F06HQF	Generate sequence of complex plane rotations
F06HRF	Generate complex elementary reflection
F06HTF	Apply complex elementary reflection

F06JDF Multiply complex vector by real scalar (CSSCAL/ZDSCAL)
F06JJF Compute Euclidean norm of complex vector (SCNRM2/DZNRM2)
F06JKF Sum the absolute values of complex vector elements (SCASUM/DZASUM)
F06JLF Index, real vector element with largest absolute value (ISAMAX/IDAMAX)
F06JMF Index, complex vector element with largest absolute value (ICAMAX/IZAMAX)
F06KCF Multiply complex vector by real diagonal matrix
F06KDF Multiply complex vector by real scalar, preserving input vector
F06KFF Copy real vector to complex vector
F06KJF Update Euclidean norm of complex vector in scaled form
F06KLF Last non-negligible element of real vector
F06KPF Apply real plane rotation to two complex vectors
F06PAF Matrix-vector product, real rectangular matrix (SGEMV/DGEMV)
F06PBF Matrix-vector product, real rectangular band matrix (SGBMV/DGBMV)
F06PCF Matrix-vector product, real symmetric matrix (SSYMV/DSYMV)
F06PDF Matrix-vector product, real symmetric band matrix (SSBMV/DSBMV)
F06PEF Matrix-vector product, real symmetric packed matrix (SSPMV/DSPMV)
F06PFF Matrix-vector product, real triangular matrix (STRMV/DTRMV)
F06PGF Matrix-vector product, real triangular band matrix (STBMV/DTBMV)
F06PHF Matrix-vector product, real triangular packed matrix (STPMV/DTPMV)
F06PJF System of equations, real triangular matrix (STRSV/DTRSV)
F06PKF System of equations, real triangular band matrix (STBSV/DTBSV)
F06PLF System of equations, real triangular packed matrix (STPSV/DTPSV)
F06PMF Rank-1 update, real rectangular matrix (SGER/DGER)
F06PPF Rank-1 update, real symmetric matrix (SSYR/DSYR)
F06PQF Rank-1 update, real symmetric packed matrix (SSPR/DSPR)
F06PRF Rank-2 update, real symmetric matrix (SSYR2/DSYR2)
F06PSF Rank-2 update, real symmetric packed matrix (SSPR2/DSPR2)
F06QFF Matrix copy, real rectangular or trapezoidal matrix
F06QHF Matrix initialisation, real rectangular matrix
F06QJF Permute rows or columns, real rectangular matrix, permutations represented by an integer array
F06QKF Permute rows or columns, real rectangular matrix, permutations represented by a real array
F06QMF Orthogonal similarity transformation of a real symmetric matrix as a sequence of plane rotations
F06QPF QR factorization by sequence of plane rotations, rank-1 update of real upper triangular matrix
F06QQF QR factorization by sequence of plane rotations, real upper triangular matrix augmented by a full row
F06QRF QR or RQ factorization by sequence of plane rotations, real upper Hessenberg matrix
F06QSF QR or RQ factorization by sequence of plane rotations, real upper spiked matrix
F06QTF QR factorization of UZ or RQ factorization of ZU, U real upper triangular, Z a sequence of plane rotations
F06QVF Compute upper Hessenberg matrix by sequence of plane rotations, real upper triangular matrix
F06QWF Compute upper spiked matrix by sequence of plane rotations, real upper triangular matrix
F06QXF Apply sequence of plane rotations, real rectangular matrix
F06RAF 1-norm, ∞-norm, Frobenius norm, largest absolute element, real general matrix
F06RBF 1-norm, ∞-norm, Frobenius norm, largest absolute element, real band matrix
F06RCF 1-norm, ∞-norm, Frobenius norm, largest absolute element, real symmetric matrix
F06RDF 1-norm, ∞-norm, Frobenius norm, largest absolute element, real symmetric matrix, packed storage
F06REF 1-norm, ∞-norm, Frobenius norm, largest absolute element, real symmetric band matrix
F06RJF 1-norm, ∞-norm, Frobenius norm, largest absolute element, real trapezoidal/triangular matrix
F06RKF 1-norm, ∞-norm, Frobenius norm, largest absolute element, real triangular matrix, packed storage
F06RLF 1-norm, ∞-norm, Frobenius norm, largest absolute element, real triangular band matrix
F06RMF 1-norm, ∞-norm, Frobenius norm, largest absolute element, real Hessenberg matrix
F06SAF Matrix-vector product, complex rectangular matrix (CGEMV/ZGEMV)
F06SBF Matrix-vector product, complex rectangular band matrix (CGBMV/ZGBMV)
F06SCF Matrix-vector product, complex Hermitian matrix (CHEMV/ZHEMV)
F06SDF Matrix-vector product, complex Hermitian band matrix (CHBMV/ZHBMV)
F06SEF Matrix-vector product, complex Hermitian packed matrix (CHPMV/ZHPMV)

F06SFF	Matrix-vector product, complex triangular matrix (CTRMV/ZTRMV)
F06SGF	Matrix-vector product, complex triangular band matrix (CTBMV/ZTBMV)
F06SHF	Matrix-vector product, complex triangular packed matrix (CTPMV/ZTPMV)
F06SJF	System of equations, complex triangular matrix (CTRSV/ZTRSV)
F06SKF	System of equations, complex triangular band matrix (CTBSV/ZTBSV)
F06SLF	System of equations, complex triangular packed matrix (CTPSV/ZTPSV)
F06SMF	Rank-1 update, complex rectangular matrix, unconjugated vector (CGERU/ZGERU)
F06SNF	Rank-1 update, complex rectangular matrix, conjugated vector (CGERC/ZGERC)
F06SPF	Rank-1 update, complex Hermitian matrix (CHER/ZHER)
F06SQF	Rank-1 update, complex Hermitian packed matrix (CHPR/ZHPR)
F06SRF	Rank-2 update, complex Hermitian matrix (CHER2/ZHER2)
F06SSF	Rank-2 update, complex Hermitian packed matrix (CHPR2/ZHPR2)
F06TFF	Matrix copy, complex rectangular or trapezoidal matrix
F06THF	Matrix initialisation, complex rectangular matrix
F06TMF	Unitary similarity transformation of a Hermitian matrix as a sequence of plane rotations
F06TPF	QR factorization by sequence of plane rotations, rank-1 update of complex upper triangular matrix
F06TQF	$QRx k$ factorization by sequence of plane rotations, complex upper triangular matrix augmented by a full row
F06TRF	QR or RQ factorization by sequence of plane rotations, complex upper Hessenberg matrix
F06TSF	QR or RQ factorization by sequence of plane rotations, complex upper spiked matrix
F06TTF	QR factorization of UZ or RQ factorization of ZU, U complex upper triangular, Z a sequence of plane rotations
F06TVF	Compute upper Hessenberg matrix by sequence of plane rotations, complex upper triangular matrix
F06TWF	Compute upper spiked matrix by sequence of plane rotations, complex upper triangular matrix
F06TXF	Apply sequence of plane rotations, complex rectangular matrix, real cosine and complex sine
F06TYF	Apply sequence of plane rotations, complex rectangular matrix, complex cosine and real sine
F06UAF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex general matrix
F06UBF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex band matrix
F06UCF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex Hermitian matrix
F06UDF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex Hermitian matrix, packed storage
F06UEF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex Hermitian band matrix
F06UFF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex symmetric matrix
F06UGF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex symmetric matrix, packed storage
F06UHF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex symmetric band matrix
F06UJF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex trapezoidal/triangular matrix
F06UKF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex triangular matrix, packed storage
F06ULF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex triangular band matrix
F06UMF	1-norm, ∞-norm, Frobenius norm, largest absolute element, complex Hessenberg matrix
F06VJF	Permute rows or columns, complex rectangular matrix, permutations represented by an integer array
F06VKF	Permute rows or columns, complex rectangular matrix, permutations represented by a real array
F06VXF	Apply sequence of plane rotations, complex rectangular matrix, real cosine and sine
F06YAF	Matrix-matrix product, two real rectangular matrices (SGEMM/DGEMM)
F06YCF	Matrix-matrix product, one real symmetric matrix, one real rectangular matrix (SSYMM/DSYMM)
F06YFF	Matrix-matrix product, one real triangular matrix, one real rectangular matrix (STRMM/DTRMM)
F06YJF	Solves a system of equations with multiple right-hand sides, real triangular coefficient matrix (STRSM/DTRSM)
F06YPF	Rank-k update of a real symmetric matrix (SSYRK/DSYRK)

F06YRF Rank-2k update of a real symmetric matrix (SSYR2K/DSYR2K)
F06ZAF Matrix-matrix product, two complex rectangular matrices (CGEMM/ZGEMM)
F06ZCF Matrix-matrix product, one complex Hermitian matrix, one complex rectangular matrix (CHEMM/ZHEMM)
F06ZFF Matrix-matrix product, one complex triangular matrix, one complex rectangular matrix (CTRMM/ZTRMM)
F06ZJF Solves system of equations with multiple right-hand sides, complex triangular coefficient matrix (CTRSM/ZTRSM)
F06ZPF Rank-k update of a complex Hermitian matrix (CHERK/ZHERK)
F06ZRF Rank-2k update of a complex Hermitian matrix (CHER2K/ZHER2K)
F06ZTF Matrix-matrix product, one complex symmetric matrix, one complex rectangular matrix (CSYMM/ZSYMM)
F06ZUF Rank-k update of a complex symmetric matrix (CSYRK/ZSYRK)
F06ZWF Rank-2k update of a complex symmetric matrix (CSYR2K/ZHER2K)

Chapter F07 – Linear Equations (LAPACK)

F07ADF *LU* factorization of real m by n matrix (SGETRF/DGETRF)
F07AEF Solution of real system of linear equations, multiple right-hand sides, matrix already factorized by F07ADF (SGETRS/DGETRS)
F07AGF Estimate condition number of real matrix, matrix already factorized by F07ADF (SGECON/DGECON)
F07AHF Refined solution with error bounds of real system of linear equations, multiple right-hand sides (SGERFS/DGERFS)
F07AJF Inverse of a real matrix, matrix already factorized by F07ADF (SGETRI/DGETRI)
F07ARF *LU* factorization of complex m by n matrix (CGETRF/ZGETRF)
F07ASF Solution of complex system of linear equations, multiple right-hand sides, matrix already factorized by F07ARF (CGETRS/ZGETRS)
F07AUF Estimate condition number of complex matrix, matrix already factorized by F07ARF (CGECON/ZGECON)
F07AVF Refined solution with error bounds of complex system of linear equations, multiple right-hand sides (CGERFS/ZGERFS)
F07AWF Inverse of a complex matrix, matrix already factorized by F07ARF (CGETRI/ZGETRI)
F07BDF *LU* factorization of real m by n band matrix (SGBTRF/DGBTRF)
F07BEF Solution of real band system of linear equations, multiple right-hand sides, matrix already factorized by F07BDF (SGBTRS/DGBTRS)
F07BGF Estimate condition number of real band matrix, matrix already factorized by F07BDF (SGBCON/DGBCON)
F07BHF Refined solution with error bounds of real band system of linear equations, multiple right-hand sides (SGBRFS/DGBRFS)
F07BRF *LU* factorization of complex m by n band matrix (CGBTRF/ZGBTRF)
F07BSF Solution of complex band system of linear equations, multiple right-hand sides, matrix already factorized by F07BRF (CGBTRS/ZGBTRS)
F07BUF Estimate condition number of complex band matrix, matrix already factorized by F07BRF (CGBCON/ZGBCON)
F07BVF Refined solution with error bounds of complex band system of linear equations, multiple right-hand sides (CGBRFS/ZGBRFS)
F07FDF Cholesky factorization of real symmetric positive-definite matrix (SPOTRF/DPOTRF)
F07FEF Solution of real symmetric positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07FDF (SPOTRS/DPOTRS)
F07FGF Estimate condition number of real symmetric positive-definite matrix, matrix already factorized by F07FDF (SPOCON/DPOCON)
F07FHF Refined solution with error bounds of real symmetric positive-definite system of linear equations, multiple right-hand sides (SPORFS/DPORFS)
F07FJF Inverse of a real symmetric positive-definite matrix, matrix already factorized by F07FDF (SPOTRI/DPOTRI)
F07FRF Cholesky factorization of complex Hermitian positive-definite matrix (CPOTRF/ZPOTRF)

F07FSF	Solution of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07FRF (CPOTRS/ZPOTRS)
F07FUF	Estimate condition number of complex Hermitian positive-definite matrix, matrix already factorized by F07FRF (CPOCON/ZPOCON)
F07FVF	Refined solution with error bounds of complex Hermitian positive-definite system of linear equations, multiple right-hand sides (CPORFS/ZPORFS)
F07FWF	Inverse of a complex Hermitian positive-definite matrix, matrix already factorized by F07FRF (CPOTRI/ZPOTRI)
F07GDF	Cholesky factorization of a real symmetric positive-definite matrix, packed storage (SPPTRF/DPPTRF)
F07GEF	Solution of real symmetric positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07GDF, packed storage (SPPTRS/DPPTRS)
F07GGF	Estimate condition number of real symmetric positive-definite matrix, matrix already factorized by F07GDF, packed storage (SPPCON/DPPCON)
F07GHF	Refined solution with error bounds of real symmetric positive-definite system of linear equations, multiple right-hand sides, packed storage (SPPRFS/DPPRFS)
F07GJF	Inverse of a real symmetric positive-definite matrix, matrix already factorized by F07GDF, packed storage (SPPTRI/DPPTRI)
F07GRF	Cholesky factorization of complex Hermitian positive-definite matrix, packed storage (CPPTRF/ZPPTRF)
F07GSF	Solution of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07GRF, packed storage (CPPTRS/ZPPTRS)
F07GUF	Estimate condition number of complex Hermitian positive-definite matrix, matrix already factorized by F07GRF, packed storage (CPPCON/ZPPCON)
F07GVF	Refined solution with error bounds of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, packed storage (CPPRFS/ZPPRFS)
F07GWF	Inverse of a complex Hermitian positive-definite matrix, matrix already factorized by F07GRF, packed storage (CPPTRI/ZPPTRI)
F07HDF	Cholesky factorization of real symmetric positive-definite band matrix (SPBTRF/DPBTRF)
F07HEF	Solution of real symmetric positive-definite band system of linear equations, multiple right-hand sides, matrix already factorized by F07HDF (SPBTRS/DPBTRS)
F07HGF	Estimate condition number of real symmetric positive-definite band matrix, matrix already factorized by F07HDF (SPBCON/DPBCON)
F07HHF	Refined solution with error bounds of real symmetric positive-definite band system of linear equations, multiple right-hand sides (SPBRFS/DPBRFS)
F07HRF	Cholesky factorization of complex Hermitian positive-definite band matrix (CPBTRF/ZPBTRF)
F07HSF	Solution of complex Hermitian positive-definite band system of linear equations, multiple right-hand sides, matrix already factorized by F07HRF (CPBTRS/ZPBTRS)
F07HUF	Estimate condition number of complex Hermitian positive-definite band matrix, matrix already factorized by F07HRF (CPBCON/ZPBCON)
F07HVF	Refined solution with error bounds of complex Hermitian positive-definite band system of linear equations, multiple right-hand sides (CPBRFS/ZPBRFS)
F07MDF	Bunch–Kaufman factorization of real symmetric indefinite matrix (SSYTRF/DSYTRF)
F07MEF	Solution of real symmetric indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07MDF (SSYTRS/DSYTRS)
F07MGF	Estimate condition number of real symmetric indefinite matrix, matrix already factorized by F07MDF (SSYCON/DSYCON)
F07MHF	Refined solution with error bounds of real symmetric indefinite system of linear equations, multiple right-hand sides (SSYRFS/DSYRFS)
F07MJF	Inverse of a real symmetric indefinite matrix, matrix already factorized by F07MDF (SSYTRI/DSYTRI)
F07MRF	Bunch–Kaufman factorization of complex Hermitian indefinite matrix (CHETRF/ZHETRF)
F07MSF	Solution of complex Hermitian indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07MRF (CHETRS/ZHETRS)

F07MUF Estimate condition number of complex Hermitian indefinite matrix, matrix already factorized by F07MRF (CHECON/ZHECON)

F07MVF Refined solution with error bounds of complex Hermitian indefinite system of linear equations, multiple right-hand sides (CHERFS/ZHERFS)

F07MWF Inverse of a complex Hermitian indefinite matrix, matrix already factorized by F07MRF (CHETRI/ZHETRI)

F07NRF Bunch–Kaufman factorization of complex symmetric matrix (CSYTRF/ZSYTRF)

F07NSF Solution of complex symmetric system of linear equations, multiple right-hand sides, matrix already factorized by F07NRF (CSYTRS/ZSYTRS)

F07NUF Estimate condition number of complex symmetric matrix, matrix already factorized by F07NRF (CSYCON/ZSYCON)

F07NVF Refined solution with error bounds of complex symmetric system of linear equations, multiple right-hand sides (CSYRFS/ZSYRFS)

F07NWF Inverse of a complex symmetric matrix, matrix already factorized by F07NRF (CSYTRI/ZSYTRI)

F07PDF Bunch–Kaufman factorization of real symmetric indefinite matrix, packed storage (SSPTRF/DSPTRF)

F07PEF Solution of real symmetric indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07PDF, packed storage (SSPTRS/DSPTRS)

F07PGF Estimate condition number of real symmetric indefinite matrix, matrix already factorized by F07PDF, packed storage (SSPCON/DSPCON)

F07PHF Refined solution with error bounds of real symmetric indefinite system of linear equations, multiple right-hand sides, packed storage (SSPRFS/DSPRFS)

F07PJF Inverse of a real symmetric indefinite matrix, matrix already factorized by F07PDF, packed storage (SSPTRI/DSPTRI)

F07PRF Bunch–Kaufman factorization of complex Hermitian indefinite matrix, packed storage (CHPTRF/ZHPTRF)

F07PSF Solution of complex Hermitian indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07PRF, packed storage (CHPTRS/ZHPTRS)

F07PUF Estimate condition number of complex Hermitian indefinite matrix, matrix already factorized by F07PRF, packed storage (CHPCON/ZHPCON)

F07PVF Refined solution with error bounds of complex Hermitian indefinite system of linear equations, multiple right-hand sides, packed storage (CHPRFS/ZHPRFS)

F07PWF Inverse of a complex Hermitian indefinite matrix, matrix already factorized by F07PRF, packed storage (CHPTRI/ZHPTRI)

F07QRF Bunch–Kaufman factorization of complex symmetric matrix, packed storage (CSPTRF/ZSPTRF)

F07QSF Solution of complex symmetric system of linear equations, multiple right-hand sides, matrix already factorized by F07QRF, packed storage (CSPTRS/ZSPTRS)

F07QUF Estimate condition number of complex symmetric matrix, matrix already factorized by F07QRF, packed storage (CSPCON/ZSPCON)

F07QVF Refined solution with error bounds of complex symmetric system of linear equations, multiple right-hand sides, packed storage (CSPRFS/ZSPRFS)

F07QWF Inverse of a complex symmetric matrix, matrix already factorized by F07QRF, packed storage (CSPTRI/ZSPTRI)

F07TEF Solution of real triangular system of linear equations, multiple right-hand sides (STRTRS/DTRTRS)

F07TGF Estimate condition number of real triangular matrix (STRCON/DTRCON)

F07THF Error bounds for solution of real triangular system of linear equations, multiple right-hand sides (STRRFS/DTRRFS)

F07TJF Inverse of a real triangular matrix (STRTRI/DTRTRI)

F07TSF Solution of complex triangular system of linear equations, multiple right-hand sides (CTRTRS/ZTRTRS)

F07TUF Estimate condition number of complex triangular matrix (CTRCON/ZTRCON)

F07TVF Error bounds for solution of complex triangular system of linear equations, multiple right-hand sides (CTRRFS/ZTRRFS)

F07TWF Inverse of a complex triangular matrix (CTRTRI/ZTRTRI)
F07UEF Solution of real triangular system of linear equations, multiple right-hand sides, packed storage (STPTRS/DTPTRS)
F07UGF Estimate condition number of real triangular matrix, packed storage (STPCON/DTPCON)
F07UHF Error bounds for solution of real triangular system of linear equations, multiple right-hand sides, packed storage (STPRFS/DTPRFS)
F07UJF Inverse of a real triangular matrix, packed storage (STPTRI/DTPTRI)
F07USF Solution of complex triangular system of linear equations, multiple right-hand sides, packed storage (CTPTRS/ZTPTRS)
F07UUF Estimate condition number of complex triangular matrix, packed storage (CTPCON/ZTPCON)
F07UVF Error bounds for solution of complex triangular system of linear equations, multiple right-hand sides, packed storage (CTPRFS/ZTPRFS)
F07UWF Inverse of a complex triangular matrix, packed storage (CTPTRI/ZTPTRI)
F07VEF Solution of real band triangular system of linear equations, multiple right-hand sides (STBTRS/DTBTRS)
F07VGF Estimate condition number of real band triangular matrix (STBCON/DTBCON)
F07VHF Error bounds for solution of real band triangular system of linear equations, multiple right-hand sides (STBRFS/DTBRFS)
F07VSF Solution of complex band triangular system of linear equations, multiple right-hand sides (CTBTRS/ZTBTRS)
F07VUF Estimate condition number of complex band triangular matrix (CTBCON/ZTBCON)
F07VVF Error bounds for solution of complex band triangular system of linear equations, multiple right-hand sides (CTBRFS/ZTBRFS)

Chapter F08 – Least-squares and Eigenvalue Problems (LAPACK)

F08AEF QR factorization of real general rectangular matrix (SGEQRF/DGEQRF)
F08AFF Form all or part of orthogonal Q from QR factorization determined by F08AEF or F08BEF (SORGQR/DORGQR)
F08AGF Apply orthogonal transformation determined by F08AEF or F08BEF (SORMQR/DORMQR)
F08AHF LQ factorization of real general rectangular matrix (SGELQF/DGELQF)
F08AJF Form all or part of orthogonal Q from LQ factorization determined by F08AHF (SORGLQ/DORGLQ)
F08AKF Apply orthogonal transformation determined by F08AHF (SORMLQ/DORMLQ)
F08ASF QR factorization of complex general rectangular matrix (CGEQRF/ZGEQRF)
F08ATF Form all or part of unitary Q from QR factorization determined by F08ASF or F08BSF (CUNGQR/ZUNGQR)
F08AUF Apply unitary transformation determined by F08ASF or F08BSF (CUNMQR/ZUNMQR)
F08AVF LQ factorization of complex general rectangular matrix (CGELQF/ZGELQF)
F08AWF Form all or part of unitary Q from LQ factorization determined by F08AVF (CUNGLQ/ZUNGLQ)
F08AXF Apply unitary transformation determined by F08AVF (CUNMLQ/ZUNMLQ)
F08BEF Form all or part of orthogonal Q from QR factorization determined by F08AEF or F08BEF (SORGQR/DORGQR)
F08BSF Form all or part of unitary Q from QR factorization determined by F08ASF or F08BSF (CUNGQR/ZUNGQR)
F08FEF Orthogonal reduction of real symmetric matrix to symmetric tridiagonal form (SSYTRD/DSYTRD)
F08FFF Generate orthogonal transformation matrix from reduction to tridiagonal form determined by F08FEF (SORGTR/DORGTR)
F08FGF Apply orthogonal transformation determined by F08FEF (SORMTR/DORMTR)
F08FSF Unitary reduction of complex Hermitian matrix to real symmetric tridiagonal form (CHETRD/ZHETRD)
F08FTF Generate unitary transformation matrix from reduction to tridiagonal form determined by F08FSF (CUNGTR/ZUNGTR)
F08FUF Apply unitary transformation matrix determined by F08FSF (CUNMTR/ZUNMTR)
F08GEF Orthogonal reduction of real symmetric matrix to symmetric tridiagonal form, packed storage (SSPTRD/DSPTRD)

F08GFF Generate orthogonal transformation matrix from reduction to tridiagonal form determined by F08GEF (SOPGTR/DOPGTR)

F08GGF Apply orthogonal transformation determined by F08GEF (SOPMTR/DOPMTR)

F08GSF Unitary reduction of complex Hermitian matrix to real symmetric tridiagonal form, packed storage (CHPTRD/ZHPTRD)

F08GTF Generate unitary transformation matrix from reduction to tridiagonal form determined by F08GSF (CUPGTR/ZUPGTR)

F08GUF Apply unitary transformation matrix determined by F08GSF (CUPMTR/ZUPMTR)

F08HEF Orthogonal reduction of real symmetric band matrix to symmetric tridiagonal form (SSBTRD/DBSTRD)

F08HSF Unitary reduction of complex Hermitian band matrix to real symmetric tridiagonal form (CHBTRD/ZHBTRD)

F08JEF All eigenvalues and eigenvectors of real symmetric tridiagonal matrix, reduced from real symmetric matrix using implicit QL or QR (SSTEQR/DSTEQR)

F08JFF All eigenvalues of real symmetric tridiagonal matrix, root-free variant of QL or QR (SSTERF/DSTERF)

F08JGF All eigenvalues and eigenvectors of real symmetric positive definite tridiagonal matrix, reduced from real symmetric positive definite matrix (SPTEQR/DPTEQR)

F08JJF Selected eigenvalues of real symmetric tridiagonal matrix by bisection (SSTEBZ/DSTEBZ)

F08JKF Selected eigenvectors of real symmetric tridiagonal matrix by inverse iteration, storing eigenvectors in real array (SSTEIN/DSTEIN)

F08JSF All eigenvalues and eigenvectors of real symmetric tridiagonal matrix, reduced from complex Hermitian matrix, using implicit QL or QR (CSTEQR/ZSTEQR)

F08JUF All eigenvalues and eigenvectors of real symmetric positive definite tridiagonal matrix, reduced from complex Hermitian positive definite matrix (CPTEQR/ZPTEQR)

F08JXF Selected eigenvectors of real symmetric tridiagonal matrix by inverse iteration, storing eigenvectors in complex array (CSTEIN/ZSTEIN)

F08KEF Orthogonal reduction of real general rectangular matrix to bidiagonal form (SGEBRD/DGEBRD)

F08KFF Generate orthogonal transformation matrices from reduction to bidiagonal form determined by F08KEF (SORGBR/DORGBR)

F08KGF Apply orthogonal transformations from reduction to bidiagonal form determined by F08KEF (SORMBR/DORMBR)

F08KSF Unitary reduction of complex general rectangular matrix to bidiagonal form (CGEBRD/ZGEBRD)

F08KTF Generate unitary transformation matrices from reduction to bidiagonal form determined by F08KSF (CUNGBR/ZUNGBR)

F08KUF Apply unitary transformations from reduction to bidiagonal form determined by F08KSF (CUNMBR/ZUNMBR)

F08MEF SVD of real bidiagonal matrix reduced from real general matrix (SBDSQR/DBDSQR)

F08MSF SVD of real bidiagonal matrix reduced from complex general matrix (CBDSQR/ZBDSQR)

F08NEF Orthogonal reduction of real general matrix to upper Hessenberg form (SGEHRD/DGEHRD)

F08NFF Generate orthogonal transformation matrix from reduction to Hessenberg form determined by F08NEF (SORGHR/DORGHR)

F08NGF Apply orthogonal transformation matrix from reduction to Hessenberg form determined by F08NEF (SORMHR/DORMHR)

F08NHF Balance real general matrix (SGEBAL/DGEBAL)

F08NJF Transform eigenvectors of real balanced matrix to those of original matrix supplied to F08NHF (SGEBAK/DGEBAK)

F08NSF Unitary reduction of complex general matrix to upper Hessenberg form (CGEHRD/ZGEHRD)

F08NTF Generate unitary transformation matrix from reduction to Hessenberg form determined by F08NSF (CUNGHR/ZUNGHR)

F08NUF Apply unitary transformation matrix from reduction to Hessenberg form determined by F08NSF (CUNMHR/ZUNMHR)

F08NVF Balance complex general matrix (CGEBAL/ZGEBAL)

F08NWF Transform eigenvectors of complex balanced matrix to those of original matrix supplied to F08NVF (CGEBAK/ZGEBAK)

F08PEF Eigenvalues and Schur factorization of real upper Hessenberg matrix reduced from real general matrix (SHSEQR/DHSEQR)

F08PKF Selected right and/or left eigenvectors of real upper Hessenberg matrix by inverse iteration (SHSEIN/DHSEIN)

F08PSF Eigenvalues and Schur factorization of complex upper Hessenberg matrix reduced from complex general matrix (CHSEQR/ZHSEQR)

F08PXF Selected right and/or left eigenvectors of complex upper Hessenberg matrix by inverse iteration (CHSEIN/ZHSEIN)

F08QFF Reorder Schur factorization of real matrix using orthogonal similarity transformation (STREXC/DTREXC)

F08QGF Reorder Schur factorization of real matrix, form orthonormal basis of right invariant subspace for selected eigenvalues, with estimates of sensitivities (STRSEN/DTRSEN)

F08QHF Solve real Sylvester matrix equation $AX + XB = C$, A and B are upper quasi-triangular or transposes (STRSYL/DTRSYL)

F08QKF Left and right eigenvectors of a real upper quasi-triangular matrix (STREVC/DTREVC)

F08QLF Estimates of sensitivities of selected eigenvalues and eigenvectors of real upper quasi-triangular matrix (STRSNA/DTRSNA)

F08QTF Reorder Schur factorization of complex matrix using unitary similarity transformation (CTREXC/ZTREXC)

F08QUF Reorder Schur factorization of complex matrix, form orthonormal basis of right invariant subspace for selected eigenvalues, with estimates of sensitivities (CTRSEN/ZTRSEN)

F08QVF Solve complex Sylvester matrix equation $AX + XB = C$, A and B are upper triangular or conjugate-transposes (CTRSYL/ZTRSYL)

F08QXF Left and right eigenvectors of a complex upper triangular matrix (CTREVC/ZTREVC)

F08QYF Estimates of sensitivities of selected eigenvalues and eigenvectors of complex upper triangular matrix (CTRSNA/ZTRSNA)

F08SEF Reduction to standard form of real symmetric-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$, B factorized by F07FDF (SSYGST/DSYGST)

F08SSF Reduction to standard form of complex Hermitian-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$, B factorized by F07FRF (CHEGST/ZHEGST)

F08TEF Reduction to standard form of real symmetric-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$, packed storage, B factorized by F07GDF (SSPGST/DSPGST)

F08TSF Reduction to standard form of complex Hermitian-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$, packed storage, B factorized by F07GRF (CHPGST/ZHPGST)

Chapter F11 – Sparse Linear Algebra

F11BAF Real sparse nonsymmetric linear systems, set-up for F11BBF

F11BBF Real sparse nonsymmetric linear systems, preconditioned RGMRES, CGS or Bi-CGSTAB

F11BCF Real sparse nonsymmetric linear systems, diagnostic for F11BBF

F11DAF Real sparse nonsymmetric linear systems, incomplete LU factorization

F11DBF Solution of linear system involving incomplete LU preconditioning matrix generated by F11DAF

F11DCF Solution of real sparse nonsymmetric linear system, RGMRES, CGS or Bi-CGSTAB method, preconditioner computed by F11DAF (Black Box)

F11DDF Solution of linear system involving pre-conditioning matrix gnerated by applying SSOR to real sparse nonsymmetric matrix

F11DEF Solution of real sparse nonsymmetric linear system, RGMRES, CGS, or Bi-CGSTAB method, Jacobi or SSOR preconditioner (Black Box)

F11GAF Real sparse symmetric linear systems, set-up for F11GBF

F11GBF Real sparse symmetric linear systems, pre-conditioned conjugate gradient or Lanczos

F11GCF Real sparse symmetric linear systems, diagnostic for F11GBF

F11JAF Real sparse symmetric matrix, incomplete Cholesky factorization

F11JBF Solution of linear system involving incomplete Cholseky preconditioning matrix generated by F11JAF

F11JCF	Solution of real sparse symmetric linear system, conjugate gradient/Lanczos method, preconditioner computed by F11JAF (Black Box)
F11JDF	Solution of linear system involving preconditioning matrix generated by applying SSOR to real sparse symmetric matrix
F11JEF	Solution of real sparse symmetric linear system, conjugate gradient/Lanczos method, Jacobi or SSOR preconditioner (Black Box)
F11XAF	Real sparse nonsymmetric matrix vector multiply
F11XEF	Real sparse symmetric matrix vector multiply
F11ZAF	Real sparse nonsymmetric matrix reorder routine
F11ZBF	Real sparse symmetric matrix reorder routine

Chapter G01 – Simple Calculations and Statistical Data

G01AAF	Mean, variance, skewness, kurtosis etc, one variable, from raw data
G01ABF	Mean, variance, skewness, kurtosis etc, two variables, from raw data
G01ADF	Mean, variance, skewness, kurtosis etc, one variable, from frequency table
G01AEF	Frequency table from raw data
G01AFF	Two-way contingency table analysis, with χ^2/Fisher's exact test
G01AGF	Lineprinter scatterplot of two variables
G01AHF	Lineprinter scatterplot of one variable against Normal scores
G01AJF	Lineprinter histogram of one variable
G01ALF	Computes a five-point summary (median, hinges and extremes)
G01ARF	Constructs a stem and leaf plot
G01ASF	Constructs a box and whisker plot
G01BJF	Binomial distribution function
G01BKF	Poisson distribution function
G01BLF	Hypergeometric distribution function
G01DAF	Normal scores, accurate values
G01DBF	Normal scores, approximate values
G01DCF	Normal scores, approximate variance–covariance matrix
G01DDF	Shapiro and Wilk's W test for Normality
G01DHF	Ranks, Normal scores, approximate Normal scores or exponential (Savage) scores
G01EAF	Computes probabilities for the standard Normal distribution
G01EBF	Computes probabilities for Student's t-distribution
G01ECF	Computes probabilities for χ^2 distribution
G01EDF	Computes probabilities for F-distribution
G01EEF	Computes upper and lower tail probabilities and probability density function for the beta distribution
G01EFF	Computes probabilities for the gamma distribution
G01EMF	Computes probability for the Studentized range statistic
G01EPF	Computes bounds for the significance of a Durbin–Watson statistic
G01ERF	Computes probability for Von Mises distribution
G01EYF	Computes probabilities for the one-sample Kolmogorov–Smirnov distribution
G01EZF	Computes probabilities for the two-sample Kolmogorov–Smirnov distribution
G01FAF	Computes deviates for the standard Normal distribution
G01FBF	Computes deviates for Student's t-distribution
G01FCF	Computes deviates for the χ^2 distribution
G01FDF	Computes deviates for the F-distribution
G01FEF	Computes deviates for the beta distribution
G01FFF	Computes deviates for the gamma distribution
G01FMF	Computes deviates for the Studentized range statistic
G01GBF	Computes probabilities for the non-central Student's t-distribution
G01GCF	Computes probabilities for the non-central χ^2 distribution
G01GDF	Computes probabilities for the non-central F-distribution
G01GEF	Computes probabilities for the non-central beta distribution
G01HAF	Computes probability for the bivariate Normal distribution
G01HBF	Computes probabilities for the multivariate Normal distribution
G01JCF	Computes probability for a positive linear combination of χ^2 variables

G01JDF Computes lower tail probability for a linear combination of (central) χ^2 variables
G01MBF Computes reciprocal of Mills' Ratio
G01NAF Cumulants and moments of quadratic forms in Normal variables
G01NBF Moments of ratios of quadratic forms in Normal variables, and related statistics

Chapter G02 – Correlation and Regression Analysis

G02BAF Pearson product-moment correlation coefficients, all variables, no missing values
G02BBF Pearson product-moment correlation coefficients, all variables, casewise treatment of missing values
G02BCF Pearson product-moment correlation coefficients, all variables, pairwise treatment of missing values
G02BDF Correlation-like coefficients (about zero), all variables, no missing values
G02BEF Correlation-like coefficients (about zero), all variables, casewise treatment of missing values
G02BFF Correlation-like coefficients (about zero), all variables, pairwise treatment of missing values
G02BGF Pearson product-moment correlation coefficients, subset of variables, no missing values
G02BHF Pearson product-moment correlation coefficients, subset of variables, casewise treatment of missing values
G02BJF Pearson product-moment correlation coefficients, subset of variables, pairwise treatment of missing values
G02BKF Correlation-like coefficients (about zero), subset of variables, no missing values
G02BLF Correlation-like coefficients (about zero), subset of variables, casewise treatment of missing values
G02BMF Correlation-like coefficients (about zero), subset of variables, pairwise treatment of missing values
G02BNF Kendall/Spearman non-parametric rank correlation coefficients, no missing values, overwriting input data
G02BPF Kendall/Spearman non-parametric rank correlation coefficients, casewise treatment of missing values, overwriting input data
G02BQF Kendall/Spearman non-parametric rank correlation coefficients, no missing values, preserving input data
G02BRF Kendall/Spearman non-parametric rank correlation coefficients, casewise treatment of missing values, preserving input data
G02BSF Kendall/Spearman non-parametric rank correlation coefficients, pairwise treatment of missing values
G02BTF Update a weighted sum of squares matrix with a new observation
G02BUF Computes a weighted sum of squares matrix
G02BWF Computes a correlation matrix from a sum of squares matrix
G02BXF Computes (optionally weighted) correlation and covariance matrices
G02BYF Computes partial correlation/variance-covariance matrix from correlation/variance-covariance matrix computed by G02BXF
G02CAF Simple linear regression with constant term, no missing values
G02CBF Simple linear regression without constant term, no missing values
G02CCF Simple linear regression with constant term, missing values
G02CDF Simple linear regression without constant term, missing values
G02CEF Service routines for multiple linear regression, select elements from vectors and matrices
G02CFF Service routines for multiple linear regression, re-order elements of vectors and matrices
G02CGF Multiple linear regression, from correlation coefficients, with constant term
G02CHF Multiple linear regression, from correlation-like coefficients, without constant term
G02DAF Fits a general (multiple) linear regression model
G02DCF Add/delete an observation to/from a general linear regression model
G02DDF Estimates of linear parameters and general linear regression model from updated model
G02DEF Add a new variable to a general linear regression model
G02DFF Delete a variable from a general linear regression model
G02DGF Fits a general linear regression model for new dependent variable
G02DKF Estimates and standard errors of parameters of a general linear regression model for given constraints
G02DNF Computes estimable function of a general linear regression model and its standard error

G02EAF Computes residual sums of squares for all possible linear regressions for a set of independent variables
G02ECF Calculates R^2 and C_P values from residual sums of squares
G02EEF Fits a linear regression model by forward selection
G02FAF Calculates standardized residuals and influence statistics
G02FCF Computes Durbin–Watson test statistic
G02GAF Fits a generalized linear model with Normal errors
G02GBF Fits a generalized linear model with binomial errors
G02GCF Fits a generalized linear model with Poisson errors
G02GDF Fits a generalized linear model with gamma errors
G02GKF Estimates and standard errors of parameters of a general linear model for given constraints
G02GNF Computes estimable function of a generalized linear model and its standard error
G02HAF Robust regression, standard M-estimates
G02HBF Robust regression, compute weights for use with G02HDF
G02HDF Robust regression, compute regression with user-supplied functions and weights
G02HFF Robust regression, variance-covariance matrix following G02HDF
G02HKF Calculates a robust estimation of a correlation matrix, Huber's weight function
G02HLF Calculates a robust estimation of a correlation matrix, user-supplied weight function plus derivatives
G02HMF Calculates a robust estimation of a correlation matrix, user-supplied weight function

Chapter G03 – Multivariate Methods

G03AAF Performs principal component analysis
G03ACF Performs canonical variate analysis
G03ADF Performs canonical correlation analysis
G03BAF Computes orthogonal rotations for loading matrix, generalized orthomax criterion
G03BCF Computes Procrustes rotations
G03CAF Computes the maximum likelihood estimates of the parameters of a factor analysis model, factor loadings, communalities and residual correlations
G03CCF Computes factor score coefficients (for use after G03CAF)
G03DAF Computes test statistic for equality of within-group covariance matrices and matrices for discriminant analysis
G03DBF Computes Mahalanobis squared distances for group or pooled variance-covariance matrices (for use after G03DAF)
G03DCF Allocates observations to groups according to selected rules (for use after G03DAF)
G03EAF Computes distance matrix
G03ECF Hierarchical cluster analysis
G03EFF K-means cluster analysis
G03EHF Constructs dendrogram (for use after G03ECF)
G03EJF Computes cluster indicator variable (for use after G03ECF)
G03FAF Performs principal co-ordinate analysis, classical metric scaling
G03FCF Performs non-metric (ordinal) multidimensional scaling
G03ZAF Produces standardized values (z-scores) for a data matrix

Chapter G04 – Analysis of Variance

G04AGF Two-way analysis of variance, hierarchical classification, subgroups of unequal size
G04BBF Analysis of variance, randomized block or completely randomized design, treatment means and standard errors
G04BCF Analysis of variance, general row and column design, treatment means and standard errors
G04CAF Analysis of variance, complete factorial design, treatment means and standard errors
G04DAF Computes sum of squares for contrast between means
G04DBF Computes confidence intervals for differences between means computed by G04BBF or G04BCF
G04EAF Computes orthogonal polynomials or dummy variables for factor/classification variable

Chapter G05 – Random Number Generators

G05CAF Pseudo-random real numbers, uniform distribution over (0,1)
G05CBF Initialise random number generating routines to give repeatable sequence
G05CCF Initialise random number generating routines to give non-repeatable sequence
G05CFF Save state of random number generating routines
G05CGF Restore state of random number generating routines
G05DAF Pseudo-random real numbers, uniform distribution over (a, b)
G05DBF Pseudo-random real numbers, (negative) exponential distribution
G05DCF Pseudo-random real numbers, logistic distribution
G05DDF Pseudo-random real numbers, Normal distribution
G05DEF Pseudo-random real numbers, lognormal distribution
G05DFF Pseudo-random real numbers, Cauchy distribution
G05DHF Pseudo-random real numbers, χ^2 distribution
G05DJF Pseudo-random real numbers, Student's t-distribution
G05DKF Pseudo-random real numbers, F-distribution
G05DPF Pseudo-random real numbers, Weibull distribution
G05DRF Pseudo-random integer, Poisson distribution
G05DYF Pseudo-random integer from uniform distribution
G05DZF Pseudo-random logical (boolean) value
G05EAF Set up reference vector for multivariate Normal distribution
G05EBF Set up reference vector for generating pseudo-random integers, uniform distribution
G05ECF Set up reference vector for generating pseudo-random integers, Poisson distribution
G05EDF Set up reference vector for generating pseudo-random integers, binomial distribution
G05EEF Set up reference vector for generating pseudo-random integers, negative binomial distribution
G05EFF Set up reference vector for generating pseudo-random integers, hypergeometric distribution
G05EGF Set up reference vector for univariate ARMA time series model
G05EHF Pseudo-random permutation of an integer vector
G05EJF Pseudo-random sample from an integer vector
G05EWF Generate next term from reference vector for ARMA time series model
G05EXF Set up reference vector from supplied cumulative distribution function or probability distribution function
G05EYF Pseudo-random integer from reference vector
G05EZF Pseudo-random multivariate Normal vector from reference vector
G05FAF Generates a vector of random numbers from a uniform distribution
G05FBF Generates a vector of random numbers from an (negative) exponential distribution
G05FDF Generates a vector of random numbers from a Normal distribution
G05FEF Generates a vector of pseudo-random numbers from a beta distribution
G05FFF Generates a vector of pseudo-random numbers from a gamma distribution
G05FSF Generates vector of pseudo-random variates from Von Mises distribution
G05GAF Computes random orthogonal matrix
G05GBF Computes random correlation matrix
G05HDF Generates a realisation of a multivariate time series from a VARMA model

Chapter G07 – Univariate Estimation

G07AAF Computes confidence interval for the parameter of a binomial distribution
G07ABF Computes confidence interval for the parameter of a Poisson distribution
G07BBF Computes maximum likelihood estimates for parameters of the Normal distribution from grouped and/or censored data
G07BEF Computes maximum likelihood estimates for parameters of the Weibull distribution
G07CAF Computes t-test statistic for a difference in means between two Normal populations, confidence interval
G07DAF Robust estimation, median, median absolute deviation, robust standard deviation
G07DBF Robust estimation, M-estimates for location and scale parameters, standard weight functions
G07DCF Robust estimation, M-estimates for location and scale parameters, user-defined weight functions
G07DDF Computes a trimmed and winsorized mean of a single sample with estimates of their variance
G07EAF Robust confidence intervals, 1 sample
G07EBF Robust confidence intervals, 2 sample

Chapter G08 – Nonparametric Statistics

G08AAF Sign test on two paired samples
G08ACF Median test on two samples of unequal size
G08AEF Friedman two-way analysis of variance on k matched samples
G08AFF Kruskal–Wallis one-way analysis of variance on k samples of unequal size
G08AGF Performs the Wilcoxon one-sample (matched pairs) signed rank test
G08AHF Performs the Mann–Whitney U test on two independent samples
G08AJF Computes the exact probabilities for the Mann–Whitney U statistic, no ties in pooled sample
G08AKF Computes the exact probabilities for the Mann–Whitney U statistic, ties in pooled sample
G08ALF Performs the Cochran Q test on cross-classified binary data
G08BAF Mood's and David's tests on two samples of unequal size
G08CBF Performs the one-sample Kolmogorov–Smirnov test for standard distributions
G08CCF Performs the one-sample Kolmogorov–Smirnov test for a user-supplied distribution
G08CDF Performs the two-sample Kolmogorov–Smirnov test
G08CGF Performs the χ^2 goodness of fit test, for standard continuous distributions
G08DAF Kendall's coefficient of concordance
G08EAF Performs the runs up or runs down test for randomness
G08EBF Performs the pairs (serial) test for randomness
G08ECF Performs the triplets test for randomness
G08EDF Performs the gaps test for randomness
G08RAF Regression using ranks, uncensored data
G08RBF Regression using ranks, right-censored data

Chapter G10 – Smoothing in Statistics

G10ABF Fit cubic smoothing spline, smoothing parameter given
G10ACF Fit cubic smoothing spline, smoothing parameter estimated
G10BAF Kernel density estimate using Gaussian kernel
G10CAF Compute smoothed data sequence using running median smoothers
G10ZAF Reorder data to give ordered distinct observations

Chapter G11 – Contingency Table Analysis

G11AAF χ^2 statistics for two-way contingency table
G11BAF Computes multiway table from set of classification factors using selected statistic
G11BBF Computes multiway table from set of classification factors using given percentile/quantile
G11BCF Computes marginal tables for multiway table computed by G11BAF or G11BBF
G11SAF Contingency table, latent variable model for binary data
G11SBF Frequency count for G11SAF

Chapter G12 – Survival Analysis

G12AAF Computes Kaplan–Meier (product-limit) estimates of survival probabilities
G12BAF Fits Cox's proportional hazard model

Chapter G13 – Time Series Analysis

G13AAF Univariate time series, seasonal and non-seasonal differencing
G13ABF Univariate time series, sample autocorrelation function
G13ACF Univariate time series, partial autocorrelations from autocorrelations
G13ADF Univariate time series, preliminary estimation, seasonal ARIMA model
G13AEF Univariate time series, estimation, seasonal ARIMA model (comprehensive)
G13AFF Univariate time series, estimation, seasonal ARIMA model (easy-to-use)
G13AGF Univariate time series, update state set for forecasting
G13AHF Univariate time series, forecasting from state set
G13AJF Univariate time series, state set and forecasts, from fully specified seasonal ARIMA model
G13ASF Univariate time series, diagnostic checking of residuals, following G13AEF or G13AFF
G13AUF Computes quantities needed for range-mean or standard deviation-mean plot
G13BAF Multivariate time series, filtering (pre-whitening) by an ARIMA model

G13BBF Multivariate time series, filtering by a transfer function model
G13BCF Multivariate time series, cross-correlations
G13BDF Multivariate time series, preliminary estimation of transfer function model
G13BEF Multivariate time series, estimation of multi-input model
G13BGF Multivariate time series, update state set for forecasting from multi-input model
G13BHF Multivariate time series, forecasting from state set of multi-input model
G13BJF Multivariate time series, state set and forecasts from fully specified multi-input model
G13CAF Univariate time series, smoothed sample spectrum using rectangular, Bartlett, Tukey or Parzen lag window
G13CBF Univariate time series, smoothed sample spectrum using spectral smoothing by the trapezium frequency (Daniell) window
G13CCF Multivariate time series, smoothed sample cross spectrum using rectangular, Bartlett, Tukey or Parzen lag window
G13CDF Multivariate time series, smoothed sample cross spectrum using spectral smoothing by the trapezium frequency (Daniell) window
G13CEF Multivariate time series, cross amplitude spectrum, squared coherency, bounds, univariate and bivariate (cross) spectra
G13CFF Multivariate time series, gain, phase, bounds, univariate and bivariate (cross) spectra
G13CGF Multivariate time series, noise spectrum, bounds, impulse response function and its standard error
G13DBF Multivariate time series, multiple squared partial autocorrelations
G13DCF Multivariate time series, estimation of VARMA model
G13DJF Multivariate time series, forecasts and their standard errors
G13DKF Multivariate time series, updates forecasts and their standard errors
G13DLF Multivariate time series, differences and/or transforms (for use before G13DCF)
G13DMF Multivariate time series, sample cross-correlation or cross-covariance matrices
G13DNF Multivariate time series, sample partial lag correlation matrices, χ^2 statistics and significance levels
G13DPF Multivariate time series, partial autoregression matrices
G13DSF Multivariate time series, diagnostic checking of residuals, following G13DCF
G13DXF Calculates the zeros of a vector autoregressive (or moving average) operator
G13EAF Combined measurement and time update, one iteration of Kalman filter, time-varying, square root covariance filter
G13EBF Combined measurement and time update, one iteration of Kalman filter, time-invariant, square root covariance filter

Chapter H – Operations Research

H02BBF Integer programming problem, branch and bound method
H02BFF Interpret MPSX data file defining IP or LP problem, optimize and print solution
H02BUF Converts MPSX data file defining IP or LP problem to format required by H02BBF or E04MFF
H02BVF Prints IP or LP solutions with user specified names for rows and columns
H02BZF Integer programming solution, supplies further information on solution obtained by H02BBF
H03ABF Transportation problem, modified 'stepping stone' method
H03ADF Shortest path problem, Dijkstra's algorithm

Chapter M01 – Sorting

M01CAF Sort a vector, real numbers
M01CBF Sort a vector, integer numbers
M01CCF Sort a vector, character data
M01DAF Rank a vector, real numbers
M01DBF Rank a vector, integer numbers
M01DCF Rank a vector, character data
M01DEF Rank rows of a matrix, real numbers
M01DFF Rank rows of a matrix, integer numbers
M01DJF Rank columns of a matrix, real numbers
M01DKF Rank columns of a matrix, integer numbers

M01DZF Rank arbitrary data
M01EAF Rearrange a vector according to given ranks, real numbers
M01EBF Rearrange a vector according to given ranks, integer numbers
M01ECF Rearrange a vector according to given ranks, character data
M01ZAF Invert a permutation
M01ZBF Check validity of a permutation
M01ZCF Decompose a permutation into cycles

Chapter P01 – Error Trapping

P01ABF Return value of error indicator/terminate with error message

Chapter S – Approximations of Special Functions

S01BAF $\ln(1+x)$
S01EAF Complex exponential, e^z
S07AAF $\tan x$
S09AAF $\arcsin x$
S09ABF $\arccos x$
S10AAF $\tanh x$
S10ABF $\sinh x$
S10ACF $\cosh x$
S11AAF $\text{arctanh} x$
S11ABF $\text{arcsinh} x$
S11ACF $\text{arccosh} x$
S13AAF Exponential integral $E_1(x)$
S13ACF Cosine integral $\text{Ci}(x)$
S13ADF Sine integral $\text{Si}(x)$
S14AAF Gamma function
S14ABF Log Gamma function
S14ACF $\psi(x) - \ln x$
S14ADF Scaled derivatives of $\psi(x)$
S14BAF Incomplete gamma functions $P(a,x)$ and $Q(a,x)$
S15ABF Cumulative normal distribution function $P(x)$
S15ACF Complement of cumulative normal distribution function $Q(x)$
S15ADF Complement of error function $\text{erfc}(x)$
S15AEF Error function $\text{erf}(x)$
S15AFF Dawson's integral
S15DDF Scaled complex complement of error function, $\exp(-z^2)\text{erfc}(-iz)$
S17ACF Bessel function $Y_0(x)$
S17ADF Bessel function $Y_1(x)$
S17AEF Bessel function $J_0(x)$
S17AFF Bessel function $J_1(x)$
S17AGF Airy function $\text{Ai}(x)$
S17AHF Airy function $\text{Bi}(x)$
S17AJF Airy function $\text{Ai}'(x)$
S17AKF Airy function $\text{Bi}'(x)$
S17DCF Bessel functions $Y_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
S17DEF Bessel functions $J_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
S17DGF Airy functions $\text{Ai}(z)$ and $\text{Ai}'(z)$, complex z
S17DHF Airy functions $\text{Bi}(z)$ and $\text{Bi}'(z)$, complex z
S17DLF Hankel functions $H^{(j)}_{\nu+a}(z)$, $j = 1, 2$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
S18ACF Modified Bessel function $K_0(x)$
S18ADF Modified Bessel function $K_1(x)$
S18AEF Modified Bessel function $I_0(x)$
S18AFF Modified Bessel function $I_1(x)$
S18CCF Modified Bessel function $e^x K_0(x)$
S18CDF Modified Bessel function $e^x K_1(x)$
S18CEF Modified Bessel function $e^{-|x|} I_0(x)$

S18CFF Modified Bessel function $e^{-|x|}I_1(x)$
S18DCF Modified Bessel functions $K_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
S18DEF Modified Bessel functions $I_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
S19AAF Kelvin function ber x
S19ABF Kelvin function bei x
S19ACF Kelvin function ker x
S19ADF Kelvin function kei x
S20ACF Fresnel integral $S(x)$
S20ADF Fresnel integral $C(x)$
S21BAF Degenerate symmetrised elliptic integral of 1st kind $R_C(x, y)$
S21BBF Symmetrised elliptic integral of 1st kind $R_F(x, y, z)$
S21BCF Symmetrised elliptic integral of 2nd kind $R_D(x, y, z)$
S21BDF Symmetrised elliptic integral of 3rd kind $R_J(x, y, z, r)$
S21CAF Jacobian elliptic functions sn, cn and dn

Chapter X01 – Mathematical Constants

X01AAF Provides the mathematical constant π
X01ABF Provides the mathematical constant γ (Euler's Constant)

Chapter X02 – Machine Constants

X02AHF The largest permissable argument for sin and cos
X02AJF The machine precision
X02AKF The smallest positive model number
X02ALF The largest positive model number
X02AMF The safe range parameter
X02ANF The safe range parameter for complex floating-point arithmetic
X02BBF The largest representable integer
X02BEF The maximum number of decimal digits that can be represented
X02BHF The floating-point model parameter, b
X02BJF The floating-point model parameter, p
X02BKF The floating-point model parameter $e_{\min}$
X02BLF The floating-point model parameter $e_{\max}$
X02DAF Switch for taking precautions to avoid underflow
X02DJF The floating-point model parameter ROUNDS

Chapter X03 – Inner Products

X03AAF Real inner product added to initial value, basic/additional precision
X03ABF Complex inner product added to initial value, basic/additional precision

Chapter X04 – Input/Output Utilities

X04AAF Return or set unit number for error messages
X04ABF Return or set unit number for advisory messages
X04BAF Write formatted record to external file
X04BBF Read formatted record from external file
X04CAF Print a real general matrix (easy-to-use)
X04CBF Print a real general matrix (comprehensive)
X04CCF Print a real packed triangular matrix (easy-to-use)
X04CDF Print a real packed triangular matrix (comprehensive)
X04CEF Print a real packed banded matrix (easy-to-use)
X04CFF Print a real packed banded matrix (comprehensive)
X04DAF Print a complex general matrix (easy-to-use)
X04DBF Print a complex general matrix (comprehensive)
X04DCF Print a complex packed triangular matrix (easy-to-use)
X04DDF Print a complex packed triangular matrix (comprehensive)
X04DEF Print a complex packed banded matrix (easy-to-use)
X04DFF Print a complex packed banded matrix (comprehensive)

Chapter X05 – Date and Time Utilities

Withdrawn Routines

This document lists all those routines that have been present in earlier Marks of the Library (back as far as Mark 6), but have since been withdrawn. The document gives the names of the routines which are now recommended as their replacements. Another document Advice on Replacement Calls for Withdrawn/Superseded Routines gives more detailed guidance for those routines withdrawn since Mark 13.

Withdrawn Routine	**Mark of Withdrawal**	**Recommended Replacement**
C02ADF	15	C02AFF
C02AEF	16	C02AGF
C05AAF	9	C05ADF
C05ABF	9	C05ADF
C05ACF	9	C05ADF
C05NAF	10	C05NBF or C05NCF
C05PAF	8	C05PBF or C05PCF
C06AAF	9	C06ECF or C06FRF
C06ABF	9	C06EAF or C06FPF
C06ACF	12	C06EKF or C06FKF
C06ADF	12	C06FFF
D01AAF	8	D01AJF
D01ABF	8	D01AJF
D01ACF	9	D01BDF
D01ADF	8	D01BAF or D01BBF
D01AEF	8	D01BAF or D01BBF
D01AFF	8	D01BAF or D01BBF
D01AGF	9	D01AJF
D01FAF	11	D01GBF
D02AAF	8	D02PDF and related routines
D02ABF	8	D02PCF and related routines
D02ADF	9	D02HAF or D02GAF
D02AFF	9	D02TGF
D02AHF	8	D02CJF or D02QFF
D02AJF	8	D02EJF or D02NBF and related routines
D02BAF	18	D02PCF and associated D02P routines
D02BBF	18	D02PCF and associated D02P routines
D02BDF	18	D02PCF and associated D02P routines
D02CAF	18	D02CJF
D02CBF	18	D02CJF
D02CGF	18	D02CJF
D02CHF	18	D02CJF
D02EAF	18	D02EJF
D02EBF	18	D02EJF
D02EGF	18	D02EJF
D02EHF	18	D02EJF
D02PAF	18	D02PDF and associated D02P routines
D02QAF	14	D02QFF, D02QWF and D02QXF
D02QBF	13	D02NBF and related routines
D02QDF	17	D02NBF or D02NCF
D02QQF	17	not needed except with D02QDF
D02XGF	14	D02QZF
D02XHF	14	D02QZF
D02XAF	18	D02PXF and associated D02P routines
D02XBF	18	D02PXF and associated D02P routines
D02YAF	18	D02PDF and associated D02P routines
D03PAF	17	D03PCF

D03PBF	17	D03PCF
D03PGF	17	D03PCF
E01ACF	15	E01DAF and E02DEF
E01ADF	9	E01BAF
E02DBF	16	E02DEF
E04AAF	7	E04ABF
E04BAF	7	E04BBF
E04CDF	7	E04UCF
E04CEF	7	E04JAF
E04CFF	8	E04UCF
E04CGF	13	E04JAF
E04DBF	13	E04DGF
E04DCF	7	E04UCF or E04KDF
E04DDF	8	E04UCF or E04KDF
E04DEF	13	E04KAF
E04DFF	13	E04KCF
E04EAF	8	E04LBF
E04EBF	13	E04LAF
E04FAF	8	E04FCF or E04FDF
E04FBF	7	E04FCF or E04FDF
E04FDF	18	E04FYF
E04GAF	8	E04GBF, E04GCF, E04GDF or E04GEF
E04GCF	18	E04GYF
E04HAF	7	E04UCF
E04HBF	16	no longer required
E04JAF	18	E04JYF
E04JBF	16	E04UCF
E04KBF	16	E04UCF
E04MBF	18	E04MFF
E04NAF	18	E04NFF
E04UAF	13	E04UCF
E04VAF	12	E04UCF
E04VBF	12	E04UCF
E04VCF	17	E04UCF
E04VDF	17	E04UCF
E04WAF	12	E04UCF
E04ZAF	12	E04ZCF
E04ZBF	12	no longer required
F01AAF	17	F07ADF (SGETRF/DGETRF) and F07AJF (SGETRI/DGETRI)
F01ACF	16	F01ABF
F01AEF	18	F07FDF (SPOTRF/DPOTRF) and F08SEF (SSYGST/DSYGST)
F01AFF	18	F06YJF (STRSM/DTRSM)
F01AGF	18	F08FEF (SSYTRD/DSYTRD)
F01AHF	18	F08FGF (SORMTR/DORMTR)
F01AJF	18	F08FEF (SSYTRD/DSYTRD) and F08FFF (SORGTR/DORGTR)
F01AKF	18	F08NEF (SGEHRD/DGEHRD)
F01ALF	18	F08NGF (SORMHR/DORMHR)
F01AMF	18	F08NSF (CGEHRD/ZGEHRD)
F01ANF	18	F08NTF (CUNMHR/ZUNMHR)
F01APF	18	F08NFF (SORGHR/DORGHR)
F01ATF	18	F08NHF (SGEBAL/DGEBAL)
F01AUF	18	F08NJF (SGEBAK/DGEBAK)
F01AVF	18	F08NVF (CGEBAL/ZGEBAL)
F01AWF	18	F08NWF (CGEBAK/ZGEBAK)
F01AXF	18	F08BEF (SGEQPF/CGEQPF)
F01AYF	18	F08GEF (SSPTRD/DSPTRD)
F01AZF	18	F08GGF (SOPMTR/DOPMTR)
F01BCF	18	F08FSF (CHETRD/ZHETRD) and F08FTF (CUNGTR/ZUNGTR)

F01BDF	18	F07FDF (SPOTRF/DPOTRF) and F08SEF (SSYGST/DSYGST)
F01BEF	18	F06YFF (STRMM/DTRMM)
F01BFF	8	F07GDF (SPPTRF/DPPTRF) or F07PDF (SSPTRF/DSPTRF)
F01BHF	9	F02WEF
F01BJF	9	F08HEF (SSBTRD/DSBTRD)
F01BKF	9	F02WDF
F01BMF	9	F07BDF (SGBTRF/DGBTRF)
F01BNF	17	F07FRF (CPOTRF/ZPOTRF)
F01BPF	17	F07FRF (CPOTRF/ZPOTRF) and F07FWF (CPOTRI/ZPOTRI)
F01BQF	16	F07GDF (SPPTRF/DPPTRF) or F07PDF (SSPTRF/DSPTRF)
F01BTF	18	F07ADF (SGETRF/DGETRF)
F01BWF	18	F08HEF (SSBTRD/DSBTRD)
F01BXF	17	F07FDF (SPOTRF/DPOTRF)
F01CAF	14	F06QHF
F01CBF	14	F06QHF
F01CCF	7	F06QFF
F01CDF	15	F01CTF
F01CEF	15	F01CTF
F01CFF	14	F06QFF
F01CGF	15	F01CTF
F01CHF	15	F01CTF
F01CJF	8	F01CRF
F01CLF	16	F06YAF (SGEMM/DGEMM)
F01CMF	14	F06QFF
F01CNF	13	F06EFF (SCOPY/DCOPY)
F01CPF	13	F06EFF (SCOPY/DCOPY)
F01CQF	13	F06FBF
F01CSF	13	F06PEF (SSPMV/DSPMV)
F01DAF	13	F06EAF (SDOT/DDOT)
F01DBF	13	X03AAF
F01DCF	13	F06GAF (CDOTU/ZDOTU)
F01DDF	13	X03ABF
F01DEF	14	F06EAF (SDOT/DDOT)
F01LBF	18	F07BDF (SGBTRF/DGBTRF)
F01LZF	15	F08KEF (SGEBRD/DGEBRD) and F08KFF (SORGBR/DORGBR) or F08KGF (SORMBR/DORMBR)
F01NAF	17	F07BRF (CGBTRF/ZGBTRF)
F01QAF	15	F08AEF (SGEQRF/DGEQRF)
F01QBF	15	F01QJF
F01QCF	18	F08AEF (SGEQRF/DGEQRF)
F01QDF	18	F08AGF (SORMQR/DORMQR)
F01QEF	18	F08AFF (SORGQR/DORGQR)
F01QFF	18	F08BEF (SGEQPF/DGEQPF)
F01RCF	18	F08ASF (CGEQRF/ZGEQRF)
F01RDF	18	F08AUF (CUNMQR/ZUNMQR)
F01REF	18	F08ATF (CUNGQR/ZUNGQR)
F01RFF	18	F08BSF (CGEQPF/ZGEQPF)
F02AAF	18	F02FAF
F02ABF	18	F02FAF
F02ADF	18	F02FDF
F02AEF	18	F02FDF
F02AFF	18	F02EBF
F02AGF	18	F02EBF
F02AHF	8	F02ECF
F02AJF	18	F02GBF
F02AKF	18	F02GBF
F02ALF	8	F02GCF
F02AMF	18	F08JEF (SSTEQR/DSTEQR)

F02ANF	18	F08PSF (CHSEQR/ZHSEQR)
F02APF	18	F08PEF (SHSEQR/DHSEQR)
F02AQF	18	F08PEF (SHSEQR/DHSEQR) and F08QKF (STREVC/DTREVC)
F02ARF	18	F08PSF (CHSEQR/ZHSEQR) and F08QXF (CTREVC/ZTREVC)
F02ATF	8	F08PKF (SHSEIN/DHSEIN)
F02AUF	8	F08PXF (CHSEIN/ZHSEIN)
F02AVF	18	F08JFF (SSTERF/DSTERF)
F02AWF	18	F02HAF
F02AXF	18	F02HAF
F02AYF	18	F08JSF (CSTEQR/ZSTEQR)
F02BEF	18	F08JJF (SSTEBZ/DSTEBZ) and F08JKF (SSTEIN/DSTEIN)
F02BFF	18	F08JJF (SSTEBZ/DSTEBZ)
F02BKF	18	F08PKF (SHSEIN/DHSEIN)
F02BLF	18	F08PXF (CHSEIN/ZHSEIN)
F02BMF	9	F08HEF (SSBTRD/DSBTRD) and F08JJF (SSTEBZ/DSTEBZ)
F02SWF	18	F08KEF (SGEBRD/DGEBRD)
F02SXF	18	F08KFF (SORGBR/DORGBR) or F08KGF (SORMBR/DORMBR)
F02SYF	18	F08MEF (SBDSQR/DBDSQR)
F02SZF	15	F08MEF (SBDSQR/DBDSQR)
F02UWF	18	F08KSF (CGEBRD/ZGEBRD)
F02UXF	18	F08KTF (CUNGBR/ZUNGBR) or F08KUF (CUNMBR/ZUNMBR)
F02UYF	18	F08MSF (CBDSQR/ZBDSQR)
F02WAF	16	F02WEF
F02WBF	14	F02WEF
F02WCF	14	F02WEF
F03AGF	17	F07HDF (SPBTRF/DPBTRF)
F03AHF	17	F07ARF (CGETRF/ZGETRF)
F03AJF	8	F01BRF
F03AKF	8	F01BSF
F03ALF	9	F07BDF (SGBTRF/DGBTRF)
F03AMF	17	none – see the F03 Chapter Introduction
F04AKF	17	F07ASF (CGETRS/ZGETRS)
F04ALF	17	F07HEF (SPBTRS/DPBTRS)
F04ANF	18	F08AGF (SORMQR/DORMQR) and F06PJF (STRSV/DTRSV)
F04APF	8	F04AXF
F04AQF	16	F07GEF (SPPTRS/DPPTRS) or F07PEF (SSPTRS/DSPTRS)
F04AUF	9	F04JGF
F04AVF	9	F07BEF (SGBTRS/DGBTRS)
F04AWF	17	F07FSF (CPOTRS/ZPOTRS)
F04AYF	18	F07AEF (SGETRS/DGETRS)
F04AZF	17	F07FEF (SPOTRS/DPOTRS)
F04LDF	18	F07BEF (SGBTRS/DGBTRS)
F04NAF	17	F07BSF (CGBTRS/ZGBTRS)
F05ABF	14	F06EJF (SNRM2/DNRM2)
F06QGF	16	F06RAF, F06RCF and F06RJF
F06VGF	16	F06UAF, F06UCF and F06UJF
G01ACF	9	G04BBF
G01BAF	16	G01EBF
G01BBF	16	G01EDF
G01BCF	16	G01ECF
G01BDF	16	G01EEF
G01CAF	16	G01FBF
G01CBF	16	G01FDF
G01CCF	16	G01FCF
G01CDF	16	G01FEF
G01CEF	18	G01FAF
G02CJF	16	G02DAF and G02DGF
G04ADF	17	G04BCF

G04AEF	17	G04BBF
G04AFF	17	G04CAF
G05AAF	7	G05CAF
G05ABF	7	G05DAF
G05ACF	7	G05DBF
G05ADF	7	G05DDF
G05AEF	7	G05DDF
G05AFF	7	G05DEF
G05AGF	7	G05DFF
G05AHF	7	G05FFF
G05AJF	7	G05FFF
G05AKF	7	G05FFF
G05ALF	7	G05FEF
G05AMF	7	G05FEF
G05ANF	7	G05DHF
G05APF	7	G05DJF
G05AQF	7	G05DKF
G05ARF	7	G05EXF
G05ASF	7	G05EDF
G05ATF	7	G05EBF
G05AUF	7	G05EFF
G05AVF	7	G05ECF
G05AWF	7	G05EXF
G05AZF	7	G05EYF
G05BAF	7	G05CBF
G05BBF	7	G05CCF
G05DGF	16	G05FFF
G05DLF	16	G05FEF
G05DMF	16	G05FEF
G08ABF	16	G08AGF
G08ADF	16	G08AHF, G08AKF and G08AJF
G08CAF	16	G08CBF
G13DAF	17	G13DMF
H01ABF	12	E04MFF
H01ADF	12	E04MFF
H01AEF	9	E04MFF
H01AFF	12	E04MFF
H01BAF	12	E04MFF
H02AAF	12	E04NCF
H02BAF	15	H02BBF
M01AAF	13	M01DAF
M01ABF	13	M01DAF
M01ACF	13	M01DBF
M01ADF	13	M01DBF
M01AEF	13	M01DEF and M01EAF
M01AFF	13	M01DEF and M01EAF
M01AGF	13	M01DFF and M01EBF
M01AHF	13	M01DFF and M01EBF
M01AJF	16	M01DAF, M01ZAF and M01CAF
M01AKF	16	M01DAF, M01ZAF and M01CAF
M01ALF	13	M01DBF, M01ZAF and M01CBF
M01AMF	13	M01DBF, M01ZAF and M01CBF
M01ANF	13	M01CAF
M01APF	16	M01CAF
M01AQF	13	M01CBF
M01ARF	13	M01CBF
M01BAF	13	M01CCF
M01BBF	13	M01CCF

M01BCF	13	M01CCF
M01BDF	13	M01CCF
P01AAF	13	P01ABF
X02AAF	16	X02AJF
X02ABF	16	X02AKF
X02ACF	16	X02ALF
X02ADF	14	X02AJF and X02AKF
X02AEF	14	X02AMF
X02AFF	14	X02AMF
X02AGF	16	X02AMF
X02BAF	14	X02BHF
X02BCF	14	X02AMF
X02BDF	14	X02AMF
X02CAF	17	not needed except with F01BTF and F01BXF

If you need assistance in obtaining documentation of withdrawn routines, please contact your site representative.

Advice on Replacement Calls for Withdrawn/Superseded Routines

The following list illustrates how a call to routine, which has been withdrawn or superseded since Mark 13, may be replaced by a call to a new routine. The list indicates the minimum change necessary, but many of the replacement routines have additional flexibility and users may wish to take advantage of new features. It is strongly recommended that users consult the routine documents.

C02 – Zeros of Polynomials

C02ADF
Withdrawn at Mark 15

```
Old: CALL C02ADF(AR,AC,N,REZ,IMZ,TOL,IFAIL)
New: CALL C02AFF(A,N-1,SCALE,Z,W,IFAIL)
```

The coefficients are stored in the ***real*** array A of dimension $(2, N + 1)$ rather than in the arrays AR and AC, the zeros are returned in the ***real*** array Z of dimension (2,N) rather than in the arrays REZ and IMZ, and W is a ***real*** work array of dimension $(4 * (N + 1))$.

C02AEF
Withdrawn at Mark 15

```
Old: CALL C02AEF(A,N,REZ,IMZ,TOL,IFAIL)
New: CALL C02AGF(A,N-1,SCALE,Z,W,IFAIL)
```

The zeros are returned in the ***real*** array Z of dimension (2,N) rather than in the arrays REZ and IMZ, and W is a ***real*** work array of dimension $(2 * (N + 1))$.

D02 – Ordinary Differential Equations

D02BAF
Withdrawn at Mark 18

```
Old: CALL D02BAF(X,XEND,N,Y,TOL,FCN,W,IFAIL)
New: DO 10 L = 1,N
       THRES(L) = TOL
  10 CONTINUE
     CALL D02PVF(N,X,Y,XEND,TOL,THRES,2,'usualtask',.FALSE.,
     +           0.0e0,W,14*N,IFAIL)
     CALL D02PCF(FCN,XEND,X,Y,YP,YMAX,W,IFAIL)
```

THRES, YP and YMAX are ***real*** arrays of length N and the length of array W needs extending to length 14*N.

D02BBF
Withdrawn at Mark 18

```
Old: CALL D02BBF(X,XEND,N,Y,TOL,IRELAB,FCN,OUTPUT,W,IFAIL)
New: CALL D02PVF(N,X,Y,XEND,TOL,THRES,2,'usualtask',.FALSE.,
     +           0.0e0,W,14*N,IFAIL)
     ... set XWANT ...
  10 CONTINUE
     CALL D02PCF(FCN,XWANT,X,Y,YP,YMAX,W,IFAIL)
     IF (XWANT.LT.XEND) THEN
       ... reset XWANT ...
       GO TO 10
     ENDIF
```

THRES, YP and YMAX are ***real*** arrays of length N and the length of array W needs extending to length 14*N.

D02BDF
Withdrawn at Mark 18

```
Old: CALL D02BDF(X,XEND,N,Y,TOL,IRELAB,FCN,STIFF,YNORM,W,
   +            IW,M,OUTPUT,IFAIL)
New: CALL D02PVF(N,X,Y,XEND,TOL,THRES,2,'usualtask',.TRUE.,
   +            0.0e0,W,32N,IFAIL)
     ... set XWANT ...
  10 CONTINUE
     CALL D02PCF(FCN,XWANT,X,Y,YP,YMAX,IFAIL)
     IF (XWANT.LT.XEND) THEN
       ... reset XWANT ...
       GO TO 10
     ENDIF
     CALL D02PZF(RMSERR,ERRMAX,TERRMX,W,IFAIL)
```

THRES, YP, YMAX and RMSERR are ***real*** arrays of length N and W is now a ***real*** one-dimensional array of length 32*N.

D02CAF
Withdrawn at Mark 18

```
Old: CALL D02CAF(X,XEND,N,Y,TOL,FCN,W,IFAIL)
New: CALL D02CJF(X,XEND,N,Y,FCN,TOL,'M',D02CJX,D02CJW,W,IFAIL)
```

D02CJX is a subroutine provided in the NAG Fortran Library and D02CJW is a ***real*** function also provided. Both must be declared as EXTERNAL. The array W needs to be 5 elements greater in length.

D02CBF
Withdrawn at Mark 18

```
Old: CALL D02CBF(X,XEND,N,Y,TOL,IRELAB,FCN,OUTPUT,W,IFAIL)
New: CALL D02CJF(X,XEND,N,Y,FCN,TOL,RELABS,OUTPUT,D02CJW,W,IFAIL)
```

D02CJW is a ***real*** function provided in the NAG Fortran Library and must be declared as EXTERNAL. The array W needs to be 5 elements greater on length. The integer parameter IRELAB (which can take values 0, 1, 2) is provided for by the new CHARACTER*1 argument RELABS (whose corresponding values are 'M', 'A' and 'R').

D02CGF
Withdrawn at Mark 18

```
Old: CALL D02CGF(X,XEND,N,Y,TOL,HMAX,M,VAL,FCN,W,IFAIL)
New: CALL D02CJF(X,XEND,N,Y,FCN,TOL,'M',D02CJX,G,W,IFAIL)
     .
     .
     .
     real FUNCTION G(X,Y)
     real X,Y(*)
     G = Y(M)-VAL
     END
```

D02CJX is a subroutine provided in the NAG Fortran Library and should be declared as EXTERNAL. Note the functionality of HMAX is no longer available directly. Checking the value of Y(M)−VAL at intervals of length HMAX can be effected by a user-supplied procedure OUTPUT in place of D02CJX in the call described above. See the document for D02CJF for more details.

D02CHF
Withdrawn at Mark 18

```
Old: CALL D02CHF(X,XEND,N,Y,TOL,IRELAB,HMAX,FCN,G,W,IFAIL)
New: CALL D02CJF(X,XEND,N,Y,FCN,TOL,RELABS,D02CJX,G,W,IFAIL)
```

D02CJX is a subroutine provided by the NAG Fortran Library and should be declared as EXTERNAL. The functionality of HMAX can be provided as described under the replacement call for D02CGF above. The relationship between the parameters IRELAB and RELABS is described under the replacement call for D02CBF.

D02EAF
Withdrawn at Mark 18

```
Old: CALL D02EAF(X,XEND,N,Y,TOL,FCN,W,IW,IFAIL)
New: CALL D02EJF(X,XEND,N,Y,FCN,TOL,'M',D02EJX,D02EJW,D02EJY,W,IW,
    +            IFAIL)
```

D02EJY and D02EJX are subroutines provided in the NAG Fortran Library and D02EJW is a ***real*** function also provided. All must be declared as EXTERNAL.

D02EBF
Withdrawn at Mark 18

```
Old: CALL D02EBF(X,XEND,N,Y,TOL,IRELAB,FCN,MPED,PEDERV,OUTPUT,W,IW,
    +            IFAIL)
New: CALL D02EJF(X,XEND,N,Y,FCN,PEDERV,TOL,RELABS,OUTPUT,D02EJW,W,IW,
    +            IFAIL)
```

D02EJW is a ***real*** function provided in the NAG Fortran Library and must be declared as EXTERNAL. The integer parameter IRELAB (which can take values 0, 1, 2) is provided for by the new CHARACTER*1 argument RELABS (whose corresponding values are 'M', 'A' and 'R'). If MPED = 0 in the call of D02EBF then PEDERV must be the routine D02EJY, which is supplied in the Library and should be declared as EXTERNAL.

D02EGF
Withdrawn at Mark 18

```
Old: CALL D02EGF(X,XEND,N,Y,TOL,HMAX,M,VAL,FCN,W,IW,IFAIL)
New: CALL D02EJF(X,XEND,N,Y,FCN,D02EJY,TOL,'M',D02EJX,G,W,IW,IFAIL)
     .
     .
     .
     real FUNCTION G(X,Y)
     real X,Y(*)
     G = Y(M)-VAL
     END
```

D02EJY and D02EJX are subroutines provided in the NAG Fortran Library and should be declared as EXTERNAL. Note the functionality of HMAX is no longer available directly. Checking the value of Y(M)−VAL at intervals of length HMAX can be effected by a user-supplied procedure OUTPUT in place of D02EJX in the call described above. See the document for D02EJF for more details.

D02EHF
Withdrawn at Mark 18

```
Old: CALL D02EHF(X,XEND,N,Y,TOL,IRELAB,HMAX,MPED,PEDERV,FCN,G,W,IFAIL)
New: CALL D02EJF(X,XEND,N,Y,FCN,PEDERV,TOL,RELABS,D02EJX,G,W,IFAIL)
```

D02EJX is a subroutine provided by the NAG Fortran Library and should be declared as EXTERNAL. The functionality of HMAX can be provided as described under the replacement call for D02EGF above. The relationship between the parameters IRELAB and RELABS is described under the replacement call for D02EBF. If MPED = 0 in the call of D02EHF then PEDERV must be the routine D02EJY, which is supplied in the Library and should be declared as EXTERNAL.

D02PAF
Withdrawn at Mark 18

Existing programs should be modified to call D02PVF and D02PDF. The interfaces are significantly different and therefore precise details of a replacement call cannot be given. Please consult the appropriate routine documents.

D02QAF
Withdrawn at Mark 14

Existing programs should be modified to call D02QWF and D02QFF. The interfaces are significantly different and therefore precise details of a replacement call cannot be given. Please consult the appropriate routine documents.

D02QBF
Withdrawn at Mark 13

Existing programs should be modified to call D02NSF, D02NVF and D02NBF. The interfaces are significantly different and therefore precise details of a replacement call cannot be given. Please consult the appropriate routine documents.

D02QDF
Withdrawn at Mark 17

Existing programs should be modified to call D02NSF, D02NVF and D02NBF, or D02NTF, D02NVF and D02NCF. The interfaces are significantly different and therefore precise details of a replacement call cannot be given. Please consult the appropriate routine documents.

D02QQF
Withdrawn at Mark 17

Not needed except with D02QDF.

D02XAF, D02XBF
Withdrawn at Mark 18

Not needed except with D02PAF. The equivalent routine is D02PXF.

D02XGF, D02XHF
Withdrawn at Mark 14

Not needed except with D02QAF. The equivalent routine is D02QZF.

D02YAF
Withdrawn at Mark 18

There is no precise equivalent to this routine. The closest alternative routine is D02PDF.

D03 – Partial Differential Equations

D03PAF, D03PBF, D03PGF
Withdrawn at Mark 17

Should be modified to call D03PCF. The replacement routine is designed to solve a broader class of problems. Therefore it is not possible to give a precise replacement call for these routines. Users are advised to consult the appropriate routine documents for more details.

E01 – Interpolation

E01ACF
Withdrawn at Mark 15

```
Old: CALL E01ACF(A,B,X,Y,F,VAL,VALL,IFAIL,XX,WORK,AM,D,IG1,M1,N1)
New: CALL E01DAF(N1,M1,X,Y,F,PX,PY,LAMDA,MU,C,WRK,IFAIL)
     A1(1) = A
     B1(1) = B
     M = 1
     CALL E02DEF(M,PX,PY,A1,B1,LAMDA,MU,C,FF,WRK,IWRK,IFAIL)
     VAL = FF(1)
     VALL = VAL
```

where PX, PY and M are INTEGER variables, LAMDA is a ***real*** array of dimension (N1 + 4), MU is a ***real*** array of dimension (M1 + 4), C is a ***real*** array of dimension (N1*M1), WRK is a ***real*** array of dimension ((N1+6)*(M1+6)), A1, B1 and FF are ***real*** arrays of dimension (1), IWRK is an INTEGER array of dimension (M1).

The above new calls duplicate almost exactly the effect of the old call, except that the new routines produce a single interpolated value for each point, rather than the two alternative values VAL and VALL produced by the old routine. By attempting this duplication, however, efficiency is probably being sacrificed. In general it is preferable to evaluate the interpolating function provided by E01DAF at a set of M points, supplied in arrays A1 and B1, rather than at a single point. In this case, A1, B1 and FF must be dimensioned of length M.

Note also that E01ACF uses natural splines, i.e., splines having zero second derivatives at the ends of the ranges. This is likely to be slightly unsatisfactory, and E01DAF does not have this problem. It does mean however that results produced by E01DAF may not be exactly the same as those produced by E01ACF.

E01SEF
Superseded at Mark 18
Scheduled for withdrawal at Mark 20

```
Old: CALL E01SEF(M,X,Y,F,RNW,RNQ,NW,NQ,FNODES,MINNQ,WRK,IFAIL)
New: CALL E01SGF(M,X,Y,F,NW,NQ,IQ,LIQ,RQ,LRQ,IFAIL)
```

E01SEF is superseded by E01SGF which gives improved accuracy, facilities for obtaining gradient values and a consistent interface with the Mark 18 routine E01TGF for interpolation of scattered data in three dimensions.

The interpolant generated by the two routines will not be identical, but similar results may be obtained by using the same values of NW and NQ. Details of the interpolant are passed to the evaluator through the arrays IQ and RQ rather than FNODES and RNW.

E01SFF
Superseded at Mark 18
Scheduled for withdrawal at Mark 20

```
Old: CALL E01SFF(M,X,Y,F,RNW,FNODES,PX,PY,PF,IFAIL)
New: CALL E01SHF(M,X,Y,F,IQ,LIQ,RQ,LRQ,1,PX,PY,PF,QX,QY,IFAIL)
```

The two routines will not produce identical results due to differences in the generation routines E01SEF and E01SGF. Details of the interpolant are passed from E01SGF through the arrays IQ and RQ rather than FNODES and RNW.

E01SHF also returns gradient values in QX and QY and allows evaluation at arrays of points rather than just single points.

E02 – Curve and Surface Fitting

E02DBF
Withdrawn at Mark 16

```
Old: CALL E02DBF(M,PX,PY,X,Y,FF,LAMDA,MV,POINT,NPOINT,C,NC,IFAIL)
New: CALL E02DEF(M,PX,PY,X,Y,LAMDA,MU,C,FF,WRK,IWRK,IFAIL)
```

where WRK is a ***real*** array of dimension (PY – 4), and IWRK is an INTEGER array of dimension (PY – 4).

E04 – Minimizing or Maximizing a Function

E04CGF
Withdrawn at Mark 13

```
Old: CALL E04CGF(N,X,F,IW,LIW,W,LW,IFAIL)
New: CALL E04JAF(N,1,W,W(N+1),X,F,IW,LIW,W(2*N+1),LW-2*N,IFAIL)
```

E04DBF
Withdrawn at Mark 13

```
Old: CALL E04DBF(N,X,F,G,XTOL,FEST,DUM,W,FUNCT,MONIT,MAXCAL,IFAIL)
New: CALL E04DGF(N,OBJFUN,ITER,F,G,X,IWORK,WORK,IUSER,USER,IFAIL)
```

The subroutine providing function and gradient values to E04DGF is OBJFUN: it has a different parameter list to FUNCT, but can be constructed simply as:

```
      SUBROUTINE OBJFUN(MODE,N,XC,FC,GC,NSTATE,IUSER,USER)
      INTEGER    MODE, N, NSTATE, IUSER(*)
      real       XC(N), FC, GC(N), USER(*)
C
      CALL FUNCT(N,XC,FC,GC)
      RETURN
      END
```

The parameters IWORK and WORK are workspace parameters for E04DGF and must have lengths at least (N + 1) and (12*N) respectively. IUSER and USER must be declared as arrays each of length at least (1).

There is no parameter MONIT to E04DGF, but monitoring output may be obtained by calling an option setting routine. Similarly, values for FEST and MAXCAL may be supplied by calling an option setting routine. See the routine document for further information.

E04DEF
Withdrawn at Mark 13

```
Old: CALL E04DEF(N,X,F,G,IW,LIW,W,LW,IFAIL)
New: CALL E04KAF(N,1,W,W(N+1),X,F,G,IW,LIW,W(2*N+1),LW-2*N,IFAIL)
```

E04DFF
Withdrawn at Mark 13

```
Old: CALL E04DFF(N,X,F,G,IW,LIW,W,LW,IFAIL)
New: CALL E04KCF(N,1,W,W(N+1),X,F,G,IW,LIW,W(2*N+1),LW-2*N,IFAIL)
```

E04EBF
Withdrawn at Mark 13

```
Old: CALL E04EBF(N,X,F,G,IW,LIW,W,LW,IFAIL)
New: CALL E04LAF(N,1,W,W(N+1),X,F,G,IW,LIW,W(2*N+1),LW-2*N,IFAIL)
```

E04FDF
Superseded at Mark 18
Schedule for withdrawal at Mark 19

```
Old: CALL E04FDF(M,N,X,FSUMSQ,IW,LIW,W,LW,IFAIL)
New: CALL E04FYF(M,N,LSFUN,X,FSUMSQ,W,LW,IUSER,USER,IFAIL)
```

LSFUN appears in the parameter list instead of the fixed-name subroutine LSFUN1 of E04FDF. LSFUN must be declared as external in the calling subprogram. In addition it has an extra two parameters, IUSER and USER, over and above those of LSFUN1. It may be derived from LSFUN1 as follows:

```
      SUBROUTINE LSFUN(M,N,XC,FVECC,IUSER,USER)
      INTEGER    M, N, IUSER(*)
      real       XC(N), FVECC(M), USER(*)

      CALL LSFUN1(M,N,XC,FVECC)

      RETURN
      END
```

In general the extra parameters, IUSER and USER, should be declared in the calling program as IUSER(1) and USER(1), but will not need initialising.

E04GCF
Superseded at Mark 18
Scheduled for withdrawal at Mark 19

```
Old: CALL E04GCF(M,N,X,FSUMSQ,IW,LIW,W,LW,IFAIL)
New: CALL E04GYF(M,N,LSFUN,X,FSUMSQ,W,LW,IUSER,USER,IFAIL)
```

LSFUN appears in the parameter list instead of the fixed-name subroutine LSFUN2 of E04GCF. LSFUN must be declared as external in the calling subprogram. In addition it has an extra two parameters, IUSER and USER, over and above those of LSFUN2. It may be derived from LSFUN2 as follows:

```
SUBROUTINE LSFUN(M,N,XC,FVECC,FJACC,LJC,IUSER,USER)
INTEGER    M, N, LJC, IUSER(*)
real       XC(N), FVECC(M), FJACC(LJC,N), USER(*)

CALL LSFUN2(M,N,XC,FVECC,FJACC,LJC)

RETURN
END
```

In general the extra parameters, IUSER and USER, should be declared in the calling program as IUSER(1) and USER(1), but will not need initialising. If however, the array IW was used to pass information through E04GCF into LSFUN2, or get information from LSFUN2, then the array IUSER should be declared appropriately and used for this purpose.

E04GEF
Superseded at Mark 18
Scheduled for withdrawal at Mark 19

```
Old: CALL E04GEF(M,N,X,FSUMSQ,IW,LIW,W,LW,IFAIL)
New: CALL E04GZF(M,N,LSFUN,X,FSUMSQ,W,LW,IUSER,USER,IFAIL)
```

LSFUN appears in the parameter list instead of the fixed-name subroutine LSFUN2 of E04GEF. LSFUN must be declared as external in the calling subprogram. In addition it has an extra two parameters, IUSER and USER, over and above those of LSFUN2. It may be derived from LSFUN2 as follows:

```
SUBROUTINE LSFUN(M,N,X,FVECC,FJACC,LJC,IUSER,USER)
INTEGER    M, N, LJC, IUSER(*)
real       XC(N), FVECC(M), FJACC(LJC,N), USER(*)

CALL LSFUN2(M,N,XC,FVECC,FJACC,LJC)

RETURN
END
```

In general the extra parameters, IUSER and USER, should be declared in the calling program as IUSER(1) and USER(1), but will not need initialising. If however, the array IW was used to pass information through E04GEF into LSFUN2, or get information from LSFUN2, then the array IUSER should be declared appropriately and used for this purpose.

E04HBF
Withdrawn at Mark 16
Only required in conjunction with E04JBF.

E04HFF Superseded at Mark 18
Scheduled for withdrawal at Mark 19

```
Old: CALL E04HFF(M,N,X,FSUMSQ,IW,LIW,W,LW,IFAIL)
New: CALL E04HYF(M,N,LSFUN,LSHES,X,FSUMSQ,W,LW,IUSER,USER,IFAIL)
```

LSFUN and LSHES appear in the parameter list instead of the fixed-name subroutines LSFUN2 and LSHES2 of E04HFF. LSFUN and LSHES must both be declared as external in the calling subprogram. In addition they have an extra two parameters, IUSER and USER, over and above those of LSFUN2 and LSHES2. They may be derived from LSFUN2 and LSHES2 as follows:

```
SUBROUTINE LSFUN(M,N,XC,FVECC,FJACC,LJC,IUSER,USER)
INTEGER    M, N, LJC, IUSER(*)
real       XC(N), FVECC(M), FJACC(LJC,N), USER(*)

CALL LSFUN2(M,N,XC,FVECC,FJACC,LJC)

RETURN
END

SUBROUTINE LSHES(M,N,FVECC,XC,B,LB,IUSER,USER)
INTEGER    M, N, LB, IUSER(*)
real       FVECC(M), XC(N), B(LB), USER(*)

CALL LSHES2(M,N,FVECC,XC,B,LB)

RETURN
END
```

In general, the extra parameters, IUSER and USER, should be declared in the calling program as IUSER(1) and USER(1), but will not need initialising. If however, the array IW was used to pass information through E04HFF into LSFUN2 or LSHES2, or to get information from LSFUN2, then the array IUSER should be declared appropriately and used for this purpose.

E04JAF
Superseded at Mark 18
Scheduled for withdrawal at Mark 19

```
Old: CALL E04JAF(N,IBOUND,BL,BU,X,F,IW,LIW,LW,IFAIL)
New: CALL E04JYF(N,IBOUND,FUNCT,BL,BU,X,F,IW,LIW,W,LW,IUSER,USER,IFAIL)
```

FUNCT appears in the parameter list instead of the fixed-name subroutine FUNCT1 of E04JAF. FUNCT must be declared as external in the calling subprogram. In addition it has an extra two parameters, IUSER and USER, over and above those of FUNCT1. It may be derived from FUNCT1 as follows:

```
SUBROUTINE FUNCT(N,XC,FC,IUSER,USER)
INTEGER    N, IUSER(*)
real       XC(N), FC, USER(*)

CALL FUNCT1(N,XC,FC)

RETURN
END
```

The extra parameters, IUSER and USER, should be declared in the calling program as IUSER(1) and USER(1), but will not need initialising.

E04JBF
Withdrawn at Mark 16

No comparative calls are given between E04JBF and E04UCF since both routines have considerable flexibility and can be called with many different options. E04UCF allows some values to be passed to it, not through the parameter list, but as 'optional parameters', supplied through calls to E04UDF or E04UEF. Names of optional parameters are given here in bold type.

E04UCF is a more powerful routine than E04JBF, in that it allows for general linear and nonlinear constraints, and for some or all of the first derivatives to be supplied; however when replacing E04JBF, only the simple bound constraints are relevant, and only function values are assumed to be available.

Therefore E04UCF must be called with NCLIN = NCNLN = 0, with dummy arrays of size (1) supplied as the arguments A, C and CJAC, and with the name of the auxiliary routine E04UDM (UDME04 in some implementations) as the argument CONFUN. The optional parameter **Derivative Level** must be set to 0.

The subroutine providing function values to E04UCF in OBJFUN. It has a different parameter list to FUNCT, but can be constructed as follows:

```
      SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE,IUSER,USER)
      INTEGER    MODE, N, NSTATE, IUSER(*)
      real       X(N), OBJF, OBJGRD(N), USER(*)
      INTEGER    IFLAG,IW(1)
      real       W(1)
C
      IFLAG = 0
      CALL FUNCT(IFLAG,N,X,OBJF,OBJGRD,IW,1,W,1)
      IF (IFLAG.LT.0) MODE = IFLAG
      RETURN
      END
```

(This assumes that the arrays IW and W are not used to communicate between FUNCT and the calling program; E04UCF supplied the arrays IUSER and USER for this purpose.)

The functions of the parameters BL and BU are similar, but E04UCF has no parameter corresponding to IBOUND; all elements of BL and BU must be set (as when IBOUND = 0 in the call to E04JBF). The optional parameter **Infinite bound size** must be set to $1.0e+6$ if there are any infinite bounds. The function of the parameter ISTATE is similar but the specification is slightly different. The parameters F and G are equivalent to OBJF and OBJGRD of E04UCF. E04UCF does not allow a user-supplied routine MONIT, but intermediate output is provided by the routine, under the control of the optional parameters **Major print level** and **Minor print level**.

Most of the 'tuning' parameters in E04JBF have their counterparts as 'optional parameters' to E04UCF, as indicated in the following list, but the correspondence is not exact and the specifications must be read carefully.

IPRINT	**Minor print level**
INTYPE	**Cold start/Warm start**
MAXCAL	**Minor iteration limit** (note that this counts iterations rather than function calls)
ETA	**Line search tolerance**
XTOL	**Optimality tolerance** (note that this specifies the accuracy in F rather than the accuracy in X)
STEPMX	**Step limit**
DELTA	**Difference interval**

E04KAF
Superseded at Mark 18
Scheduled for withdrawal at Mark 19

```
Old: CALL E04KAF(N,IBOUND,BL,BU,X,F,G,IW,LIW,W,LW,IFAIL)
New: CALL E04KYF(N,IBOUND,FUNCT2,BL,BU,X,F,G,IW,LIW,W,LW,IUSER,USER,IFAIL)
```

FUNCT appears in the parameter list instead of the fixed-name subroutine FUNCT2 of E04KAF. FUNCT must be declared as external in the calling subprogram. In addition it has an extra two parameters, IUSER and USER, over and above those of FUNCT2. It may be derived from FUNCT2 as follows:

```
      SUBROUTINE FUNCT(N,XC,FC,GC,IUSER,USER)
      INTEGER    N, IUSER(*)
      real       XC(N), FC, GC(N), USER(*)

      CALL FUNCT2(N,XC,FC,GC)

      RETURN
      END
```

The extra parameters, IUSER and USER, should be declared in the calling program as IUSER(1) and USER(1), but will not need initialising.

E04KBF
Withdrawn at Mark 16

No comparative calls are given between E04KBF and E04UCF since both routines have considerable flexibility and can be called with many different options. Most of the advice given for replacing E04JBF (see above) applies also to E04KBF, and only the differences are given here.

The optional parameter **Derivative Level** must be set to 1.

The subroutine providing both function and gradient values to E04UCF is OBJFUN. It has a different parameter list to FUNCT, but can be constructed as follows:

```
      SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE,IUSER,USER)
      INTEGER    MODE, N, NSTATE, IUSER(*)
      real       X(N), OBJF, OBJGRD(N), USER(*)
      INTEGER    IW(1)
      real       W(1)
C
      CALL FUNCT(MODE,N,X,OBJF,OBJGRD,IW,1,W,1)
      RETURN
      END
```

E04MBF
Withdrawn at Mark 18

```
Old: CALL E04MBF(ITMAX,MSGLVL,N,NCLIN,NCTOTL,NROWA,A,BL,BU,CVEC,
    +            LINOBJ,X,ISTATE,OBJLP,CLAMDA,IWORK,LIWORK,WORK,
    +            LWORK,IFAIL)
New: CALL E04MFF(N,NCLIN,A,NROWA,BL,BU,CVEC,ISTATE,X,ITER,OBJLP,
    +            AX,CLAMDA,IWORK,LIWORK,WORK,LWORK,IFAIL)
```

The parameter NCTOTL is no longer required. Values for ITMAX, MSGLVL and LINOBJ may be supplied by calling an option setting routine.

E04MFF contains two additional parameters as follows:

ITER – INTEGER.

AX(*) – ***real*** array of dimension at least max(1,NCLIN).

The minimum value of the parameter LIWORK must be increased from 2×N to 2×N + 3. The minimum value of the parameter LWORK may also need to be changed. See the routine documents for further information.

E04NAF
Withdrawn at Mark 18

```
Old: CALL E04NAF(ITMAX,MSGLVL,N,NCLIN,NCTOTL,NROWA,NROWH,NCOLH,
    +            BIGBND,A,BL,BU,CVEC,FEATOL,HESS,QPHESS,COLD,LP,
    +            ORTHOG,X,ISTATE,ITER,OBJ,CLAMDA,IWORK,LIWORK,
    +            WORK,LWORK,IFAIL)
New: CALL E04NFF(N,NCLIN,A,NROWA,BL,BU,CVEC,HESS,NROWH,QPHESS,
    +            ISTATE,X,ITER,OBJ,AX,CLAMDA,IWORK,LIWORK,WORK,
    +            LWORK,IFAIL)
```

The specification of the subroutine QPHESS must also be changed as follows.

```
Old: SUBROUTINE QPHESS(N,NROWH,NCOLH,JTHCOL,HESS,X,HX)
     INTEGER    N, NROWH, NCOLH, JTHCOL
     real       HESS(NROWH,NCOLH), X(N), HX(N)
New: SUBROUTINE QPHESS(N,JTHCOL,HESS,NROWH,X,HX)
     INTEGER    N, JTHCOL, NROWH
     real       HESS(NROWH,*), X(N), HX(N)
```

The parameters NCTOTL, NCOLH and ORTHOG are no longer required. Values for ITMAX, MSGLVL, BIGBND, FEATOL, COLD and LP may be supplied by calling an option setting routine.

E04NFF contains one additional parameter as follows:

AX(*) – ***real*** array of dimension at least max(1,NCLIN).

The minimum value of the parameter LIWORK must be increased from 2×N to 2×N + 3. The minimum value of the parameter LWORK may also need to be changed. See the routine documents for further information.

E04UAF
Withdrawn at Mark 13

No comparative calls are given between E04UAF and E04UCF since both routines have considerable flexibility and can be called with many different options. However users of E04UAF should have no difficulty in making the transition. Most of the 'tuning' parameters in E04UAF have their counterparts as optional parameters to E04UCF, and these may be provided by calling an option setting routine prior to the call to E04UCF. The subroutines providing function and constraint values to E04UCF are OBJFUN and CONFUN respectively: they have different parameter lists to FUNCT1 and CON1, but can be constructed simply as:

```
      SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE,IUSER,USER)
      INTEGER    MODE, N, NSTATE, IUSER(*)
      real       X(N), OBJF, OBJGRD(N), USER(*)
C
      CALL FUNCT1(MODE,N,X,OBJF)
      RETURN
      END
      SUBROUTINE CONFUN(MODE,NCNLN,N,NROWJ,NEEDC,X,C,CJAC.NSTATE,
     +                  IUSER,USER)
      INTEGER    MODE, NCNLN, N, NROWJ, NEEDC(*), NSTATE, IUSER(*)
      real       X(X), C(*), CJAC(NROWJ,*), USER(*)
C
      CALL CON1(MODE,N,NCNLN,X,C)
      RETURN
      END
```

The parameters OBJGRD, NEEDC, CJAC, IUSER and USER are the same as those for E04UCF itself. It is important to note that, unlike FUNCT1 and CON1, a call to CONFUN is not preceded by a call to OBJFUN with the same values in X, so that FUNCT1 and CON1 will need to be modified if this property was being utilized. It should also be noted that E04UCF allows general linear constraints to be supplied separately from nonlinear constraints, and indeed this is to be encouraged, but the above call to CON1 assumes that linear constraints are being regarded as nonlinear.

E04UPF
Superseded at Mark 17
Scheduled for withdrawal at Mark 19

```
Old: CALL E04UPF(M,N,NCLIN,LDA,LDCJ,LDFJ,LDR,A,BL,BU,
    +            CONFUN,OBJFUN,ITER,ISTATE,C,CJAC,F,FJAC,
    +            CLAMDA,OBJF,R,X,IWORK,LIWORK,WORK,LWORK,
    +            IUSER,USER,IFAIL)
New: CALL E04UNF(M,N,NCLIN,LDA,LDCJ,LDFJ,LDR,A,BL,BU,Y,
    +            CONFUN,OBJFUN,ITER,ISTATE,C,CJAC,F,FJAC,
    +            CLAMDA,OBJF,R,X,IWORK,LIWORK,WORK,LWORK,
    +            IUSER,USER,IFAIL)
```

E04UNF contains one additional parameter as follows:

Y(M) – ***real*** array.

Note that a call to E04UPF is the same as a call to E04UNF with $Y(i) = 0.0$, for $i = 1, 2, \ldots, M$.

E04VCF
Withdrawn at Mark 17

```
Old: CALL E04VCF(ITMAX,MSGLVL,N,NCLIN,NCNLN,NCTOTL,NROWA,NROWJ,
    +            NROWR,BIGBND,EPSAF,ETA,FTOL,A,BL,BU,FEATOL,
    +            CONFUN,OBJFUN,COLD,FEALIN,ORTHOG,X,ISTATE,R,ITER,
    +            C,CJAC,OBJF,OBJGRD,CLAMDA,IWORK,LIWORK,WORK,LWORK,
    +            IFAIL)
New: CALL E04UCF(N,NCLIN,NCNLN,NROWA,NROWJ,NROWR,A,BL,BU,CONFUN,
    +            OBJFUN,ITER,ISTATE,C,CJAC,CLAMDA,OBJF,OBJGRD,R,X,
    +            IWORK,LIWORK,WORK,LWORK,IUSER,USER,IFAIL)
```

The specification of the subroutine OBJFUN must also be changed as follows.

```
Old: SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE)
     INTEGER    MODE, N, NSTATE
     real       X(N), OBJF, OBJGRD(N)
New: SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE,IUSER,USER)
     INTEGER    MODE, N, NSTATE, IUSER(*)
     real       X(N), OBJF, OBJGRD(N), USER(*)
```

If NCNLN > 0, the specification of the subroutine CONFUN must also be changed as follows.

```
Old: SUBROUTINE CONFUN(MODE,NCNLN,N,NROWJ,X,C,CJAC,NSTATE)
     INTEGER    MODE, NCNLN, N, NROWJ, NSTATE
     real       X(N), C(NROWJ), CJAC(NROWJ,N)
New: SUBROUTINE CONFUN(MODE,NCNLN,N,NROWJ,NEEDC,X,C,CJAC,NSTATE,
    +                   IUSER,USER)
     INTEGER    MODE, NCNLN, N, NROWJ, NEEDC(NCNLN), NSTATE, IUSER(*)
     real       X(N), C(NCNLN), CJAC(NROWJ,N), USER(*)
```

If NCNLN = 0, then the name of the dummy routine E04VDM (VDME04 in some implementations) may need to be changed to E04UDM (UDME04 in some implementations) in the calling program.

The parameters NCTOTL, EPSAF, FEALIN and ORTHOG are no longer required. Values for ITMAX, MSGLVL, BIGBND, ETA, FTOL, COLD and FEATOL may be supplied by calling an option setting routine.

E04UCF contains two additional parameters as follows:

IUSER(*) – INTEGER array of dimension at least 1.

USER(*) – ***real*** array of dimension at least 1.

The minimum value of the parameter LIWORK must be increased from 3×N + NCLIN + NCNLN to 3×N + NCLIN + 2×NCNLN. The minimum value of the parameter LWORK may also need to be changed. See the routine documents for further information.

E04VDF
Withdrawn at Mark 17

```
Old: IFAIL = 110
     CALL E04VDF(ITMAX,MSGLVL,N,NCLIN,NCNLN,NCTOTL,NROWA,NROWJ,
    +            CTOL,FTOL,A,BL,BU,CONFUN,OBJFUN,X,ISTATE,C,CJAC,
    +            CJAC,OBJF,OBJGRD,CLAMDA,IWORK,LIWORK,WORK,LWORK,
    +            IFAIL)
New: IFAIL = -1
     CALL E04UCF(N,NCLIN,NCNLN,NROWA,NROWJ,N,A,BL,BU,CONFUN,OBJFUN,
    +            ITER,ISTATE,C,CJAC,CLAMDA,OBJF,OBJGRD,R,X,IWORK,
    +            LIWORK,WORK,LWORK,IUSER,USER,IFAIL)
```

The specification of the subroutine OBJFUN must also be changed as follows.

```
Old: SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE)
     INTEGER    MODE, N, NSTATE
     real       X(N), OBJF, OBJGRD(N)
New: SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE,IUSER,USER)
     INTEGER    MODE, N, NSTATE, IUSER(*)
     real       X(N), OBJF, OBJGRD(N), USER(*)
```

If NCNLN > 0, the specification of the subroutine CONFUN must also be changed as follows.

```
Old: SUBROUTINE CONFUN(MODE,NCNLN,N,NROWJ,X,C,CJAC,NSTATE)
     INTEGER    MODE, NCNLN, N, NROWJ, NSTATE
     real       X(N), C(NROWJ), CJAC(NROWJ,N)
New: SUBROUTINE CONFUN(MODE,NCNLN,N,NROWJ,NEEDC,X,C,CJAC,NSTATE,
    +                   IUSER,USER)
     INTEGER    MODE, NCNLN, N, NROWJ, NEEDC(NCNLN), NSTATE, IUSER(*)
     real       X(N), C(NCNLN), CJAC(NROWJ,N), USER(*)
```

If NCNLN = 0, then the name of the dummy routine E04VDM (VDME04 in some implementations) may need to be changed to E04UDM (UDME04 in some implementations) in the calling program.

The parameter NCTOTL is no longer required. Values for ITMAX, MSGLVL, CTOL and FTOL may be supplied by calling an option setting routine.

E04UCF contains four additional parameters as follows:

ITER – INTEGER.

R(N,N) – ***real*** array.

IUSER(*) – INTEGER array of dimension at least 1.

USER(*) – ***real*** array of dimension at least 1.

The minimum value of the parameter LIWORK must be increased from 3×N + NCLIN + NCNLN to 3×N + NCLIN + 2×NCNLN. The minimum value of the parameter LWORK may also need to be changed. See the routine documents for further information.

F01 – Matrix Operations, Including Inversion

F01AAF
Withdrawn at Mark 17

```
Old: CALL F01AAF(A,IA,N,X,IX,WKSPCE,IFAIL)
New: CALL sgetrf(N,N,A,IA,IPIV,IFAIL)
     CALL F06QFF('General',N,N,A,IA,X,IX)
     CALL sgetri(N,X,IX,IPIV,WKSPCE,LWORK,IFAIL)
```

where IPIV is an INTEGER vector of length N, and the INTEGER LWORK is the length of array WKSPCE, which must be at least max(1,N). In the replacement calls, F07ADF (SGETRF/DGETRF) computes the *LU* factorization of the matrix *A*, F06QFF copies the factorization from A to X, and F07AJF (SGETRI/DGETRI) overwrites X by the inverse of *A*. If the original matrix *A* is no longer required, the call to F06QFF is not necessary, and references to X and IX in the call of F07AJF (SGETRI/DGETRI) may be replaced by references to A and IA, in which case A will be overwritten by the inverse.

F01ACF
Withdrawn at Mark 16

```
Old: CALL F01ACF(N,EPS,A,IA,B,IB,Z,L,IFAIL)
New: CALL F01ABF(A,IA,N,B,IB,Z,IFAIL)
```

The number of iterative refinement corrections returned by F01ACF in L is no longer available. The parameter EPS is no longer required.

F01AEF
Withdrawn at Mark 18

```
Old: CALL F01AEF(N,A,IA,B,IB,DL,IFAIL)
New: DO 20 J = 1, N
        DO 10 I = J, N
           A(I,J) = A(J,I)
           B(I,J) = B(J,I)
 10     CONTINUE
        DL(J) = B(J,J)
 20  CONTINUE
     CALL spotrf('L',N,B,IB,INFO)
     IF (INFO.EQ.0) THEN
        CALL ssygst(1,'L',N,A,IA,B,IB,INFO)
     ELSE
        IFAIL = 1
     END IF
     CALL sswap(N,DL,1,B,IB+1)
```

IFAIL is set to 1 if the matrix B is not positive-definite. It is essential to test IFAIL.

F01AFF
Withdrawn at Mark 18

```
Old: CALL F01AFF(N,M1,M2,B,IB,DL,Z,IZ)
New: CALL sswap(N,DL,1,B,IB+1)
     CALL strsm('L','L','T','N',N,M2-M1+1,1.0e0,B,IB,Z(1,M1),IZ)
     CALL sswap(N,DL,1,B,IB+1)
```

F01AGF
Withdrawn at Mark 18

```
Old: CALL F01AGF(N,TOL,A,IA,D,E,E2)
New: CALL ssytrd('L',N,A,IA,D,E(2),TAU,WORK,LWORK,INFO)
     E(1) = 0.0e0
     DO 10 I = 1, N
        E2(I) = E(I)*E(I)
 10  CONTINUE
```

where TAU is a ***real*** array of length at least (N−1), WORK is a ***real*** array of length at least (1) and LWORK is its actual length.

Note that the tridiagonal matrix computed by F08FEF (SSYTRD/DSYTRD) is different from that computed by F01AGF, but it has the same eigenvalues.

F01AHF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F01AGF has been replaced by a call to F08FEF (SSYTRD/DSYTRD) as shown above.

```
Old: CALL F01AHF(N,M1,M2,A,IA,E,Z,IZ)
New: CALL sormtr('L','L','N',N,M2-M1+1,A,IA,TAU,Z(1,M1),IZ,WORK,
    +            LWORK,INFO)
```

where WORK is a ***real*** array of length at least (M2−M1+1), and LWORK is its actual length.

F01AJF
Withdrawn at Mark 18

```
Old: CALL F01AJF(N,TOL,A,IA,D,E,Z,IZ)
New: CALL ssytrd('L',N,A,IA,D,E(2),TAU,WORK,LWORK,INFO)
     E(1) = 0.0e0
     CALL F06QFF('L',N,N,A,IA,Z,IZ)
     CALL dorgtr('L',N,Z,IZ,TAU,WORK,LWORK,INFO)
```

where TAU is a ***real*** array of length at least (N−1), WORK is a ***real*** array of length at least (N−1) and LWORK is its actual length.

Note that the tridiagonal matrix T and the orthogonal matrix Q computed by F08FEF (SSYTRD/DSYTRD) and F08FFF (SORGTR/DORGTR) are different from those computed by F01AJF, but they satisfy the same relation $Q^T AQ = T$.

F01AKF
Withdrawn at Mark 18

```
Old: CALL F01AKF(N,K,L,A,IA,INTGER)
New: CALL sgehrd(N,K,L,A,IA,TAU,WORK,LWORK,INFO)
```

where TAU is a ***real*** array of length at least (N−1), WORK is a ***real*** array of length at least (N) and LWORK is its actual length.

Note that the Hessenberg matrix computed by F08NEF (SGEHRD/DGEHRD) is different from that computed by F01AKF, because F08NEF (SGEHRD/DGEHRD) uses orthogonal transformations, whereas F01AKF uses stabilized elementary transformations.

F01ALF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F01AKF has been replaced by a call to F08NEF (SGEHRD/DGEHRD) as indicated above.

```
Old: CALL F01ALF(K,L,IR,A,IA,INTGER,Z,IZ,N)
New: CALL sormhr('L','N',N,IR,K,L,A,IA,TAU,Z,IZ,WORK,LWORK,INFO)
```

where WORK is a ***real*** array of length at least (IR) and LWORK is its actual length.

F01AMF
Withdrawn at Mark 18

```
Old: CALL F01AMF(N,K,L,AR,IAR,AI,IAI,INTGER)
New: DO 20 J = 1, N
        DO 10 I = 1, N
           A(I,J) = cmplx(AR(I,J),AI(I,J))
 10     CONTINUE
 20  CONTINUE
     CALL cgehrd(N,K,L,A,IA,TAU,WORK,LWORK,INFO)
```

where A is a ***complex*** array of dimension (IA,N), TAU is a ***complex*** array of length at least (N−1), WORK is a ***complex*** array of length at least (N) and LWORK is its actual length.

Note that the Hessenberg matrix computed by F08NSF (CGEHRD/ZGEHRD) is different from that computed by F01AMF, because F08NSF (CGEHRD/ZGEHRD) uses orthogonal transformations, whereas F01AMF uses stabilized elementary transformations.

F01ANF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F01AMF has been replaced by a call to F08NSF (CGEHRD/ZGEHRD) as indicated above.

```
Old: CALL F01ANF(K,L,IR,AR,IAR,AI,IAI,INTGER,ZR,IZR,ZI,IZI,N)
New: CALL cunhmr('L','N',N,IR,K,L,A,IA,TAU,Z,IZ,WORK,LWORK,INFO)
     DO 20 J = 1, IR
        DO 10 I = 1, N
           ZR(I,J) = real(Z(I,J))
           ZI(I,J) = imag(Z(I,J))
 10     CONTINUE
 20  CONTINUE
```

where A is a ***complex*** array of dimension (IA,N), TAU is a ***complex*** array of length at least (N−1), Z is a ***complex*** array of dimension (IZ,IR), WORK is a ***complex*** array of length at least (IR) and LWORK is its actual length.

F01APF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F01AKF has been replaced by a call to F08NEF (SGEHRD/DGEHRD) as indicated above.

```
Old: CALL F01APF(N,K,L,INTGER,H,IH,V,IV)
New: CALL F06QFF('L',N,N,H,IH,V,IV)
     CALL sorghr(N,K,L,V,IV,TAU,WORK,LWORK,INFO)
```

where WORK is a ***real*** array of length at least (N), and LWORK is its actual length.

Note that the orthogonal matrix formed by F08NFF (SORGHR/DORGHR) is not the same as the non-orthogonal matrix formed by F01APF. See F01AKF above.

F01ATF
Withdrawn at Mark 18

```
Old: CALL F01ATF(N,IB,A,IA,K,L,D)
New: CALL sgebal('B',N,A,IA,K,L,D,INFO)
```

Note that the balanced matrix returned by F08NHF (SGEBAL/DGEBAL) may be different from that returned by F01ATF.

F01AUF
Withdrawn at Mark 18

```
Old: CALL F01AUF(N,K,L,M,D,Z,IZ)
New: CALL sgebak('B','R',N,K,L,D,M,Z,IZ,INFO)
```

F01AVF
Withdrawn at Mark 18

```
Old: CALL F01AVF(N,IB,AR,IAR,AI,IAI,K,L,D)
New: DO 20 J = 1, N
        DO 10 I = 1, N
           A(I,J) = cmplx(AR(I,J),AI(I,J))
 10     CONTINUE
 20  CONTINUE
     CALL cgebal('B',N,A,IA,K,L,D,INFO)
     DO 20 J = 1, N
        DO 10 I = 1, N
           AR(I,J) = real(A(I,J))
           AI(I,J) = imag(A(I,J))
 10     CONTINUE
 20  CONTINUE
```

where A is a ***complex*** array of dimension (IA,N).

Note that the balanced matrix returned by F08NVF (CGEBAL/ZGEBAL) may be different from that returned by F01AVF.

F01AWF
Withdrawn at Mark 18

```
Old: CALL F01AWF(N,K,L,M,D,ZR,IZR,ZI,IZI)
New: DO 20 J = 1, M
        DO 10 I = 1, N
           Z(I,J) = cmplx(ZR(I,J),ZI(I,J))
 10     CONTINUE
 20  CONTINUE
     CALL cgebak('B','R',N,K,L,D,M,Z,IZ,INFO)
     DO 40 J = 1, M
        DO 30 I = 1, N
```

```
            ZR(I,J) = real(Z(I,J))
            ZI(I,J) = imag(Z(I,J))
   30     CONTINUE
   40  CONTINUE
```

where Z is a ***complex*** array of dimension (IZ,M).

F01AXF
Withdrawn at Mark 18

```
Old: CALL F01AXF(M,N,QR,IQR,ALPHA,IPIV,Y,E,IFAIL)
New: CALL sgeqpf(M,N,QR,IQR,IPIV,Y,WORK,INFO)
     CALL scopy(N,QR,IQR+1,ALPHA,1)
```

where WORK is a ***real*** array of length at least (3*N).

Note that the details of the Householder matrices returned by F08BEF (SGEQPF/DGEQPF) are different from those returned by F01AXF, but they determine the same orthogonal matrix Q.

F01AYF
Withdrawn at Mark 18

```
Old: CALL F01AYF(N,TOL,A,IA,D,E,E2)
New: CALL ssptrd('U',N,A,D,E(2),TAU,INFO)
     E(1) = 0.0e0
     DO 10 I = 1, N
        E2(I) = E(I)*E(I)
  10 CONTINUE
```

where TAU is a ***real*** array of length at least (N−1).

F01AZF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F01AYF has been replaced by a call to F08GEF (SSPTRD/DSPTRD) as shown above.

```
Old: CALL F01AZF(N,M1,M2,A,IA,Z,IZ)
New: CALL sopmtr('L','U','N',N,M2-M1+1,A,TAU,Z(1,M1),IZ,WORK,INFO)
```

where WORK is a ***real*** array of length at least (M2−M1+1).

F01BCF
Withdrawn at Mark 18

```
Old: CALL F01BCF(N,TOL,AR,IAR,AI,IAI,D,E,WK1,WK2)
New: DO 20 J = 1, N
        DO 10 I = 1, N
           A(I,J) = cmplx(AR(I,J),AI(I,J))
 10     CONTINUE
 20  CONTINUE
     CALL chetrd('L',N,A,IA,D,E(2),TAU,WORK,LWORK,INFO)
     E(1) = 0.0e0
     CALL cungtr('L',N,A,IA,TAU,WORK,LWORK,INFO)
     DO 40 J = 1, N
        DO 30 I = 1, N
           AR(I,J) = real(A(I,J))
           AI(I,J) = imag(A(I,J))
 30     CONTINUE
 40  CONTINUE
```

where A is a ***complex*** array of dimension (IA,N), TAU is a ***complex*** array of length at least (N−1), WORK is a ***complex*** array of length at least (N−1), and LWORK is its actual length.

Note that the tridiagonal matrix T and the unitary matrix Q computed by F08FSF (CHETRD/ZHETRD) and F08FTF (CUNGTR/ZUNGTR) are different from those computed by F01BCF, but they satisfy the same relation $Q^H AQ = T$.

F01BDF
Withdrawn at Mark 18

```
Old: CALL F01BDF(N,A,IA,B,IB,DL,IFAIL)
New: DO 20 J = 1, N
        DO 10 I = J, N
           A(I,J) = A(J,I)
           B(I,J) = B(J,I)
 10     CONTINUE
        DL(J) = B(J,J)
 20  CONTINUE
     CALL spotrf('L',N,B,IB,INFO)
     IF (INFO.EQ.0) THEN
        CALL ssygst(2,'L',N,A,IA,B,IB,INFO)
     ELSE
        IFAIL = 1
     END IF
     CALL sswap(N,DL,1,B,IB+1)
```

IFAIL is set to 1 if the matrix B is not positive-definite. It is essential to test IFAIL.

F01BEF
Withdrawn at Mark 18

```
Old: CALL F01BEF(N,M1,M2,B,IB,DL,V,IV)
New: CALL sswap(N,DL,1,B,IB+1)
     CALL strmm('L','L','N','N',N,M2-M1+1,1.0e0,B,IB,V(1,M1),IV)
     CALL sswap(N,DL,1,B,IB+1)
```

F01BNF
Withdrawn at Mark 17

```
Old: CALL F01BNF(N,A,IA,P,IFAIL)
New: CALL cpotrf('Upper',N,A,IA,IFAIL)
```

where, before the call, array A contains the upper triangle of the matrix to be factorized rather than the lower triangle (note that the elements of the upper triangle are the complex conjugates of the elements of the lower triangle). The ***real*** array P is no longer required; the upper triangle of A is overwritten by the upper triangular factor U, including the diagonal elements (which are not reciprocated).

F01BPF
Withdrawn at Mark 17

```
Old: CALL F01BPF(N,A,IA,V,IFAIL)
New: CALL cpotrf('Upper',N,A,IA,IFAIL)
     CALL cpotri('Upper',N,A,IA,IFAIL)
```

where, before the calls, the upper triangle of the matrix to be inverted must be contained in rows 1 to N of A, rather than the lower triangle being in rows 2 to $N+1$ (note that the elements of the upper triangle are the complex conjugates of the elements of the lower triangle). The workspace vector V is no longer required.

F01BQF
Withdrawn at Mark 16

The replacement routines do not have exactly the same functionality as F01BQF; if this functionality is genuinely required, please contact NAG.

(a) where the symmetric matrix is known to be positive-definite (if the matrix is in fact not positive-definite, the replacement routine will return a positive value in IFAIL)

```
Old: CALL F01BQF(N,EPS,RL,IRL,D,IFAIL)
New: CALL spptrf('Lower',N,RL,IFAIL)
```

(b) where the matrix is not positive-definite (the replacement routine forms an LDL^T factorization where D is block diagonal, rather than a Cholesky factorization)

```
Old: CALL F01BQF(N,EPS,RL,IRL,D,IFAIL)
New: CALL ssptrf('Lower',N,RL,IPIV,IFAIL)
```

For the replacement calls in both (a) and (b), the array RL must now hold the complete lower triangle of the symmetric matrix, including the diagonal elements, which are no longer required to be stored in the redundant array D. The declared dimension of RL must be increased from at least N(N − 1)/2 to at least N(N + 1)/2. It is important to note that for the calls of F07GDF (SPPTRF/DPPTRF) and F07PDF (SSPTRF/DSPTRF), the lower triangle of the matrix must be stored packed by column instead of by row. The dimension parameter IRL is no longer required. For the call of F07PDF (SSPTRF/DSPTRF), the INTEGER array IPIV of length N must be supplied.

F01BTF
Withdrawn at Mark 18

```
Old: CALL F01BTF(N,A,IA,P,DP,IFAIL)
New: CALL sgetrf(N,N,A,IA,IPIV,IFAIL)
```

where IPIV is an INTEGER array of length N which holds the indices of the pivot elements, and the array P is no longer required. It may be important to note that after a call of F07ADF (SGETRF/DGETRF), A is overwritten by the upper triangular factor U and the off-diagonal elements of the unit lower triangular factor L, whereas the factorization returned by F01BTF gives U the unit diagonal. The permutation determinant DP returned by F01BTF is not computed by F07ADF (SGETRF/DGETRF). If this value is required, it may be calculated after a call of F07ADF (SGETRF/DGETRF) by code similar to the following:

```
      DP = 1.0e0
      DO 10 I = 1, N
         IF (I.NE.IPIV(I)) DP = -DP
   10 CONTINUE
```

F01BWF
Withdrawn at Mark 18

```
Old: CALL F01BWF(N,M1,A,IA,D,E)
New: CALL ssbtrd('N','U',N,M1-1,A,IA,D,E(2),Q,1,WORK,INFO)
     E(1) = 0.0e0
```

where Q is a dummy ***real*** array of length (1) (not used in this call), and WORK is a ***real*** array of length at least (N).

Note that the tridiagonal matrix computed by F08HEF (SSBTRD/DSBTRD) is different from that computed by F01BWF, but it has the same eigenvalues.

F01BXF
Withdrawn at Mark 17

```
Old: CALL F01BXF(N,A,IA,P,IFAIL)
New: CALL spotrf('Upper',N,A,IA,IFAIL)
```

where, before the call, array A contains the upper triangle of the matrix to be factorized rather than the lower triangle. The array P is no longer required; the upper triangle of A is overwritten by the upper triangular factor U, including the diagonal elements (which are not reciprocated).

F01CAF
Withdrawn at Mark 14

```
Old: CALL F01CAF(A,M,N,IFAIL)
New: CALL F06QHF('General',M,N,0.0e0,0.0e0,A,M)
```

F01CBF
Withdrawn at Mark 14

```
Old: CALL F01CBF(A,M,N,IFAIL)
New: CALL F06QHF('General',M,N,0.0e0,1.0e0,A,M)
```

F01CDF
Withdrawn at Mark 15

```
Old: CALL F01CDF(A,B,C,M,N,IFAIL)
New: CALL F01CTF('N','N',M,N,1.0e0,B,M,1.0e0,C,M,A,M,IFAIL)
```

F01CEF
Withdrawn at Mark 15

```
Old: CALL F01CEF(A,B,C,M,N,IFAIL)
New: CALL F01CTF('N','N',M,N,1.0e0,B,M,-1.0e0,C,M,A,M,IFAIL)
```

F01CFF
Withdrawn at Mark 14

```
Old: CALL F01CFF(A,MA,NA,P,Q,B,MB,NB,M1,M2,N1,N2,IFAIL)
New: CALL F06QFF('General',M2-M1+1,N2-N1+1,B(M1,N1),MB,A(P,Q),MA)
```

F01CGF
Withdrawn at Mark 15

```
Old: CALL F01CGF(A,MA,NA,P,Q,B,MB,NB,M1,M2,N1,N2,IFAIL)
New: CALL F01CTF('N','N',M2-M1+1,N2-N1+1,1.0e0,A(P,Q),MA,1.0e0,
    +            B(M1,N1),MB,A(P,Q),MA,IFAIL)
```

F01CHF
Withdrawn at Mark 15

```
Old: CALL F01CHF(A,MA,NA,P,Q,B,MB,NB,M1,M2,N1,N2,IFAIL)
New: CALL F01CTF('N','N',M2-M1+1,N2-N1+1,1.0e0,A(P,Q),MA,-1.0e0,
    +            B(M1,N1),MB,A(P,Q),MA,IFAIL)
```

F01CLF
Withdrawn at Mark 16

```
Old: CALL F01CLF(A,B,C,N,P,M,IFAIL)
New: CALL sgemm('N','T',N,P,M,1.0e0,B,N,C,P,0.0e0,A,N)
```

F01CMF
Withdrawn at Mark 14

```
Old: CALL F01CMF(A,LA,B,LB,M,N)
New: CALL F06QFF('General',M,N,A,LA,B,LB)
```

F01CNF
Withdrawn at Mark 13

```
Old: CALL F01CNF(V,M,A,LA,I)
New: CALL scopy(M,V,1,A(I,1),LA)
```

F01CPF
Withdrawn at Mark 13

```
Old: CALL F01CPF(A,B,N)
New: CALL scopy(N,A,1,B,1)
```

F01CQF
Withdrawn at Mark 13

```
Old: CALL F01CQF(A,N)
New: CALL F06FBF(N,0.0e0,A,1)
```

F01CSF
Withdrawn at Mark 13

```
Old: CALL F01CSF(A,LA,B,N,C)
New: CALL sspmv('U',N,1.0e0,A,B,1,0.0e0,C,1)
```

F01DAF
Withdrawn at Mark 13

```
Old: F01DAF(L,M,C1,IRA,ICB,A,IA,B,IB,N)
New: C1 + sdot(M-L+1,A(IRA,L)IA,B(L,ICB),1)
```

F01DBF
Withdrawn at Mark 13

```
Old: D = F01DBF(L,M,C1,IRA,ICB,A,IA,B,IB,N)
New: CALL X03AAF(A(IRA,L),(M-L)*IA+1,B(L,ICB),M-L+1,IA,1,C1,0.0e0,D,
    +            D2,.TRUE.,IFAIL)
```

(here D2 is a new ***real*** variable whose value is not used).

F01DCF
Withdrawn at Mark 13

```
Old: CALL F01DCF(L,M,CX,IRA,ICB,A,IA,B,IB,N,CR,CI)
New: DX = CX - cdotu(M-L+1,A(IRA,L),IA,B(L,ICB),1)
     CR = real(DX)
     CI = imag(DX)
```

(here DX is a new ***complex*** variable).

F01DDF
Withdrawn at Mark 13

```
Old: CALL F01DDF(L,M,CX,IRA,ICB,A,IA,B,IB,N,CR,CI)
New: CALL X03ABF(A(IRA,L),(M-L)*IA+1,B(L,ICB),M-L+1,IA,1,-CX,DX,
    +            .TRUE.,IFAIL)
     CR = -real(DX)
     CI = -imag(DX)
```

(here DX is a new ***complex*** variable).

F01DEF
Withdrawn at Mark 14

```
Old: F01DEF(A,B,N)
New: sdot(N,A,1,B,1)
```

F01LBF
Withdrawn at Mark 18

```
Old: CALL F01LBF(N,M1,M2,A,IA,AL,IL,IN,IV,IFAIL)
New: CALL sgbtrf(N,N,M1,M2,A,IA,IN,IFAIL)
```

where the size of array A must now have a leading dimension IA of at least $2 \times$ M1+M2 + 1. The array AL, its associated dimension parameter IL, and the parameter IV are not required for F07BDF (SGBTRF/DGBTRF) because this routine overwrites A by both the L and U factors. The scheme by which the matrix is packed into the array is completely different from that used by F01LBF; the relevant routine document should be consulted for details..

F01LZF
Withdrawn at Mark 15

```
Old: CALL F01LZF(N,A,NRA,C,NRC,WANTB,B,WANTQ,WANTY,Y,NRY,LY,WANTZ,Z,
    +            NRZ,NCZ,D,E,WORK1,WORK2,IFAIL)
New: CALL sgebrd(N,N,A,NRA,D,E(2),TAUQ,TAUP,WORK1,LWORK,INFO)
     IF (WANTB) THEN
        CALL sormbr('Q','L','T',N,1,NA,NRA,TAUQ,B,N,WORK1,LWORK,INFO)
     ELSE IF (WANTQ) THEN
        CALL sorgbr('Q',N,N,N,A,NRA,TAUQ,WORK,LWORK,INFO)
     ELSE IF (WANTY) THEN
        CALL sormbr('Q','R','N',LY,N,N,A,NRA,TAUQ,Y,NRY,WORK1,LWORK,
    +               INFO)
     ELSE IF (WANTZ) THEN
        CALL sormbr('P','L','T',N,NCZ,N,A,NRA,TAUP,Z,NRZ,WORK1,LWORK,
    +               INFO)
     END IF
```

where TAUQ and TAUP are real arrays of length at least (N) and LWORK is the actual length of WORK1. The parameter WORK2 is no longer required.

F01MAF
Superseded at Mark 17
Scheduled for withdrawal at Mark 19

Existing programs should be modified to call F11JAF. The interfaces are significantly different and therefore precise details of a replacement call cannot be given. Please consult the appropriate routine document.

F01NAF
Withdrawn at Mark 17

```
Old: CALL F01NAF(N,ML,MU,A,NRA,TOL,IN,SCALE,IFAIL)
New: CALL cgbtrf(N,N,ML,MU,A,NRA,IN,IFAIL)
```

where the parameter TOL and array SCALE are no longer required. The input matrix must be stored using the same scheme as for F01NAF, except in rows ML + 1 to $2 \times$ ML + MU + 1 of A instead of rows 1 to ML + MU + 1. In F07BRF(CGBTRF/ZGBTRF), the value returned in IN(N) has no significance as an indicator of near-singularity of the matrix.

F01QAF
Withdrawn at Mark 15

```
Old: CALL F01QAF(M,N,A,NRA,C,NRC,Z,IFAIL)
New: CALL sgeqrf(M,N,A,NRA,Z,WORK,LWORK,INFO)
```

where WORK is a real array of length at least (LWORK). The parameters C and NRC are no longer required.

Note that the representation of the matrix Q is not identical, but subsequent calls to routines F08AFF (SORGQR/DORGQR) and F08AGF (SORMQR/DORMQR) may be used to obtain Q explicitly and to transform by Q or Q^T respectively.

F01QBF
Withdrawn at Mark 15

```
Old: CALL F01QBF(M,N,A,NRA,C,NRC,WORK,IFAIL)
New: CALL F06QFF('General',M,N,A,NRA,C,NRC)
     CALL F01QJF(M,N,C,NRC,WORK,IFAIL)
```

The call to F06QFF simply copies the M by N part of A to C. This may be omitted if it is desired to use the same arrays for A and C. Note that the representation of the orthogonal matrix Q is not identical, but following F01QJF routine F01QKF may be used to form Q.

F01QCF
Withdrawn at Mark 18

```
Old: CALL F01QCF(M,N,A,LDA,ZETA,IFAIL)
New: CALL sgeqrf(M,N,A,LDA,ZETA,WORK,LWORK,INFO)
```

where WORK is a ***real*** array of length at least (N), and LWORK is its actual length.

The subdiagonal elements of A and the elements of ZETA returned by F08AEF (SGEQRF/DGEQRF) are not the same as those returned by F01QCF. Subsequent calls to F01QDF or F01QEF must also be replaced by calls to F08AGF (SORMQR/DORMQR) or F08AFF (SORGQR/DORGQR) as shown below.

F01QDF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F01QCF has been replaced by a call to F08AEF (SGEQRF/DGEQRF) as shown above. It also assumes that the 2nd argument of F01QDF (WHERET) is 'S', which is appropriate if the contents of A and ZETA have not been changed after the call of F01QCF.

```
Old: CALL F01QDF(TRANS,'S',M,N,A,LDA,ZETA,NCOLB,B,LDB,WORK,IFAIL)
New: CALL sormqr('L',TRANS,M,NCOLB,N,A,LDA,ZETA,B,LDB,WORK,LWORK,INFO)
```

where LWORK is the actual length of WORK.

F01QEF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F01QCF has been replaced by a call to F08AEF (SGEQRF/DGEQRF) as shown above. It also assumes that the 1st argument of F01QEF (WHERET) is 'S', which is appropriate if the contents of A and ZETA have not been changed after the call of F01QCF.

```
Old: CALL F01QEF('S',M,N,NCOLQ,A,LDA,ZETA,WORK,IFAIL)
New: CALL sorgqr(M,NCOLQ,N,A,LDA,ZETA,WORK,LWORK,INFO)
```

where LWORK is the actual length of WORK.

F01QFF
Withdrawn at Mark 18

The following replacement assumes that the 1st argument of F01QFF (PIVOT) is 'C'. There is no direct replacement if PIVOT = 'S'.

```
Old: CALL F01QFF('C',M,N,A,LDA,ZETA,PERM,WORK,IFAIL)
New: DO 10 I = 1, N
        PERM(I) = 0
  10 CONTINUE
     CALL sgeqpf(M,N,A,LDA,PERM,ZETA,WORK,INFO)
```

where WORK is a ***real*** array of length at least (3*N) (F01QFF only requires WORK to be of length (2*N)).

The subdiagonal elements of A and the elements of ZETA returned by F08BEF (SGEQPF/DGEQPF) are not the same as those returned by F01QFF. Subsequent calls to F01QDF or F01QEF must also be replaced by calls to F08AGF (SORMQR/DORMQR) or F08AFF (SORGQR/DORGQR) as shown above. Note also that the array PERM returned by F08BEF (SGEQPF/DGEQPF) holds details of the interchanges in a different form than that returned by F01QFF.

F01RCF
Withdrawn at Mark 18

```
Old: CALL F01RCF(M,N,A,LDA,THETA,IFAIL)
New: CALL cgeqrf(M,N,A,LDA,THETA,WORK,LWORK,INFO)
```

where WORK is a ***complex*** array of length at least (N), and LWORK is its actual length.

The subdiagonal elements of A and the elements of THETA returned by F08ASF (CGEQRF/ZGEQRF) are not the same as those returned by F01RCF. Subsequent calls to F01RDF or F01REF must also be replaced by calls to F08AUF (CUNMQR/ZUNMQR) or F08ATF (CUNGQR/ZUNGQR) as shown below.

F01RDF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F01RCF has been replaced by a call to F08ASF (CGEQRF/ZGEQRF) as shown above. It also assumes that the 2nd argument of F01RDF (WHERET) is 'S', which is appropriate if the contents of A and THETA have not been changed after the call of F01RCF.

```
Old: CALL F01RDF(TRANS,'S',M,N,A,LDA,THETA,NCOLB,B,LDB,WORK,IFAIL)
New: CALL cunmqr('L',TRANS,M,NCOLB,N,A,LDA,THETA,B,LDB,WORK,LWORK,
    +            INFO)
```

where LWORK is the actual length of WORK.

F01REF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F01RCF has been replaced by a call to F08ASF (CGEQRF/ZGEQRF) as shown above. It also assumes that the 1st argument of F01REF (WHERET) is 'S', which is appropriate if the contents of A and THETA have not been changed after the call of F01RCF.

```
Old: CALL F01REF('S',M,N,NCOLQ,A,LDA,THETA,WORK,IFAIL)
New: CALL cungqr(M,NCOLQ,N,A,LDA,THETA,WORK,LWORK,INFO)
```

where LWORK is the actual length of WORK.

F01RFF
Withdrawn at Mark 18

The following replacement assumes that the 1st argument of F01RFF (PIVOT) is 'C'. There is no direct replacement if PIVOT = 'S'.

```
Old: CALL F01RFF('C',M,N,A,LDA,THETA,PERM,WORK,IFAIL)
New: DO 10 I = 1, N
        PERM(I) = 0
  10 CONTINUE
     CALL cgeqpf(M,N,A,LDA,PERM,THETA,CWORK,WORK,INFO)
```

where CWORK is a ***complex*** array of length at least (N).

The subdiagonal elements of A and the elements of THETA returned by F08BSF (CGEPQF/ZGEPQF) are not the same as those returned by F01RFF. Subsequent calls to F01RDF or F01REF must also be replaced by calls to F08AUF (CUNMQR/ZUNMQR) or F08ATF (CUNGQR/ZUNGQR) as shown above. Note also that the array PERM returned by F08BSF (CGEPQF/ZGEPQF) holds details of the interchanges in a different form than that returned by F01RFF.

F02 – Eigenvalues and Eigenvectors

Notes:

1. Replacement routines require complex matrices to be stored in ***complex*** arrays, whereas most of the corresponding old routines require the real and imaginary parts to be stored separately in two ***real*** arrays.
2. Replacement routines for computing eigenvectors may scale the eigenvectors in a different manner from the old routines, and hence at first glance the eigenvectors may appear to disagree completely; they may indeed be different, but they are equally acceptable as eigenvectors; some replacement routines may also return the eigenvalues (and the corresponding eigenvectors) in a different order.

3. Replacement routines in Chapter F07 and Chapter F08 have a parameter INFO, which has a different specification to the usual NAG error-handling parameter IFAIL. See the F07 or F08 Chapter Introduction for details.

F02AAF
Withdrawn at Mark 18

```
Old: CALL F02AAF(A,IA,N,R,E,IFAIL)
New: CALL F02FAF('N','L',N,A,IA,R,WORK,LWORK,IFAIL)
```

where WORK is a ***real*** array of length at least (3*N) and LWORK is its actual length.

F02ABF
Withdrawn at Mark 18

```
Old: CALL F02ABF(A,IA,B,IB,N,R,V,IV,E,IFAIL)
New: CALL F06QFF('L',N,N,A,IA,V,IV)
     CALL F02FAF('V','L',N,V,IV,R,WORK,LWORK,IFAIL)
```

where WORK is a ***real*** array of length at least (3*N) and LWORK is its actual length. If F02ABF was called with the same array supplied for V and A, then the call to F06QFF (which copies A to V) may be omitted.

F02ADF
Withdrawn at Mark 18

```
Old: CALL F02ADF(A,IA,B,IB,N,R,DE,IFAIL)
New: CALL F02FDF(1,'N','U',N,A,IA,B,IB,R,WORK,LWORK,IFAIL)
```

where WORK is a ***real*** array of length at least (3*N) and LWORK is its actual length.

Note that the call to F02FDF will overwrite the upper triangles of the arrays A and B and leave the subdiagonal elements unchanged, whereas the call to F02ADF overwrites the lower triangle and leaves the elements above the diagonal unchanged.

F02AEF
Withdrawn at Mark 18

```
Old: CALL F02AEF(A,IA,N,R,V,IV,DL,E,IFAIL)
New: CALL F06QFF('U',N,N,A,IA,V,IV)
     CALL F02FDF(1,'V','U',N,V,IV,B,IB,R,WORK,LWORK,IFAIL)
```

where WORK is a ***real*** array of length at least (3*N) and LWORK is its actual length.

Note that the call to F02FDF will overwrite the upper triangle of the array B and leave the subdiagonal elements unchanged, whereas the call to F02ADF overwrites the lower triangle and leaves the elements above the diagonal unchanged. The call to F06QFF copies A to V, so A is left unchanged. If F02AEF was called with the same array supplied for V and A, then the call to F06QFF may be omitted.

F02AFF
Withdrawn at Mark 18

```
Old: CALL F02AFF(A,IA,N,RR,RI,INTGER,IFAIL)
New: CALL F02EBF('N',N,A,IA,RR,RI,VR,1,VI,1,WORK,LWORK,IFAIL)
```

where VR and VI are dummy arrays of length (1) (not used in this call), WORK is a ***real*** array of length at least (4*N) and LWORK is its actual length; the iteration counts (returned by F02AFF in the array INTGER) are not available from F02EBF.

F02AGF
Withdrawn at Mark 18

```
Old: CALL F02AGF(A,IA,N,RR,RI,VR,IVR,VI,IVI,INTGER,IFAIL)
New: CALL F02EBF('V',N,A,IA,RR,RI,VR,IVR,VI,IVI,WORK,LWORK,IFAIL)
```

where WORK is a ***real*** array of length at least (4*N) and LWORK is its actual length; the iteration counts (returned by F02AGF in the array INTGER) are not available from F02EBF.

F02AJF
Withdrawn at Mark 18

```
Old: CALL F02AJF(AR,IAR,AI,IAI,N,RR,RI,INTGER,IFAIL)
New: DO 20 J = 1, N
        DO 10 I = 1, N
           A(I,J) = cmplx(AR(I,J),AI(I,J))
 10     CONTINUE
 20  CONTINUE
     CALL F02GBF('N',N,A,IA,R,V,1,RWORK,WORK,LWORK,IFAIL)
     DO 30 I = 1, N
        RR(I) = real(R(I))
        RI(I) = imag(R(I))
 30  CONTINUE
```

where A is a ***complex*** array of dimension (IA,N), R is a ***complex*** array of dimension (N), V is a dummy ***complex*** array of length (1) (not used in this call), RWORK is a ***real*** array of length at least (2*N), WORK is a ***complex*** array of length at least (2*N) and LWORK is its actual length.

F02AKF
Withdrawn at Mark 18

```
Old: CALL F02AKF(AR,IAR,AI,IAI,N,RR,RI,VR,IVR,VI,IVI,INTGER,IFAIL)
New: DO 20 J = 1, N
        DO 10 I = 1, N
           A(I,J) = cmplx(AR(I,J),AI(I,J))
  10    CONTINUE
  20 CONTINUE
     CALL F02GBF('V',N,A,IA,R,V,IV,RWORK,WORK,LWORK,IFAIL)
     DO 40 J = 1, N
        RR(J) = real(R(J))
        RI(J) = imag(R(J))
        DO 30 I = 1, N
           VR(I,J) = real(V(I,J))
           VI(I,J) = imag(V(I,J))
  30    CONTINUE
  40 CONTINUE
```

where A is a ***complex*** array of dimension (IA,N), R is a ***complex*** array of length (N), V is a ***complex*** array of dimension (IV,N), RWORK is a ***real*** array of length at least (2*N), WORK is a ***complex*** array of length at least (2*N) and LWORK is its actual length.

F02AMF
Withdrawn at Mark 18

```
Old: CALL F02AMF(N,EPS,D,E,V,IV,IFAIL)
New: CALL ssteqr('V',N,D,E(2),V,IV,WORK,INFO)
```

where WORK is a ***real*** array of length at least (2*(N−1)).

F02ANF
Withdrawn at Mark 18

```
Old: CALL F02ANF(N,EPS,HR,IHR,HI,IHI,RR,RI,IFAIL)
New: DO 20 J = 1, N
        DO 10 I = 1, N
           H(I,J) = cmplx(HR(I,J),HI(I,J))
  10    CONTINUE
  20 CONTINUE
     CALL chseqr('E','N',N,1,N,H,IH,R,Z,1,WORK,1,INFO)
     DO 30 I = 1, N
        RR(I) = real(R(I))
        RI(I) = imag(R(I))
  30 CONTINUE
```

where H is a ***complex*** array of dimension (IH,N), R is a ***complex*** array of length (N), Z is a dummy ***complex*** array of length (1) (not used in this call), and WORK is a ***complex*** array of length at least (N).

F02APF
Withdrawn at Mark 18

```
Old: CALL F02APF(N,EPS,H,IH,RR,RI,ICNT,IFAIL)
New: CALL shseqr('E','N',N,1,N,H,IH,RR,RI,Z,1,WORK,1,INFO)
```

where Z is a dummy ***real*** array of length (1) (not used in this call), and WORK is a ***real*** array of length at least (N); the iteration counts (returned by F02APF in the array ICNT) are not available from F08PEF (SHSEQR/DHSEQR).

F02AQF
Withdrawn at Mark 18

```
Old: CALL F02AQF(N,K,L,EPS,H,IH,V,IV,RR,RI,INTGER,IFAIL)
New: CALL shseqr('S','V',N,K,L,H,IH,RR,RI,V,IV,WORK,1,INFO)
     CALL strevc('R','O',SELECT,N,H,IH,V,IV,V,IV,N,M,WORK,INFO)
```

where SELECT is a dummy logical array of length (1) (not used in this call), and WORK is a ***real*** array of length at least (N); the iteration counts (returned by F02AQF in the array INTGER) are not available from F08PEF (SHSEQR/DHSEQR); M is an integer which is set to N by F08QKF (STREVC/DTREVC).

F02ARF
Withdrawn at Mark 18

```
Old: CALL F02ARF(N,K,L,EPS,INTGER,HR,IHR,HI,IHI,RR,RI,VR,IVR,VI,
    +            IVI, IFAIL)
New: DO 20 J = 1, N
        DO 10 I = 1, N
           H(I,J) = cmplx(HR(I,J),HI(I,J))
 10     CONTINUE
 20  CONTINUE
     CALL chseqr('S','V',N,K,L,H,IH,R,V,IV,WORK,1,INFO)
     CALL ctrevc('R','O',SELECT,N,H,IH,V,IV,V,IV,N,M,WORK,INFO)
     DO 40 J = 1, N
        RR(J) = real(R(J))
        RI(J) = imag(R(J))
        DO 30 I = 1, N
           VR(I,J) = real(V(I,J))
           VI(I,J) = imag(V(I,J))
 30     CONTINUE
 40  CONTINUE
```

where H is a ***complex*** array of dimension (IH,N), R is a ***complex*** array of length (N), V is a ***complex*** array of dimension (IV,N), WORK is a ***complex*** array of length at least (2*N) and RWORK is a ***real*** array of length at least (N); M is an integer which is set to N by F08QXF (CTREVC/ZTREVC).

If F02ARF was preceded by a call to F01AMF to reduce a full complex matrix to Hessenberg form, then the call to F01AMF must also be replaced by calls to F08NSF (CGEHRD/ZGEHRD) and F08NTF (CUNGHR/ZUNGHR).

F02AVF
Withdrawn at Mark 18

```
Old: CALL F02AVF(N,EPS,D,E,IFAIL)
New: CALL ssterf(N,D,E(2),INFO)
```

F02AWF
Withdrawn at Mark 18

```
Old: CALL F02AWF(AR,IAR,AI,IAI,N,R,WK1,WK2,WK3,IFAIL)
```

```
New: DO 20 J = 1, N
        DO 10 I = 1, N
           A(I,J) = cmplx(AR(I,J),AI(I,J))
  10    CONTINUE
  20 CONTINUE
     CALL F02HAF('N','L',N,A,IA,R,RWORK,WORK,LWORK,IFAIL)
```

where A is a ***complex*** array of dimension (IA,N), RWORK is a ***real*** array of length at least (3*N), WORK is a ***complex*** array of length at least (2*N) and LWORK is its actual length.

F02AXF
Withdrawn at Mark 18

```
Old: CALL F02AXF(AR,IAR,AI,IAI,N,R,VR,IVR,VI,IVI,WK1,WK2,WK3,IFAIL)
New: DO 20 J = 1, N
        DO 10 I = 1, N
           A(I,J) = cmplx(AR(I,J),AI(I,J))
  10    CONTINUE
  20 CONTINUE
     CALL F06TFF('L',N,N,A,IA,V,IV)
     CALL F02HAF('V','L',N,V,IV,R,RWORK,WORK,LWORK,IFAIL)
     DO 40 J = 1, N
        DO 30 I = 1, N
           VR(I,J) = real(V(I,J))
           VI(I,J) = imag(V(I,J))
  30    CONTINUE
  40 CONTINUE
```

where A is a ***complex*** array of dimension (IA,N), V is a ***complex*** array of dimension (IV,N), RWORK is a ***real*** array of length at least (3*N), WORK is a ***complex*** array of length at least (2*N) and LWORK is its actual length. If F02AXF was called with the same arrays supplied for VR and AR and for VI and AI, then the call to F06TFF (which copies A to V) may be omitted.

F02AYF
Withdrawn at Mark 18

```
Old: CALL F02AYF(N,EPS,D,E,VR,IVR,VI,IVI,IFAIL)
New: CALL csteqr('V',N,D,E(2),V,IV,WORK,INFO)
     DO 40 J = 1, N
        DO 30 I = 1, N
           VR(I,J) = real(V(I,J))
           VI(I,J) = imag(V(I,J))
  30    CONTINUE
  40 CONTINUE
```

where V is a ***complex*** array of dimension (IV,N), and WORK is a ***real*** array of length at least (2*(N−1)).

F02BBF
Superseded at Mark 17
Scheduled for withdrawal at Mark 19

```
Old: CALL F02BBF(A,IA,N,ALB,UB,M,MM,R,V,IV,D,E,E2,X,G,C,
    +            ICOUNT,IFAIL)
New: CALL F02FCF('Vectors','Value','Lower',N,A,IA,ALB,UB,0,0,
    +            M,MM,R,V,IV,WORK,LWORK,IWORK,IFAIL)
```

where R must have dimension (N), WORK is a ***real*** array of length at least (8*N), LWORK is its actual length, and IWORK is an integer array of length at least (5*N). Note that in the call to F02BBF R needs only to be of dimension (M).

F02BCF
Superseded at Mark 17
Scheduled for withdrawal at Mark 19

```
Old: CALL F02BCF(A,IA,N,ALB,UB,M,MM,RR,RI,VR,IVR,VI,IVI,
    +            INTGER,ICNT,C,B,IB,U,V,IFAIL)
New: CALL F02ECF('Moduli',N,A,IA,ALB,UB,M,MM,RR,RI,VR,IVR,
    +            VI,IVI,WORK,LWORK,ICNT,C,IFAIL)
```

where WORK is a ***real*** array of length at least (N*(N+4)) and LWORK is its actual length.

F02BDF
Superseded at Mark 17
Scheduled for withdrawal at Mark 19

```
Old: CALL F02BDF(AR,IAR,AI,IAI,N,ALB,UB,M,MM,RR,RI,VR,IVR,
    +            VI,IVI,INTGER,C,BR,IBR,BI,IBI,U,V,IFAIL)
New: DO 20 J = 1, N
        DO 10 I = 1, N
           A(I,J) = cmplx(AR(I,J),AI(I,J))
  10    CONTINUE
  20 CONTINUE
     CALL F02GCF('Moduli', N, A, IA, ALB, UB, M, MM, R, V, IV, WORK,
    +            LWORK, RWORK, INTGER, C, IFAIL)
     DO 30 I = 1, N
        RR(I) = real(R(I))
        RI(I) = imag(R(I))
  30 CONTINUE
     DO 50 J = 1, MM
        DO 40 I = 1, N
           VR(I,J) = real(V(I,J))
           VI(I,J) = imag(V(I,J))
  40    CONTINUE
  50 CONTINUE
```

where A is a ***complex*** array of dimension (IA,N), R is a ***complex*** array of dimension (N), V is a ***complex*** array of dimension (IV,M), WORK is a ***complex*** array of length at least (N*(N+2)), LWORK is its actual length, and RWORK is a ***real*** array of length at least (2*N).

F02BEF
Withdrawn at Mark 18

```
Old: CALL F02BEF(N,D,ALB,UB,EPS,EPS1,E,E2,M,MM,R,V,IV,ICOUNT,X,C,
    +            IFAIL)
New: CALL sstebz('V','B',N,ALB,UB,0,0,EPS1,D,E(2),MM,NSPLIT,R,IBLOCK,
    +            ISPLIT,X,IWORK,INFO)
     CALL sstein(N,D,E(2),MM,R,IBLOCK,ISPLIT,V,IV,X,IWORK,IFAILV,INFO)
```

where NSPLIT is an integer variable, IBLOCK, ISPLIT and IFAILV are integer arrays of length at least (N), and IWORK is an integer array of length at least (3*N).

F02BFF
Withdrawn at Mark 18

```
Old: CALL F02BFF(D,E,E2,N,M1,M2,MM12,EPS1,EPS,EPS2,IZ,R,WU)
New: CALL sstebz('I','E',N,0.0e0,0.0e0,M1,M2,EPS1,D,E(2),M,
    +            NSPLIT,R,IBLOCK,ISPLIT,WORK,IWORK,INFO)
```

where M and NSPLIT are integer variables, IBLOCK and ISPLIT are integer arrays of length at least (N), WORK is a ***real*** array of length at least (4*N), and IWORK is an integer array of length at least (3*N).

F02BKF
Withdrawn at Mark 18

```
Old: CALL F02BKF(N,M,H,IH,RI,C,RR,V,IV,B,IB,U,W,IFAIL)
New: CALL shsein('R','Q','N',C,N,H,IH,RR,RI,V,IV,V,IV,M,M2,B,IFAILR,
    +            IFAILR,INFO)
```

where M2 is an integer variable, and IFAILR is an integer array of length at least (N).

Note that the array C may be modified by F08PKF (SHSEIN/DHSEIN) if there are complex conjugate pairs of eigenvalues.

F02BLF
Withdrawn at Mark 18

```
Old: CALL F02BLF(N,M,HR,IHR,HI,IHI,RI,C,RR,VR,IVR,VI,IVI,BR,IBR,BI,
    +            IBI,U,W,IFAIL)
New: DO 20 J = 1, N
        R(J) = cmplx(RR(J),RI(J))
        DO 10 I = 1, N
           H(I,J) = cmplx(HR(I,J),HI(I,J))
  10    CONTINUE
  20 CONTINUE
     CALL chsein('R','Q','N',C,N,H,IH,R,V,IV,V,IV,M,M2,WORK,RWORK,
    +            IFAILR,IFAILR,INFO)
     DO 30 I = 1, N
        RR(I) = real(R(I))
  30 CONTINUE
     DO 50 J = 1, M
        DO 40 I = 1, N
           VR(I,J) = real(V(I,J))
           VI(I,J) = imag(V(I,J))
  40    CONTINUE
  50 CONTINUE
```

where H is a ***complex*** array of dimension (IH,N), R is a ***complex*** array of length (N), V is a ***complex*** array of dimension (IV,M), M2 is an integer variable, WORK is a ***complex*** array of length at least (N*N), RWORK is a ***real*** array of length at least (N), and IFAILR is an integer array of length at least (N).

F02SWF
Withdrawn at Mark 18

The following replacement ignores the triangular structure of A, and therefore references the subdiagonal elements of A; however on many machines the replacement code will be more efficient.

```
Old: CALL F02SWF(N,A,LDA,D,E,NCOLY,Y,LDY,WANTQ,Q,LDQ,IFAIL)
New: DO 20 J = 1, N
        DO 10 I = J+1, N
           A(I,J) = 0.0e0
  10    CONTINUE
  20 CONTINUE
     CALL sgebrd(N,N,A,LDA,D,E,TAUQ,TAUP,WORK,LWORK,INFO)
     IF (WANTQ) THEN
        CALL F06QFF('L',N,N,A,LDA,Q,LDQ)
        CALL sorgbr('Q',N,N,N,Q,LDQ,TAUQ,WORK,LWORK,INFO)
     END IF
     IF (NCOLY.GT.0) THEN
        CALL sormbr('Q','L','T',N,NCOLY,N,A,LDA,TAUQ,Y,LDY,
    +               WORK,LWORK,INFO)
     END IF
```

where TAUQ, TAUP and WORK are ***real*** arrays of length at least (N), and LWORK is the actual length of WORK.

F02SXF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F02SWF has been replaced by a call to F08KEF as shown above.

```
Old: CALL F02SXF(N,A,LDA,NCOLY,Y,LDY,WORK,IFAIL)
New: IF (NCOLY.EQ.0) THEN
        CALL sorgbr('P',N,N,N,A,LDA,TAUP,WORK,LWORK,INFO)
     ELSE
        CALL sormbr('P','L','T',N,NCOLY,N,A,LDA,TAUP,Y,LDY,WORK,
    +                LWORK,INFO)
     END IF
```

F02SYF
Withdrawn at Mark 18

```
Old: CALL F02SYF(N,D,E,NCOLB,B,LDB,NROWY,Y,LDY,NCOLZ,Z,LDZ,WORK,
    +            IFAIL)
New: CALL sbdsqr('U',N,NCOLZ,NROWY,NCOLB,D,E,Z,LDZ,Y,LDY,B,LDB,WORK,
    +            INFO)
```

where WORK is a ***real*** array of length at least (4*(N−1)) unless NCOLB = NROWY = NCOLZ = 0.

F02SZF
Withdrawn at Mark 15

```
Old: CALL F02SZF(N,D,E,SV,WANTB,B,WANTY,Y,NRY,LY,WANTZ,Z,NRZ,NCZ,
    +            WORK1,WORK2,WORK3,IFAIL)
New: IF (WANTB) THEN
        NCC = 1
     ELSE
        NCC = 0
     END IF
     IF (WANTY) THEN
        NRU = LY
     ELSE
        NRU = 0
     END IF
     IF (WANTZ) THEN
        NCVT = NCZ
     ELSE
        NCVT = 0
     END IF
     CALL sbdsqr('U',N,NCVT,NRU,NCC,D,E(2),Z,NRZ,Y,NRY,B,N,WORK,INFO)
```

WORK must be a one-dimensional ***real*** array of length at least (lwork) given by:

$lwork = 1$ when WANTB, WANTY and WANTZ are all false;

$lwork = max(1, 4x(N-1))$ otherwise.

The parameters WORK1, WORK2 and WORK3 are no longer required.

F02UWF
Withdrawn at Mark 18

The following replacement ignores the triangular structure of A, and therefore references the subdiagonal elements of A; however on many machines the replacement code will be more efficient.

```
Old: CALL F02UWF(N,A,LDA,D,E,NCOLY,Y,LDY,WANTQ,Q,LDQ,WORK,IFAIL)
New: DO 20 J = 1, N
        DO 10 I = J+1, N
           A(I,J) = 0.0e0
```

```
10    CONTINUE
20 CONTINUE
   CALL cgebrd(N,N,A,LDA,D,E,TAUQ,TAUP,WORK,LWORK,INFO)
   IF (WANTQ) THEN
      CALL F06TFF('L',N,N,A,LDA,Q,LDQ)
      CALL cungbr('Q',N,N,N,Q,LDQ,TAUQ,WORK,LWORK,INFO)
   END IF
   IF (NCOLY.GT.0) THEN
      CALL cunmbr('Q','L','C',N,NCOLY,N,A,LDA,TAUQ,Y,LDY,
  +                WORK,LWORK,INFO)
   END IF
```

where TAUQ and TAUP are ***complex*** arrays of length at least (N), and LWORK is the actual length of WORK.

F02UXF
Withdrawn at Mark 18

The following replacement is valid only if the previous call to F02UWF has been replaced by a call to F08KSF (CGEBRD/ZGEBRD) as shown above.

```
Old: CALL F02UXF(N,A,LDA,NCOLY,Y,LDY,RWORK,CWORK,IFAIL)
New: IF (NCOLY.EQ.0) THEN
        CALL cungbr('P',N,N,N,A,LDA,TAUP,CWORK,LWORK,INFO)
     ELSE
        CALL cunmbr('P','L','C',N,NCOLY,N,A,LDA,TAUP,Y,LDY,CWORK,
    +                LWORK,INFO)
     END IF
```

where LWORK is the actual length of CWORK.

F02UYF
Withdrawn at Mark 18

```
Old: CALL F02UYF(N,D,E,NCOLB,B,LDB,NROWY,Y,LDY,NCOLZ,Z,LDZ,WORK,
    +            IFAIL)
New: CALL cbdsqr('U',N,NCOLZ,NROWY,NCOLB,D,E,Z,LDZ,Y,LDY,B,LDB,WORK,
    +            INFO)
```

where WORK is a ***real*** array of length at least (4*(N−1)) unless NCOLB = NROWY = NCOLZ = 0.

F02WAF
Withdrawn at Mark 16

```
Old: CALL F02WAF(M,N,A,NRA,WANTB,B,SV,WORK,LWORK,IFAIL)
New: IF (WANTB) THEN
        NCOLB = 1
     ELSE
        NCOLB = 0
     END IF
     CALL F02WEF(M,N,A,NRA,NCOLB,B,M,.FALSE.,WORK,1,SV,.TRUE.,
    +            WORK,1,RWORK,IFAIL)
```

RWORK must be a one-dimensional ***real*** array of length at least *lwork* given by:

$lwork = \max(3 \times (N-1), 1)$ when WANTB is false;

$lwork = \max(5 \times (N-1), 2)$ when WANTB is true.

If, in the call to F02WAF, LWORK satisfies these conditions then F02WEF may be called with RWORK as WORK.

F02WBF
Withdrawn at Mark 14

```
Old: CALL F02WBF(M,N,A,NRA,WANTB,B,SV,WORK,LWORK,IFAIL)
New: IF (WANTB) THEN
        NCOLB = 1
     ELSE
        NCOLB = 0
     END IF
     CALL F02WEF(M,N,A,NRA,NCOLB,B,M,.FALSE.,WORK,1,SV,.TRUE.,
    +            WORK,1,RWORK,IFAIL)
```

RWORK must be a one-dimensional ***real*** array of length at least *lwork* given by:

lwork = max(3 × (M − 1), 1) when M = N and WANTB is false;

lwork = max(5 × (M − 1), 1) when M = N and WANTB is true;

lwork = $M^2 + 3 \times (M - 1)$ when M < N and WANTB is false;

lwork = $M^2 + 5 \times (M - 1)$ when M < N and WANTB is true.

In the cases where WANTB is false F02WEF may be called with RWORK as WORK, but when WANTB is true the user should check that, in the call to F02WBF, LWORK satisfies the above conditions before replacing RWORK with WORK.

F02WCF
Withdrawn at Mark 14

```
Old: CALL F02WCF(M,N,MINMN,A,NRA,Q,NRQ,SV,PT,NRPT,WORK,LWORK,
    +            IFAIL)
New: IF (M.GE.N) THEN
        CALL F06QFF('General',M,N,A,NRA,Q,NRQ)
        CALL F02WEF(M,N,Q,NRQ,0,WORK,1,.TRUE.,WORK,1,SV,.TRUE.,
    +               PT,NRPT,RWORK,IFAIL)
     ELSE
        CALL F06QFF('General',M,N,A,NRA,PT,NRPT)
        CALL F02WEF(M,N,PT,NRPT,0,WORK,1,.TRUE.,Q,NRQ,SV,.TRUE.,
    +               WORK,1,RWORK,IFAIL)
     END IF
```

RWORK must be a one-dimensional ***real*** array of length at least *lwork* given by:

lwork = $N^2 + 5 \times (N - 1)$ when M ≥ N

lwork = $M^2 + 5 \times (M - 1)$ when M < N.

If, in the call to F02WCF, LWORK satisfies these conditions then F02WEF may be called with RWORK as WORK.

F03 – Determinants

F03AGF
Withdrawn at Mark 17

```
Old: CALL F03AGF(N,M,A,IA,RL,IL,M1,D1,ID,IFAIL)
New: CALL spbtrf('Lower',N,M,A,IA,IFAIL)
```

where the array RL and its associated dimension parameter IL, and the parameters M1, D1 and ID are no longer required. In F07HDF (SPBTRF/DPBTRF), the array A holds the matrix packed using a different scheme to that used by F03AGF; see the routine document for details. F07HDF (SPBTRF/DPBTRF) overwrites A with the Cholesky factor L (without reciprocating diagonal elements) rather than returning L in the array RL. F07HDF (SPBTRF/DPBTRF) does not compute the determinant of the input matrix, returned as D1 × 2.0^{ID} by F03AGF. If this is required, it may be calculated after the call of F07HDF (SPBTRF/DPBTRF) by code similar to the following. The code computes the determinant by multiplying the diagonal elements of the factor L, taking care to avoid possible overflow or underflow.

```
      D1 = 1.0e0
      ID = 0
      DO 30 I = 1, N
         D1 = D1*A(1,I)**2
10       IF (D1.GE.1.0e0) THEN
            D1 = D1*0.0625e0
            ID = ID + 4
            GO TO 10
         END IF
20       IF (D1.LT.0.0625e0) THEN
            D1 = D1*16.0e0
            ID = ID - 4
            GO TO 20
         END IF
30    CONTINUE
```

F03AHF
Withdrawn at Mark 17

```
Old: CALL F03AHF(N,A,IA,DETR,DETI,ID,RINT,IFAIL)
New: CALL cgetrf(N,N,A,IA,IPIV,IFAIL)
```

where IPIV is an INTEGER array of length N which holds the indices of the pivot elements, and the array RINT is no longer required. It may be important to note that after a call of F07ARF (CGETRF/ZGETRF), A is overwritten by the upper triangular factor U and the off-diagonal elements of the unit lower triangular factor L, whereas the factorization returned by F03AHF gives U the unit diagonal. F07ARF (CGETRF/ZGETRF) does not compute the determinant of the input matrix, returned as ***cmplx***(DETR,DETI)$\times 2.0^{\mathrm{ID}}$ by F03AHF. If this is required, it may be calculated after a call of F07HDF (SPBTRF/DPBTRF) by code similar to the following, where DET is a ***complex*** variable. The code computes the determinant by multiplying the diagonal elements of the factor U, taking care to avoid possible overflow or underflow.

```
      DET = cmplx(1.0e0,0.0e0)
      ID = 0
      DO 30 I = 1, N
         IF (IPIV(I).NE.I) DET = -DET
         DET = DET*A(I,I)
10       IF (MAX(ABS(real(DET)),ABS(imag(DET))).GE.1.0e0) THEN
            DET = DET*0.0625e0
            ID = ID + 4
            GO TO 10
         END IF
20       IF (MAX(ABS(real(DET)),ABS(imag(DET))).LT.0.0625e0) THEN
            DET = DET*16.0e0
            ID = ID - 4
            GO TO 20
         END IF
30    CONTINUE
      DETR = real(DET)
      DETI = imag(DET)
```

F03AMF
Withdrawn at Mark 17

```
Old: CALL F01BNF(N,A,IA,P,IFAIL)
     CALL F03AMF(N,TEN,P,D1,D2)
New: CALL cpotrf('Upper',N,A,IA,IFAIL)
     D1 = 1.0e0
     D2 = 0.0e0
     DO 30 I = 1, N
        D1 = D1*real(A(I,I))**2
  10    IF (D1.GE.1.0e0) THEN
           D1 = D1*0.0625e0
           D2 = D2 + 4
           GO TO 10
        END IF
  20    IF (D1.LT.0.0625e0) THEN
           D1 = D1*16.0e0
           D2 = D2 - 4
           GO TO 20
      END IF
  30 CONTINUE
     IF (TEN) THEN
        I = D2
        D2 = D2*LOG10(2.0e0)
        D1 = D1*2.0e0**(I-D2/LOG10(2.0e0))
     END IF
```

F03AMF computes the determinant of a Hermitian positive-definite matrix after factorization by F01BNF, and has no replacement routine. F01BNF has been superseded by F07FRF (CPOTRF/ZPOTRF). To compute the determinant of such a matrix, in the same form as that returned by F03AMF, code similar to the above may be used. The code computes the determinant by multiplying the (real) diagonal elements of the factor U, taking care to avoid possible overflow or underflow.

Note that before the call of F07FRF (CPOTRF/ZPOTRF), array A contains the upper triangle of the matrix rather than the lower triangle.

F04 – Simultaneous Linear Equations

F04AKF
Withdrawn at Mark 17

```
Old: CALL F04AKF(N,IR,A,IA,P,B,IB)
New: CALL cgetrs('No Transpose',N,IR,A,IA,IPIV,B,IB,INFO)
```

It is assumed that the matrix has been factorized by a call of F07ARF (CGETRF/ZGETRF) rather than F03AHF; see the F03 Chapter Introduction for details. IPIV is an INTEGER array of length N, as returned by F07ARF (CGETRF/ZGETRF), and the array P is no longer required. INFO is an INTEGER diagnostic parameter; see the F07ASF (CGETRS/ZGETRS) routine document for details.

F04ALF
Withdrawn at Mark 17

```
Old: CALL F04ALF(N,M,IR,RL,IRL,M1,B,IB,X,IX)
New: CALL F06QFF('General',N,IR,B,IB,X,IX)
     CALL spbtrs('Lower',N,M,IR,A,IA,X,IX,INFO)
```

It is assumed that the matrix has been factorized by a call of F07HDF (SPBTRF/DPBTRF) rather than F03AGF; see the F03 Chapter Introduction for details. A is the factorized matrix as returned by F07HDF (SPBTRF/DPBTRF). The array RL, its associated dimension parameter IRL, and the parameter M1 are no longer required. INFO is an INTEGER diagnostic parameter; see the F07HEF (SPBTRS/DPBTRS) routine document for details. If the original right-hand side matrix B is no longer required, the call to

F06QFF is not necessary, and references to X and IX in the call of F07HEF (SPBTRS/DPBTRS) may be replaced by references to B and IB, in which case B will be overwritten by the solution.

F04ANF
Withdrawn at Mark 18

```
Old: CALL F04ANF(M,N,QR,IQR,ALPHA,IPIV,B,X,Z)
New: CALL scopy(N,ALPHA,1,QR,IQR+1)
     CALL sormqr('L','T',M,1,N,QR,IQR,Y,B,M,Z,N,INFO)
     CALL strsv('U','N','N',N,QR,IQR,B,1)
     DO 10 I = 1, N
        X(IPIV(I)) = B(I)
  10 CONTINUE
```

where Y must be the same ***real*** array as was used as the 7th argument in the previous call of F01AXF.

This replacement is valid only if the previous call to F01AXF has been replaced by a call to F08BEF (SGEQPF/DGEQPF) as shown above.

F04AQF
Withdrawn at Mark 16

may be replaced by calls to F06EFF (SCOPY/DCOPY), and F07GEF (SPPTRS/DPPTRS) or F07PEF (SSPTRS/DSPTRS), depending on whether the symmetric matrix has previously been factorized by F07GDF (SPPTRF/DPPTRF) or F07PDF (SSPTRF/DSPTRF) (see the description above of how to replace calls to F01BQF).

(a) where the symmetric matrix has been factorized by F07GDF (SPPTRF/DPPTRF)

```
Old: CALL F04AQF(N,M,RL,D,B,X)
New: CALL scopy(N,B,1,X,1)
     CALL spptrs('Lower',N,1,RL,X,N,INFO)
```

(b) where the symmetric matrix has been factorized by F07PDF (SSPTRF/DSPTRF)

```
Old: CALL F04AQF(N,M,RL,D,B,X)
New: CALL scopy(N,B,1,X,1)
     CALL ssptrs('Lower',N,1,RL,IPIV,X,N,INFO)
```

In both (a) and (b), the array RL must be as returned by the relevant factorization routine. The INTEGER parameter INFO is a diagnostic parameter. The INTEGER array IPIV in (b) must be as returned by F07PDF (SSPTRF/DSPTRF). The dimension parameter M, and the array D, are no longer required. If the right-hand-side array B is not needed after solution of the equations, the call to F06EFF (SCOPY/DCOPY), which simply copies array B to X, is not necessary. References to X in the calls of F07GEF (SPPTRS/DPPTRS) and F07PEF (SSPTRS/DSPTRS) may then be replaced by references to B, in which case B will be overwritten by the solution vector.

F04AWF
Withdrawn at Mark 17

```
Old: CALL F04AWF(N,IR,A,IA,P,B,IB,X,IX)
New: CALL F06TFF('General',N,IR,B,IB,X,IX)
     CALL cpotrs('Upper',N,IR,A,IA,X,IX,INFO)
```

It is assumed that the matrix has been factorized by a call of F07FRF (CPOTRF/ZPOTRF) rather than F01BNF; see the F01 Chapter Introduction for details. A is the factorized matrix as returned by F07FRF (CPOTRF/ZPOTRF). The array P is no longer required. INFO is an INTEGER diagnostic parameter; see the F07FSF (CPOTRS/ZPOTRS) routine document for details. If the original right-hand side array B is no longer required, the call to F06TFF is not necessary, and references to X and IX in the call of F07FSF (CPOTRS/ZPOTRS) may be replaced by references to B and IB, in which case B will be overwritten by the solution.

F04AYF
Withdrawn at Mark 18

```
Old: CALL F04AYF(N,IR,A,IA,P,B,IB,IFAIL)
New: CALL sgetrs('No Transpose',N,IR,A,IA,IPIV,B,IB,IFAIL)
```

It is assumed that the matrix has been factorized by a call of F07ADF (SGETRF/DGETRF) rather than F01BTF. IPIV is an INTEGER array of length N, and the array P is no longer required.

F04AZF
Withdrawn at Mark 17

```
Old: CALL F04AZF(N,IR,A,IA,P,B,IB,IFAIL)
New: CALL spotrs('Upper',N,IR,A,IA,B,IB,IFAIL)
```

It is assumed that the matrix has been factorized by a call of F07FDF (SPOTRF/DPOTRF) rather than F01BXF. The array P is no longer required.

F04LDF
Withdrawn at Mark 18

```
Old: CALL F04LDF(N,M1,M2,IR,A,IA,AL,IL,IN,B,IB,IFAIL)
New: CALL sgbtrs('No Transpose',N,M1,M2,IR,A,IA,IN,B,IB,IFAIL)
```

It is assumed that the matrix has been factorized by a call of F07BDF (SGBTRF/DGBTRF) rather than F01LBF. The array AL and its associated dimension parameter IL are no longer required.

F04MAF
Superseded at Mark 17
Scheduled for withdrawal at Mark 19

Existing programs should be modified to call F11JCF. The interfaces are significantly different and therefore precise details of a replacement call cannot be given. Please consult the appropriate routine document.

F04MBF
Superseded at Mark 17
Scheduled for withdrawal at Mark 19

If a user-defined preconditioner is required existing programs should be modified to call F11GAF, F11GBF and F11GCF. Otherwise F11JCF or F11JEF may be used. The interfaces for these routines are significantly different from that for F04MBF and therefore precise details of a replacement call cannot be given. Please consult the appropriate routine document.

F04NAF
Withdrawn at Mark 17

```
Old: CALL F04NAF(JOB,N,ML,MU,A,NRA,IN,B,TOL,IFAIL)
New: JOB = ABS(JOB)
     IF (JOB.EQ.1) THEN
        CALL cgbtrs('No Transpose',N,ML,MU,1,A,NRA,IN,B,N,IFAIL)
     ELSE IF (JOB.EQ.2) THEN
        CALL cgbtrs('Conjugate Transpose',N,ML,MU,1,A,NRA,IN,B,N,IFAIL)
     ELSE IF (JOB.EQ.3) THEN
        CALL ctbsv('Upper','No Transpose','Non-unit',N,ML+MU,A,NRA,B,1)
     END IF
```

It is assumed that the matrix has been factorized by a call of F07BRF (CGBTRF/ZGBTRF) rather than F01NAF. The replacement routines do not have the functionality to perturb diagonal elements of the triangular factor U, as specified by a negative value of JOB in F04NAF. The parameter TOL is therefore no longer useful. If this functionality is genuinely required, please contact NAG.

F05 – Orthogonalisation

F05ABF
Withdrawn at Mark 14

```
Old: U = F05ABF(X,N)
New: U = snrm2(N,X,1)
```

F06 – Linear Algebra Support Routines

F06QGF
Withdrawn at Mark 16

```
Old: ANORM = F06QGF(NORM,MATRIX,M,N,A,LDA)
New: C = MATRIX(1:1)
     IF ( (C.EQ.'G') .OR. (C.EQ.'g') ) THEN
        ANORM = F06RAF(NORM,M,N,A,LDA,WORK1)
     ELSE IF ( (C.EQ.'H') .OR. (C.EQ.'h') .OR. (C.EQ.'S') .OR.
    +          (C.EQ.'s')) THEN
        ANORM = F06RCF(NORM,'U',N,A,LDA,WORK2)
     ELSE IF ( (C.EQ.'E') .OR. (C.EQ.'e') .OR. (C.EQ.'Y') .OR.
    +     (C.EQ.'y')) THEN
        ANORM = F06RCF(NORM,'L',N,N,A,LDA,WORK1)
     ELSE IF ( (C.EQ.'U') .OR. (C.EQ.'u') ) THEN
        ANORM = F06RJF(NORM,'U','N',M,N,A,LDA,WORK1)
     ELSE IF ( (C.EQ.'L') .OR. (C.EQ.'l') ) THEN
        ANORM = F06RJF(NORM,'L','N',M,N,A,LDA,WORK1)
     END IF
```

C must be declared as `CHARACTER*1`, WORK1 as a ***real*** array of dimension (1) and WORK2 as a ***real*** array of dimension (N).

F06VGF
Withdrawn at Mark 16

```
Old: ANORM = F06VGF(NORM,MATRIX,M,N,A,LDA)
New: C = MATRIX(1:1)
     IF ( (C.EQ.'G') .OR. (C.EQ.'g') ) THEN
        ANORM = F06UAF(NORM,M,N,A,LDA,WORK1)
     ELSE IF ( (C.EQ.'H') .OR. (C.EQ.'h') .OR. (C.EQ.'S') .OR.
    +          (C.EQ.'s')) THEN
        ANORM = F06UCF(NORM,'U',N,A,LDA,WORK2)
     ELSE IF ( (C.EQ.'E') .OR. (C.EQ.'e') .OR. (C.EQ.'Y') .OR.
    +     (C.EQ.'y')) THEN
        ANORM = F06UCF(NORM,'L',N,A,LDA,WORK1)
     ELSE IF ( (C.EQ.'U') .OR. (C.EQ.'u') ) THEN
        ANORM = F06UJF(NORM,'U','N',M,N,A,LDA,WORK1)
     ELSE IF ( (C.EQ.'L') .OR. (C.EQ.'l') ) THEN
        ANORM = F06UJF(NORM,'L','N',M,N,A,LDA,WORK1)
     END IF
```

C must be declared as `CHARACTER*1`, WORK1 as a ***real*** array of dimension (1) and WORK2 as a ***real*** array of dimension (N).

G01 – Simple Calculations on Statistical Data

G01BAF
Withdrawn at Mark 16

```
Old: P = G01BAF(IDF,T,IFAIL)
New: P = G01EBF('Lower-tail',T,real(IDF),IFAIL)
```

G01BBF
Withdrawn at Mark 16

```
Old: P = G01BBF(I1,I2,A,IFAIL)
New: P = G01EDF('Upper-tail',A,real(I1),real(I2),IFAIL)
```

G01BCF
Withdrawn at Mark 16

```
Old: P = G01BCF(X,N,IFAIL)
New: P = G01ECF('Upper-tail',X,real(N),IFAIL)
```

G01BDF
Withdrawn at Mark 16

```
Old: P = G01BDF(X,A,B,IFAIL)
New: CALL G01EEF(X,A,B,TOL,P,Q,PDF,IFAIL)
```

where TOL is set to the accuracy required by the user and Q and PDF are additional output quantities.

Note. The values of A and B must be $\leq 10^6$.

G01CAF
Withdrawn at Mark 16

```
Old: T = G01CAF(P,N,IFAIL)
New: T = G01FBF('Lower-tail',P,real(N),IFAIL)
```

G01CBF
Withdrawn at Mark 16

```
Old: F = G01CBF(P,M,N,IFAIL)
New: F = G01FDF(P,real(M),real(N),IFAIL)
```

G01CCF
Withdrawn at Mark 16

```
Old: X = G01CCF(P,N,IFAIL)
New: X = G01FCF(P,real(N),IFAIL)
```

G01CDF
Withdrawn at Mark 16

```
Old: X = G01CDF(P,A,B,IFAIL)
New: X = G01FEF(P,A,B,TOL,IFAIL)
```

where TOL is set to the accuracy required by the user.

Note. The values of A and B must be $\leq 10^6$.

G01CEF
Withdrawn at Mark 18

```
Old: X = G01CEF(P,IFAIL)
New: X = G01FAF('Lower-tail',P,IFAIL)
```

G02 – Correlation and Regression Analysis

G02CJF
Withdrawn at Mark 16

```
Old:       CALL G02CJF(X,IX,Y,IY,N,M,IR,THETA,IT,SIGSQ,C,IC,IPIV,
          +            WK1,WK2,IFAIL)
New: C     set the first M elements of ISX to 1
           CALL F06DBF(M,1,ISX,1)
     C     THEN
           TOL = X02AJF()
           CALL G02DAF('Zero','Unweighted',N,X,IX,M,ISX,M,Y,WT,
          +            RSS,IDF,THETA,SE,COV,RES,H,C,IC,SVD,IRANK,
          +            P,TOL,WK,IFAIL)
```

```
      SIGSQ(1) = RSS/IDF
C     there are two or more dependent variables,
C     i.e., IR is greater than or equal to 2 then:
      DO 20 I = 2, IR
         CALL G02DGF('Unweighted',N,WT,RSS,IP,IRANK,COV,C,IC,SVD,
     +               P,Y(1,I),THETA(1,I),SE,RES,WK,IFAIL)
         SIGSQ(I) = RSS/IDF
   20 CONTINUE
```

For unweighted regression, as is used here, WT may be any ***real*** array and will not be referenced, e.g. SIGSQ could be used.

The array C no longer contains $(X^TX)^{-1}$; however, $(X^TX)^{-1}$ scaled by $\hat{\sigma}^2$ is returned in packed form in array COV. The upper triangular part of C will now contain a factorization of X^TX.

The ***real*** arrays SE(M), COV(M*(M + 1)/2), RES(N), H(N), P(M*(M + 2)), the logical variable SVD and the INTEGER variable IRANK are additional outputs. There is also a single ***real*** workspace WK(5*(M − 1) + M * M).

G04 – Analysis of Variance

G04ADF
Withdrawn at Mark 17

```
Old: CALL G04ADF(DATA,VAR,AMR,AMC,AMT,LCODE,IA,N,NN)
New: IFAIL = 0
     CALL G04BCF(1,N,N,DATA,N,IT,GMEAN,AMT,TABLE,6,C,NMAX,
    +            IREP,RPMEAN,AMR,AMC,R,EF,0.0,0,WK,IFAIL)
```

The arrays AMR, AMC and AMT contain the means of the rows, columns and treatments rather than the totals. The values equivalent to those returned in the array VAR of G04ADF are returned in the second column of the two-dimensional array TABLE starting at the second row, e.g., VAR(1) = TABLE(2,2). The two dimensional integer array LCODE (containing the treatment codes) has been replaced by the one-dimensional array IT. These arrays will be the equivalent if IA = N. The following additional declarations are required.

```
 real       GMEAN
 INTEGER    IFAIL
 real       C(NMAX,NMAX), EF(NMAX), TABLE(6,5), R(NMAX*NMAX),
+           RPMEAN(1), WK(NMAX*NMAX+NMAX)
 INTEGER    IREP(NMAX), IT(NMAX*NMAX)
```

where NMAX is an integer such that NMAX ≥ N.

G04AEF
Withdrawn at Mark 17

```
Old: CALL G04AEF(Y,N,K,NOBS,GBAR,GM,SS,IDF,F,FP,IFAIL)
New: CALL G04BBF(N,Y,0,K,IT,GM,BMEAN,GBAR,TABLE,4,C,KMAX,NOBS,
    +            R,EF,0.0e0,0,WK,IFAIL)
```

The values equivalent to those returned by G04AEF in the arrays IDF and SS are returned in the first and second columns of TABLE starting at row 2 and the values equivalent to those returned in the scalars F and FP are returned in TABLE(2,4) and TABLE(2,5) respectively. NOBS is output from G04BBF rather than input. The groups are indicated by the array IT. The following code illustrates how IT can be computed from NOBS.

```
      IJ = 0
      DO 40 I = 1, K
         DO 20 J = 1, NOBS(I)
            IJ = IJ + 1
            IT(IJ) = I
20       CONTINUE
40    CONTINUE
```

The following additional declarations are required.

```
 real        BMEAN(1),C(KMAX,KMAX),EF(KMAX),R(NMAX),TABLE(4,5),
+            WK(KMAX*KMAX+KMAX)
 INTEGER     IT(NMAX)
```

NMAX and KMAX are integers such that NMAX $\geq$ N and KMAX $\geq$ K.

G04AFF
Withdrawn at Mark 17

```
Old: CALL G04AFF(Y,IY1,IY2,M,NR,NC,ROW,COL,CELL,ICELL,GM,SS,IDF,F,FP,
    +            IFAIL)
New: CALL G04CAF(M*NR*NC,Y1,2,LFAC,1,2,0,6,TABLE,ITOTAL,TMEAN,MAXT,E,
    +            IMEAN,SEMEAN,BMEAN,R,IWK,IFAIL)
```

Where Y1 is a one-dimensional array containing the observations in the same order as Y, if IY1 = M and IY2 = NR then these are equivalent. LFAC is an integer array such that LFAC(1) = NC and LFAC(2) = NR. The following indicates how the results equivalent to those produced by G04AFF can be extracted from the results produced by G04CAF.

G04AFF	G04CAF
ROW(i)	TMEAN(IMEAN(1)+i), i = 1,2,...,NR
COL(j)	TMEAN(j), j = 1,2,...,NC
CELL(i,j)	TMEAN(IMEAN(2)+(j-1)*NR+i), i = 1,2,...,NR; j = 1,2,...,NC
GM	BMEAN(1)
SS(1)	TABLE(3,2)
SS(2)	TABLE(2,2)
SS(i)	TABLE(4,2)
IDF(1)	TABLE(3,1)
IDF(2)	TABLE(2,1)
IDF(i)	TABLE(4,1)
F(1)	TABLE(3,4)
F(2)	TABLE(2,4)
F(3)	TABLE(4,4)
FP(1)	TABLE(3,5)
FP(2)	TABLE(2,5)
FP(3)	TABLE(4,5)

Note how rows and columns have swapped.

The following additional declarations are required.

```
 real        TABLE(6,5), R(NMAX), TMEAN(MAXT), E(MAXT), BMEAN(1),
+            SEMEAN(5)
 INTEGER     IMEAN(5), IWK(NMAX+6), LFAC(2)
```

NMAX and MAXT are integers such that NMAX $\geq$ M $\times$ NR $\times$ NC and MAXT $\geq$ NR + NC + NR $\times$ NC.

G05 – Random Number Generators

G05DGF
Withdrawn at Mark 16

```
Old: X = G05DGF(G,H,IFAIL)
New: CALL G05FFF(G,H,1,X(1),IFAIL)
```

where X must now be declared as an array of length at least 1.

G05DLF
Withdrawn at Mark 16

```
Old: X = G05DLF(G,H,IFAIL)
New: CALL G05FEF(G,H,1,X(1),IFAIL)
```

where X must now be declared as an array of length at least 1.

G05DMF
Withdrawn at Mark 16

```
Old: X = G05DMF(G,H,IFAIL)
New: CALL G05FEF(G,H,1,X(1),IFAIL)
     IF (X(1).LT.1.0e0) X(1) = X(1)/(1.0e0-X(1))
```

where X must now be declared as an array of length at least 1. If the value of X(1) returned by G05FEF is 1.0, appropriate action should be taken. Alternatively the ratio of gamma variates can be used i.e.,

```
CALL G05FFF(G,1.0e0,1,X(1),IFAIL1)
CALL G05FFF(H,1.0e0,1,Y(1),IFAIL2)
IF (Y(1).NE.0.0e0) X(1) = X(1)/Y(1)
```

where Y must be declared as an array of length at least 1.

G08 – Nonparametric Statistics

G08ABF
Withdrawn at Mark 16

```
Old: CALL G08ABF(X,Y,N,W1,W2,W,N1,P,IFAIL)
New: DO 20 I = 1, N
        Z(I) = X(I) - Y(I)
  20 CONTINUE
     XME = 0.0e0
     CALL G08AGF(N,Z,XME,'Lower-tail','No-zeros',W,WNOR,P,
    +            N1,W1,IFAIL)
```

W1 is a ***real*** work array of dimension (3*N). The ***real*** array W2 is no longer required. WNOR returns the normalized Wilcoxon test statistic. The ***real*** array Z, of dimension (N), contains the difference between the paired sample observations, and by setting the ***real*** variable XME to zero the routine may be used to test whether the medians of the two matched or paired samples are equal.

G08ADF
Withdrawn at Mark 16

```
Old: CALL G08ADF(X,N,N1,W,U,P,IFAIL)
New: N2 = N - N1
     CALL G08AHF(N1,X,N2,X(N1+1),'Lower-tail',U,UNOR,P,
    +            TIES,RANKS,W,IFAIL)
```

The observations from the two independent samples must be stored in two separate ***real*** arrays, of dimensions N1 and N2, where N2 = N − N1, rather than consecutively in one array as in the superseded routine G08ADF.

UNOR returns the normalized Mann–Whitney *U* statistic. The LOGICAL parameter TIES indicates whether ties were present in the pooled sample or not and RANKS, a ***real*** array of dimension (N1 + N2), returns the ranks of the pooled sample.

Both G08ADF and its replacement routine G08AHF return approximate tail probabilities for the test statistic. To compute exact tail probabilities G08AJF may be used if there are no ties in the pooled sample and G08AKF may be used if there are ties in the pooled sample.

G08CAF
Withdrawn at Mark 16

```
Old: CALL G08CAF(N,X,NULL,NP,P,NEST,NTYPE,D,PROB,S,IND,IFAIL)
New: CALL G08CBF(N,X,DIST,PAR,NEST,NTYPE,D,Z,PROB,S,IFAIL)
```

The following table indicates how existing choices for the null distribution, indicated through the INTEGER variable NULL in G08CAF, may be made in G08CBF using the character variable DIST.

null distribution	G08CAF - NULL	G08CBF - DIST
uniform	1	'U'
Normal	2	'N'
Poisson	3	'P'
exponential	4	'E'

PAR is a ***real*** array of dimension (1) for both the one and two parameter distributions, but only the first element of PAR is actually referenced (used) if the chosen null distribution has only one parameter. The input parameter NP is no longer required.

On exit S contains the sample observations sorted into ascending order. It no longer contains the sample cumulative distribution function but this may be computed from S.

G13 – Time Series Analysis

G13DAF
Withdrawn at Mark 17

```
Old:        CALL G13DAF(X,NXM,NX,NSM,NS,NL,ICR,C0,C,IFAIL)
New: C      First transpose the data matrix X
     C      note NSM is used as the first dimension of the array W
            DO 20 I = 1, NS
               CALL F06EFF(NX,X,(1,I),1,W(I,1),NSM)
         20 CONTINUE
     C      then if ICR = 0 in the call to G13DAF
            CALL G13DMF('V-Covariances',NS,NX,W,NSM,NL,WMEAN,C0,C,IFAIL)
     C      else if ICR = 1 in the call to G13DAF
            CALL G13DMF('R-Correlations',NS,NX,W,NSM,NL,WMEAN,C0,C,IFAIL)
```

Note that in G13DAF the NS series are stored in the columns of X whereas in G13DMF these series are stored in rows; hence it is necessary to transpose the data array.

The ***real*** array WMEAN must be of length NS, and on output stores the means of each of the NS series.

The diagonal elements of C0 store the variances of the series if covariances are requested, but the standard deviations if correlations are requested.

H – Operations Research

H02BAF
Withdrawn at Mark 15

```
Old:    CALL H02BAF(A,MM,N1,M,N,200,L,X,NUMIT,OPT,IFAIL)
New: C  M, N and MM must be set before these declaration statements
        INTEGER    MAXDPT, LIWORK, LRWORK, ITMAX, MSGLVL, MAXNOD, INTFST
        PARAMETER  (LIWORK = (25+N+M)*MAXDPT + 5*N + M + 4)
        PARAMETER  (LRWORK = MAXDPT*(N+2) + 2*N*N + 13*N + 12*M)
        INTEGER    INTVAR(N), IWORK(LIWORK)
        real       BIGBND, TOLFES, TOLIV, ROPT
        real       RA(MM,N), RX(N), CVEC(N), BL(N+M), BU(N+M),
       +           RWORK(LRWORK)
        IFAIL = 0
        DO 10 J = 1, N
           INTVAR(J) = 1
```

```
            CVEC(J) = A(1,J)
            RX(J) = 1.0e0
            DO 20 I = 1, M
               RA(I,J) = A(I+1,J)
20          CONTINUE
10       CONTINUE

         BIGBND = 1.0e20
         DO 30 I = 1, N
            BL(I) = 0.0e0
            BU(I) = BIGBND
30       CONTINUE
         DO 40 I = N+1, N+M
            BU(I) = A(I-N+1,N+1)
            BL(I) = -BIGBND
40       CONTINUE
         ITMAX = 0
         MSGLVL = 0
         MAXNOD = 0
         INTFST = 0
         TOLIV = 0.0e0
         TOLFES = 0.0e0
         MAXDPT = 3*N/2

         CALL H02BBF(ITMAX,MSGLVL,N,M,RA,MM,BL,BU,INTVAR,CVEC,MAXNOD,
        +            INTFST,MAXDPT,TOLIV,TOLFES,BIGBND,RX,ROPT,IWORK,
        +            LIWORK,RWORK,LRWORK,IFAIL)
         L = 1
         IF (IFAIL.EQ.0) L = 0
         IF (IFAIL.EQ.4) L = 2

         IF (L.EQ.0) THEN
            DO 50 I = 1, N
               X(I) = RX(I)
50          CONTINUE
            OPT = ROPT
         ENDIF
```

The code indicates the minimum changes necessary, but H02BBF has additional flexibility and users may wish to take advantage of new features. It is strongly recommended that users consult the routine document.

M01 – Sorting

M01AAF
Withdrawn at Mark 13

```
Old: CALL M01AAF(A,M,N,IP,IST,IFAIL)
New: CALL M01DAF(A(M),1,N-M+1,'A',IP(M),IFAIL)
```

The array IST is no longer needed.

M01ABF
Withdrawn at Mark 13

```
Old: CALL M01ABF(A,M,N,IP,IST,IFAIL)
New: CALL M01DAF(A(M),1,N-M+1,'D',IP(M),IFAIL)
```

The array IST is no longer needed.

M01ACF
Withdrawn at Mark 13

```
Old: CALL M01ACF(IA,M,N,IP,IST,IFAIL)
New: CALL M01DBF(IA(M),1,N-M+1,'A',IP(M),IFAIL)
```

The array IST is no longer needed.

M01ADF
Withdrawn at Mark 13

```
Old: CALL M01ADF(IA,M,N,IP,IST,IFAIL)
New: CALL M01DBF(IA(M),1,N-M+1,'D',IP(M),IFAIL)
```

The array IST is no longer needed.

M01AEF
Withdrawn at Mark 13

```
Old: CALL M01AEF(A,NR,NC,IC,T,TT,IFAIL)
New: CALL M01DEF(A,NR,1,NR,IC,IC,'A',IRANK,IFAIL)
     DO 10 I = 1, NC
        CALL M01EAF(A(1,I),1,NR,IRANK,IFAIL)
  10 CONTINUE
```

The ***real*** arrays T and TT are no longer needed, but a new integer array IRANK of length NR is required.

M01AFF
Withdrawn at Mark 13

```
Old: CALL M01AFF(A,NR,NC,IC,T,TT,IFAIL)
New: CALL M01DEF(A,NR,1,NR,IC,IC,'D',IRANK,IFAIL)
     DO 10 I = 1, NC
        CALL M01EAF(A(1,I),1,NR,IRANK,IFAIL)
  10 CONTINUE
```

The ***real*** arrays T and TT are no longer needed, but a new integer array IRANK of length NR is required.

M01AGF
Withdrawn at Mark 13

```
Old: CALL M01AGF(IA,NR,NC,IC,K,L,IFAIL)
New: CALL M01DFF(IA,NR,1,NR,IC,IC,'A',IRANK,IFAIL)
     DO 10 I = 1, NC
        CALL M01EBF(IA(1,I),1,NR,IRANK,IFAIL)
  10 CONTINUE
```

The integer arrays K and L are no longer needed, but a new integer array IRANK of length NR is required.

M01AHF
Withdrawn at Mark 13

```
Old: CALL M01AHF(IA,NR,NC,IC,K,L,IFAIL)
New: CALL M01DFF(IA,NR,1,NR,IC,IC,'D',IRANK,IFAIL)
     DO 10 I = 1, NC
        CALL M01EBF(IA(1,I),1,NR,IRANK,IFAIL)
  10 CONTINUE
```

The integer arrays K and L are no longer needed, but a new integer array IRANK of length NR is required.

M01AJF
Withdrawn at Mark 16

```
Old: CALL M01AJF(A,W,IND,INDW,N,NW,IFAIL)
New: CALL M01DAF(A,1,N,'A',IND,IFAIL)
     CALL M01ZAF(IND,1,N,IFAIL)
     CALL M01CAF(A,1,N,'A',IFAIL)
```

The arrays W and INDW are no longer needed.

M01AKF
Withdrawn at Mark 16

```
Old: CALL M01AKF(A,W,IND,INDW,N,NW,IFAIL)
New: CALL M01DAF(A,1,N,'D',IND,IFAIL)
     CALL M01ZAF(IND,1,N,IFAIL)
     CALL M01CAF(A,1,N,'D',IFAIL)
```

The arrays W and INDW are no longer needed.

M01ALF
Withdrawn at Mark 13

```
Old: CALL M01ALF(IA,IW,IND,INDW,N,NW,IFAIL)
New: CALL M01DBF(IA,1,N,'A',IND,IFAIL)
     CALL M01ZAF(IND,1,N,IFAIL)
     CALL M01CBF(IA,1,N,'A',IFAIL)
```

The arrays IW and INDW are no longer needed.

M01AMF
Withdrawn at Mark 13

```
Old: CALL M01AMF(IA,IW,IND,INDW,N,NW,IFAIL)
New: CALL M01DBF(IA,1,N,'D',IND,IFAIL)
     CALL M01ZAF(IND,1,N,IFAIL)
     CALL M01CBF(IA,1,N,'D',IFAIL)
```

The arrays IW and INDW are no longer needed.

M01ANF
Withdrawn at Mark 13

```
Old: CALL M01ANF(A,I,J,IFAIL)
New: CALL M01CAF(A,I,J,'A',IFAIL)
```

M01APF
Withdrawn at Mark 16

```
Old: CALL M01APF(A,I,J,IFAIL)
New: CALL M01CAF(A,I,J,'D',IFAIL)
```

M01AQF
Withdrawn at Mark 13

```
Old: CALL M01AQF(IA,I,J,IFAIL)
New: CALL M01CBF(IA,I,J,'A',IFAIL)
```

M01ARF
Withdrawn at Mark 13

```
Old: CALL M01ARF(IA,I,J,IFAIL)
New: CALL M01CBF(IA,I,J,'D',IFAIL)
```

The character-sorting routines M01BAF, M01BBF, M01BCF and M01BDF have no exact replacements, because they require the data to be stored in an integer array, whereas the new character-sorting routines require the data to be stored in a character array. The following advice assumes that calling programs are modified so that the data is stored in a character array CH instead of in an integer array IA; ***nchar*** denotes the machine-dependent number of characters stored in an integer variable. The new routines sort according to the ASCII collating sequence, which may differ from the machine-dependent collating sequence used by the old routines.

M01BAF
Withdrawn at Mark 13

```
Old: CALL M01BAF(IA,I,J,IFAIL)
New: CALL M01CCF(CH,I,J,1,nchar,'D',IFAIL)
```

assuming that each element of the character array CH corresponds to one element of the integer array IA.

M01BBF
Withdrawn at Mark 13

```
Old: CALL M01BBF(IA,I,J,IFAIL)
New: CALL M01CCF(CH,I,J,1,nchar,'A',IFAIL)
```

assuming that each element of the character array CH corresponds to one element of the integer array IA.

M01BCF
Withdrawn at Mark 13

```
Old: CALL M01BCF(IA,NR,NC,L1,L2,LC,IUC,IT,ITT,IFAIL)
New: CALL M01CCF(CH,LC,IUC,(L1-1)*nchar-1,L2*nchar,'D',IFAIL)
```

provided that each element of the character array CH corresponds to a whole column of the integer array IA. The arrays IT and ITT are no longer needed. The call of M01CCF will fail if NR*$nchar$ exceeds 255.

M01BDF
Withdrawn at Mark 13

```
Old: CALL M01BDF(IA,NR,NC,L1,L2,LC,IUC,IT,ITT,IFAIL)
New: CALL M01CCF(CH,LC,IUC,(L1-1)*nchar-1,L2*nchar,'A',IFAIL)
```

provided that each element of the character array CH corresponds to a whole column of the integer array IA. The arrays IT and ITT are no longer needed. The call of M01CCF will fail if NR*$nchar$ exceeds 255.

X02 – Machine Constants

X02AAF
Withdrawn at Mark 16

```
Old: X02AAF(X)
New: X02AJF()
```

X02ABF
Withdrawn at Mark 16

```
Old: X02ABF(X)
New: X02AKF()
```

X02ACF
Withdrawn at Mark 16

```
Old: X02ACF(X)
New: X02ALF()
```

X02ADF
Withdrawn at Mark 14

```
Old: X02ADF(X)
New: X02AKF()/X02AJF()
```

X02AEF*
Withdrawn at Mark 14

```
Old: X02AEF(X)
New: LOG(X02AMF())
```

X02AFF*
Withdrawn at Mark 14

```
Old: X02AFF(X)
New: -LOG(X02AMF())
```

X02AGF*
Withdrawn at Mark 16

```
Old: X02AGF(X)
New: X02AMF()
```

X02BAF
Withdrawn at Mark 14

```
Old: X02BAF(X)
New: X02BHF()
```

X02BCF*
Withdrawn at Mark 14

```
Old: X02BCF(X)
New: -LOG(X02AMF())/LOG(2.0)
```

X02BDF*
Withdrawn at Mark 14

```
Old: X02BDF(X)
New: LOG(X02AMF())/LOG(2.0)
```

X02CAF
Withdrawn at Mark 17

This routine is no longer required.

Note. In the case of the routines marked with an asterisk (*), the replacement expressions may not return the same value, but the value will be sufficiently close, and safe, for the purposes for which it is used in the Library.

Acknowledgements

Below we list the names of those people who have made a substantial contribution to the design, development and validation of software that is included in the current Mark of the Library (in the designated chapters).

The list includes the names of those who have collaborated with NAG specifically to develop software for the Library; and also the names of the authors of public-domain software that has been adapted for inclusion in the Library. It gives the institutions at which the individuals were working at the time they made their contributions, not necessarily their present addresses. It does not include the names of those – too numerous to mention individually – who have contributed ideas, criticisms, reports of errors, or suggestions for improvements to the software; nor does it cover work done by NAG full-time staff or those who are responsible for implementing the Library on different machines.

We acknowledge with gratitude the contributions of all these people to the current Mark of the Library.

J P Abbott, University of Nottingham	(F03, F04)
D Amos, Sandia National Laboratories, Albuquerque	(S)
E Anderson, University of Tennessee and Cray Research	(F07, F08)
G T Anthony, National Physical Laboratory	(E01, E02)
Z Bai, New York University, University of Tennessee and University of Kentucky	(F07, F08)
H M Barber, National Physical Laboratory	(E04)
I Barrodale, University of Victoria	(E02)
R H Bartels, Johns Hopkins University	(E02)
W Barth, Technische Hochschule Darmstadt	(F02)
M Berzins, University of Leeds	(D02, D03)
G P Bhattacharjee, Indian Institute of Technology, Kharagpur	(G01)
C Bischof, Argonne National Laboratory	(F07, F08)
J M Boyle, Argonne National Laboratory	(F02)
R W Brankin, University of Manchester	(D02)
R Bulirsch, Technische Hochschule, München	(S)
J C P Bus, Mathematisch Centrum, Amsterdam	(C05)
P A Businger, Bell Telephone Laboratories	(F01, F04)
B C Carlson, Iowa State University	(S)
R E Carlson, Lawrence Livermore Laboratory	(E01)
A R Conn, University of Waterloo	(E02)
A Cook, Middlesex University	(G03)
M G Cox, National Physical Laboratory	(E01, E02, F01, F04)
B Curtis, National Physical Laboratory	(E02)
P Curtis, National Physical Laboratory	(E01, E02)
C Daly, University of Lancaster	(G13)
T J Dekker, University of Amsterdam	(C05)
L M Delves, University of Liverpool	(D01)
J W Demmel, New York University and University of California, Berkeley	(F07, F08)
P M Dew, University of Leeds	(D03)
R Dias Da Cunha, University of Rio Grande do Sul	(F11)
P Dierckx, University of Leuven	(E02)
D R Divgi, Syracuse University	(G01)
D S Dodson, Convex Computer Corporation	(F06)
E de Doncker-Kapenga, University of Leuven and Western Michigan University	(D01)
J J Dongarra, Argonne National Laboratory, University of Tennessee and Oak Ridge National Laboratory	(F02, F06, F07, F08)
J R Dormand, Teeside Polytechnic	(D02)
M Drew, University of Birmingham	(H)
I S Duff, AERE Harwell	(F01, F04, F06)
B Ford, University of Nottingham	(E01, F01, F02)
R Franke, Naval Postgraduate School	(E01)
F N Fritsch, Lawrence Livermore National Laboratory	(E01)
R M Furzeland, Shell Research Ltd	(D02)

B S Garbow, Argonne National Laboratory (C05, C06, F02)
W Gautschi, Purdue University (S)
A C Genz, University of Kent and Washington State University (D01)
P E Gill, National Physical Laboratory, Stanford University and University of California at San Diego (E04)
G Giunta, Argonne National Laboratory (C06)
I Gladwell, University of Manchester and Southern Methodist University (C05, D02, E01, E02, F01, F04)
G H Golub, University of Stanford (F01, F04)
N I M Gould, National Physical Laboratory (E04)
S R Graham, National Physical Laboratory (E04)
P R Graves-Morris, University of Kent (E01, E02)
A Greenbaum, New York University (F07, F08)
P Griffiths, University of Oxford (G02)
R G Grimes, Boeing Computer Services (F06)
K Halstead, University of Warwick (G01)
S J Hammarling, Middlesex Polytechnic and National Physical Laboratory (E04, F01, F02, F04)
D C Handscomb, University of Oxford (G05)
R J Hanson, Sandia National Laboratories, Albuquerque (F06)
J A Hartigan, Yale University (G03)
R W Hatfield, University of Nottingham (E01)
R I Hay, University of Manchester (F01, F04)
J G Hayes, National Physical Laboratory (E01, E02)
L Hayes, University of Oxford (F01, F02, F03, F04, F05)
N J Higham, University of Manchester (F04, F07)
I D Hill, Medical Research Council (G01)
K E Hillstrom, Argonne National Laboratory (C05)
B T Hinde, National Physical Laboratory (E04)
A C Hindmarsh, Lawrence Livermore National Laboratory (D02)
D C Hoaglin, Harvard University (G01, G10)
S M Hodson, National Physical Laboratory (D02, E04)
T R Hopkins, University of Kent (E01, F11)
M Hurley, University of Lancaster (G13)
M F Hutchinson, CSIRO, Canberra (G10)
A N Jack, University of Nottingham (E02)
D A H Jacobs, Central Electricity Research Laboratory (D03)
D K Kahaner, National Bureau of Standards (D01)
M S Keech, University of Birmingham (S)
M Kennedy, Queen's University, Belfast and Open University (D01)
P D Kenward, National Physical Laboratory (F01, F02)
D R Kincaid, University of Texas, Austin (F06)
L Knüsel, University of Munich (G01)
F T Krogh, Jet Propulsion Laboratory (F06)
C L Lawson, Jet Propulsion Laboratory (F06)
M Lentini, California Institute of Technology (D02)
S A Lill, University of Liverpool (E04)
G R Lindfield, Massey University (C05)
J Lloyd-Jones, University of Manchester (G08)
E M R Long, National Physical Laboratory (E04)
J N Lyness, Argonne National Laboratory (C06, D01, D04)
A McKenney, New York University (F07, F08)
N M Maclaren, University of Cambridge (G01, G05, M01)
J R Magnus, London School of Economics and Center for Economic Research, Tilburg (G01)
K L Majumder, Space Applications Centre, Ahmedabad (G01)
A Marazzi, University of Lausanne (G02, G07)
R S Martin, National Physical Laboratory (F01, F02, F03, F04)
G F Miller, National Physical Laboratory (D01, D05)
C B Moler, University of New Mexico (F02)

J J Moré, Argonne National Laboratory (C05)
A Murli, University of Naples (C06)
N Munksgaard, AERE Harwell (F01, F04)
W Murray, National Physical Laboratory and Stanford University (E04)
N Neumann, University of Göttingen (G08)
P Nicholson, University of Leeds (G08)
P J Nikolai, US Air Force Flight Dynamics Laboratory (F02)
E M Notis, Iowa State University (S)
M R O'Donohoe, Cambridge University (C06, D01)
S Ostrouchov, University of Tennessee (F07, F08)
C C Paige, McGill University (F04)
J M Parkinson, University of Leeds (E04)
B N Parlett, University of California, Berkeley (F01)
T N L Patterson, The Queen's University of Belfast (D01)
A Paver, University of Middlesex (G01)
S V Pennington, University of Leeds (D03)
V Pereyra, University of Caracas (D02)
B Pesaran, Bank of England (G01)
G Peters, National Physical Laboratory (F01, F02, F03, F04)
A N Pettit, Loughborough University (G01, G08)
L Petzold, Lawrence Livermore National Laboratory (D02)
C Phillips, University of Liverpool (E02)
W Phillips, University of Oxford (F01, F02, F03, F04, F05)
R Piessens, University of Leuven (D01)
R A Pitfield, National Physical Laboratory (D03)
P J Prince, Teeside Polytechnic (D02)
J D Pryce, University of Bristol (D02)
G Radicati, IBM ECSEC, Rome (F07)
M Razzaz, University of Birmingham (S)
J K Reid, AERE Harwell (D03, F01, F04)
C Reinsch, Technical University Munich (F01, F02)
R J Renka, Oak Ridge National Laboratory (E01)
D G Rhead, University of Nottingham (E04)
A Riley, University of Nottingham (G04)
F D K Roberts, University of Victoria (E02)
K Robinson, Rutherford Appleton Laboratory (C06)
D Roose, University of Leuven (D01)
J P Royston, Medical Research Council (G01)
G Sande, University of Chicago (C06)
M A Saunders, Stanford University (E04, F04)
J L Schonfelder, University of Birmingham and University of Liverpool (S)
W L Seward, University of Oxford and University of Waterloo (D02)
L F Shampine, Sandia National Laboratories, Albuquerque and Southern Methodist University (D02)
B L Shea, University of Aberystwyth, London School of Economics and Manchester Metropolitan University (G05, G11, G13)
B W Silverman, University of Bath (G10)
J W Sinclair, University of Dundee (E02)
M A Singer, National Physical Laboratory (E01)
A J Skellern, Loughborough University (G01, G08)
B T Smith, Argonne National Laboratory (C02)
D C Sorensen, Argonne National Laboratory and Rice University (F07, F08)
P N Swarztrauber, National Centre for Atmospheric Research (D03)
R A Sweet, University of Colorado at Denver (D03)
A Swift, Massey University (C05)
G T Symm, National Physical Laboratory (D03, D05, F01)
H J Symm, National Physical Laboratory (F01, F02)
D B Taylor, University of Edinburgh (F01, F02, F03, F04)

N Temme, CWI, Amsterdam (S)
G E Thomas, University of Nottingham (G01)
C P Thompson, AERE Harwell (D03)
G Tunnicliffe-Wilson, University of Lancaster (G13)
C W Überhuber, Technical University Vienna (D01)
P F Velleman, Cornell University (G01, G10)
J E Walsh, University of Manchester (D03, S)
H A Watts, Sandia National Laboratories, Albuquerque (D02)
P Wesseling, Delft University of Technology (D03)
J H Wilkinson, National Physical Laboratory (F01, F02, F03, F04)
D J Winstanley, University of Kent (E01)
M A Wong, Yale University (G03)
M H Wright, Stanford University and National Physical Laboratory (E04)
I Wynne-Jones, Imperial College of Science and Technology (C06)

Background to The Numerical Algorithms Group (NAG)

The Numerical Algorithms Group Ltd is a not-for-profit company which aims to provide products and services to meet the mathematical software and related needs of computer users especially scientists, engineers, planners and analysts. This document provides a general overview of the company and its products.

1 The Early Years

The NAG project began in 1970 as a collaborative venture between the Universities of Birmingham, Leeds, Manchester, Nottingham and Oxford, and the Atlas Computer Laboratory (now part of the Science and Engineering Research Council's Rutherford Appleton Laboratory). The original aim was to develop a library of numerical and statistical subroutines for use on their ICL 1906A/S machines. Because of the different language emphasis in the centres and the difficulty of mixed-language programming it was decided to develop the library in both Algol 60 and ANSI Fortran. The Mark 1 Library contained 98 user-callable routines and was released on 1st October 1971.

Universities with other types of computer became interested in the activity, and various machine-range implementations were initiated from Mark 2 of the Library onwards. Hence, the early aims of NAG could be summarised as follows:

(a) to create a balanced, general-purpose library of algorithms meeting the numerical and statistical needs of computer users;
(b) to support the Library with documentation giving advice on problem identification, algorithm selection, and routine usage, together with a comprehensive set of example programs;
(c) to provide a substantial test suite for certification of the Library;
(d) to implement the Library as widely as user demand required.

In adopting these objectives, NAG committed itself to a long-term programme of library contents development with a strong emphasis on documentation, testing and portability. This emphasis has been maintained throughout the subsequent years of NAG's development, and is reflected in the present-day range of NAG products.

Both the contribution activity and the implementation process were co-ordinated from Nottingham until August 1973. The central office of the project then moved to Oxford University, and at this point the name of the project changed from 'The Nottingham Algorithms Group' to 'The Numerical Algorithms Group'.

2 The Formation of NAG as a Company

Until mid-1975, the distribution of the Library was, for funding reasons, restricted to UK university computing sites and certain related installations. Since the Library and its supporting facilities appeared to be useful to scientific computer users generally, it was decided to make the Library more widely available. A Library Service based on Mark 5 was initiated in 1976.

To facilitate the provision of such a service, NAG became a not-for-profit company, limited by guarantee, in March 1976. This particular company structure was chosen to reflect NAG's continuing commitment to collaborative work in the development of software for the benefit of the scientific and research community. NAG has no shareholders or owners, but about 300 members voluntarily guarantee the company. All financial surpluses can thus be re-invested in resources to enhance development of existing and new products and services.

After the Company was formed, the nature of the resources within NAG continued as before, namely the close collaboration between full-time staff and a large number of specialists in numerical and statistical computing. Originally, these specialists were situated in university departments or government research institutions throughout the United Kingdom, but since then co-operation has been established with a much wider body of specialist expertise worldwide.

To promote the use of the NAG Library Service in North America, an associated company, The Numerical Algorithms Group (USA) Incorporated, was established in 1978. A second subsidiary company, The

Numerical Algorithms Group (Deutschland) GmbH, was created in 1990. The NAG IRIS Explorer Center Japan (IECJ) was set up in 1994 to provide a full range of product sales, services, training, consultancy and support for IRIS Explorer. NAG also has a number of distributors in different countries around the world.

3 Developing and Extending the Product Range

(a) Numerical and Statistical libraries

In keeping with its original objectives, NAG has developed its original product, the NAG Fortran Library, through a succession of major Marks (editions). Eighteen Marks have been released between 1970 and 1997, and Mark 18 contains over 1100 user-callable routines. This active campaign of development will continue both in terms of new contents and implementation on an ever-widening range of computer systems, from personal computers to supercomputers and special architecture systems.

These libraries have been prepared to use the appropriate current standard Fortran language, e.g. specific Fortran 77 constructs were introduced at Mark 12. The adoption of the Fortran 90 standard now provides further scope for enhanced library facilities. NAG has already implemented the Fortran Library for use with Fortran 90 compilers including the NAGWare f90 Compiler (see below). However, NAG has also developed NAG *fl*90, a new Fortran 90 Library which will take advantage of features of the new standard to provide greater flexibility and ease-of-use. The third release, containing the equivalent of about 450 Fortran (77) routines, will be completed in 1997.

NAG has also been developing further software for use in parallel computing environments. The NAG Parallel Library makes use of PVM (Parallel Virtual Machine) or MPI (Message Passing Interface) to allow high level routines to run on distributed memory systems or networks of workstations. For users wishing to exploit the power and scalability of shared memory systems (also referred to as symmetric multi-processor computers) the Fortran SMP Library is also now available.

Alternative language libraries have also continued to feature in the NAG product range. In 1973, NAG began to develop an Algol 68 Library, Mark 1 of which was released on ICL 1900 machines in March 1976, and for which a total of four Marks were prepared. These developments coincided with reduced interest in Algol 60; Mark 8 of the NAG Algol Library was made available in 1981, but no further Marks have been produced. A Pascal Library was released in 1986. A collaborative project for the creation of an Ada Library was also initiated, and Mark 2 of this library is now available.

Towards the end of the 1980s, a growing interest in the use of C for scientific computing led NAG to start the development of a C Library. This has continued, with Mark 4 (containing about 275 user-callable functions) now available.

At the same time, a number of specialised topic libraries have been developed jointly by NAG and other organisations. These provide facilities for solution of problems in Control Engineering, Sparse Matrices, Data Approximation and Finite Elements.

For some years NAG has supported the Fortran Library and Graphics Library with the provision of an On-line Information Supplement containing a reduced form of the library manuals. We have now developed a new range of electronic documentation products providing hypertext forms of the manuals supporting the libraries for Fortran, NAG *fl*90, Parallel, Fortran SMP and C. These TextWare products are available with browsers for UNIX and PC Windows systems.

(b) Statistical Packages

Because of NAG's expertise in software validation, porting, sales and support, third party software developers often enter into agreements with NAG, whereby the company undertakes the implementation and distribution of their software products. The first of these were the widely used statistical packages GLIM (developed by the Royal Statistical Society) and Genstat (developed by Rothamsted Experimental Station). Over the years, this collaboration has grown so that NAG and the developers share knowledge for the design of technical contents, graphical facilities, user interface, etc.

(c) Software Tools

Because of NAG's commitment to the production of high quality software which is robust and easily portable, the need for an extensive range of reliable software processing tools was recognised from the beginning. NAG developed a number of tools for its own use, and this activity widened into collaboration with the Toolpack project formed in 1978. This project was supported in the US by the National Science Foundation and the Department of Energy, and in the UK by the Science and Engineering Research Council. The first release of Fortran 77 processing tools was made available as a public domain distribution service in 1985, and a subsequent release was made in 1986.

Over the following years, NAG's use of these tools increased, and new versions were developed which were faster, more robust and able to cope with processing much larger source code. The portability base on which tools were produced was modified to ease implementation and support, and to provide a simpler model of the host environment. These technological developments were then made available to other Fortran programmers when the f77 Tool Suite was released for sale as the first product in the NAGWare range of software support systems, in 1991.

While always making use of a number of programming languages, NAG has recognised the special place of Fortran to serve the requirements of the scientific computing community. To further this support, NAG became closely involved in the discussions concerning future Fortran language standards, which culminated in the approval of the Fortran 90 standard. Within a very short period of time, NAG was able to release the NAGWare f90 compiler, which as a fully standard-conforming compiler gave the Fortran community the chance to take advantage of the new constructs very quickly. This compiler is available for both workstations and personal computers, and the next release (due in 1997) will provide full Fortran 95 support.

Our work on the compiler and the Fortran 90 library has naturally led to the development of a suite of tools for processing Fortran 90 code. These have also been released, as the NAGWare f90 Tool Suite. A toolset specifically for use when converting code from Fortran 77 to Fortran 90 is also available.

Another aspect of software processing has been the building of links between systems, particularly to provide the power and flexibility of NAG library software to users of other products. This has already led to the development, in collaboration with the National Physical Laboratory, of a system for generating gateways allowing Fortran subroutines to be called from MATLAB by simple MATLAB-style commands. This NAGWare Gateway Generator is now available.

(d) Visualisation and Graphics

Users of the NAG Fortran Library soon expressed a requirement to be able to produce plots of data or results from within their programs. NAG, therefore, developed a Graphics Library to provide this functionality while ensuring that the underlying numerical computations were sufficiently rigorous to give high-quality accurate plots, e.g. of contours, perspective surface views, etc. Mark 4 of this library includes facilities for production of 'publication quality' output.

While there is continuing demand for this type of graphics subroutine library, the requirement for sophisticated data visualisation systems has grown recently (along with the development of single-user workstations powerful enough to support such work). At the forefront of these products is IRIS Explorer, which was developed by Silicon Graphics Inc as both a data visualisation system and as a visual programming environment. In 1992 Silicon Graphics invited NAG to port IRIS Explorer to a range of other UNIX workstations; again this is evidence of the recognition of NAG's expertise in software portability. In 1995 NAG took over the development, porting and support of IRIS Explorer worldwide, and new releases (for both UNIX and Windows NT) continue.

(e) Symbolic Computing

AXIOM is another example of NAG accepting responsibility for material developed elsewhere. The IBM T J Watson Research Centre developed a powerful symbolic computing package, Scratchpad II, which provides a mathematical approach to problem solving, together with advanced graphical capabilities and hypertext documentation. A number of collaborators around the world had contributed to this work, and wished to continue to do so. NAG, with its extensive experience of co-ordinating such activities, was asked by IBM to develop Scratchpad II into the product AXIOM. Release 2 of AXIOM, including a new compiler and links to NAG numerical software, is now available for a range of UNIX workstations as well as Windows NT/95.

(f) Simulation

The NAG numerical libraries contain a number of routines for the solution of ordinary and partial differential equations, but NAG is now planning extensions to this functionality by the provision of appropriate software packages, e.g. for mesh generation. Fastflo, the first product in this new family, is a finite element package for the numerical solution of systems of PDEs in two or three dimensions; it was released in Summer 1997.

To complement this range of products NAG's expertise in mathematical computation and software engineering is available on a consultancy basis and has been used to great effect in a wide range of industries.

4 User Support

NAG continues to place considerable emphasis on all forms of user support. In addition to the comprehensive documentation supplied with each product, a number of other services are available. These include:

(a) Response Centre

This acts as a focal point for all enquiries about any aspect of NAG activities, as well as providing specific technical, servicing and sales support. It can be contacted by telephone, fax, e-mail, Internet and letter at NAG Ltd and NAG Inc. All queries are logged and monitored to ensure NAG can identify areas of concern or difficulty for our users.

(b) Publications

"Network" is a news-sheet distributed regularly to all supported customers to highlight new developments within NAG and topics of general interest. Newsletters for particular products are also produced, and Technical Reports on specific topics are available.

(c) Web site (http://www.nag.co.uk/ or http://www.nag.com/)

The NAG Web site includes product information and demonstrations, technical documentation, servicing information, and articles of general interest.

(d) Meetings

NAG organises a variety of seminars, workshops and training courses at suitable locations around the world. These may focus on a particular product, such as Genstat or the NAGWare f90 Compiler, or provide a more general overview of developments in scientific computing.

(e) User Groups

The NAG Users Association (NAGUA) is an independent organisation open to all users of NAG products and services. NAG provides support, mainly by giving presentations at meetings and seminars arranged by NAGUA. Similarly, NAG supports other groups where either hardware or software is the common factor amongst members.

As well as offering these facilities as a service to users, NAG values the feedback on our products which such contacts generate. This information, backed up by surveys on specific topics, e.g. documentation usage, is used to ensure NAG is aware of changing user requirements and can respond accordingly.

For further information about NAG and its existing or planned products, please do not hesitate to contact us.

Keywords in Context for the NAG Fortran 77 Library

Single 1-D complex discrete Fourier transform, extra workspace...	C06FCF
Single 1-D complex discrete Fourier transform, no extra...	C06ECF
1-D complex discrete Fourier transform of...	C06FFF
Multiple 1-D complex discrete Fourier transforms	C06FRF
1-D Gaussian quadrature	D01BAF
Single 1-D Hermitian discrete Fourier transform, extra...	C06FBF
Single 1-D Hermitian discrete Fourier transform, no extra...	C06EBF
Multiple 1-D Hermitian discrete Fourier transforms	C06FQF
1-D quadrature, adaptive, finite interval, allowing for...	D01ALF
1-D quadrature, adaptive, finite interval, method...	D01AKF
1-D quadrature, adaptive, finite interval, strategy due...	D01AHF
1-D quadrature, adaptive, finite interval, strategy due...	D01AJF
1-D quadrature, adaptive, finite interval, variant of...	D01ATF
1-D quadrature, adaptive, finite interval, variant of...	D01AUF
1-D quadrature, adaptive, finite interval, weight...	D01AQF
1-D quadrature, adaptive, finite interval, weight...	D01ANF
1-D quadrature, adaptive, finite interval, weight...	D01APF
1-D quadrature, adaptive, infinite or semi-infinite...	D01AMF
1-D quadrature, adaptive, semi-infinite interval, weight...	D01ASF
1-D quadrature, integration of function defined by data...	D01GAF
1-D quadrature, non-adaptive, finite interval	D01BDF
1-D quadrature, non-adaptive, finite interval with...	D01ARF
Single 1-D real discrete Fourier transform, extra workspace for...	C06FAF
Single 1-D real discrete Fourier transform, no extra workspace	C06EAF
Multiple 1-D real discrete Fourier transforms	C06FPF
Elliptic PDE, Laplace's equation, 2-D arbitrary domain	D03EAF
2-D complex discrete Fourier transform	C06FUF
Sort 2-D data into panels for fitting bicubic splines	E02ZAF
...of finite difference equations by SIP, five-point 2-D molecule, iterate to convergence	D03EBF
...of finite difference equations by SIP, five-point 2-D molecule, one iteration	D03UAF
2-D quadrature, finite region	D01DAF
Elliptic PDE, Helmholtz equation, 3-D Cartesian co-ordinates	D03FAF
3-D complex discrete Fourier transform	C06FXF
...of finite difference equations by SIP, seven-point 3-D molecule, iterate to convergence	D03ECF
...of finite difference equations by SIP, seven-point 3-D molecule, one iteration	D03UBF
Nonlinear convolution Volterra–Abel equation, 1st kind, weakly singular	D05BEF
Nonlinear convolution Volterra–Abel equation, 2nd kind, weakly singular	D05BDF
Generate weights for use in solving weakly singular Abel type equations	D05BYF
Calculation of weights and abscissae for Gaussian quadrature rules, general choice...	D01BCF
Pre-computed weights and abscissae for Gaussian quadrature rules, restricted...	D01BBF
Robust estimation, median, median absolute deviation, robust standard deviation	G07DAF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex band matrix	F06UBF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex general matrix	F06UAF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex Hermitian band matrix	F06UEF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex Hermitian matrix	F06UCF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex Hermitian matrix, packed...	F06UDF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex Hessenberg matrix	F06UMF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex symmetric band matrix	F06UHF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex symmetric matrix	F06UFF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex symmetric matrix, packed...	F06UGF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex trapezoidal/triangular matrix	F06UJF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex triangular band matrix	F06ULF
1-norm, ∞-norm, Frobenius norm, largest absolute element, complex triangular matrix, packed...	F06UKF
1-norm, ∞-norm, Frobenius norm, largest absolute element, real band matrix	F06RBF
1-norm, ∞-norm, Frobenius norm, largest absolute element, real general matrix	F06RAF
1-norm, ∞-norm, Frobenius norm, largest absolute element, real Hessenberg matrix	F06RMF
1-norm, ∞-norm, Frobenius norm, largest absolute element, real symmetric band matrix	F06REF
1-norm, ∞-norm, Frobenius norm, largest absolute element, real symmetric matrix	F06RCF
1-norm, ∞-norm, Frobenius norm, largest absolute element, real symmetric matrix, packed storage	F06RDF
1-norm, ∞-norm, Frobenius norm, largest absolute element, real trapezoidal/triangular matrix	F06RJF
1-norm, ∞-norm, Frobenius norm, largest absolute element, real triangular band matrix	F06RLF
1-norm, ∞-norm, Frobenius norm, largest absolute element, real triangular matrix, packed...	F06RKF
Elements of real vector with largest and smallest absolute value	F06FLF
Index, complex vector element with largest absolute value (ICAMAX/IZAMAX)	F06JMF
Index, real vector element with largest absolute value (ISAMAX/IDAMAX)	F06JLF
Sum the absolute values of complex vector elements...	F06JKF
Sum the absolute values of real vector elements (SASUM/DASUM)	F06EKF
Acceleration of convergence of sequence, Shanks'...	C06BAF
Normal scores, accurate values	G01DAF
ODEs, IVP, Adams method, until function of solution is zero,...	D02CJF
ODEs, IVP, Adams method with root-finding (forward communication,...	D02QFF
ODEs, IVP, Adams method with root-finding (reverse communication,...	D02QGF
1-D quadrature, adaptive, finite interval, allowing for singularities...	D01ALF
1-D quadrature, non-adaptive, finite interval	D01BDF
1-D quadrature, adaptive, finite interval, method suitable for...	D01AKF
1-D quadrature, adaptive, finite interval, strategy due to Patterson,...	D01AHF
1-D quadrature, adaptive, finite interval, strategy due to Piessens and...	D01AJF
1-D quadrature, adaptive, finite interval, variant of D01AJF efficient...	D01ATF
1-D quadrature, adaptive, finite interval, variant of D01AKF efficient...	D01AUF
1-D quadrature, adaptive, finite interval, weight function $1/(x-c)$,...	D01AQF
1-D quadrature, adaptive, finite interval, weight function $\cos(\omega$...	D01ANF
1-D quadrature, adaptive, finite interval, weight function with...	D01APF
1-D quadrature, non-adaptive, finite interval with provision for indefinite...	D01ARF
1-D quadrature, adaptive, infinite or semi-infinite interval	D01AMF
Multi-dimensional adaptive quadrature over hyper-rectangle	D01FCF
Multi-dimensional adaptive quadrature over hyper-rectangle, multiple...	D01EAF
1-D quadrature, adaptive, semi-infinite interval, weight function...	D01ASF
Add a new variable to a general linear regression model	G02DEF
Add a scalar times a complex sparse vector to another...	F06GTF
Add a scalar times a real sparse vector to another real...	F06ETF
Add scalar times complex vector to complex vector...	F06GCF
Add scalar times real vector to real vector...	F06ECF
Add/delete an observation to/from a general linear...	G02DCF
Real inner product added to initial value, basic/additional precision	X03AAF
Complex inner product added to initial value, basic/additional precision	X03ABF

GAMS Index for the NAG Fortran 77 Library

This index classifies NAG Fortran 77 Library routines according to Version 2 of the GAMS classification scheme described in [1]. Note that only those GAMS classes which contain Library routines, either directly or in a subclass, are included below.

A Arithmetic, error analysis
A3 Real
A3a Standard precision
F06BLF Compute quotient of two real scalars, with overflow flag
A4 Complex
A4a Standard precision
A02ABF Modulus of a complex number
A02ACF Quotient of two complex numbers
F06CLF Compute quotient of two complex scalars, with overflow flag
A7 Sequences (e.g., convergence acceleration)
C06BAF Acceleration of convergence of sequence, Shanks' transformation and epsilon algorithm
C Elementary and special functions (*search also class L5*)
C1 Integer-valued functions (e.g., factorial, binomial coefficient, permutations, combinations, floor, ceiling)
C10 Bessel functions
C10a J, Y, H_1, H_2
C10a1 Real argument, integer order
S17ACF Bessel function $Y_0(x)$
S17ADF Bessel function $Y_1(x)$
S17AEF Bessel function $J_0(x)$
S17AFF Bessel function $J_1(x)$
C10a4 Complex argument, real order
S17DCF Bessel functions $Y_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
S17DEF Bessel functions $J_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
S17DLF Hankel functions $H^{(j)}_{\nu+a}(z)$, $j = 1, 2$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
C10b I, K
C10b1 Real argument, integer order
S18ACF Modified Bessel function $K_0(x)$
S18ADF Modified Bessel function $K_1(x)$
S18AEF Modified Bessel function $I_0(x)$
S18AFF Modified Bessel function $I_1(x)$
S18CCF Modified Bessel function $e^x K_0(x)$
S18CDF Modified Bessel function $e^x K_1(x)$
S18CEF Modified Bessel function $e^{-|x|} I_0(x)$
S18CFF Modified Bessel function $e^{-|x|} I_1(x)$
C10b4 Complex argument, real order
S18DCF Modified Bessel functions $K_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
S18DEF Modified Bessel functions $I_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0, 1, 2, \ldots$
C10c Kelvin functions
S19AAF Kelvin function ber x
S19ABF Kelvin function bei x
S19ACF Kelvin function ker x
S19ADF Kelvin function kei x
C10d Airy and Scorer functions
S17AGF Airy function $\mathrm{Ai}(x)$
S17AHF Airy function $\mathrm{Bi}(x)$
S17AJF Airy function $\mathrm{Ai}'(x)$
S17AKF Airy function $\mathrm{Bi}'(x)$
S17DGF Airy functions $\mathrm{Ai}(z)$ and $\mathrm{Ai}'(z)$, complex z
S17DHF Airy functions $\mathrm{Bi}(z)$ and $\mathrm{Bi}'(z)$, complex z
C13 Jacobian elliptic functions, theta functions
S21CAF Jacobian elliptic functions sn, cn and dn
C14 Elliptic integrals
S21BAF Degenerate symmetrised elliptic integral of 1st kind $R_C(x, y)$
S21BBF Symmetrised elliptic integral of 1st kind $R_F(x, y, z)$
S21BCF Symmetrised elliptic integral of 2nd kind $R_D(x, y, z)$
S21BDF Symmetrised elliptic integral of 3rd kind $R_J(x, y, z, r)$
C2 Powers, roots, reciprocals
A02AAF Square root of a complex number
C3 Polynomials
C3a Orthogonal
C3a2 Chebyshev, Legendre

References

[1] Boisvert R F, Howe S E and Kahaner D K (1990) The guide to available mathematical software problem classification scheme. *Report NISTIR 4475* Applied and Computational Mathematics Division, National Institute of Standards and Technology.

[2] Boisvert R F, Howe S E and Kahaner D K (1985) GAMS — a framework for the management of scientific software. *ACM Trans. Math. Software* **11** 313–355.

[3] Boisvert R F (1989) The guide to available mathematical software advisory system. *Math. Comput. Simul.* **31** 453–464.

Implementation-specific Details for Users of the NAG Fortran Library

The NAG Fortran Library is available in a number of different implementations, each certified under a particular computing system; the NAG Fortran Library Manual is generally applicable to all of them. Any information that applies solely to a specific implementation (e.g. the IBM 360/370 Fortran Double Precision Implementation) is provided in printed form and in a Users' Note file on the Library Release Tape for that implementation; i.e. the information is distributed in machine-readable form to installations which use that implementation.

Your installation must make that information available, either by giving you access to the Users' Note file via the computing system or by including the information in local user documentation. In either case, we strongly recommend that you obtain a copy of the information and place it behind the tabbed divider provided in your NAG Fortran Library Manual. Please ensure that the information is up-to-date; if the note relates to your implementation but to a previous Mark please discard it (see the Contents at the front of Volume 1 of the Manual for the current Mark).

Chapter A02 – Complex Arithmetic

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
A02AAF	2	Square root of a complex number
A02ABF	2	Modulus of a complex number
A02ACF	2	Quotient of two complex numbers

Chapter A02

Complex Arithmetic

Contents

1 Scope of the Chapter

This chapter provides facilities for arithmetic operations involving complex numbers.

2 Background to the Problems

Of the several representations used for complex numbers, perhaps the most common is $a + ib$, where a and b are real numbers, and i represents the **imaginary** number $\sqrt{-1}$. The number a is the **real part**, and ib the **imaginary part**.

For the basic arithmetic operations of addition, subtraction and multiplication, the inclusion of routines was not considered worthwhile. Their coding would be short and no special techniques need be used.

In complex number operations of a more complicated nature, special precautions may have to be taken to avoid unnecessary overflow and underflow at intermediate stages of the computation. This has led to the inclusion of routines in this chapter.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

The routines were originally written for use by NAG Library routines which compute eigensystems of real and complex matrices (Chapter F02). They may, however, be of general use to programmers using complex numbers.

Fortran programmers may prefer to use the COMPLEX facilities in that language rather than carrying the real and imaginary parts of the numbers in different variables.

4 Index

A02AAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

A02AAF evaluates the square root of the complex number $x = (x_r, x_i)$.

2. Specification

```
SUBROUTINE A02AAF (XR, XI, YR, YI) '
real             XR, XI, YR, YI
```

3. Description

The method of evaluating $y = \sqrt{x}$ depends on the value of x_r.

For $x_r \geq 0$,

$$y_r = \sqrt{\frac{x_r + \sqrt{x_r^2 + x_i^2}}{2}}, \qquad y_i = \frac{x_i}{2y_r}.$$

For $x_r < 0$,

$$y_i = \text{sign}(x_i) \times \sqrt{\frac{|x_r| + \sqrt{x_r^2 + x_i^2}}{2}}, \qquad y_r = \frac{x_i}{2y_i}.$$

Overflow is avoided when squaring x_i and x_r by calling A02ABF to evaluate $\sqrt{x_r^2 + x_i^2}$.

4. References

[1] WILKINSON, J.H. and REINSCH, C.
Handbook for Automatic Computation, (Vol. II, Linear Algebra).
Springer-Verlag, pp. 357-358, 1971.

5. Parameters

1: XR – ***real***. *Input*
2: XI – ***real***. *Input*

On entry: x_r and x_i, the real and imaginary parts of x, respectively.

3: YR – ***real***. *Output*
4: YI – ***real***. *Output*

On exit: y_r and y_i, the real and imaginary parts of y, respectively.

6. Error Indicators and Warnings

None.

7. Accuracy

The result should be correct to ***machine precision***.

8. Further Comments

The time taken by the routine is negligible.

9. Example

To find the square root of $-1.7 + 2.6i$.

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     A02AAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             XI, XR, YI, YR
*     .. External Subroutines ..
      EXTERNAL         A02AAF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'A02AAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) XR, XI
*
      CALL A02AAF(XR,XI,YR,YI)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '   XR     XI       YR       YI'
      WRITE (NOUT,99999) XR, XI, YR, YI
      STOP
*
99999 FORMAT (1X,2F6.1,2F9.4)
      END
```

9.2. Program Data

```
A02AAF Example Program Data
 -1.7 2.6
```

9.3. Program Results

```
 A02AAF Example Program Results

    XR     XI       YR       YI
  -1.7    2.6   0.8386   1.5502
```

A02ABF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

A02ABF returns the value of the modulus of the complex number $x = (x_r, x_i)$.

2. Specification

```
real FUNCTION A02ABF (XR, XI)
real             XR, XI
```

3. Description

The function evaluates $\sqrt{x_r^2+x_i^2}$ by using $a\sqrt{1+\left(\frac{b}{a}\right)^2}$ where a is the larger of x_r and x_i, and b is the smaller of x_r and x_i. This ensures against unnecessary overflow and loss of accuracy when calculating $(x_r^2+x_i^2)$.

4. References

[1] WILKINSON, J.H. and REINSCH, C.
Handbook for Automatic Computation, (Vol. II, Linear Algebra).
Springer-Verlag, pp. 357-358, 1971.

5. Parameters

1: XR – ***real***. *Input*

2: XI – ***real***. *Input*

On entry: x_r and x_i, the real and imaginary parts of x, respectively.

6. Error Indicators and Warnings

None.

7. Accuracy

The result should be correct to ***machine precision***.

8. Further Comments

None.

9. Example

To find the modulus of $-1.7+2.6i$.

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     A02ABF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             XI, XR, Y
```

```
*     .. External Functions ..
      real               A02ABF
      EXTERNAL           A02ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'A02ABF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) XR, XI
      Y = A02ABF(XR,XI)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '   XR     XI        Y'
      WRITE (NOUT,99999) XR, XI, Y
      STOP
*
99999 FORMAT (1X,2F6.1,F9.4)
      END
```

9.2. Program Data

```
A02ABF Example Program Data
 -1.7 2.6
```

9.3. Program Results

```
 A02ABF Example Program Results

    XR     XI        Y
  -1.7    2.6    3.1064
```

A02ACF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

A02ACF divides one complex number, $x = (x_r,x_i)$, by a second complex number, $y = (y_r,y_i)$, returning the result in $z = (z_r,z_i)$.

2. Specification

```
SUBROUTINE A02ACF (XR, XI, YR, YI, ZR, ZI)
real             XR, XI, YR, YI, ZR, ZI
```

3. Description

$z = \dfrac{x}{y}$ is calculated using the following formulae:

If $|y_r| > |y_i|$,

$$z_r = \frac{x_r + \theta x_i}{\theta y_i + y_r}, \qquad z_i = \frac{x_i - \theta x_r}{\theta y_i + y_r} \qquad \text{where } \theta = \frac{y_i}{y_r}$$

If $|y_r| \leq |y_i|$,

$$z_r = \frac{\phi x_r + x_i}{\phi y_r + y_i}, \qquad z_i = \frac{\phi x_i - x_r}{\phi y_r + y_i} \qquad \text{where } \phi = \frac{y_r}{y_i}$$

These formulae ensure that no unnecessary overflow or underflow occurs at intermediate stages of the computation.

4. References

[1] WILKINSON, J.H. and REINSCH, C.
Handbook for Automatic Computation, (Vol. II, Linear Algebra).
Springer-Verlag, pp. 357-358, 1971.

5. Parameters

1: XR – ***real***. *Input*
2: XI – ***real***. *Input*

On entry: x_r and x_i, the real and imaginary parts of x, respectively.

3: YR – ***real***. *Input*
4: YI – ***real***. *Input*

On entry: y_r and y_i, the real and imaginary parts of y, respectively.

5: ZR – ***real***. *Output*
6: ZI – ***real***. *Output*

On exit: z_r and z_i, the real and imaginary parts of z, respectively.

6. Error Indicators and Warnings

None.

7. Accuracy

The result should be correct to ***machine precision***.

8. Further Comments

The time taken by the routine is negligible.

This routine **must** not be called with YR = 0.0 and YI = 0.0.

9. Example

To find the value of $(-1.7+2.6i)/(-3.1-0.9i)$.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     A02ACF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             XI, XR, YI, YR, ZI, ZR
*     .. External Subroutines ..
      EXTERNAL         A02ACF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'A02ACF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) XR, XI, YR, YI
*
      CALL A02ACF(XR,XI,YR,YI,ZR,ZI)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    XR     XI     YR     YI       ZR       ZI'
      WRITE (NOUT,99999) XR, XI, YR, YI, ZR, ZI
      STOP
*
99999 FORMAT (1X,4F6.1,2F9.4)
      END
```

9.2. Program Data

```
A02ACF Example Program Data
 -1.7  2.6 -3.1 -0.9
```

9.3. Program Results

```
 A02ACF Example Program Results

    XR     XI     YR     YI       ZR       ZI
  -1.7    2.6   -3.1   -0.9   0.2812  -0.9203
```

Chapter C02 – Zeros of Polynomials

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
C02AFF	14	All zeros of complex polynomial, modified Laguerre method
C02AGF	13	All zeros of real polynomial, modified Laguerre method
C02AHF	14	All zeros of complex quadratic
C02AJF	14	All zeros of real quadratic

Chapter C02

Zeros of Polynomials

Contents

1 Scope of the Chapter

This chapter is concerned with computing the zeros of a polynomial with real or complex coefficients.

2 Background to the Problems

Let $f(z)$ be a polynomial of degree n with complex coefficients a_i:

$$f(z) \equiv a_0 z^n + a_1 z^{n-1} + a_2 z^{n-2} + \ldots + a_{n-1} z + a_n, \quad a_0 \neq 0.$$

A complex number z_1 is called a **zero** of $f(z)$ (or equivalently a **root** of the **equation** $f(z) = 0$), if:

$$f(z_1) = 0.$$

If z_1 is a zero, then $f(z)$ can be divided by a factor $(z - z_1)$:

$$f(z) = (z - z_1) f_1(z) \tag{1}$$

where $f_1(z)$ is a polynomial of degree $n-1$. By the Fundamental Theorem of Algebra, a polynomial $f(z)$ always has a zero, and so the process of dividing out factors $(z - z_i)$ can be continued until we have a complete **factorization** of $f(z)$:

$$f(z) \equiv a_0 (z - z_1)(z - z_2) \ldots (z - z_n).$$

Here the complex numbers $z_1, z_2, \ldots, z_n$ are the zeros of $f(z)$; they may not all be distinct, so it is sometimes more convenient to write:

$$f(z) \equiv a_0 (z - z_1)^{m_1} (z - z_2)^{m_2} \ldots (z - z_k)^{m_k}, \quad k \leq n,$$

with distinct zeros $z_1, z_2, \ldots, z_k$ and multiplicities $m_i \geq 1$. If $m_i = 1$, z_i is called a **simple** or **isolated** zero; if $m_i > 1$, z_i is called a **multiple** or **repeated** zero; a multiple zero is also a zero of the derivative of $f(z)$.

If the coefficients of $f(z)$ are all real, then the zeros of $f(z)$ are either real or else occur as pairs of conjugate complex numbers $x + iy$ and $x - iy$. A pair of complex conjugate zeros are the zeros of a quadratic factor of $f(z)$, $(z^2 + rz + s)$, with real coefficients r and s.

Mathematicians are accustomed to thinking of polynomials as pleasantly simple functions to work with. However the problem of numerically **computing** the zeros of an arbitrary polynomial is far from simple. A great variety of algorithms have been proposed, of which a number have been widely used in practice; for a fairly comprehensive survey, see Householder [1]. All general algorithms are iterative. Most converge to one zero at a time; the corresponding factor can then be divided out as in equation (1) above – this process is called **deflation** or, loosely, dividing out the zero – and the algorithm can be applied again to the polynomial $f_1(z)$. A pair of complex conjugate zeros can be divided out together – this corresponds to dividing $f(z)$ by a quadratic factor.

Whatever the theoretical basis of the algorithm, a number of practical problems arise: for a thorough discussion of some of them see Peters and Wilkinson [2] and Wilkinson [3], Chapter 2. The most elementary point is that, even if z_1 is mathematically an exact zero of $f(z)$, because of the fundamental limitations of computer arithmetic the **computed** value of $f(z_1)$ will not necessarily be exactly 0.0. In practice there is usually a small region of values of z about the exact zero at which the computed value of $f(z)$ becomes swamped by rounding errors. Moreover in many algorithms this inaccuracy in the computed value of $f(z)$ results in a similar inaccuracy in the computed step from one iterate to the next. This limits the precision with which any zero can be computed. Deflation is another potential cause of trouble, since, in the notation of equation (1), the computed coefficients of $f_1(z)$ will not be completely accurate, especially if z_1 is not an exact zero of $f(z)$; so the zeros of the computed $f_1(z)$ will deviate from the zeros of $f(z)$.

A zero is called **ill-conditioned** if it is sensitive to small changes in the coefficients of the polynomial. An ill-conditioned zero is likewise sensitive to the computational inaccuracies just mentioned. Conversely a zero is called **well-conditioned** if it is comparatively insensitive to such perturbations. Roughly speaking a zero which is well separated from other zeros is well-conditioned, while zeros which are close together are ill-conditioned, but in talking about 'closeness' the decisive factor is not the absolute distance between neighbouring zeros but their **ratio**: if the ratio is close to one the zeros are ill-conditioned. In particular, multiple zeros are ill-conditioned. A multiple zero is usually split into a cluster of zeros by perturbations in the polynomial or computational inaccuracies.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

3.1 Discussion

Four routines are available: C02AFF for polynomials with complex coefficients, C02AGF for polynomials with real coefficients, C02AHF for quadratic equations with complex coefficients and C02AJF for quadratic equations with real coefficients.

C02AFF and C02AGF both use a variant of Laguerre's Method to calculate each zero until the degree of the deflated polynomial is less than three, whereupon the remaining zeros are obtained by carefully evaluating the 'standard' closed formulae for a quadratic or linear equation.

For the solution of quadratic equations, C02AHF and C02AJF are simplified versions of the above routines.

The accuracy of the roots will depend on how ill-conditioned they are. Peters and Wilkinson [2] describe techniques for estimating the errors in the zeros after they have been computed.

4 Index

5 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have been withdrawn. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

C02ADF C02AEF

6 References

[1] Householder A S (1970) *The Numerical Treatment of a Single Nonlinear Equation* McGraw-Hill

[2] Peters G and Wilkinson J H (1971) Practical problems arising in the solution of polynomial equations *J. Inst. Maths. Applics.* **8** 16–35

[3] Wilkinson J H (1963) *Rounding Errors in Algebraic Processes* HMSO

C02AFF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C02AFF finds all the roots of a complex polynomial equation, using a variant of Laguerre's Method.

2. Specification

```
SUBROUTINE C02AFF (A, N, SCALE, Z, W, IFAIL)
INTEGER          N, IFAIL
real             A(2,N+1), Z(2,N), W(4*(N+1))
LOGICAL          SCALE
```

3. Description

The routine attempts to find all the roots of the nth degree complex polynomial equation

$$P(z) = a_0z^n + a_1z^{n-1} + a_2z^{n-2} + \ldots + a_{n-1}z + a_n = 0.$$

The roots are located using a modified form of Laguerre's Method, originally proposed by Smith [2].

The method of Laguerre [3] can be described by the iterative scheme

$$L(z_k) = z_{k+1} - z_k = \frac{-n \times P(z_k)}{P'(z_k) \pm \sqrt{H(z_k)}},$$

where $H(z_k) = (n-1)\times[(n-1)\times(P'(z_k))^2 - n\times P(z_k)P''(z_k)]$, and z_0 is specified.

The sign in the denominator is chosen so that the modulus of the Laguerre step at z_k, viz. $|L(z_k)|$, is as small as possible. The method can be shown to be cubically convergent for isolated roots (real or complex) and linearly convergent for multiple roots.

The routine generates a sequence of iterates $z_1, z_2, z_3, \ldots$, such that $|P(z_{k+1})| < |P(z_k)|$ and ensures that $z_{k+1} + L(z_{k+1})$ 'roughly' lies inside a circular region of radius $|F|$ about z_k known to contain a zero of $P(z)$; that is, $|L(z_{k+1})| \leq |F|$, where F denotes the Féjer bound (see Marden [1]) at the point z_k. Following Smith [2], F is taken to be $\min(B, 1.445\times n\times R)$, where B is an upper bound for the magnitude of the smallest zero given by

$$B = 1.0001\times\min\left(\sqrt{n}\times L(z_k), |r_1|, |a_n/a_0|^{1/n}\right),$$

r_1 is the zero X of smaller magnitude of the quadratic equation

$$(P''(z_k)/(2\times n\times(n-1)))X^2 + (P'(z_k)/n)X + \tfrac{1}{2}P(z_k) = 0$$

and the Cauchy lower bound R for the smallest zero is computed (using Newton's Method) as the positive root of the polynomial equation

$$|a_0|z^n + |a_1|z^{n-1} + |a_2|z^{n-2} + \ldots + |a_{n-1}|z - |a_n| = 0.$$

Starting from the origin, successive iterates are generated according to the rule $z_{k+1} = z_k + L(z_k)$ for $k = 1,2,3,\ldots$ and $L(z_k)$ is 'adjusted' so that $|P(z_{k+1})| < |P(z_k)|$ and $|L(z_{k+1})| \leq |F|$. The iterative procedure terminates if $P(z_{k+1})$ is smaller in absolute value than the bound on the rounding error in $P(z_{k+1})$ and the current iterate $z_p = z_{k+1}$ is taken to be a zero of $P(z)$. The deflated polynomial $\tilde{P}(z) = P(z)/(z-z_p)$ of degree $n-1$ is then formed, and the above procedure is repeated on the deflated polynomial until $n < 3$, whereupon the remaining roots are obtained via the 'standard' closed formulae for a linear ($n = 1$) or quadratic ($n = 2$) equation.

To obtain the roots of a quadratic polynomial, C02AHF can be used.

4. References

[1] MARDEN, M.
Geometry of Polynomials. Mathematical Surveys.
Am. Math. Soc., Providence, Rhode Island, USA, 3, 1966.

[2] SMITH, B.T.
ZERPOL: A Zero Finding Algorithm for Polynomials Using Laguerre's Method.
Technical Report, Department of Computer Science, University of Toronto, Canada, 1967.

[3] WILKINSON, J.H.
The Algebraic Eigenvalue Problem.
Clarendon Press, 1965.

5. Parameters

1: A(2,N+1) – ***real*** array. *Input*

On entry: if A is declared with bounds (2,0:N), then A(1,i) and A(2,i) must contain the real and imaginary parts of a_i (i.e. the coefficient of z^{n-i}), for $i = 0,1,...,n$.

Constraint: A(1,0) $\neq$ 0.0 or A(2,0) $\neq$ 0.0.

2: N – INTEGER. *Input*

On entry: the degree of the polynomial, n.

Constraint: N $\geq$ 1.

3: SCALE – LOGICAL. *Input*

On entry: indicates whether or not the polynomial is to be scaled. See Section 8 for advice on when it may be preferable to set SCALE = .FALSE. and for a description of the scaling strategy.

Suggested value: SCALE = .TRUE..

4: Z(2,N) – ***real*** array. *Output*

On exit: the real and imaginary parts of the roots are stored in Z(1,i) and Z(2,i) respectively, for $i = 1,2,...,n$.

5: W(4*(N+1)) – ***real*** array. *Workspace*

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, A(1,0) = 0.0 and A(2,0) = 0.0,
or N < 1.

IFAIL = 2

The iterative procedure has failed to converge. This error is very unlikely to occur. If it does, please contact NAG immediately, as some basic assumption for the arithmetic has been violated. See also Section 8.

IFAIL = 3

Either overflow or underflow prevents the evaluation of $P(z)$ near some of its zeros. This error is very unlikely to occur. If it does, please contact NAG immediately. See also Section 8.

7. Accuracy

All roots are evaluated as accurately as possible, but because of the inherent nature of the problem complete accuracy cannot be guaranteed.

8. Further Comments

If SCALE = .TRUE., then a scaling factor for the coefficients is chosen as a power of the base B of the machine so that the largest coefficient in magnitude approaches THRESH = B^{EMAX-P}. Users should note that no scaling is performed if the largest coefficient in magnitude exceeds THRESH, even if SCALE = .TRUE.. (For definition of B, $EMAX$ and P see the Chapter Introduction X02.)

However, with SCALE = .TRUE., overflow may be encountered when the input coefficients $a_0,a_1,a_2,...,a_n$ vary widely in magnitude, particularly on those machines for which $B^{(4\times P)}$ overflows. In such cases, SCALE should be set to .FALSE. and the coefficients scaled so that the largest coefficient in magnitude does not exceed $B^{(EMAX-2\times P)}$.

Even so, the scaling strategy used in C02AFF is sometimes insufficient to avoid overflow and/or underflow conditions. In such cases, the user is recommended to scale the independent variable (z) so that the disparity between the largest and smallest coefficient in magnitude is reduced. That is, use the routine to locate the zeros of the polynomial $d\times P(cz)$ for some suitable values of c and d. For example, if the original polynomial was $P(z) = 2^{-100}i + 2^{100}z^{20}$, then choosing $c = 2^{-10}$ and $d = 2^{100}$, for instance, would yield the scaled polynomial $i + z^{20}$, which is well-behaved relative to overflow and underflow and has zeros which are 2^{10} times those of $P(z)$.

If the routine fails with IFAIL = 2 or 3, then the real and imaginary parts of any roots obtained before the failure occurred are stored in Z in the reverse order in which they were found. Let n_R denote the number of roots found before the failure occurred. Then Z(1,n) and Z(2,n) contain the real and imaginary parts of the 1st root found, Z(1,$n-1$) and Z(2,$n-1$) contain the real and imaginary parts of the 2nd root found, ..., Z(1,n_R) and Z(2,n_R) contain the real and imaginary parts of the n_Rth root found. After the failure has occurred, the remaining $2\times(n-n_R)$ elements of Z contain a large negative number (equal to $-1/(\text{X02AMF}().\sqrt{2})$).

9. Example

To find the roots of the polynomial $a_0z^5 + a_1z^4 + a_2z^3 + a_3z^2 + a_4z + a_5 = 0$, where $a_0 = (5.0+6.0i)$, $a_1 = (30.0+20.0i)$, $a_2 = -(0.2+6.0i)$, $a_3 = (50.0+100000.0i)$, $a_4 = -(2.0-40.0i)$ and $a_5 = (10.0+1.0i)$.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C02AFF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MAXDEG
      PARAMETER        (MAXDEG=100)
      LOGICAL          SCALE
      PARAMETER        (SCALE=.TRUE.)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, N
```

```
*     .. Local Arrays ..
      real             A(2,0:MAXDEG), W(4*MAXDEG+4), Z(2,MAXDEG)
*     .. External Subroutines ..
      EXTERNAL         C02AFF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C02AFF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GT.0 .AND. N.LE.MAXDEG) THEN
         READ (NIN,*) (A(1,I),A(2,I),I=0,N)
         IFAIL = 0
*
         CALL C02AFF(A,N,SCALE,Z,W,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Degree of polynomial = ', N
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Roots of polynomial'
         WRITE (NOUT,*)
         DO 20 I = 1, N
            WRITE (NOUT,99998) 'z = ', Z(1,I), Z(2,I), '*i'
   20    CONTINUE
      ELSE
         WRITE (NOUT,*) 'N is out of range'
      END IF
      STOP
*
99999 FORMAT (1X,A,I4)
99998 FORMAT (1X,A,1P,e12.4,SP,e14.4,A)
      END
```

9.2. Program Data

```
C02AFF Example Program Data
 5
    5.0       6.0
   30.0      20.0
   -0.2      -6.0
   50.0  100000.0
   -2.0      40.0
   10.0       1.0
```

9.3. Program Results

```
 C02AFF Example Program Results

 Degree of polynomial =    5

 Roots of polynomial

 z =  -2.4328E+01   -4.8555E+00*i
 z =   5.2487E+00   +2.2736E+01*i
 z =   1.4653E+01   -1.6569E+01*i
 z =  -6.9264E-03   -7.4434E-03*i
 z =   6.5264E-03   +7.4232E-03*i
```

C02AGF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C02AGF finds all the roots of a real polynomial equation, using a variant of Laguerre's Method.

2. Specification

```
SUBROUTINE C02AGF (A, N, SCALE, Z, W, IFAIL)
INTEGER          N, IFAIL
real             A(N+1), Z(2,N), W(2*(N+1))
LOGICAL          SCALE
```

3. Description

The routine attempts to find all the roots of the nth degree real polynomial equation

$$P(z) = a_0 z^n + a_1 z^{n-1} + a_2 z^{n-2} + \ldots + a_{n-1} z + a_n = 0.$$

The roots are located using a modified form of Laguerre's Method, originally proposed by Smith [2].

The method of Laguerre [3] can be described by the iterative scheme

$$L(z_k) = z_{k+1} - z_k = \frac{-n \times P(z_k)}{P'(z_k) \pm \sqrt{H(z_k)}},$$

where $H(z_k) = (n-1)\times[(n-1)\times(P'(z_k))^2 - n\times P(z_k)P''(z_k)]$, and z_0 is specified.

The sign in the denominator is chosen so that the modulus of the Laguerre step at z_k, viz. $|L(z_k)|$, is as small as possible. The method can be shown to be cubically convergent for isolated roots (real or complex) and linearly convergent for multiple roots.

The routine generates a sequence of iterates $z_1, z_2, z_3, \ldots$, such that $|P(z_{k+1})| < |P(z_k)|$ and ensures that $z_{k+1} + L(z_{k+1})$ 'roughly' lies inside a circular region of radius $|F|$ about z_k known to contain a zero of $P(z)$; that is, $|L(z_{k+1})| \leq |F|$, where F denotes the Fejér bound (see Marden [1]) at the point z_k. Following Smith [2], F is taken to be $\min(B, 1.445\times n\times R)$, where B is an upper bound for the magnitude of the smallest zero given by

$$B = 1.0001\times\min(\sqrt{n}\times L(z_k), |r_1|, |a_n/a_0|^{1/n}),$$

r_1 is the zero X of smaller magnitude of the quadratic equation

$$(P''(z_k)/(2\times n\times(n-1)))X^2 + (P'(z_k)/n)X + \tfrac{1}{2}P(z_k) = 0$$

and the Cauchy lower bound R for the smallest zero is computed (using Newton's Method) as the positive root of the polynomial equation

$$|a_0|z^n + |a_1|z^{n-1} + |a_2|z^{n-2} + \ldots + |a_{n-1}|z - |a_n| = 0.$$

Starting from the origin, successive iterates are generated according to the rule $z_{k+1} = z_k + L(z_k)$ for $k = 1,2,3,\ldots$ and $L(z_k)$ is 'adjusted' so that $|P(z_{k+1})| < |P(z_k)|$ and $|L(z_{k+1})| \leq |F|$. The iterative procedure terminates if $P(z_{k+1})$ is smaller in absolute value than the bound on the rounding error in $P(z_{k+1})$ and the current iterate $z_p = z_{k+1}$ is taken to be a zero of $P(z)$ (as is its conjugate $\bar{z}_p$ if z_p is complex). The deflated polynomial $\tilde{P}(z) = P(z)/(z-z_p)$ of degree $n-1$ if z_p is real ($\tilde{P}(z) = P(z)/((z-z_p)(z-\bar{z}_p))$ of degree $n-2$ if z_p is complex) is then formed, and the above procedure is repeated on the deflated polynomial until $n < 3$, whereupon the remaining roots are obtained via the 'standard' closed formulae for a linear ($n = 1$) or quadratic ($n = 2$) equation.

To obtain the roots of a quadratic polynomial, C02AJF can be used.

4. References

[1] MARDEN, M.
Geometry of Polynomials. Mathematical Surveys.
Am. Math. Soc., Providence, Rhode Island, USA, 3, 1966.

[2] SMITH, B.T.
ZERPOL: A Zero Finding Algorithm for Polynomials Using Laguerre's Method.
Technical Report, Department of Computer Science, University of Toronto, Canada, 1967.

[3] WILKINSON, J.H.
The Algebraic Eigenvalue Problem.
Clarendon Press, 1965.

5. Parameters

1: A(N+1) – ***real*** array. *Input*

On entry: if A is declared with bounds (0:N), then A(i) must contain a_i (i.e. the coefficient of z^{n-i}), for $i = 0,1,...,n$.

Constraint: A(0) $\neq$ 0.0.

2: N – INTEGER. *Input*

On entry: the degree of the polynomial, n.

Constraint: N $\geq$ 1.

3: SCALE – LOGICAL. *Input*

On entry: indicates whether or not the polynomial is to be scaled. See Section 8 for advice on when it may be preferable to set SCALE = .FALSE. and for a description of the scaling strategy.

Suggested value: SCALE = .TRUE..

4: Z(2,N) – ***real*** array. *Output*

On exit: the real and imaginary parts of the roots are stored in Z(1,i) and Z(2,i) respectively, for $i = 1,2,...,n$. Complex conjugate pairs of roots are stored in consecutive pairs of elements of Z; that is, Z(1,i+1) = Z(1,i) and Z(2,i+1) = –Z(2,i).

5: W(2*(N+1)) – ***real*** array. *Workspace*

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, A(0) = 0.0,
or N < 1.

IFAIL = 2

The iterative procedure has failed to converge. This error is very unlikely to occur. If it does, please contact NAG immediately, as some basic assumption for the arithmetic has been violated. See also Section 8.

IFAIL = 3

Either overflow or underflow prevents the evaluation of $P(z)$ near some of its zeros. This error is very unlikely to occur. If it does, please contact NAG immediately. See also Section 8.

7. Accuracy

All roots are evaluated as accurately as possible, but because of the inherent nature of the problem complete accuracy cannot be guaranteed.

8. Further Comments

If SCALE = .TRUE., then a scaling factor for the coefficients is chosen as a power of the base B of the machine so that the largest coefficient in magnitude approaches THRESH = B^{EMAX-P}. Users should note that no scaling is performed if the largest coefficient in magnitude exceeds THRESH, even if SCALE = .TRUE.. (For definition of B, $EMAX$ and P see the Chapter Introduction X02.)

However, with SCALE = .TRUE., overflow may be encountered when the input coefficients $a_0,a_1,a_2,...,a_n$ vary widely in magnitude, particularly on those machines for which $B^{(4\times P)}$ overflows. In such cases, SCALE should be set to .FALSE. and the coefficients scaled so that the largest coefficient in magnitude does not exceed $B^{(EMAX-2\times P)}$.

Even so, the scaling strategy used in C02AGF is sometimes insufficient to avoid overflow and/or underflow conditions. In such cases, the user is recommended to scale the independent variable (z) so that the disparity between the largest and smallest coefficient in magnitude is reduced. That is, use the routine to locate the zeros of the polynomial $d\times P(cz)$ for some suitable values of c and d. For example, if the original polynomial was $P(z) = 2^{-100} + 2^{100}z^{20}$, then choosing $c = 2^{-10}$ and $d = 2^{100}$, for instance, would yield the scaled polynomial $1 + z^{20}$, which is well-behaved relative to overflow and underflow and has zeros which are 2^{10} times those of $P(z)$.

If the routine fails with IFAIL = 2 or 3, then the real and imaginary parts of any roots obtained before the failure occurred are stored in Z in the reverse order in which they were found. Let n_R denote the number of roots found before the failure occurred. Then Z(1,n) and Z(2,n) contain the real and imaginary parts of the 1st root found, Z(1,$n-1$) and Z(2,$n-1$) contain the real and imaginary parts of the 2nd root found, ..., Z(1,n_R) and Z(2,n_R) contain the real and imaginary parts of the n_Rth root found. After the failure has occurred, the remaining $2\times(n-n_R)$ elements of Z contain a large negative number (equal to $-1/(\text{X02AMF}().\sqrt{2})$).

9. Example

To find the roots of the 5th degree polynomial $z^5 + 2z^4 + 3z^3 + 4z^2 + 5z + 6 = 0$.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C02AGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      real             ZERO
      PARAMETER        (ZERO=0.0e0)
      INTEGER          MAXDEG
      PARAMETER        (MAXDEG=100)
      LOGICAL          SCALE
      PARAMETER        (SCALE=.TRUE.)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, N, NROOT
*     .. Local Arrays ..
      real             A(0:MAXDEG), W(2*MAXDEG+2), Z(2,MAXDEG)
```

```
*     .. External Subroutines ..
      EXTERNAL         C02AGF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C02AGF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GT.0 .AND. N.LE.MAXDEG) THEN
         READ (NIN,*) (A(I),I=0,N)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Degree of polynomial = ', N
         IFAIL = 0
*
         CALL C02AGF(A,N,SCALE,Z,W,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Roots of polynomial'
         WRITE (NOUT,*)
         NROOT = 1
   20    IF (NROOT.LE.N) THEN
            IF (Z(2,NROOT).EQ.ZERO) THEN
               WRITE (NOUT,99998) 'Z = ', Z(1,NROOT)
               NROOT = NROOT + 1
            ELSE
               WRITE (NOUT,99998) 'Z = ', Z(1,NROOT), ' +/- ',
     +           ABS(Z(2,NROOT)), '*i'
               NROOT = NROOT + 2
            END IF
            GO TO 20
         END IF
      ELSE
         WRITE (NOUT,*) 'N is out of range'
      END IF
      STOP
*
99999 FORMAT (1X,A,I4)
99998 FORMAT (1X,A,1P,e12.4,A,1P,e12.4,A)
      END
```

9.2. Program Data

```
C02AGF Example Program Data
 5
     1.0     2.0     3.0     4.0     5.0     6.0
```

9.3. Program Results

```
 C02AGF Example Program Results

 Degree of polynomial =     5

 Roots of polynomial

 Z =  -1.4918E+00
 Z =   5.5169E-01 +/-   1.2533E+00*i
 Z =  -8.0579E-01 +/-   1.2229E+00*i
```

C02AHF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C02AHF determines the roots of a quadratic equation with complex coefficients.

2. Specification

```
SUBROUTINE C02AHF (AR, AI, BR, BI, CR, CI, ZSM, ZLG, IFAIL)
INTEGER          IFAIL
real             AR, AI, BR, BI, CR, CI, ZSM(2), ZLG(2)
```

3. Description

The routine attempts to find the roots of the quadratic equation $az^2 + bz + c = 0$ (where a, b and c are complex coefficients), by carefully evaluating the 'standard' closed formula

$$z = \frac{-b \pm \sqrt{b^2-4ac}}{2a}.$$

It is based on the routine CQDRTC from Smith [1].

Note: it is not necessary to scale the coefficients prior to calling the routine.

4. References

[1] SMITH, B.T.
ZERPOL: A Zero Finding Algorithm for Polynomials Using Laguerre's Method.
Technical Report, Department of Computer Science, University of Toronto, Canada, 1967.

5. Parameters

1: AR – ***real***. *Input*

2: AI – ***real***. *Input*

On entry: AR and AI must contain the real and imaginary parts respectively of a, the coefficient of z^2.

3: BR – ***real***. *Input*

4: BI – ***real***. *Input*

On entry: BR and BI must contain the real and imaginary parts respectively of b, the coefficient of z.

5: CR – ***real***. *Input*

6: CI – ***real***. *Input*

On entry: CR and CI must contain the real and imaginary parts respectively of c, the constant coefficient.

7: ZSM(2) – ***real*** array. *Output*

On exit: the real and imaginary parts of the smallest root in magnitude are stored in ZSM(1) and ZSM(2) respectively.

8: ZLG(2) – ***real*** array. *Output*

On exit: the real and imaginary parts of the largest root in magnitude are stored in ZLG(1) and ZLG(2) respectively.

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, (AR,AI) = (0,0). In this case, ZSM(1) and ZSM(2) contain the real and imaginary parts respectively of the root $-c/b$.

IFAIL = 2

On entry, (AR,AI) = (0,0) and (BR,BI) = (0,0). In this case, ZSM(1) contains the largest machine representable number (see X02ALF) and ZSM(2) contains zero.

IFAIL = 3

On entry, (AR,AI) = (0,0) and the root $-c/b$ overflows. In this case, ZSM(1) contains the largest machine representable number (see X02ALF) and ZSM(2) contains zero.

IFAIL = 4

On entry, (CR,CI) = (0,0) and the root $-b/a$ overflows. In this case, both ZSM(1) and ZSM(2) contain zero.

IFAIL = 5

On entry, $\tilde{b}$ is so large that $\tilde{b}^2$ is indistinguishable from $\tilde{b}^2-4\tilde{a}\tilde{c}$ and the root $-b/a$ overflows, where $\tilde{b}$ = max(|BR|,|BI|), $\tilde{a}$ = max(|AR|,|AI|) and $\tilde{c}$ = max(|CR|,|CI|). In this case, ZSM(1) and ZSM(2) contain the real and imaginary parts respectively of the root $-c/b$.

If IFAIL > 0 on exit, then ZLG(1) contains the largest machine representable number (see X02ALF) and ZLG(2) contains zero.

7. Accuracy

If IFAIL = 0 on exit, then the computed roots should be accurate to within a small multiple of the ***machine precision*** except when underflow (or overflow) occurs, in which case the true roots are within a small multiple of the underflow (or overflow) threshold of the machine.

8. Further Comments

None.

9. Example

To find the roots of the quadratic equation $z^2 - (3.0-1.0i)z + (8.0+1.0i) = 0$.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C02AHF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             AI, AR, BI, BR, CI, CR
      INTEGER          IFAIL
*     .. Local Arrays ..
      real             ZLG(2), ZSM(2)
*     .. External Subroutines ..
      EXTERNAL         C02AHF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C02AHF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) AR, AI, BR, BI, CR, CI
      IFAIL = 0
*
      CALL C02AHF(AR,AI,BR,BI,CR,CI,ZSM,ZLG,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Roots of quadratic equation'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'z = ', ZSM(1), ZSM(2), '*i'
      WRITE (NOUT,99999) 'z = ', ZLG(1), ZLG(2), '*i'
      STOP
*
99999 FORMAT (1X,A,1P,e12.4,SP,e14.4,A)
      END
```

9.2. Program Data

```
C02AHF Example Program Data
 1.0    0.0   -3.0    1.0    8.0    1.0       :AR  AI  BR  BI  CR  CI
```

9.3. Program Results

```
 C02AHF Example Program Results

 Roots of quadratic equation

 z =   1.0000E+00   +2.0000E+00*i
 z =   2.0000E+00   -3.0000E+00*i
```

C02AJF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C02AJF determines the roots of a quadratic equation with real coefficients.

2. Specification

```
SUBROUTINE C02AJF (A, B, C, ZSM, ZLG, IFAIL)
INTEGER          IFAIL
real             A, B, C, ZSM(2), ZLG(2)
```

3. Description

The routine attempts to find the roots of the quadratic equation $az^2 + bz + c = 0$ (where a, b and c are real coefficients), by carefully evaluating the 'standard' closed formula

$$z = \frac{-b \pm \sqrt{b^2-4ac}}{2a}.$$

It is based on the routine QDRTC from Smith [1].

Note: it is not necessary to scale the coefficients prior to calling the routine.

4. References

[1] SMITH, B.T.
ZERPOL: A Zero Finding Algorithm for Polynomials Using Laguerre's Method.
Technical Report, Department of Computer Science, University of Toronto, Canada, 1967.

5. Parameters

1: A – ***real***. *Input*

On entry: A must contain a, the coefficient of z^2.

2: B – ***real***. *Input*

On entry: B must contain b, the coefficient of z.

3: C – ***real***. *Input*

On entry: C must contain c, the constant coefficient.

4: ZSM(2) – ***real*** array. *Output*

On exit: the real and imaginary parts of the smallest root in magnitude are stored in ZSM(1) and ZSM(2) respectively.

5: ZLG(2) – ***real*** array. *Output*

On exit: the real and imaginary parts of the largest root in magnitude are stored in ZLG(1) and ZLG(2) respectively.

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, A = 0. In this case, ZSM(1) contains the root $-c/b$ and ZSM(2) contains zero.

IFAIL = 2

On entry, A = 0 and B = 0. In this case, ZSM(1) contains the largest machine representable number (see X02ALF) and ZSM(2) contains zero.

IFAIL = 3

On entry, A = 0 and the root $-c/b$ overflows. In this case, ZSM(1) contains the largest machine representable number (see X02ALF) and ZSM(2) contains zero.

IFAIL = 4

On entry, C = 0 and the root $-b/a$ overflows. In this case, both ZSM(1) and ZSM(2) contain zero.

IFAIL = 5

On entry, b is so large that b^2 is indistinguishable from $b^2 - 4ac$ and the root $-b/a$ overflows. In this case, ZSM(1) contains the root $-c/b$ and ZSM(2) contains zero.

If IFAIL > 0 on exit, then ZLG(1) contains the largest machine representable number (see X02ALF) and ZLG(2) contains zero.

7. Accuracy

If IFAIL = 0 on exit, then the computed roots should be accurate to within a small multiple of the ***machine precision*** except when underflow (or overflow) occurs, in which case the true roots are within a small multiple of the underflow (or overflow) threshold of the machine.

8. Further Comments

None.

9. Example

To find the roots of the quadratic equation $z^2 + 3z - 10 = 0$.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C02AJF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      real             ZERO
      PARAMETER        (ZERO=0.0e0)
*     .. Local Scalars ..
      real             A, B, C
      INTEGER          IFAIL
*     .. Local Arrays ..
      real             ZLG(2), ZSM(2)
*     .. External Subroutines ..
      EXTERNAL         C02AJF
```

```
*     .. Intrinsic Functions ..
      INTRINSIC        ABS
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C02AJF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) A, B, C
      IFAIL = 0
*
      CALL C02AJF(A,B,C,ZSM,ZLG,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Roots of quadratic equation'
      WRITE (NOUT,*)
      IF (ZSM(2).EQ.ZERO) THEN
*        2 real roots.
         WRITE (NOUT,99999) 'z = ', ZSM(1)
         WRITE (NOUT,99999) 'z = ', ZLG(1)
      ELSE
*        2 complex roots.
         WRITE (NOUT,99998) 'z = ', ZSM(1), ' +/- ', ABS(ZSM(2)), '*i'
      END IF
      STOP
*
99999 FORMAT (1X,A,1P,e12.4)
99998 FORMAT (1X,A,1P,e12.4,A,e12.4,A)
      END
```

9.2. Program Data

```
C02AJF Example Program Data
 1.0    3.0   -10.0             :A  B  C
```

9.3. Program Results

```
 C02AJF Example Program Results

 Roots of quadratic equation

 z =   2.0000E+00
 z =  -5.0000E+00
```

Chapter C05 – Roots of One or More Transcendental Equations

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
C05ADF	8	Zero of continuous function in given interval, Bus and Dekker algorithm
C05AGF	8	Zero of continuous function, Bus and Dekker algorithm, from given starting value, binary search for interval
C05AJF	8	Zero of continuous function, continuation method, from a given starting value
C05AVF	8	Binary search for interval containing zero of continuous function (reverse communication)
C05AXF	8	Zero of continuous function by continuation method, from given starting value (reverse communication)
C05AZF	7	Zero in given interval of continuous function by Bus and Dekker algorithm (reverse communication)
C05NBF	9	Solution of system of nonlinear equations using function values only (easy-to-use)
C05NCF	9	Solution of system of nonlinear equations using function values only (comprehensive)
C05NDF	14	Solution of systems of nonlinear equations using function values only (reverse communication)
C05PBF	9	Solution of system of nonlinear equations using 1st derivatives (easy-to-use)
C05PCF	9	Solution of system of nonlinear equations using 1st derivatives (comprehensive)
C05PDF	14	Solution of systems of nonlinear equations using 1st derivatives (reverse communication)
C05ZAF	9	Check user's routine for calculating 1st derivatives

Chapter C05

Roots of One or More Transcendental Equations

Contents

1 Scope of the Chapter

This chapter is concerned with the calculation of real zeros of continuous real functions of one or more variables. (Complex equations must be expressed in terms of the equivalent larger system of real equations.)

2 Background to the Problems

The chapter divides naturally into two parts.

2.1 A Single Equation

The first deals with the real zeros of a real function of a single variable $f(x)$.

There are three routines with simple calling sequences. The first assumes that the user can determine an initial interval $[a, b]$ within which the desired zero lies, that is $f(a) \times f(b) < 0$, and outside which all other zeros lie. The routine then systematically subdivides the interval to produce a final interval containing the zero. This final interval has a length bounded by the user's specified error requirements; the end of the interval where the function has smallest magnitude is returned as the zero. This routine is guaranteed to converge to a **simple** zero of the function. (Here we define a simple zero as a zero corresponding to a sign-change of the function; none of the available routines are capable of making any finer distinction.) However, as with the other routines described below a non-simple zero might be determined and it is left to the user to check for this. The algorithm used is due to Bus and Dekker.

The two other routines are both designed for the case where the user is unable to specify an interval containing the simple zero. The first routine starts from an initial point and performs a search for an interval containing a simple zero. If such an interval is computed then the method described above is used next to determine the zero accurately. The second method uses a 'continuation' method based on a secant iteration. A sequence of subproblems is solved, the first of these is trivial and the last is the actual problem of finding a zero of $f(x)$. The intermediate problems employ the solutions of earlier problems to provide initial guesses for the secant iterations used to calculate their solutions.

Three other routines are also supplied. They employ reverse communication and are called by the routines described above.

2.2 Systems of Equations

The routines in the second part of this chapter are designed to solve a set of nonlinear equations in n unknowns

$$f_i(x) = 0, \quad i = 1, 2, \ldots, n, \quad x = (x_1, x_2, \ldots, x_n)^T, \tag{1}$$

where T stands for transpose.

It is assumed that the functions are continuous and differentiable so that the matrix of first partial derivatives of the functions, the **Jacobian** matrix $J_{ij}(x) = (\frac{\partial f_i}{\partial x_j})$ evaluated at the point x, exists, though it may not be possible to calculate it directly.

The functions f_i must be independent, otherwise there will be an infinity of solutions and the methods will fail. However, even when the functions are independent the solutions may not be unique. Since the methods are iterative, an initial guess at the solution has to be supplied, and the solution located will usually be the one closest to this initial guess.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

3.1 Zeros of Functions of One Variable

The routines can be divided into two classes. There are three routines (C05AVF, C05AXF and C05AZF) all written in reverse communication form and three (C05ADF, C05AGF and C05AJF) written in direct communication form. The direct communication routines are designed for inexperienced users and, in

particular, for solving problems where the function $f(x)$ whose zero is to be calculated, can be coded as a user-supplied routine. These routines find the zero by making calls to one or more of the reverse communication routines. Experienced users are recommended to use the reverse communication routines directly as they permit the user more control of the calculation. Indeed, if the zero-finding process is embedded in a much larger program then the reverse communication routines should always be used.

The recommendation as to which routine should be used depends mainly on whether the user can supply an interval $[a, b]$ containing the zero, that is $f(a) \times f(b) < 0$. If the interval can be supplied, then C05ADF (or, in reverse communication, C05AZF) should be used, in general. This recommendation should be qualified in the case when the only interval which can be supplied is very long relative to the user's error requirements **and** the user can also supply a good approximation to the zero. In this case C05AJF (or, in reverse communication, C05AXF) **may** prove more efficient (though these latter routines will not provide the error bound available from C05AZF).

If an interval containing the zero cannot be supplied then the user must choose between C05AGF (or, in reverse communication, C05AVF followed by C05AZF) and C05AJF (or, in reverse communication, C05AXF). C05AGF first determines an interval containing the zero, and then proceeds as in C05ADF; it is particularly recommended when the user does not have a good initial approximation to the zero. If a good initial approximation to the zero is available then C05AJF is to be preferred. Since neither of these latter routines has guaranteed convergence to the zero, the user is recommended to experiment with both in case of difficulty.

3.2 Solution of Sets of Nonlinear Equations

The solution of a set of nonlinear equations

$$f_i(x_1, x_2, \ldots, x_n) = 0, \quad i = 1, 2, \ldots, n \tag{2}$$

can be regarded as a special case of the problem of finding a minimum of a sum of squares

$$s(x) = \sum_{i=1}^{m} [f_i(x_1, x_2, \ldots, x_n)]^2, \quad (m \geq n). \tag{3}$$

So the routines in Chapter E04 are relevant as well as the special nonlinear equations routines.

The routines for solving a set of nonlinear equations can also be divided into classes. There are four routines (C05NBF, C05NCF, C05PBF and C05PCF) all written in direct communication form and two (C05NDF and C05PDF) written in reverse communication form. The direct communication routines are designed for inexperienced users and, in particular, these routines require the f_i (and possibly their derivatives) to be calculated in user-supplied routines. These should be set up carefully so the Library routines can work as efficiently as possible. Experienced users are recommended to use the reverse communication routines as they permit the user more control of the calculation. Indeed, if the zero-finding process is embedded in a much larger program then the reverse communication routines should always be used.

The main decision which has to be made by the user is whether to supply the derivatives $\frac{\partial f_i}{\partial x_j}$. It is advisable to do so if possible, since the results obtained by algorithms which use derivatives are generally more reliable than those obtained by algorithms which do not use derivatives.

C05PBF and C05PCF (or, in reverse communication, C05PDF) require the user to provide the derivatives, whilst C05NBF and C05NCF (or, in reverse communication, C05NDF) do not. C05NBF and C05PBF are easy-to-use routines; greater flexibility may be obtained using C05NCF and C05PCF, (or, in reverse communication, C05NDF and C05PDF), but these have longer parameter lists. C05ZAF is provided for use in conjunction with C05PBF and C05PCF to check the user-provided derivatives for consistency with the functions themselves. The user is strongly advised to make use of this routine whenever C05PBF or C05PCF is used.

Firstly, the calculation of the functions and their derivatives should be ordered so that **cancellation errors** are avoided. This is particularly important in a routine that uses these quantities to build up estimates of higher derivatives.

Secondly, **scaling** of the variables has a considerable effect on the efficiency of a routine. The problem should be designed so that the elements of x are of similar magnitude. The same comment applies to the functions, i.e., all the f_i should be of comparable size.

The accuracy is usually determined by the accuracy parameters of the routines, but the following points may be useful:

(i) Greater accuracy in the solution may be requested by choosing smaller input values for the accuracy parameters. However, if unreasonable accuracy is demanded, rounding errors may become important and cause a failure.

(ii) Some idea of the accuracies of the x_i may be obtained by monitoring the progress of the routine to see how many figures remain unchanged during the last few iterations.

(iii) An approximation to the error in the solution x, given by e where e is the solution to the set of linear equations

$$J(x)e = -f(x)$$

where $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))^T$ (see Chapter F04).

Note that the QR decomposition of J is available from C05NCF and C05PCF (or, in reverse communication, C05NDF and C05PDF) so that

$$R\,e = -Q^T f$$

and $Q^T f$ is also provided by these routines.

(iv) If the functions $f_i(x)$ are changed by small amounts ϵ_i, for $i = 1, 2, \ldots, n$, then the corresponding change in the solution x is given approximately by σ, where σ is the solution of the set of linear equations

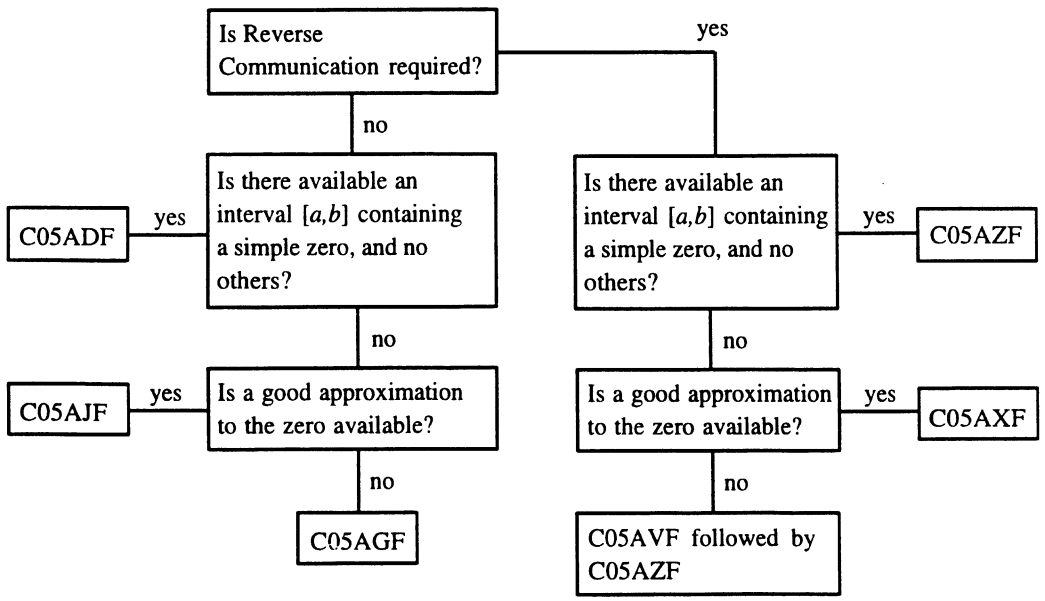
$$J(x)\sigma = -\epsilon,$$

(see Chapter F04).

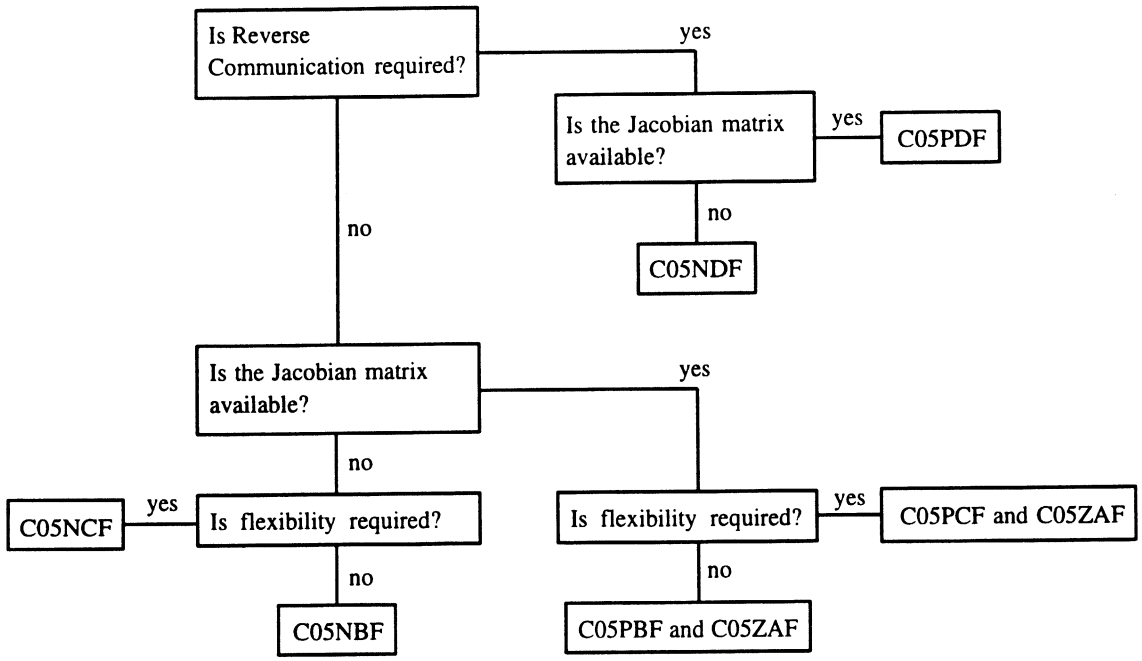
Thus one can estimate the sensitivity of x to any uncertainties in the specification of $f_i(x)$, for $i = 1, 2, \ldots, n$. As noted above, the sophisticated routines C05NCF and C05PCF (or, in reverse communication, C05NDF and C05PDF) provide the QR decomposition of J.

4 Decision Trees

(i) Functions of One Variable



(ii) Functions of Several Variables



5 Index

6 References

[1] Gill P E and Murray W (1976) Algorithms for the solution of the nonlinear least-squares problem *Report NAC 71* National Physical Laboratory

[2] Moré J J, Garbow B S, and Hillstrom K E (1974) User guide for MINPACK-1 *Technical Report ANL-80-74* Argonne National Laboratory

[3] Ortega J M and Rheinboldt W C (1970) *Iterative Solution of Nonlinear Equations in Several Variables* Academic Press

[4] Rabinowitz P (1970) *Numerical Methods for Nonlinear Algebraic Equations* Gordon and Breach

C05ADF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05ADF locates a zero of a continuous function in a given interval by a combination of the methods of linear interpolation, extrapolation and bisection.

2. Specification

```
SUBROUTINE C05ADF (A, B, EPS, ETA, F, X, IFAIL)
INTEGER          IFAIL
real             A, B, EPS, ETA, F, X
EXTERNAL         F
```

3. Description

The routine attempts to obtain an approximation to a simple zero of the function $f(x)$ given an initial interval $[a,b]$ such that $f(a) \times f(b) \leq 0$. The zero is found by calls to C05AZF whose specification should be consulted for details of the method used.

The approximation x to the zero α is determined so that one or both of the following criteria are satisfied:

(i) $|x-\alpha| <$ EPS,

(ii) $|f(x)| <$ ETA.

4. References

None.

5. Parameters

1: A – ***real***. *Input*

On entry: the lower bound of the interval, a.

2: B – ***real***. *Input*

On entry: the upper bound of the interval, b.

Constraint: B $\neq$ A.

3: EPS – ***real***. *Input*

On entry: the absolute tolerance to which the zero is required (see Section 3).

Constraint: EPS $>$ 0.0.

4: ETA – ***real***. *Input*

On entry: a value such that if $|f(x)| <$ ETA, x is accepted as the zero. ETA may be specified as 0.0 (see Section 7).

5: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must evaluate the function f whose zero is to be determined.

Its specification is:

```
real FUNCTION F(XX)
real         XX
```

1: XX – ***real***. *Input*

On entry: the point at which the function must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which C05ADF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: X – ***real***. *Output*

On exit: the approximation to the zero.

7: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, EPS $\leq$ 0.0,
or A = B,
or F(A)×F(B) > 0.0.

IFAIL = 2

Too much accuracy has been requested in the computation, that is, EPS is too small for the computer being used. The final value of X is an accurate approximation to the zero.

IFAIL = 3

A change in sign of $f(x)$ has been determined as occurring near the point defined by the final value of X. However, there is some evidence that this sign-change corresponds to a pole of $f(x)$.

IFAIL = 4

Indicates that a serious error has occurred in C05AZF. Check all routine calls. Seek expert help.

7. Accuracy

This depends on the value of EPS and ETA. If full machine accuracy is required, they may be set very small, resulting in an error exit with IFAIL = 2, although this may involve many more iterations than a lesser accuracy. The user is recommended to set ETA = 0.0 and to use EPS to control the accuracy, unless he has considerable knowledge of the size of $f(x)$ for values of x near the zero.

8. Further Comments

The time taken by the routine depends primarily on the time spent evaluating F (see Section 5).

If it is important to determine an interval of length less than EPS containing the zero, or if the function F is expensive to evaluate and the number of calls to F is to be restricted, then use of C05AZF is recommended. Use of C05AZF is also recommended when the structure of the problem to be solved does not permit a simple function F to be written: the reverse communication facilities of C05AZF are more flexible than the direct communication of F required by C05ADF.

9. Example

The example program below calculates the zero of $e^{-x} - x$ within the interval [0,1] to approximately 5 decimal places.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05ADF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             A, B, EPS, ETA, X
      INTEGER          IFAIL
*     .. External Functions ..
      real             F
      EXTERNAL         F
*     .. External Subroutines ..
      EXTERNAL         C05ADF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05ADF Example Program Results'
      A = 0.0e0
      B = 1.0e0
      EPS = 1.0e-5
      ETA = 0.0e0
      IFAIL = 1
*
      CALL C05ADF(A,B,EPS,ETA,F,X,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,99999) 'Zero =', X
      ELSE
         WRITE (NOUT,99998) 'IFAIL =', IFAIL
         IF (IFAIL.EQ.2 .OR. IFAIL.EQ.3) WRITE (NOUT,99999)
     +       'Final point = ', X
      END IF
      STOP
*
99999 FORMAT (1X,A,F12.5)
99998 FORMAT (1X,A,I3)
      END
*
      real  FUNCTION F(X)
*     .. Scalar Arguments ..
      real             X
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      F = EXP(-X) - X
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C05ADF Example Program Results

Zero =     0.56714
```

C05AGF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05AGF locates a simple zero of a continuous function from a given starting value, using a binary search to locate an interval containing a zero of the function, then a combination of the methods of linear interpolation, extrapolation and bisection to locate the zero precisely.

2. Specification

```
SUBROUTINE C05AGF (X, H, EPS, ETA, F, A, B, IFAIL)
INTEGER          IFAIL
real             X, H, EPS, ETA, F, A, B
EXTERNAL         F
```

3. Description

The routine attempts to locate an interval $[a,b]$ containing a simple zero of the function $f(x)$ by a binary search starting from the initial point x = X and using repeated calls to C05AVF. If this search succeeds, then the zero is determined to a user-specified accuracy by repeated calls to C05AZF. The specifications of routines C05AVF and C05AZF should be consulted for details of the methods used.

The approximation x to the zero α is determined so that at least one of the following criteria is satisfied:

(i) $|x-\alpha| \leq \text{EPS} \times \max(1.0, |z|)$ where z is $0(\alpha)$,

(ii) $|f(x)| < \text{ETA}$.

4. References

None.

5. Parameters

1: X – ***real***. *Input/Output*

On entry: an initial approximation to the zero.

On exit: the final approximation to the zero, unless the routine has failed, in which case it contains no useful information.

2: H – ***real***. *Input*

On entry: a step length for use in the binary search for an interval containing the zero. The maximum interval searched is [X–256.0×H, X+256.0×H].

Constraint: H must be sufficiently large that X + H ≠ X on the computer.

3: EPS – ***real***. *Input*

On entry: the tolerance to which the zero is required (see Section 3).

Constraint: EPS > 0.0.

4: ETA – ***real***. *Input*

On entry: a value such that if $|f(x)| <$ ETA, x is accepted as the zero. ETA may be specified as 0.0 (see Section 7).

5: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must evaluate the function f whose zero is to be determined.

Its specification is:

```
real FUNCTION F(XX)
real         XX
```

1: XX – ***real***. *Input*

On entry: the point at which the function must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which C05AGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: A – ***real***. *Output*

7: B – ***real***. *Output*

On exit: the lower and upper bounds respectively of the interval resulting from the binary search. If the zero is determined exactly such that $f(x) = 0.0$ or is determined so that $|f(x)| <$ ETA at any stage in the calculation, then on exit A = B = x.

8: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, either EPS ≤ 0.0, or X + H = X to machine accuracy (meaning that the search for an interval containing the zero cannot commence).

IFAIL = 2

An interval containing the zero could not be found. Increasing H and calling C05AGF again will increase the range searched for the zero. Decreasing H and calling C05AGF again will refine the mesh used in the search for the zero.

IFAIL = 3

A change of sign of $f(x)$ has been determined as occurring near the point defined by the final value of X. However, there is some evidence that this sign-change corresponds to a pole of $f(x)$.

IFAIL = 4

Too much accuracy has been requested in the computation, that is EPS is too small for the computer being used. The final value of X is an accurate approximation to the zero.

IFAIL = 5
IFAIL = 6

Indicate that a serious error has occurred in C05AVF or C05AZF respectively. Check all routine calls. Seek expert help.

7. Accuracy

This depends on EPS and ETA. If full machine accuracy is required, they may be set very small, resulting in an error exit with IFAIL = 4, although this may involve many more iterations than a lesser accuracy. The user is recommended to set ETA = 0.0 and to use EPS to control the accuracy, unless he has considerable knowledge of the size of $f(x)$ for values of x near the zero.

8. Further Comments

The time taken by the routine depends primarily on the time spent evaluating F (see Section 5). The accuracy of the initial approximation X and the value of H will have a somewhat unpredictable effect on the timing.

If it is important to determine an interval of length less than EPS containing the zero, or if the function F is expensive to evaluate and the number of calls to F is to be restricted, then use of C05AVF followed by C05AZF is recommended. Use of this combination is also recommended when the structure of the problem to be solved does not permit a simple function F to be written; the reverse communication facilities of these routines are more flexible than the direct communication of F required by C05AGF.

If the iteration terminates with successful exit and A = B = X there is no guarantee that the value returned in X corresponds to a simple zero and the user should check whether it does.

One way to check this is to compute the derivative of f at the point X, preferably analytically, or, if this is not possible, numerically, perhaps by using a central difference estimate.

If $f'(X) = 0.0$, then X must correspond to a multiple zero of f rather than a simple zero.

9. Example

The example program below calculates the zero of $x - e^{-x}$ to approximately five decimal places starting from X = 1.0 and using an initial search step H = 0.1.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05AGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             A, B, EPS, ETA, H, X
      INTEGER          IFAIL
*     .. External Functions ..
      real             F
      EXTERNAL         F
*     .. External Subroutines ..
      EXTERNAL         C05AGF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05AGF Example Program Results'
      X = 1.0e0
      H = 0.1e0
      EPS = 1.0e-5
      ETA = 0.0e0
      IFAIL = 1
*
      CALL C05AGF(X,H,EPS,ETA,F,A,B,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,99999) 'Root is ', X
         WRITE (NOUT,99998) 'Interval searched is (', A, ',', B, ')'
      ELSE
         WRITE (NOUT,99997) 'IFAIL =', IFAIL
         IF (IFAIL.EQ.3 .OR. IFAIL.EQ.4) WRITE (NOUT,99999)
     +       'Final value = ', X
      END IF
      STOP
*
99999 FORMAT (1X,A,F13.5)
99998 FORMAT (1X,A,F8.5,A,F8.5,A)
99997 FORMAT (1X,A,I3)
      END
```

```
*
      real  FUNCTION F(X)
*     .. Scalar Arguments ..
      real              X
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Executable Statements ..
      F = X - EXP(-X)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C05AGF Example Program Results

Root is        0.56714
Interval searched is ( 0.50000, 0.90000)
```

C05AJF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05AJF attempts to locate a zero of a continuous function by a continuation method using a secant iteration.

2. Specification

```
SUBROUTINE C05AJF (X, EPS, ETA, F, NFMAX, IFAIL)
INTEGER          NFMAX, IFAIL
real             X, EPS, ETA, F
EXTERNAL         F
```

3. Description

The routine attempts to obtain an approximation to a zero α of the function $f(x)$ given an initial approximation x to α. The zero is found by a call to C05AXF whose specification should be consulted for details of the method used.

The approximation x to the root α is determined so that at least one of the following criteria is satisfied:

(i) $|x-\alpha| \sim$ EPS,

(ii) $|f(x)| \leq$ ETA.

4. References

None.

5. Parameters

1: X – ***real.*** *Input/Output*

On entry: an initial approximation to the zero.

On exit: the final approximation to the zero, unless an error exit has occurred, in which case it contains no useful information.

2: EPS – ***real.*** *Input*

On entry: an absolute tolerance to control the accuracy to which the zero is determined. In general, the smaller the value of EPS the more accurate X will be as an approximation to α. Indeed, for very small positive values of EPS, it is likely that the final approximation will satisfy $|X-\alpha| <$ EPS. The user is advised to call the routine with more than one value for EPS to check the accuracy obtained.

Constraint: EPS > 0.0.

3: ETA – ***real.*** *Input*

On entry: a value such that if $|f(x)| <$ ETA, then x is returned as the final approximation to the zero. ETA may be specified as 0.0 (see Section 7).

4: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must evaluate the function f whose zero is to be determined.

Its specification is:

```
real FUNCTION F(XX)
real         XX
```

1: XX – ***real***. *Input*

On entry: the point at which the function must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which C05AJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: NFMAX – INTEGER. *Input*

On entry: the maximum permitted number of calls to F from C05AJF. If F is inexpensive to evaluate, NFMAX should be given a large value (say > 1000).

Constraint: NFMAX > 0.

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, EPS ≤ 0.0,
or NFMAX ≤ 0.

IFAIL = 2

An internally calculated scale factor has the wrong order of magnitude for the problem. If this error exit occurs, the user is advised to call C05AXF instead where different scale values can be tried.

IFAIL = 3

Either the function $f(x)$ given by F has no zero near X or too much accuracy has been requested in calculating the zero. The first is a more likely cause of this error exit and the user should check the coding of F and make an independent investigation of its behaviour near X. The second can be alleviated by increasing EPS.

IFAIL = 4

More than NFMAX calls have been made to the function F. This error exit can occur because NFMAX is too small for the problem (essentially because X is too far away from the zero) or for either of the reasons given under IFAIL = 3 above. If NFMAX is increased considerably and this error exit occurs again at approximately the same final value of X, then it is likely that one of the reasons given under IFAIL = 3 is the cause.

IFAIL = 5

Indicates that a serious error has occurred in C05AXF. Check all subroutine calls. Seek expert help.

7. Accuracy

This depends on the values of EPS and ETA. If full machine accuracy is required, they may be set very small, possibly resulting in an error exit with IFAIL = 3 or 4, although this may involve many more iterations than a lesser accuracy. The user is recommended to set ETA = 0.0 and to use EPS to control the accuracy unless he has considerable knowledge of the size of $f(x)$ for values of x near the zero.

8. Further Comments

The time taken by the routine depends primarily on the time spent evaluating the function F (see Section 5) and on how close the initial value of X is to the zero.

If a more flexible way of specifying the function F is required or if the user wishes to have closer control of the calculation, then the reverse communication routine C05AXF is recommended instead of C05AJF.

9. Example

The example program below calculates the zero of $f(x) = e^{-x} - x$ from a starting value X = 1.0. Two calculations are made with EPS = 1.0E–3 and 1.0E–4 for comparison purposes, with ETA = 0.0 in both cases.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05AJF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             EPS, ETA, X
      INTEGER          IFAIL, K, NFMAX
*     .. External Functions ..
      real             F
      EXTERNAL         F
*     .. External Subroutines ..
      EXTERNAL         C05AJF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05AJF Example Program Results'
      WRITE (NOUT,*)
      DO 20 K = 3, 4
         EPS = 10.0e0**(-K)
         X = 1.0e0
         ETA = 0.0e0
         NFMAX = 200
         IFAIL = 1
*
         CALL C05AJF(X,EPS,ETA,F,NFMAX,IFAIL)
*
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,99998) 'With EPS = ', EPS, '   root = ', X
         ELSE
            WRITE (NOUT,99999) 'IFAIL =', IFAIL
            IF (IFAIL.EQ.3 .OR. IFAIL.EQ.4) THEN
               WRITE (NOUT,99998) 'With EPS = ', EPS, ' final value = ',
     +           X
            END IF
         END IF
   20 CONTINUE
      STOP
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,A,e10.2,A,F14.5)
      END
```

```
*
      real  FUNCTION F(X)
*     .. Scalar Arguments ..
      real              X
*     .. Intrinsic Functions ..
      INTRINSIC       EXP
*     .. Executable Statements ..
      F = EXP(-X) - X
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C05AJF Example Program Results

With EPS =    0.10E-02    root =         0.56715
With EPS =    0.10E-03    root =         0.56715
```

C05AVF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05AVF attempts to locate an interval containing a simple zero of a continuous function using a binary search. It uses reverse communication for evaluating the function.

2. Specification

```
SUBROUTINE C05AVF (X, FX, H, BOUNDL, BOUNDU, Y, C, IND, IFAIL)
INTEGER          IND, IFAIL
real             X, FX, H, BOUNDL, BOUNDU, Y, C(11)
```

3. Description

The user must supply an initial point X and a step H. The routine attempts to locate a short interval [X,Y] $\subset$ [BOUNDL,BOUNDU] containing a simple zero of $f(x)$.

(On exit we may have X > Y; X is determined as the first point encountered in a binary search where the sign of $f(x)$ differs from the sign of $f(x)$ at the initial input point X.) The routine attempts to locate a zero of $f(x)$ using H, 0.1×H, 0.01×H and 0.001×H in turn as its basic step before quitting with an error exit if unsuccessful.

C05AVF returns to the calling program for each evaluation of $f(x)$. On each return the user should set FX = f(X) and call C05AVF again.

4. References

None.

5. Parameters

Note: this routine uses **reverse communication**. Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the **parameter IND**. Between intermediate exits and re-entries, **all parameters other than FX must remain unchanged.**

1: X – ***real.*** *Input/Output*

On initial entry: the best available approximation to the zero.

Constraint: X must lie in the closed interval [BOUNDL,BOUNDU] (see below).

On intermediate exit: X contains the point at which f must be evaluated before re-entry to the routine.

On final exit: X contains one end of an interval containing the zero, the other end being in Y (below), unless an error has occurred. If IFAIL = 4, X and Y are the endpoints of the largest interval searched. If a zero is located exactly, its value is returned in X (and in Y).

2: FX – ***real.*** *Input*

On initial entry: if IND = 1, FX need not be set.

If IND = −1, FX must contain f(X) for the initial value of X.

On intermediate re-entry: FX must contain f(X) for the current value of X.

3: H – ***real.*** *Input/Output*

On initial entry: a basic step-size which is used in the binary search for an interval containing a zero. The basic step-sizes H, 0.1×H, 0.01×H and 0.001×H are used in turn when searching for the zero.

Constraint: either X + H or X − H must lie inside the closed interval [BOUNDL,BOUNDU] (see below).

H must be sufficiently large that X + H ≠ X on the computer.

On final exit: H is undefined.

4: BOUNDL – ***real.*** *Input*

5: BOUNDU – ***real.*** *Input*

On initial entry: BOUNDL and BOUNDU must contain respectively lower and upper bounds for the interval of search for the zero.

Constraint: BOUNDL < BOUNDU.

6: Y – ***real.*** *Input/Output*

On initial entry: Y need not be set.

On final exit: Y contains the closest point found to the final value of X, such that $f(X) \times f(Y) \leq 0$. If a value X is found such that $f(X) = 0$, then Y = X. On final exit with IFAIL = 4, X and Y are the endpoints of the largest interval searched.

7: C(11) – ***real*** array. *Workspace*

(On final exit with IFAIL = 0 or 4, C(1) contains $f(Y)$.)

8: IND – INTEGER. *Input/Output*

On initial entry: IND must be set to 1 or –1:

if IND = 1, FX need not be set;

if IND = –1, FX must contain $f(X)$.

On intermediate exit: IND contains 2 or 3. The calling program must evaluate f at X, storing the result in FX, and re-enter C05AVF with all other parameters unchanged.

On final exit: IND contains 0.

Constraint: on entry IND = –1, 1, 2 or 3.

9: IFAIL – INTEGER. *Input/Output*

On initial entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On final exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, BOUNDU ≤ BOUNDL,
or X ∉ [BOUNDL,BOUNDU],
or both X + H and X – H ∉ [BOUNDL,BOUNDU].

IFAIL = 2

On initial entry, H is too small to be used to perturb the initial value of X in the search.

IFAIL = 3

The parameter IND is incorrectly set on initial or intermediate entry.

IFAIL = 4

The routine has been unable to determine an interval containing a simple zero starting from the initial value of X and using the step H. A user who has prior knowledge that a simple zero lies in the interval [BOUNDL,BOUNDU], should vary X and H in an attempt to find it. (See also Section 8.)

7. Accuracy

This routine is not intended to be used to obtain accurate approximations to the zero of $f(x)$ but rather to locate an interval containing a zero. This interval can then be used as input to an accurate rootfinder such as C05AZF or C05ADF. The size of the interval determined depends somewhat unpredictably on the choice of X and H. The closer X is to the root and the **smaller** the initial value of H, then, in general, the smaller (more accurate) the interval determined; however, the accuracy of this statement depends to some extent on the behaviour of $f(x)$ near x = X and on the size of H.

8. Further Comments

For most problems, the time taken on each call to C05AVF will be negligible compared with the time spent evaluating $f(x)$ between calls to C05AVF. However, the initial choices of X and H will clearly affect the number of evaluations of $f(x)$. In general, the closer X is to the root and the **larger** the initial value of H then the less the time taken. (However taking H large can affect the accuracy and reliability of the routine, see below.)

The user is expected to choose BOUNDL and BOUNDU as physically (or mathematically) realistic limits on the interval of search. For example, it may be known, from physical arguments, that no zero of $f(x)$ of interest will lie outside [BOUNDL,BOUNDU]. Alternatively, $f(x)$ may be more expensive to evaluate for some values of X than for others and such expensive evaluations can sometimes be avoided by careful choice of BOUNDL and BOUNDU.

The choice of BOUNDL and BOUNDU affects the search only in that these values provide physical limitations on the search values and that the search is terminated if it seems, from the available information about $f(x)$, that the zero lies outside [BOUNDL,BOUNDU]. In this case (IFAIL = 4 on exit), only one of f(BOUNDL) and f(BOUNDU) may have been evaluated and a zero close to the other end of the interval could be missed. The actual interval searched is returned in the parameters X and Y and the user can call C05AVF again to search the remainder of the original interval.

Though C05AVF is intended primarily for determining an interval containing a zero of $f(x)$, it may be used to shorten a known interval. This could be useful if, for example, a large interval containing the zero is known and it is also known that the root lies close to one end of the interval; by setting X to this end of the interval and H small, a short interval will usually be determined. However, it is worth noting that once any interval containing a zero has been determined, a call to C05AZF will usually be the most efficient way to calculate an interval of specified length containing the zero. To assist in this determination, the information in X, Y, FX and C(1) on successful exit from C05AVF is in the correct form for a call to routine C05AZF with IND = −1.

If the calculation terminates because f(X) = 0.0, then on return Y is set to X. (In fact, Y = X on return only in this case.) In this case, there is no guarantee that the value in X corresponds to a **simple** zero and the user should check whether it does.

One way to check this is to compute the derivative of f at the point X, preferably analytically, or, if this is not possible, numerically, perhaps by using a central difference estimate.

If f'(X) = 0.0, then X must correspond to a multiple zero of f rather than a simple zero.

9. Example

To find a subinterval of [0.0,4.0] containing a zero of $x^2 - 3x + 2$. The zero nearest to 3.0 is required and so we set X = 3.0 initially.

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05AVF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             BOUNDL, BOUNDU, FX, H, X, Y
      INTEGER          IFAIL, IND
*     .. Local Arrays ..
      real             C(11)
*     .. External Subroutines ..
      EXTERNAL         C05AVF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05AVF Example Program Results'
      WRITE (NOUT,*)
      X = 3.0e0
      H = 0.1e0
      BOUNDL = 0.0e0
      BOUNDU = 4.0e0
      IFAIL = 1
      IND = 1
*
   20 CALL C05AVF(X,FX,H,BOUNDL,BOUNDU,Y,C,IND,IFAIL)
*
      IF (IND.NE.0) THEN
         FX = X*X - 3.0e0*X + 2.0e0
         GO TO 20
      ELSE
         IF (IFAIL.GT.0) THEN
            WRITE (NOUT,99997) 'Error exit,  IFAIL =', IFAIL
         ELSE
            WRITE (NOUT,*) 'Interval containing root is (Y,X) where'
            WRITE (NOUT,99999) 'Y =', Y, '   X =', X
            WRITE (NOUT,*) 'Values of f at Y and X are'
            WRITE (NOUT,99998) 'f(Y) =', C(1), '   f(X) =', FX
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,F12.4,A,F12.4)
99998 FORMAT (1X,A,F12.2,A,F12.2)
99997 FORMAT (1X,A,I2)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 C05AVF Example Program Results

 Interval containing root is (Y,X) where
 Y =      2.5000   X =      1.7000
 Values of f at Y and X are
 f(Y) =        0.75   f(X) =       -0.21
```

C05AXF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05AXF attempts to locate a zero of a continuous function using a continuation method based on a secant iteration. It uses reverse communication for evaluating the function.

2. Specification

```
SUBROUTINE C05AXF (X, FX, TOL, IR, SCALE, C, IND, IFAIL)
INTEGER          IR, IND, IFAIL
real             X, FX, TOL, SCALE, C(26)
```

3. Description

This routine uses a modified version of an algorithm given in Swift and Lindfield [1] to compute a zero α of a continuous function $f(x)$. The algorithm used is based on a continuation method in which a sequence of problems

$$f(x) - \theta_r f(x_0), \qquad r = 0,1,...,m$$

are solved, where $1 = \theta_0 > \theta_1 > ... > \theta_m = 0$ (the value of m is determined as the algorithm proceeds) and where x_0 is the user's initial estimate for the zero of $f(x)$. For each θ_r the current problem is solved by a robust secant iteration using the solution from earlier problems to compute an initial estimate.

The user must supply an error tolerance TOL. TOL is used directly to control the accuracy of solution of the final problem ($\theta_m = 0$) in the continuation method, and $\sqrt{\text{TOL}}$ is used to control the accuracy in the intermediate problems ($\theta_1, \theta_2, ..., \theta_{m-1}$).

4. References

[1] SWIFT, A. and LINDFIELD, G.R.
Comparison of a Continuation Method for the Numerical Solution of a Single Nonlinear Equation.
Comput. J., 21, pp. 359-362, 1978.

5. Parameters

Note: this routine uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the **parameter IND.** Between intermediate exits and re-entries, **all parameters other than FX must remain unchanged.**

1: X – ***real.*** *Input/Output*

On initial entry: an initial approximation to the zero.

On intermediate exit: the point at which f must be evaluated before re-entry to the routine.

On final exit: the final approximation to the zero.

2: FX – ***real.*** *Input*

On initial entry: if IND = 1, FX need not be set.

If IND = −1, FX must contain f(X) for the initial value of X.

On intermediate re-entry: FX must contain f(X) for the current value of X.

3: TOL – ***real.*** *Input*

On initial entry: a value which controls the accuracy to which the zero is determined. This parameter is used in determining the convergence of the secant iteration used at each stage of the continuation process. It is used directly when solving the last problem ($\theta_m = 0$ in Section 3), and $\sqrt{\text{TOL}}$ is used for the problem defined by θ_r, $r < m$. Convergence to the accuracy specified by TOL is not guaranteed, and so the user is recommended to find the zero using at least two values for TOL to check the accuracy obtained.

Constraint: TOL > 0.0.

4: IR – INTEGER. *Input*

On initial entry: IR indicates the type of error test required, as follows. Solving the problem defined by θ_r, $1 \leq r \leq m$, involves computing a sequence of secant iterates $x_r^0, x_r^1, \ldots$. This sequence will be considered to have converged only if:

for IR = 0, $|x_r^{(i+1)} - x_r^{(i)}| \leq \text{EPS} \times \max(1.0, |x_r^{(i)}|)$,

for IR = 1, $|x_r^{(i+1)} - x_r^{(i)}| \leq \text{EPS}$,

for IR = 2, $|x_r^{(i+1)} - x_r^{(i)}| \leq \text{EPS} \times |x_r^{(i)}|$,

for some $i > 1$; here EPS is either TOL or $\sqrt{\text{TOL}}$ as discussed above. Note that there are other subsidiary conditions (not given here) which must also be satisfied before the secant iteration is considered to have converged.

Constraint: IR = 0, 1 or 2.

5: SCALE – ***real.*** *Input*

On initial entry: a factor for use in determining a significant approximation to the derivative of $f(x)$ at $x = x_0$, the initial value. A number of difference approximations to $f'(x_0)$ are calculated using

$$f'(x_0) \sim (f(x_0+h) - f(x_0))/h$$

where $|h| < |\text{SCALE}|$ and h has the same sign as SCALE. A significance (cancellation) check is made on each difference approximation and the approximation is rejected if insignificant.

Suggested value: the square root of the ***machine precision.***

Constraint: SCALE must be sufficiently large that X + SCALE ≠ X on the computer.

6: C(26) – ***real*** array. *Workspace*

(C(5) contains the current value, θ_r, and C(7) contains a value, λ_r, used in the secant iteration (see Swift and Lindfield [1]); these values may be useful in the event of an error exit.)

7: IND – INTEGER. *Input/Output*

On initial entry: IND must be set to 1 or –1:

if IND = 1, FX need not be set;

if IND = –1, FX must contain $f(\text{X})$.

On intermediate exit: IND contains 2, 3 or 4. The calling program must evaluate f at X, storing the result in FX, and re-enter C05AXF with all other parameters unchanged.

On final exit: IND contains 0.

Constraint: on entry IND = –1, 1, 2, 3 or 4.

8: IFAIL – INTEGER. *Input/Output*

On initial entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On final exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, TOL $\leq$ 0.0,
or IR $\neq$ 0, 1 or 2.

IFAIL = 2

The parameter IND is incorrectly set on initial or intermediate entry.

IFAIL = 3

SCALE is too small, or significant derivatives of f cannot be computed (this can happen when f is almost constant and non-zero, for any value of SCALE).

IFAIL = 4

The current problem in the continuation sequence cannot be solved, see C(5) for the value of θ_r. The most likely explanation is that the current problem has no solution, either because the original problem had no solution or because the continuation path passes through a set of insoluble problems. This latter reason for failure should occur rarely, and not at all if the initial approximation to the zero is sufficiently close. Other possible explanations are that TOL is too small and hence the accuracy requirement is too stringent, or that TOL is too large and the initial approximation too poor, leading to successively worse intermediate solutions.

IFAIL = 5

Continuation away from the initial point is not possible. This error exit will usually occur if the problem has not been properly posed or the error requirement is extremely stringent.

IFAIL = 6

The final problem (with $\theta_m = 0$) cannot be solved. It is likely that too much accuracy has been requested, or that the zero is at $\alpha = 0$ and IR = 2.

7. Accuracy

The accuracy of the approximation to the zero depends on TOL and IR. In general decreasing TOL will give more accurate results. Care must be exercised when using the relative error criterion (IR = 2).

If the zero is at $X = 0$, or if the initial value of X and the zero bracket the point X = 0, it is likely that an error exit with IFAIL = 4, 5 or 6 will occur.

As discussed in Section 6, it is possible to request too much or too little accuracy. Since it is not possible to achieve more than machine accuracy, a value of TOL $\ll$ ***machine precision*** should not be input and may lead to an error exit with IFAIL = 4, 5 or 6. For the reasons discussed under IFAIL = 4 in Section 6, TOL should not be taken too large, say no larger than TOL = 1.0E–3.

8. Further Comments

For most problems, the time taken on each call to C05AXF will be negligible compared with the time spent evaluating $f(x)$ between calls to C05AXF. However, the initial value of X and the choice of TOL will clearly affect the timing. The closer that X is to the root, the less evaluations of f required. The effect of the choice of TOL will not be large, in general, unless TOL is very small, in which case the timing will increase.

If the results obtained from this routine seem unreliable or inaccurate, the user should consider using C05AZF (possibly combined with C05AVF to obtain an interval containing the zero).

One way to check this is to compute the derivative of f at the point X, preferably analytically, or, if this is not possible, numerically, perhaps by using a central difference estimate.

If $f'(X) = 0.0$, then X must correspond to a multiple zero of f rather than a simple zero.

9. Example

To calculate a zero of $x - e^{-x}$ with initial approximation $x_0 = 1.0$, and TOL = 1.0E−3 and 1.0E−4.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05AXF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             F, SCALE, TOL, X
      INTEGER          I, IFAIL, IND, IR
*     .. Local Arrays ..
      real             C(26)
*     .. External Functions ..
      real             X02AJF
      EXTERNAL         X02AJF
*     .. External Subroutines ..
      EXTERNAL         C05AXF
*     .. Intrinsic Functions ..
      INTRINSIC        EXP, SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05AXF Example Program Results'
      SCALE = SQRT(X02AJF())
      IR = 0
      DO 40 I = 3, 4
         TOL = 10.0e0**(-I)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'TOL = ', TOL
         WRITE (NOUT,*)
         X = 1.0e0
         IFAIL = 1
         IND = 1
*
   20    CALL C05AXF(X,F,TOL,IR,SCALE,C,IND,IFAIL)
*
         IF (IND.NE.0) THEN
            F = X - EXP(-X)
            GO TO 20
         ELSE
            IF (IFAIL.GT.0) THEN
               WRITE (NOUT,99998) 'Error exit, IFAIL =', IFAIL
               IF (IFAIL.EQ.4 .OR. IFAIL.EQ.6) THEN
                  WRITE (NOUT,99997) 'Final value = ', X, ' THETA = ',
     +              C(5), ' LAMBDA = ', C(7)
               END IF
            ELSE
```

```
                  WRITE (NOUT,99997) 'Root is ', X
               END IF
            END IF
         40 CONTINUE
            STOP
      *
      99999 FORMAT (1X,A,e10.4)
      99998 FORMAT (1X,A,I2)
      99997 FORMAT (1X,A,F14.5,A,F10.2,A,F10.2)
            END
```

9.2. Program Data

None.

9.3. Program Results

```
 C05AXF Example Program Results

 TOL = 0.1000E-02

 Root is         0.56715

 TOL = 0.1000E-03

 Root is         0.56715
```

C05AZF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05AZF locates a simple zero of a continuous function on a given interval by a combination of the methods of linear interpolation, linear extrapolation and bisection. It uses reverse communication for evaluating the function.

2. Specification

```
SUBROUTINE C05AZF (X, Y, FX, TOLX, IR, C, IND, IFAIL)
INTEGER          IR, IND, IFAIL
real             X, Y, FX, TOLX, C(17)
```

3. Description

The user must supply an initial interval [X,Y] containing a simple zero of the function $f(x)$ (the choice of X and Y must be such that $f(X) \times f(Y) \leq 0.0$). The routine combines the methods of bisection, linear interpolation and linear extrapolation (see Dahlquist and Bjorck [1]), to find a sequence of subintervals of the initial interval such that the final interval [X,Y] contains the zero and |X–Y| is less than some tolerance specified by TOLX and IR (see Section 5). In fact, since the intervals [X,Y] are determined only so that $f(X) \times f(Y) \leq 0$, it is possible that the final interval may contain a discontinuity or a pole of f (violating the requirement that f be continuous). C05AZF checks if the sign change is likely to correspond to a pole of f and gives an error return in this case.

C05AZF returns to the calling program for each evaluation of $f(x)$. On each return the user should set FX = f(X) and call C05AZF again.

The routine is a modified version of procedure 'zeroin' given by Bus and Dekker [2].

4. References

[1] DAHLQUIST, G. and BJORCK, A.
Numerical Methods.
Prentice-Hall, 1974.

[2] BUS, J.C.P. and DEKKER, T.J.
Two Efficient Algorithms with Guaranteed Convergence for Finding a Zero of a Function.
ACM Trans. Math. Software, 1, pp. 330-345, 1975.

5. Parameters

Note: this routine uses **reverse communication**. Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the **parameter IND**. Between intermediate exits and re-entries, **all parameters other than FX must remain unchanged.**

1: X – ***real.*** *Input/Output*

2: Y – ***real.*** *Input/Output*

On initial entry: X and Y must define an initial interval containing the zero, such that $f(X) \times f(Y) \leq 0$. It is not necessary that X < Y.

On intermediate exit: X contains the point at which f must be evaluated before re-entry to the routine.

On final exit: X and Y define a smaller interval containing the zero, such that $f(X) \times f(Y) \leq 0$, and |X–Y| satisfies the accuracy specified by TOLX and IR, unless an error has occurred. If IFAIL = 4, X and Y generally contain very good approximations to a pole; if IFAIL = 5, X and Y generally contain very good approximations to the zero (see Section 6). If a point X is found such that $f(X) = 0$, then on final exit X = Y (in this case there is no guarantee that X is a simple zero).

3: FX – ***real.*** *Input/Output*

On initial entry: if IND = 1, FX need not be set.

If IND = −1, FX must contain $f(X)$ for the initial value of X.

On intermediate re-entry: FX must contain $f(X)$ for the current value of X.

On exit: FX is unchanged, except that after initial entry with IND = −1 FX contains the input value of C(1).

4: TOLX – ***real.*** *Input*

On initial entry: the accuracy to which the zero is required. The type of error test is specified by IR (below).

Constraint: TOLX > 0.

5: IR – INTEGER. *Input*

On initial entry: indicates the type of error test as follows:

if IR = 0, the test is: $|X-Y| \leq 2.0 \times TOLX \times \max(1.0, |Z|)$;

if IR = 1, the test is: $|X-Y| \leq 2.0 \times TOLX$;

if IR = 2, the test is: $|X-Y| \leq 2.0 \times TOLX \times |Z|$.

Here Z is the value of x for which $|f(x)|$ is currently known to have the smallest value; Z is calculated internally to C05AZF.

Suggested value: IR = 0.

Constraint: IR = 0, 1 or 2.

6: C(17) – ***real*** array. *Input/Output*

On initial entry: if IND = 1, no elements of C need be set.

If IND = −1, C(1) must contain $f(Y)$, other elements of C need not be set.

On final exit: C is undefined.

7: IND – INTEGER. *Input/Output*

On initial entry: IND must be set to 1 or −1:

if IND = 1, FX and C(1) need not be set;

if IND = −1, FX and C(1) must contain $f(X)$ and $f(Y)$ respectively.

On intermediate exit: IND contains 2, 3 or 4. The calling program must evaluate f at X, storing the result in FX, and re-enter C05AZF with all other parameters unchanged.

On final exit: IND contains 0.

Constraint: on entry IND = −1, 1, 2, 3 or 4.

8: IFAIL – INTEGER. *Input/Output*

On initial entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On final exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, $f(X)$ and $f(Y)$ have the same sign, with $f(X) \neq 0.0$.

IFAIL = 2

On entry, IND ≠ −1, 1, 2, 3 or 4.

IFAIL = 3

On entry, TOLX $\leq$ 0.0,
or IR $\neq$ 0, 1 or 2.

IFAIL = 4

An interval [X,Y] has been determined satisfying the error tolerance specified by TOLX and IR and such that $f(X) \times f(Y) \leq 0$. However, from observation of the values of f during the calculation of [X,Y], it seems that the interval [X,Y] contains a pole rather than a zero. Note that this error exit is not completely reliable: the error exit may be taken in extreme cases when [X,Y] contains a zero, or the error exit may not be taken when [X,Y] contains a pole. Both these cases occur most frequently when TOLX is large.

IFAIL = 5

The tolerance TOLX is too small for the problem being solved. This indicator is only set when the length of the interval [X,Y] containing the zero has been reduced as much as possible without satisfying the accuracy requirement (see Section 3 and Section 5). The values X and Y returned are usually both very good approximations to the zero.

7. Accuracy

The accuracy of the final value X as an approximation of the zero is determined by TOLX and IR as described above. A relative accuracy criterion (IR = 2) should not be used when the initial values X and Y are of different orders of magnitude. In this case a change of origin of the independent variable may be appropriate. For example, if the initial interval [X,Y] is transformed linearly to the interval [1,2], then the zero can be determined to a precise number of figures using an absolute (IR = 1) or relative (IR = 2) error test and the effect of the transformation back to the original interval can also be determined. Except for the accuracy check, such a transformation has no effect on the calculation of the zero.

8. Further Comments

For most problems, the time taken on each call to C05AZF will be negligible compared with the time spent evaluating $f(x)$ between calls to C05AZF.

If the calculation terminates because $f(X) = 0.0$, then on return Y is set to X. (In fact, Y = X on return only in this case and, possibly, when IFAIL = 5.) There is no guarantee that the value returned in X corresponds to a **simple** root and the user should check whether it does.

One way to check this is to compute the derivative of f at the point X, preferably analytically, or, if this is not possible, numerically, perhaps by using a central difference estimate.

If $f'(X) = 0.0$, then X must correspond to a multiple zero of f rather than a simple zero.

9. Example

To calculate a zero of $e^{-x} - x$ with an initial interval [0,1], TOLX = 1.0E−5 and a mixed error test.

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05AZF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             FX, TOLX, X, Y
      INTEGER          IFAIL, IND, IR
```

```
*     .. Local Arrays ..
      real              C(17)
*     .. External Functions ..
      real              F
      EXTERNAL          F
*     .. External Subroutines ..
      EXTERNAL          C05AZF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05AZF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) ' Iterations'
      WRITE (NOUT,*)
      TOLX = 1.0e-5
      X = 0.0e0
      Y = 1.0e0
      IR = 0
      IFAIL = 1
      IND = 1
*
   20 CALL C05AZF(X,Y,FX,TOLX,IR,C,IND,IFAIL)
*
      IF (IND.NE.0) THEN
         IF (IND.LT.2 .OR. IND.GT.4) THEN
            WRITE (NOUT,99997) 'Failure with IND=', IND, ' at X=', X
         ELSE
            FX = F(X)
            WRITE (NOUT,99999) ' X=', X, '   FX=', FX, '   IND=', IND
            GO TO 20
         END IF
      ELSE
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,*) ' Solution'
            WRITE (NOUT,*)
            WRITE (NOUT,99998) ' X=', X, '   Y=', Y
         ELSE
            WRITE (NOUT,99997) 'IFAIL = ', IFAIL
            IF (IFAIL.EQ.4 .OR. IFAIL.EQ.5) WRITE (NOUT,99998) 'X =', X,
     +           '  Y =', Y
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,F8.5,A,e12.4,A,I2)
99998 FORMAT (1X,A,F8.5,A,F8.5)
99997 FORMAT (1X,A,I2,A,F10.4)
      END
*
      real  FUNCTION F(X)
*     .. Scalar Arguments ..
      real              X
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Executable Statements ..
      F = EXP(-X) - X
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C05AZF Example Program Results

 Iterations

X= 0.00000   FX=  0.1000E+01   IND= 2
X= 1.00000   FX= -0.6321E+00   IND= 3
X= 0.61270   FX= -0.7081E-01   IND= 4
X= 0.56384   FX=  0.5182E-02   IND= 4
X= 0.56717   FX= -0.4242E-04   IND= 4
X= 0.56714   FX= -0.2538E-07   IND= 4
X= 0.56714   FX=  0.7810E-05   IND= 4

 Solution

X= 0.56714   Y= 0.56714
```

C05NBF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05NBF is an easy-to-use routine to find a solution of a system of nonlinear equations by a modification of the Powell hybrid method.

2. Specification

```
SUBROUTINE C05NBF (FCN, N, X, FVEC, XTOL, WA, LWA, IFAIL)
INTEGER          N, LWA, IFAIL
real             X(N), FVEC(N), XTOL, WA(LWA)
EXTERNAL         FCN
```

3. Description

The system of equations is defined as:

$$f_i(x_1,x_2,\ldots,x_n) = 0, \qquad \text{for } i = 1,2,\ldots,n.$$

C05NBF is based upon the MINPACK routine HYBRD1 (Moré *et al.* [1]). It chooses the correction at each step as a convex combination of the Newton and scaled gradient directions. Under reasonable conditions this guarantees global convergence for starting points far from the solution and a fast rate of convergence. The Jacobian is updated by the rank-1 method of Broyden. At the starting point the Jacobian is approximated by forward differences, but these are not used again until the rank-1 method fails to produce satisfactory progress. For more details see Powell [2].

4. References

[1] MORÉ, J.J., GARBOW, B.S. and HILLSTROM, K.E.
User Guide for MINPACK-1.
Argonne National Laboratory, ANL-80-74.

[2] POWELL, M.J.D.
A Hybrid Method for Nonlinear Algebraic Equations.
In, 'Numerical Methods for Nonlinear Algebraic Equations', Rabinowitz, P. (ed).
Gordon and Breach, 1970.

5. Parameters

1: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must return the values of the functions f_i at a point x.

Its specification is:

```
SUBROUTINE FCN(N, X, FVEC, IFLAG)
INTEGER    N, IFLAG
real       X(N), FVEC(N)
```

1: N – INTEGER. *Input*

On entry: the number of equations, n.

2: X(N) – ***real*** array. *Input*

On entry: the components of the point x at which the functions must be evaluated.

3: FVEC(N) – ***real*** array. *Output*

On exit: the function values $f_i(x)$ (unless IFLAG is set to a negative value by FCN).

4: IFLAG – INTEGER. *Input/Output*

On entry: IFLAG > 0.

On exit: in general, IFLAG should not be reset by FCN. If, however, the user wishes to terminate execution (perhaps because some illegal point X has been reached), then IFLAG should be set to a negative integer. This value will be returned through IFAIL.

FCN must be declared as EXTERNAL in the (sub)program from which C05NBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: N – INTEGER. *Input*

On entry: the number of equations, n.

Constraint: N > 0.

3: X(N) – ***real*** array. *Input/Output*

On entry: an initial guess at the solution vector.

On exit: the final estimate of the solution vector.

4: FVEC(N) – ***real*** array. *Output*

On exit: the function values at the final point, X.

5: XTOL – ***real***. *Input*

On entry: the accuracy in X to which the solution is required.

Suggested value: the square root of the ***machine precision***.

Constraint: XTOL ≥ 0.0.

6: WA(LWA) – ***real*** array. *Workspace*

7: LWA – INTEGER. *Input*

On entry: the dimension of the array WA.

Constraint: LWA ≥ N×(3×N+13)/2.

8: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL < 0

The user has set IFLAG negative in FCN. The value of IFAIL will be the same as the user's setting of IFLAG.

IFAIL = 1

On entry, N ≤ 0,
or XTOL < 0.0,
or LWA < N×(3×N+13)/2.

IFAIL = 2

There have been at least 200×(N+1) evaluations of FCN. Consider restarting the calculation from the final point held in X.

IFAIL = 3

No further improvement in the approximate solution X is possible; XTOL is too small.

IFAIL = 4

The iteration is not making good progress. This failure exit may indicate that the system does not have a zero, or that the solution is very close to the origin (see Section 7). Otherwise, rerunning C05NBF from a different starting point may avoid the region of difficulty.

7. Accuracy

If $\hat{x}$ is the true solution, C05NBF tries to ensure that

$$\|x-\hat{x}\| \leq \text{XTOL}\times\|\hat{x}\|.$$

If this condition is satisfied with XTOL = 10^{-k} then the larger components of x have k significant decimal digits. There is a danger that the smaller components of x may have large relative errors, but the fast rate of convergence of C05NBF usually avoids this possibility.

If XTOL is less than ***machine precision***, and the above test is satisfied with the ***machine precision*** in place of XTOL, then the routine exits with IFAIL = 3.

Note: this convergence test is based purely on relative error, and may not indicate convergence if the solution is very close to the origin.

The test assumes that the functions are reasonably well behaved. If this condition is not satisfied, then C05NBF may incorrectly indicate convergence. The validity of the answer can be checked, for example, by rerunning C05NBF with a tighter tolerance.

8. Further Comments

The time required by C05NBF to solve a given problem depends on n, the behaviour of the functions, the accuracy requested and the starting point. The number of arithmetic operations executed by C05NBF to process each call of FCN is about $11.5\times n^2$. Unless FCN can be evaluated quickly, the timing of C05NBF will be strongly influenced by the time spent in FCN.

Ideally the problem should be scaled so that at the solution the function values are of comparable magnitude.

9. Example

To determine the values $x_1,\ldots,x_9$ which satisfy the tridiagonal equations:

$$(3-2x_1)x_1 - 2x_2 = -1$$
$$-x_{i-1} + (3-2x_i)x_i - 2x_{i+1} = -1, \qquad i = 2,3,\ldots,8$$
$$-x_8 + (3-2x_9)x_9 = -1.$$

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05NBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LWA
      PARAMETER        (N=9,LWA=(N*(3*N+13))/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
```

```
*     .. Local Scalars ..
      real               FNORM, TOL
      INTEGER            I, IFAIL, J
*     .. Local Arrays ..
      real               FVEC(N), WA(LWA), X(N)
*     .. External Functions ..
      real               F06EJF, X02AJF
      EXTERNAL           F06EJF, X02AJF
*     .. External Subroutines ..
      EXTERNAL           C05NBF, FCN
*     .. Intrinsic Functions ..
      INTRINSIC          SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05NBF Example Program Results'
      WRITE (NOUT,*)
*     The following starting values provide a rough solution.
      DO 20 J = 1, N
         X(J) = -1.0e0
   20 CONTINUE
      TOL = SQRT(X02AJF())
      IFAIL = 1
*
      CALL C05NBF(FCN,N,X,FVEC,TOL,WA,LWA,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         FNORM = F06EJF(N,FVEC,1)
         WRITE (NOUT,99999) 'Final 2-norm of the residuals =', FNORM
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final approximate solution'
         WRITE (NOUT,*)
         WRITE (NOUT,99998) (X(J),J=1,N)
      ELSE
         WRITE (NOUT,99997) 'IFAIL = ', IFAIL
         IF (IFAIL.GT.1) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Approximate solution'
            WRITE (NOUT,*)
            WRITE (NOUT,99998) (X(I),I=1,N)
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,e12.4)
99998 FORMAT (1X,3F12.4)
99997 FORMAT (1X,A,I2)
      END
*
      SUBROUTINE FCN(N,X,FVEC,IFLAG)
*     .. Parameters ..
      real               ONE, TWO, THREE
      PARAMETER          (ONE=1.0e0,TWO=2.0e0,THREE=3.0e0)
*     .. Scalar Arguments ..
      INTEGER            IFLAG, N
*     .. Array Arguments ..
      real               FVEC(N), X(N)
*     .. Local Scalars ..
      INTEGER            K
*     .. Executable Statements ..
      DO 20 K = 1, N
         FVEC(K) = (THREE-TWO*X(K))*X(K) + ONE
         IF (K.GT.1) FVEC(K) = FVEC(K) - X(K-1)
         IF (K.LT.N) FVEC(K) = FVEC(K) - TWO*X(K+1)
   20 CONTINUE
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C05NBF Example Program Results

Final 2-norm of the residuals =  0.1193E-07

Final approximate solution

     -0.5707      -0.6816      -0.7017
     -0.7042      -0.7014      -0.6919
     -0.6658      -0.5960      -0.4164
```

C05NCF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05NCF is a comprehensive routine to find a solution of a system of nonlinear equations by a modification of the Powell hybrid method.

2. Specification

```
      SUBROUTINE C05NCF (FCN, N, X, FVEC, XTOL, MAXFEV, ML, MU, EPSFCN,
     1                   DIAG, MODE, FACTOR, NPRINT, NFEV, FJAC, LDFJAC,
     2                   R, LR, QTF, W, IFAIL)
      INTEGER           N, MAXFEV, ML, MU, MODE, NPRINT, NFEV, LDFJAC, LR,
     1                  IFAIL
      real              X(N), FVEC(N), XTOL, EPSFCN, DIAG(N), FACTOR,
     1                  FJAC(LDFJAC,N), R(LR), QTF(N), W(N,4)
      EXTERNAL          FCN
```

3. Description

The system of equations is defined as:

$$f_i(x_1,x_2,...,x_n) = 0, \qquad \text{for } i = 1,2,...,n.$$

C05NCF is based upon the MINPACK routine HYBRD (Moré *et al.* [1]). It chooses the correction at each step as a convex combination of the Newton and scaled gradient directions. Under reasonable conditions this guarantees global convergence for starting points far from the solution and a fast rate of convergence. The Jacobian is updated by the rank-1 method of Broyden. At the starting point the Jacobian is approximated by forward differences, but these are not used again until the rank-1 method fails to produce satisfactory progress. For more details see Powell [2].

4. References

[1] MORÉ, J.J., GARBOW, B.S. and HILLSTROM, K.E.
User Guide for MINPACK-1.
Argonne National Laboratory, ANL-80-74.

[2] POWELL, M.J.D.
A Hybrid Method for Nonlinear Algebraic Equations.
In, 'Numerical Methods for Nonlinear Algebraic Equations', Rabinowitz, P. (ed).
Gordon and Breach, 1970.

5. Parameters

1: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must return the values of the functions f_i at a point x.

Its specification is:

```
      SUBROUTINE FCN(N, X, FVEC, IFLAG)
      INTEGER    N, IFLAG
      real       X(N), FVEC(N)
```

1: N – INTEGER. *Input*

On entry: the number of equations, n

2: X(N) – ***real*** array. *Input*

On entry: the components of the point x at which the functions must be evaluated.

3: FVEC(N) – ***real*** array. *Output*

On exit: if IFLAG > 0 on entry, FVEC must contain the function values $f_i(x)$ (unless IFLAG is set to a negative value by FCN).

If IFLAG = 0 on entry, FVEC must not be changed.

4: IFLAG – INTEGER. *Input/Output*

On entry: IFLAG ≥ 0:

if IFLAG = 0, X and FVEC are available for printing (see NPRINT below);

if IFLAG > 0, FVEC must be updated

On exit: in general IFLAG should not be reset by FCN. If, however, the user wishes to terminate execution (perhaps because some illegal point X has been reached), then IFLAG should be set to a negative integer. This value will be returned through IFAIL.

FCN must be declared as EXTERNAL in the (sub)program from which C05NCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: N – INTEGER. *Input*

On entry: the number of equations, *n*.

Constraint: N > 0.

3: X(N) – ***real*** array. *Input/Output*

On entry: an initial guess at the solution vector.

On exit: the final estimate of the solution vector.

4: FVEC(N) – ***real*** array. *Output*

On exit: the function values at the final point, X.

5: XTOL – ***real***. *Input*

On entry: the accuracy in X to which the solution is required.

Suggested value: the square root of the ***machine precision***.

Constraint: XTOL ≥ 0.0.

6: MAXFEV – INTEGER. *Input*

On entry: the maximum number of calls to FCN with IFLAG ≠ 0. C05NCF will exit with IFAIL = 2, if, at the end of an iteration, the number of calls to FCN exceeds MAXFEV.

Suggested value: MAXFEV = 200×(N+1).

Constraint: MAXFEV > 0.

7: ML – INTEGER. *Input*

On entry: the number of subdiagonals within the band of the Jacobian matrix. (If the Jacobian is not banded, or you are unsure, set ML = N−1.)

Constraint: ML ≥ 0.

8: MU – INTEGER. *Input*

On entry: the number of superdiagonals within the band of the Jacobian matrix. (If the Jacobian is not banded, or you are unsure, set MU = N−1.)

Constraint: MU ≥ 0.

9: EPSFCN – ***real.*** *Input*

On entry: a rough estimate of the largest relative error in the functions. It is used in determining a suitable step for a forward difference approximation to the Jacobian. If EPSFCN is less than ***machine precision*** then ***machine precision*** is used. Consequently a value of 0.0 will often be suitable.

Suggested value: EPSFCN = 0.0.

10: DIAG(N) – ***real*** array. *Input/Output*

On entry: if MODE = 2 (see below), DIAG must contain multiplicative scale factors for the variables.

Constraint: DIAG(i) > 0.0, for i = 1,2,...,n.

On exit: the scale factors actually used (computed internally if MODE ≠ 2).

11: MODE – INTEGER. *Input*

On entry: indicates whether or not the user has provided scaling factors in DIAG. If MODE = 2 the scaling must have been specified in DIAG. Otherwise, the variables will be scaled internally.

12: FACTOR – ***real.*** *Input*

On entry: FACTOR must specify a quantity to be used in determining the initial step bound. In most cases, FACTOR should lie between 0.1 and 100.0. (The step bound is FACTOR×$\|$DIAG×X$\|_2$ if this is non-zero; otherwise the bound is FACTOR.)

Suggested value: FACTOR = 100.0.

Constraint: FACTOR > 0.0.

13: NPRINT – INTEGER. *Input*

On entry: indicates whether special calls to FCN, with IFLAG set to 0, are to be made for printing purposes. If NPRINT ≤ 0, then no calls are made. If NPRINT > 0, then FCN is called at the beginning of the first iteration, every NPRINT iterations thereafter and immediately prior to the return from C05NCF.

14: NFEV – INTEGER. *Output*

On exit: the number of calls made to FCN.

15: FJAC(LDFJAC,N) – ***real*** array. *Output*

On exit: the orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

16: LDFJAC – INTEGER. *Input*

On entry: the first dimension of the array FJAC as declared in the (sub)program from which C05NCF is called.

Constraint: LDFJAC ≥ N.

17: R(LR) – ***real*** array. *Output*

On exit: the upper triangular matrix R produced by the QR factorization of the final approximate Jacobian, stored row-wise.

18: LR – INTEGER. *Input*

On entry: the dimension of the array R as declared in the (sub)program from which C05NCF is called.

Constraint: LR ≥ N×(N+1)/2.

19: QTF(N) – ***real*** array. *Output*

On exit: the vector $Q^T f$.

20: W(N,4) – ***real*** array. *Workspace*

21: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL < 0

This indicates an exit from C05NCF because the user has set IFLAG negative in FCN. The value of IFAIL will be the same as the user's setting of IFLAG.

IFAIL = 1

On entry, N ≤ 0,
or XTOL < 0.0,
or MAXFEV ≤ 0,
or ML < 0,
or MU < 0,
or FACTOR ≤ 0.0,
or LDFJAC < N,
or LR < N×(N+1)/2,
or MODE = 2 and DIAG(i) ≤ 0.0 for some i, i = 1,2,...,N.

IFAIL = 2

There have been at least MAXFEV evaluations of FCN. Consider restarting the calculation from the final point held in X.

IFAIL = 3

No further improvement in the approximate solution X is possible; XTOL is too small.

IFAIL = 4

The iteration is not making good progress, as measured by the improvement from the last 5 Jacobian evaluations.

IFAIL = 5

The iteration is not making good progress, as measured by the improvement from the last 10 iterations.

The values IFAIL = 4 and IFAIL = 5 may indicate that the system does not have a zero, or that the solution is very close to the origin (see Section 7). Otherwise, rerunning C05NCF from a different starting point may avoid the region of difficulty.

7. Accuracy

If $\hat{x}$ is the true solution and D denotes the diagonal matrix whose entries are defined by the array DIAG, then C05NCF tries to ensure that

$$\|D(x-\hat{x})\|_2 \leq \text{XTOL} \times \|D\hat{x}\|_2.$$

If this condition is satisfied with XTOL = 10^{-k} then the larger components of Dx have k significant decimal digits. There is a danger that the smaller components of Dx may have large relative errors, but the fast rate of convergence of C05NCF usually avoids this possibility.

If XTOL is less than the ***machine precision*** and the above test is satisfied with the ***machine precision*** in place of XTOL, then the routine exits with IFAIL = 3.

Note: this convergence test is based purely on relative error, and may not indicate convergence if the solution is very close to the origin.

The test assumes that the functions are reasonably well behaved. If this condition is not satisfied, then C05NCF may incorrectly indicate convergence. The validity of the answer can be checked for example, by rerunning C05NCF with a tighter tolerance.

8. Further Comments

The time required by C05NCF to solve a given problem depends on n, the behaviour of the functions, the accuracy requested and the starting point. The number of arithmetic operations executed by C05NCF to process each call of FCN is about $11.5 \times n^2$. Unless FCN can be evaluated quickly, the timing of C05NCF will be strongly influenced by the time spent in FCN.

Ideally the problem should be scaled so that, at the solution, the function values are of comparable magnitude.

The number of function evaluations required to evaluate the Jacobian may be reduced if the user can specify ML and MU.

9. Example

To determine the values $x_1,...,x_9$ which satisfy the tridiagonal equations:

$$(3-2x_1)x_1 - 2x_2 = -1.$$
$$-x_{i-1} + (3-2x_i)x_i - 2x_{i+1} = -1, \qquad i = 2,3,...,8.$$
$$-x_8 + (3-2x_9)x_9 = -1.$$

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05NCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LDFJAC, LR
      PARAMETER        (N=9,LDFJAC=N,LR=(N*(N+1))/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             EPSFCN, FACTOR, FNORM, XTOL
      INTEGER          IFAIL, J, MAXFEV, ML, MODE, MU, NFEV, NPRINT
*     .. Local Arrays ..
      real             DIAG(N), FJAC(LDFJAC,N), FVEC(N), QTF(N), R(LR),
     +                 W(N,4), X(N)
*     .. External Functions ..
      real             F06EJF, X02AJF
      EXTERNAL         F06EJF, X02AJF
*     .. External Subroutines ..
      EXTERNAL         C05NCF, FCN
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05NCF Example Program Results'
      WRITE (NOUT,*)
*     The following starting values provide a rough solution.
      DO 20 J = 1, N
         X(J) = -1.0e0
```

```
   20 CONTINUE
      XTOL = SQRT(X02AJF())
      DO 40 J = 1, N
         DIAG(J) = 1.0e0
   40 CONTINUE
      MAXFEV = 2000
      ML = 1
      MU = 1
      EPSFCN = 0.0e0
      MODE = 2
      FACTOR = 100.0e0
      NPRINT = 0
      IFAIL = 1
*
      CALL C05NCF(FCN,N,X,FVEC,XTOL,MAXFEV,ML,MU,EPSFCN,DIAG,MODE,
     +            FACTOR,NPRINT,NFEV,FJAC,LDFJAC,R,LR,QTF,W,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         FNORM = F06EJF(N,FVEC,1)
         WRITE (NOUT,99999) 'Final 2-norm of the residuals =', FNORM
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Number of function evaluations =', NFEV
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final approximate solution'
         WRITE (NOUT,*)
         WRITE (NOUT,99997) (X(J),J=1,N)
      ELSE
         WRITE (NOUT,99996) 'IFAIL = ', IFAIL
         IF (IFAIL.GE.2) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Approximate solution'
            WRITE (NOUT,*)
            WRITE (NOUT,99997) (X(J),J=1,N)
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,e12.4)
99998 FORMAT (1X,A,I10)
99997 FORMAT (1X,3F12.4)
99996 FORMAT (1X,A,I2)
      END
*
      SUBROUTINE FCN(N,X,FVEC,IFLAG)
*     .. Parameters ..
      real              ONE, TWO, THREE
      PARAMETER         (ONE=1.0e0,TWO=2.0e0,THREE=3.0e0)
*     .. Scalar Arguments ..
      INTEGER           IFLAG, N
*     .. Array Arguments ..
      real              FVEC(N), X(N)
*     .. Local Scalars ..
      INTEGER           K
*     .. Executable Statements ..
      IF (IFLAG.EQ.0) THEN
*
*        Insert print statements here when NPRINT is positive.
*
         RETURN
      ELSE
         DO 20 K = 1, N
            FVEC(K) = (THREE-TWO*X(K))*X(K) + ONE
            IF (K.GT.1) FVEC(K) = FVEC(K) - X(K-1)
            IF (K.LT.N) FVEC(K) = FVEC(K) - TWO*X(K+1)
   20    CONTINUE
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C05NCF Example Program Results

Final 2-norm of the residuals =  0.1193E-07

Number of function evaluations =        14

Final approximate solution

     -0.5707     -0.6816     -0.7017
     -0.7042     -0.7014     -0.6919
     -0.6658     -0.5960     -0.4164
```

C05NDF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05NDF is a comprehensive reverse communication routine to find a solution of a system of nonlinear equations by a modification of the Powell hybrid method.

2. Specification

```
      SUBROUTINE C05NDF (IREVCM, N, X, FVEC, XTOL, ML, MU, EPSFCN,
     1                   DIAG, MODE, FACTOR, FJAC, LDFJAC, R, LR, QTF,
     2                   W, IFAIL)
      INTEGER          IREVCM, N, ML, MU, MODE, LDFJAC, LR, IFAIL
      real             X(N), FVEC(N), XTOL, EPSFCN, DIAG(N),
     1                 FACTOR, FJAC(LDFJAC,N), R(LR), QTF(N), W(N,4)
```

3. Description

The system of equations is defined as:

$f_i(x_1,x_2,...,x_n) = 0$, for $i = 1,2,...,n$.

C05NDF is based upon the MINPACK routine HYBRD (Moré *et al.* [1]). It chooses the correction at each step as a convex combination of the Newton and scaled gradient directions. Under reasonable conditions this guarantees global convergence for starting points far from the solution and a fast rate of convergence. The Jacobian is updated by the rank-1 method of Broyden. At the starting point the Jacobian is approximated by forward differences, but these are not used again until the rank-1 method fails to produce satisfactory progress. For more details see Powell [2].

4. References

[1] MORÉ, J.J., GARBOW, B.S. and HILLSTROM, K.E.
User Guide for MINPACK-1.
Argonne National Laboratory, ANL-80-74.

[2] POWELL, M.J.D.
A Hybrid Method for Nonlinear Algebraic Equations.
In, 'Numerical Methods for Nonlinear Algebraic Equations', Rabinowitz, P. (Ed.).
Gordon and Breach, 1970.

5. Parameters

Note: this routine uses reverse communication. Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the parameter IREVCM. Between intermediate exits and re-entries, all parameters other than FVEC must remain unchanged.

1: IREVCM – INTEGER. *Input/Output*

On initial entry: IREVCM must have the value 0.

On intermediate exit: IREVCM specifies what action the user must take before re-entering C05NDF with IREVCM **unchanged.** The value of IREVCM should be interpreted as follows:

IREVCM = 1

indicates the start of a new iteration. No action is required by the user but X and FVEC are available for printing.

IREVCM = 2

indicates that before re-entry to C05NDF, FVEC must contain the function values $f_i(x)$.

On final exit: IREVCM = 0, and the algorithm has terminated.

Constraint: IREVCM = 0, 1 or 2.

2: N – INTEGER. *Input*

On initial entry: the number of equations, n.

Constraint: N > 0.

3: X(N) – ***real*** array. *Input/Output*

On initial entry: an initial guess at the solution vector.

On intermediate exit: X contains the current point.

On final exit: the final estimate of the solution vector.

4: FVEC(N) – ***real*** array. *Input/Output*

On initial entry: FVEC must be set to the values of the functions computed at the initial point X.

On intermediate re-entry: if IREVCM = 1, FVEC must not be changed. If IREVCM = 2, FVEC must be set to the values of the functions computed at the current point X.

On final exit: the function values at the final point, X.

5: XTOL – ***real***. *Input*

On initial entry: the accuracy in X to which the solution is required.

Suggested value: the square root of the ***machine precision***.

Constraint: XTOL ≥ 0.0.

6: ML – INTEGER. *Input*

On initial entry: the number of subdiagonals within the band of the Jacobian matrix. (If the Jacobian is not banded, or you are unsure, set ML = N − 1.)

Constraint: ML ≥ 0.

7: MU – INTEGER. *Input*

On initial entry: the number of superdiagonals within the band of the Jacobian matrix. (If the Jacobian is not banded, or you are unsure, set MU = N − 1.)

Constraint: MU ≥ 0.

8: EPSFCN – ***real***. *Input*

On initial entry: the order of the largest relative error in the functions. It is used in determining a suitable step for a forward difference approximation to the Jacobian. If EPSFCN is less than ***machine precision*** then ***machine precision*** is used. Consequently a value of 0.0 will often be suitable.

Suggested value: EPSFCN = 0.0.

9: DIAG(N) – ***real*** array. *Input/Output*

On initial entry: if MODE = 2 (see below), DIAG must contain multiplicative scale factors for the variables.

Constraint: DIAG(i) > 0.0 for i = 1,2,...,n.

On intermediate exit: the scale factors actually used (computed internally if MODE ≠ 2).

10: MODE – INTEGER. *Input*

On initial entry: indicates whether or not the user has provided scaling factors in DIAG. If MODE = 2 the scaling must have been specified in DIAG. Otherwise, the variables will be scaled internally.

11: FACTOR – ***real***. *Input*

On initial entry: a quantity to be used in determining the initial step bound. In most cases, FACTOR should lie between 0.1 and 100.0. (The step bound is FACTOR×$\|DIAG \times X\|_2$ if this is non-zero; otherwise the bound is FACTOR.)

Suggested value: FACTOR = 100.0.

Constraint: FACTOR > 0.0.

12: FJAC(LDFJAC,N) – ***real*** array. *Output*

On final exit: the orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

13: LDFJAC – INTEGER. *Input*

On initial entry: the first dimension of the array FJAC as declared in the (sub)program from which C05NDF is called.

Constraint: LDFJAC ≥ N.

14: R(LR) – ***real*** array. *Output*

On final exit: the upper triangular matrix R produced by the QR factorization of the final approximate Jacobian, stored row-wise.

15: LR – INTEGER. *Input*

On initial entry: the dimension of the array R as declared in the (sub)program from which C05NDF is called.

Constraint: LR ≥ N×(N+1)/2.

16: QTF(N) – ***real*** array. *Output*

On final exit: the vector $Q^T f$.

17: W(N,4) – ***real*** array. *Workspace*

18: IFAIL – INTEGER. *Input/Output*

On initial entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On final exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, N ≤ 0,
or XTOL < 0.0,
or ML < 0,
or MU < 0,

or FACTOR ≤ 0.0,
or LDFJAC < N,
or LR < N×(N+1)/2,
or MODE = 2 and DIAG(i) ≤ 0.0 for some i, i = 1,2,...,N.

IFAIL = 2

On entry, IREVCM < 0 or IREVCM > 2.

IFAIL = 3

No further improvement in the approximate solution X is possible; XTOL is too small.

IFAIL = 4

The iteration is not making good progress, as measured by the improvement from the last 5 Jacobian evaluations.

IFAIL = 5

The iteration is not making good progress, as measured by the improvement from the last 10 iterations.

The values IFAIL = 4 and IFAIL = 5 may indicate that the system does not have a zero, or that the solution is very close to the origin (see Section 7). Otherwise, rerunning C05NDF from a different starting point may avoid the region of difficulty.

7. Accuracy

If $\hat{x}$ is the true solution and D denotes the diagonal matrix whose entries are defined by the array DIAG, then C05NDF tries to ensure that

$$\|D(x-\hat{x})\|_2 \leq \text{XTOL} \times \|D\hat{x}\|_2.$$

If this condition is satisfied with XTOL = 10^{-k} then the larger components of Dx have k significant decimal digits. There is a danger that the smaller components of Dx may have large relative errors, but the fast rate of convergence of C05NDF usually avoids this possibility.

If XTOL is less than ***machine precision*** and the above test is satisfied with the ***machine precision*** in place of XTOL, then the routine exits with IFAIL = 3.

Note that this convergence test is based purely on relative error, and may not indicate convergence if the solution is very close to the origin.

The test assumes that the functions are reasonably well behaved. If this condition is not satisfied, then C05NDF may incorrectly indicate convergence. The validity of the answer can be checked for example, by rerunning C05NDF with a tighter tolerance.

8. Further Comments

The time required by C05NDF to solve a given problem depends on n, the behaviour of the functions, the accuracy requested and the starting point. The number of arithmetic operations executed by C05NDF to process the evaluation of functions in the main program in each exit is about $11.5 \times n^2$. The timing of C05NDF will be strongly influenced by the time spent in the evaluation of the functions.

Ideally the problem should be scaled so that, at the solution, the function values are of comparable magnitude.

The number of function evaluations required to evaluate the Jacobian may be reduced if the user can specify ML and MU.

9. Example

To determine the values $x_1,...,x_9$ which satisfy the tridiagonal equations:

$$(3-2x_1)x_1 - 2x_2 = -1$$
$$-x_{i-1} + (3-2x_i)x_i - 2x_{i+1} = -1, \qquad i = 2,3,...,8$$
$$-x_8 + (3-2x_9)x_9 = -1.$$

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05NDF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LDFJAC, LR
      PARAMETER        (N=9,LDFJAC=N,LR=(N*(N+1))/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      real             ONE, TWO, THREE
      PARAMETER        (ONE=1.0e0,TWO=2.0e0,THREE=3.0e0)
*     .. Local Scalars ..
      real             EPSFCN, FACTOR, FNORM, XTOL
      INTEGER          ICOUNT, IFAIL, IREVCM, J, K, ML, MODE, MU
*     .. Local Arrays ..
      real             DIAG(N), FJAC(LDFJAC,N), FVEC(N), QTF(N), R(LR),
     +                 W(N,4), X(N)
*     .. External Functions ..
      real             F06EJF, X02AJF
      EXTERNAL         F06EJF, X02AJF
*     .. External Subroutines ..
      EXTERNAL         C05NDF
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05NDF Example Program Results'
*     The following starting values provide a rough solution.
      DO 20 J = 1, N
         X(J) = -1.0e0
   20 CONTINUE
      XTOL = SQRT(X02AJF())
      DO 40 J = 1, N
         DIAG(J) = 1.0e0
   40 CONTINUE
      ML = 1
      MU = 1
      EPSFCN = 0.0e0
      MODE = 2
      FACTOR = 100.0e0
      ICOUNT = 0
      IFAIL = 1
      IREVCM = 0
*
   60 CALL C05NDF(IREVCM,N,X,FVEC,XTOL,ML,MU,EPSFCN,DIAG,MODE,FACTOR,
     +            FJAC,LDFJAC,R,LR,QTF,W,IFAIL)
*
      IF (IREVCM.EQ.1) THEN
         ICOUNT = ICOUNT + 1
*        Insert print statements here to monitor progess if desired.
         GO TO 60
      ELSE IF (IREVCM.EQ.2) THEN
*        Evaluate functions at given point
         DO 80 K = 1, N
            FVEC(K) = (THREE-TWO*X(K))*X(K) + ONE
            IF (K.GT.1) FVEC(K) = FVEC(K) - X(K-1)
            IF (K.LT.N) FVEC(K) = FVEC(K) - TWO*X(K+1)
   80    CONTINUE
```

```
            GO TO 60
         END IF
*
         WRITE (NOUT,*)
         IF (IFAIL.EQ.0) THEN
            FNORM = F06EJF(N,FVEC,1)
            WRITE (NOUT,99999) 'Final 2-norm of the residuals after',
        +     ICOUNT, ' iterations is ', FNORM
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Final approximate solution'
            WRITE (NOUT,99998) (X(J),J=1,N)
         ELSE
            WRITE (NOUT,99999) 'IFAIL =', IFAIL
            IF (IFAIL.GE.2) THEN
               WRITE (NOUT,*) 'Approximate solution'
               WRITE (NOUT,99998) (X(J),J=1,N)
            END IF
         END IF
         STOP
*
99999 FORMAT (1X,A,I4,A,e12.4)
99998 FORMAT (5X,3F12.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 C05NDF Example Program Results

 Final 2-norm of the residuals after  11 iterations is   0.1193E-07

 Final approximate solution
          -0.5707     -0.6816     -0.7017
          -0.7042     -0.7014     -0.6919
          -0.6658     -0.5960     -0.4164
```

C05PBF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05PBF is an easy-to-use routine to find a solution of a system of nonlinear equations by a modification of the Powell hybrid method. The user must provide the Jacobian.

2. Specification

```
      SUBROUTINE C05PBF (FCN, N, X, FVEC, FJAC, LDFJAC, XTOL, WA, LWA,
     1                   IFAIL)
      INTEGER          N, LDFJAC, LWA, IFAIL
      real             X(N), FVEC(N), FJAC(LDFJAC,N), XTOL, WA(LWA)
      EXTERNAL         FCN
```

3. Description

The system of equations is defined as:

$$f_i(x_1,x_2,\ldots,x_n) = 0, \qquad \text{for } i = 1,2,\ldots,n.$$

C05PBF is based upon the MINPACK routine HYBRJ1 (Moré *et al.* [1]). It chooses the correction at each step as a convex combination of the Newton and scaled gradient directions. Under reasonable conditions this guarantees global convergence for starting points far from the solution and a fast rate of convergence. The Jacobian is updated by the rank-1 method of Broyden. At the starting point the Jacobian is calculated, but it is not recalculated until the rank-1 method fails to produce satisfactory progress. For more details see Powell [2].

4. References

[1] MORÉ, J.J., GARBOW, B.S. and HILLSTROM, K.E.
User Guide for MINPACK-1.
Argonne National Laboratory, ANL-80-74.

[2] POWELL, M.J.D.
A Hybrid Method for Nonlinear Algebraic Equations.
In, 'Numerical Methods for Nonlinear Algebraic Equations', Rabinowitz, P. (Ed).
Gordon and Breach, 1970.

5. Parameters

1: FCN – SUBROUTINE, supplied by the user. *External Procedure*

Depending upon the value of IFLAG, FCN must either return the values of the functions f_i at a point x or return the Jacobian at x.

Its specification is:

```
      SUBROUTINE FCN(N, X, FVEC, FJAC, LDFJAC, IFLAG)
      INTEGER          N, LDFJAC, IFLAG
      real             X(N), FVEC(N), FJAC(LDFJAC,N)
```

1: N – INTEGER. *Input*

On entry: the number of equations, n.

2: X(N) – ***real*** array. *Input*

On entry: the components of the point x at which the functions or the Jacobian must be evaluated.

3: FVEC(N) – ***real*** array. *Output*

On exit: if IFLAG = 1 on entry, FVEC must contain the function values $f_i(x)$ (unless IFLAG is set to a negative value by FCN).

If IFLAG = 2 on entry, FVEC must not be changed.

4: FJAC(LDFJAC,N) – ***real*** array. *Output*

On exit: if IFLAG = 2 on entry, FJAC(i,j) must contain the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i,j = 1,2,...,n$ (unless IFLAG is set to a negative value by FCN).

If IFLAG = 1 on entry, FJAC must not be changed.

5: LDFJAC – INTEGER. *Input*

On entry: the first dimension of FJAC.

6: IFLAG – INTEGER. *Input/Output*

On entry: IFLAG = 1 or 2:

if IFLAG = 1, FVEC is to be updated;

if IFLAG = 2, FJAC is to be updated.

On exit: in general, IFLAG should not be reset by FCN. If, however, the user wishes to terminate execution (perhaps because some illegal point x has been reached) then IFLAG should be set to a negative integer. This value will be returned through IFAIL.

FCN must be declared as EXTERNAL in the (sub)program from which C05PBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: N – INTEGER. *Input*

On entry: the number of equations, n.

Constraint: N > 0.

3: X(N) – ***real*** array. *Input/Output*

On entry: an initial guess at the solution vector.

On exit: the final estimate of the solution vector.

4: FVEC(N) – ***real*** array. *Output*

On exit: the function values at the final point, X.

5: FJAC(LDFJAC,N) – ***real*** array. *Output*

On exit: the orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

6: LDFJAC – INTEGER. *Input*

On entry: the first dimension of the array FJAC as declared in the (sub)program from which C05PBF is called.

Constraint: LDFJAC ≥ N.

7: XTOL – ***real***. *Input*

On entry: the accuracy in X to which the solution is required.

Suggested value: the square root of the ***machine precision***.

Constraint: XTOL ≥ 0.0.

8: WA(LWA) – ***real*** array. *Workspace*
9: LWA – INTEGER. *Input*

On entry: the dimension of the array WA.

Constraint: LWA ≥ N×(N+13)/2.

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL < 0

A negative value of IFAIL indicates an exit from C05PBF because the user has set IFLAG negative in FCN. The value of IFAIL will be the same as the user's setting of IFLAG.

IFAIL = 1

On entry, N ≤ 0,
or LDFJAC < N,
or XTOL < 0.0,
or LWA < N×(N+13)/2.

IFAIL = 2

There have been 100×(N+1) evaluations of the functions. Consider restarting the calculation from the final point held in X.

IFAIL = 3

No further improvement in the approximate solution X is possible; XTOL is too small.

IFAIL = 4

The iteration is not making good progress. This failure exit may indicate that the system does not have a zero or that the solution is very close to the origin (see Section 7). Otherwise, rerunning C05PBF from a different starting point may avoid the region of difficulty.

7. Accuracy

If $\hat{x}$ is the true solution, C05PBF tries to ensure that

$$\|x-\hat{x}\|_2 \leq \text{XTOL}\times\|\hat{x}\|_2.$$

If this condition is satisfied with XTOL = 10^{-k} then the larger components of x have k significant decimal digits. There is a danger that the smaller components of x may have large relative errors, but the fast rate of convergence of C05PBF usually avoids the possibility.

If XTOL is less than ***machine precision*** and the above test is satisfied with the ***machine precision*** in place of XTOL, then the routine exits with IFAIL = 3.

Note: this convergence test is based purely on relative error, and may not indicate convergence if the solution is very close to the origin.

The test assumes that the functions and Jacobian are coded consistently and that the functions are reasonably well behaved. If these conditions are not satisfied then C05PBF may incorrectly indicate convergence. The coding of the Jacobian can be checked using C05ZAF. If the Jacobian is coded correctly, then the validity of the answer can be checked by rerunning C05PBF with a tighter tolerance.

8. Further Comments

The time required by C05PBF to solve a given problem depends on n, the behaviour of the functions, the accuracy requested and the starting point. The number of arithmetic operations executed by C05PBF is about $11.5\times n^2$ to process each evaluation of the functions and about $1.3\times n^3$ to process each evaluation of the Jacobian. Unless FCN can be evaluated quickly, the timing of C05PBF will be strongly influenced by the time spent in FCN.

Ideally the problem should be scaled so that, at the solution, the function values are of comparable magnitude.

9. Example

To determine the values $x_1,...,x_9$ which satisfy the tridiagonal equations:

$$(3-2x_1)x_1 - 2x_2 = -1$$
$$-x_{i-1} + (3-2x_i)x_i - 2x_{i+1} = -1, \qquad i = 2,3,...,8.$$
$$-x_8 + (3-2x_9)x_9 = -1.$$

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05PBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LDFJAC, LWA
      PARAMETER        (N=9,LDFJAC=N,LWA=(N*(N+13))/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             FNORM, TOL
      INTEGER          IFAIL, J
*     .. Local Arrays ..
      real             FJAC(LDFJAC,N), FVEC(N), WA(LWA), X(N)
*     .. External Functions ..
      real             F06EJF, X02AJF
      EXTERNAL         F06EJF, X02AJF
*     .. External Subroutines ..
      EXTERNAL         C05PBF, FCN
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05PBF Example Program Results'
      WRITE (NOUT,*)
*     The following starting values provide a rough solution.
      DO 20 J = 1, N
         X(J) = -1.0e0
   20 CONTINUE
      TOL = SQRT(X02AJF())
      IFAIL = 1
*
      CALL C05PBF(FCN,N,X,FVEC,FJAC,LDFJAC,TOL,WA,LWA,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         FNORM = F06EJF(N,FVEC,1)
         WRITE (NOUT,99999) 'Final 2-norm of the residuals =', FNORM
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final approximate solution'
         WRITE (NOUT,*)
         WRITE (NOUT,99998) (X(J),J=1,N)
      ELSE
         WRITE (NOUT,99997) 'IFAIL = ', IFAIL
         IF (IFAIL.GE.2) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Approximate solution'
```

```
            WRITE (NOUT,*)
            WRITE (NOUT,99998) (X(J),J=1,N)
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,e12.4)
99998 FORMAT (1X,3F12.4)
99997 FORMAT (1X,A,I2)
      END
*
      SUBROUTINE FCN(N,X,FVEC,FJAC,LDFJAC,IFLAG)
*     .. Parameters ..
      real              ZERO, ONE, TWO, THREE, FOUR
      PARAMETER         (ZERO=0.0e0,ONE=1.0e0,TWO=2.0e0,THREE=3.0e0,
     +                  FOUR=4.0e0)
*     .. Scalar Arguments ..
      INTEGER           IFLAG, LDFJAC, N
*     .. Array Arguments ..
      real              FJAC(LDFJAC,N), FVEC(N), X(N)
*     .. Local Scalars ..
      INTEGER           J, K
*     .. Executable Statements ..
      IF (IFLAG.NE.2) THEN
         DO 20 K = 1, N
            FVEC(K) = (THREE-TWO*X(K))*X(K) + ONE
            IF (K.GT.1) FVEC(K) = FVEC(K) - X(K-1)
            IF (K.LT.N) FVEC(K) = FVEC(K) - TWO*X(K+1)
   20    CONTINUE
      ELSE
         DO 60 K = 1, N
            DO 40 J = 1, N
               FJAC(K,J) = ZERO
   40       CONTINUE
            FJAC(K,K) = THREE - FOUR*X(K)
            IF (K.GT.1) FJAC(K,K-1) = -ONE
            IF (K.LT.N) FJAC(K,K+1) = -TWO
   60    CONTINUE
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C05PBF Example Program Results

Final 2-norm of the residuals =  0.1193E-07

Final approximate solution

     -0.5707     -0.6816     -0.7017
     -0.7042     -0.7014     -0.6919
     -0.6658     -0.5960     -0.4164
```

C05PCF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05PCF is a comprehensive routine to find a solution of a system of nonlinear equations by a modification of the Powell hybrid method. The user must provide the Jacobian.

2. Specification

```
      SUBROUTINE C05PCF (FCN, N, X, FVEC, FJAC, LDFJAC, XTOL, MAXFEV,
     1                   DIAG, MODE, FACTOR, NPRINT, NFEV, NJEV, R, LR,
     2                   QTF, W, IFAIL)
      INTEGER           N, LDFJAC, MAXFEV, MODE, NPRINT, NFEV, NJEV, LR,
     1                  IFAIL
      real              X(N), FVEC(N), FJAC(LDFJAC,N), XTOL, DIAG(N),
     1                  FACTOR, R(LR), QTF(N), W(N,4)
      EXTERNAL          FCN
```

3. Description

The system of equations is defined as:

$$f_i(x_1,x_2,...,x_n) = 0, \qquad \text{for } i = 1,2,...,n.$$

C05PCF is based upon the MINPACK routine HYBRJ (Moré *et al.* [1]). It chooses the correction at each step as a convex combination of the Newton and scaled gradient directions. Under reasonable conditions this guarantees global convergence from starting points far from the solution and a fast rate of convergence. The Jacobian is updated by the rank-1 method of Broyden. At the starting point the Jacobian is calculated, but it is not recalculated until the rank-1 method fails to produce satisfactory progress. For more details see Powell [2].

4. References

[1] MORÉ, J.J., GARBOW, B.S. and HILLSTROM, K.E.
User Guide for MINPACK-1.
Argonne National Laboratory, ANL-80-74.

[2] POWELL, M.J.D.
A Hybrid Method for Nonlinear Algebraic Equations.
In: 'Numerical Methods for Nonlinear Algebraic Equations', Rabinowitz, P. (ed.).
Gordon and Breach, 1970.

5. Parameters

1: FCN – SUBROUTINE, supplied by the user. *External Procedure*

Depending upon the value of IFLAG, FCN must either return the values of the functions f_i at a point x or return the Jacobian at x.

Its specification is:

```
SUBROUTINE FCN(N, X, FVEC, FJAC, LDFJAC, IFLAG)
INTEGER    N, LDFJAC, IFLAG
real       X(N), FVEC(N), FJAC(LDFJAC,N)
```

1: N – INTEGER. *Input*

On entry: the number of equations, n.

2: X(N) – ***real*** array. *Input*

On entry: the components of the point at which the functions or the Jacobian must be evaluated.

3: FVEC(N) – ***real*** array. *Output*

On exit: if IFLAG = 1 on entry, FVEC must contain the function values $f_i(x)$ (unless IFLAG is set to a negative value by FCN).

If IFLAG = 0 or 2 on entry, FVEC must not be changed.

4: FJAC(LDFJAC,N) – ***real*** array. *Output*

On exit: if IFLAG = 2 on entry, FJAC(i,j) must contain the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i,j = 1,2,...,n$ (unless IFLAG is set to a negative value by FCN).

If IFLAG = 0 or 1 on entry, FJAC must not be changed.

5: LDFJAC – INTEGER. *Input*

On entry: the first dimension of FJAC.

6: IFLAG – INTEGER. *Input/Output*

On entry: IFLAG = 0, 1 or 2:

if IFLAG = 0, X and FVEC are available for printing (see NPRINT below);

if IFLAG = 1, FVEC is to be updated;

if IFLAG = 2, FJAC is to be updated.

On exit: in general, IFLAG should not be reset by FCN. If, however, the user wishes to terminate execution (perhaps because some illegal point X has been reached), then IFLAG should be set to a negative integer. This value will be returned through IFAIL.

FCN must be declared as EXTERNAL in the (sub)program from which C05PCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: N – INTEGER. *Input*

On entry: the number of equations, n.

Constraint: N > 0.

3: X(N) – ***real*** array. *Input/Output*

On entry: an initial guess at the solution vector.

On exit: the final estimate of the solution vector.

4: FVEC(N) – ***real*** array. *Output*

On exit: the function values at the final point, X.

5: FJAC(LDFJAC,N) – ***real*** array. *Output*

On exit: the orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

6: LDFJAC – INTEGER. *Input*

On entry: the first dimension of the array FJAC as declared in the (sub)program from which C05PCF is called.

Constraint: LDFJAC ≥ N.

7: XTOL – ***real***. *Input*

On entry: the accuracy in X to which the solution is required.

Suggested value: the square root of the ***machine precision***.

Constraint: XTOL ≥ 0.0.

8: MAXFEV – INTEGER. *Input*

On entry: the maximum number of calls to FCN with IFLAG ≠ 0. C05PCF will exit with IFAIL = 2, if, at the end of an iteration, the number of calls to FCN exceeds MAXFEV.

Suggested value: MAXFEV = 100×(N+1).

Constraint: MAXFEV > 0.

9: DIAG(N) – ***real*** array. *Input/Output*

On entry: if MODE = 2 (see below), DIAG must contain multiplicative scale factors for the variables.

Constraint: DIAG(i) > 0.0 for i = 1,2,...,n.

On exit: the scale factors actually used (computed internally if MODE ≠ 2).

10: MODE – INTEGER. *Input*

On entry: indicates whether or not the user has provided scaling factors in DIAG. If MODE = 2, the scaling must have been specified in DIAG. Otherwise, the variables will be scaled internally.

11: FACTOR – ***real***. *Input*

On entry: a quantity to be used in determining the initial step bound. In most cases, FACTOR should lie between 0.1 and 100.0. (The step bound is FACTOR×$\|$DIAG×X$\|_2$ if this is non-zero; otherwise the bound is FACTOR.)

Suggested value: FACTOR = 100.0.

Constraint: FACTOR > 0.0.

12: NPRINT – INTEGER. *Input*

On entry: indicates whether or not special calls to FCN with IFLAG = 0 are to be made for printing purposes. If NPRINT ≤ 0, then no calls are made. If NPRINT > 0, then FCN is called at the beginning of the first iteration, every NPRINT iterations thereafter and immediately prior to the return from C05PCF.

13: NFEV – INTEGER. *Output*

On exit: the number of calls made to FCN to evaluate the functions.

14: NJEV – INTEGER. *Output*

On exit: the number of calls made to FCN to evaluate the Jacobian.

15: R(LR) – ***real*** array. *Output*

On exit: the upper triangular matrix R produced by the QR factorization of the final approximate Jacobian, stored row-wise.

16: LR – INTEGER. *Input*

On entry: the dimension of the array R.

Constraint: LR ≥ N×(N+1)/2.

17: QTF(N) – ***real*** array. *Output*

On exit: the vector $Q^T f$.

18: W(N,4) – ***real*** array. *Workspace*

19: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL < 0

This indicates an exit from C05PCF because the user has set IFLAG negative in FCN. The value of IFAIL will be the same as the user's setting of IFLAG.

IFAIL = 1

On entry, N ≤ 0,
or XTOL < 0.0,
or MAXFEV ≤ 0,
or FACTOR ≤ 0.0,
or LDFJAC < N,
or LR < N×(N+1)/2,
or MODE = 2 and DIAG(i) ≤ 0.0 for some i, i = 1,2,...,N.

IFAIL = 2

There have been MAXFEV evaluations of FCN to evaluate the functions. Consider restarting the calculation from the final point held in X.

IFAIL = 3

No further improvement in the approximate solution X is possible; XTOL is too small.

IFAIL = 4

The iteration is not making good progress, as measured by the improvement from the last 5 Jacobian evaluations.

IFAIL = 5

The iteration is not making good progress, as measured by the improvement from the last 10 iterations.

The values IFAIL = 4 and IFAIL = 5 may indicate that the system does not have a zero, or that the solution is very close to the origin (see Section 7). Otherwise, rerunning C05PCF from a different starting point may avoid the region of difficulty.

7. Accuracy

If $\hat{x}$ is the true solution and D denotes the diagonal matrix whose entries are defined by the array DIAG then C05PCF tries to ensure that

$$\|D\times(x-\hat{x})\|_2 \leq \text{XTOL}\times\|D\hat{x}\|_2.$$

If this condition is satisfied with XTOL = 10^{-k} then the larger components of Dx have k significant decimal digits. There is a danger that the smaller components of Dx may have large relative errors, but the fast rate of convergence of C05PCF usually avoids this possibility.

If XTOL is less than the ***machine precision*** and the above test is satisfied with the ***machine precision*** in place of XTOL, then the routine exits with IFAIL = 3.

Note: this convergence test is based purely on relative error, and may not indicate convergence if the solution is very close to the origin.

The test assumes that the functions and the Jacobian are coded consistently and that the functions are reasonably well behaved. If these conditions are not satisfied then C05PCF may incorrectly indicate convergence. The coding of the Jacobian can be checked using C05ZAF. If the Jacobian is coded correctly, then the validity of the answer can be checked by rerunning C05PCF with a tighter tolerance.

8. Further Comments

The time required by C05PCF to solve a given problem depends on n, the behaviour of the functions, the accuracy requested and the starting point. The number of arithmetic operations executed by C05PCF is about $11.5\times n^2$ to process each evaluation of the functions and about $1.3\times n^3$ to process each evaluation of the Jacobian. Unless FCN can be evaluated quickly, the timing of C05PCF will be strongly influenced by the time spent in FCN.

Ideally the problem should be scaled so that at the solution the function values are of comparable magnitude.

9. Example

To determine the values $x_1,\ldots,x_9$ which satisfy the tridiagonal equations:

$$(3-2x_1)x_1 - 2x_2 = -1.$$
$$-x_{i-1} + (3-2x_i)x_i - 2x_{i+1} = -1, \qquad i = 2,3,\ldots,8.$$
$$-x_8 + (3-2x_9)x_9 = -1.$$

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05PCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LDFJAC, LR
      PARAMETER        (N=9,LDFJAC=N,LR=(N*(N+1))/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             FACTOR, FNORM, XTOL
      INTEGER          IFAIL, J, MAXFEV, MODE, NFEV, NJEV, NPRINT
*     .. Local Arrays ..
      real             DIAG(N), FJAC(LDFJAC,N), FVEC(N), QTF(N), R(LR),
     +                 W(N,4), X(N)
*     .. External Functions ..
      real             F06EJF, X02AJF
      EXTERNAL         F06EJF, X02AJF
*     .. External Subroutines ..
      EXTERNAL         C05PCF, FCN
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05PCF Example Program Results'
      WRITE (NOUT,*)
*     The following starting values provide a rough solution.
      DO 20 J = 1, N
         X(J) = -1.0e0
   20 CONTINUE
      XTOL = SQRT(X02AJF())
      DO 40 J = 1, N
         DIAG(J) = 1.0e0
   40 CONTINUE
      MAXFEV = 1000
      MODE = 2
      FACTOR = 100.0e0
      NPRINT = 0
      IFAIL = 1
```

```
*
      CALL C05PCF(FCN,N,X,FVEC,FJAC,LDFJAC,XTOL,MAXFEV,DIAG,MODE,FACTOR,
     +            NPRINT,NFEV,NJEV,R,LR,QTF,W,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         FNORM = F06EJF(N,FVEC,1)
         WRITE (NOUT,99999) 'Final 2-norm of the residuals =', FNORM
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Number of function evaluations =', NFEV
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Number of Jacobian evaluations =', NJEV
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final approximate solution'
         WRITE (NOUT,*)
         WRITE (NOUT,99997) (X(J),J=1,N)
      ELSE
         WRITE (NOUT,99996) 'IFAIL = ', IFAIL
         IF (IFAIL.GT.2) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,*) 'Approximate solution:'
            WRITE (NOUT,*)
            WRITE (NOUT,99997) (X(J),J=1,N)
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,e12.4)
99998 FORMAT (1X,A,I10)
99997 FORMAT (1X,3F12.4)
99996 FORMAT (1X,A,I2)
      END
*
      SUBROUTINE FCN(N,X,FVEC,FJAC,LDFJAC,IFLAG)
*     .. Parameters ..
      real              ZERO, ONE, TWO, THREE, FOUR
      PARAMETER         (ZERO=0.0e0,ONE=1.0e0,TWO=2.0e0,THREE=3.0e0,
     +                  FOUR=4.0e0)
*     .. Scalar Arguments ..
      INTEGER           IFLAG, LDFJAC, N
*     .. Array Arguments ..
      real              FJAC(LDFJAC,N), FVEC(N), X(N)
*     .. Local Scalars ..
      INTEGER           J, K
*     .. Executable Statements ..
      IF (IFLAG.EQ.0) THEN
*
*        Insert print statements here when NPRINT is positive.
*
         RETURN
      ELSE
         IF (IFLAG.NE.2) THEN
            DO 20 K = 1, N
               FVEC(K) = (THREE-TWO*X(K))*X(K) + ONE
               IF (K.GT.1) FVEC(K) = FVEC(K) - X(K-1)
               IF (K.LT.N) FVEC(K) = FVEC(K) - TWO*X(K+1)
   20       CONTINUE
         ELSE
            DO 60 K = 1, N
               DO 40 J = 1, N
                  FJAC(K,J) = ZERO
   40          CONTINUE
               FJAC(K,K) = THREE - FOUR*X(K)
               IF (K.GT.1) FJAC(K,K-1) = -ONE
               IF (K.LT.N) FJAC(K,K+1) = -TWO
   60       CONTINUE
         END IF
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C05PCF Example Program Results

Final 2-norm of the residuals =  0.1193E-07

Number of function evaluations =        11

Number of Jacobian evaluations =         1

Final approximate solution

     -0.5707      -0.6816      -0.7017
     -0.7042      -0.7014      -0.6919
     -0.6658      -0.5960      -0.4164
```

C05PDF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05PDF is a comprehensive reverse communication routine to find a solution of a system of nonlinear equations by a modification of the Powell hybrid method. The user must provide the Jacobian.

2. Specification

```
      SUBROUTINE C05PDF (IREVCM, N, X, FVEC, FJAC, LDFJAC, XTOL, DIAG,
     1                   MODE, FACTOR, R, LR, QTF, W, IFAIL)
      INTEGER          IREVCM, N, LDFJAC, MODE, LR, IFAIL
      real             X(N), FVEC(N), FJAC(LDFJAC, N), XTOL, DIAG(N),
     1                 FACTOR, R(LR), QTF(N), W(N,4)
```

3. Description

The system of equations is defined as:

$f_i(x_1,x_2,...,x_n) = 0$, for $i = 1,2,...,n$.

C05PDF is based upon the MINPACK routine HYBRJ (Moré *et al.* [1]). It chooses the correction at each step as a convex combination of the Newton and scaled gradient directions. Under reasonable conditions this guarantees global convergence from starting points far from the solution and a fast rate of convergence. The Jacobian is updated by the rank-1 method of Broyden. The Jacobian is requested to be supplied at the start of the computations, but it is not requested again. For more details see Powell [2].

4. References

[1] MORÉ, J.J., GARBOW, B.S. and HILLSTROM, K.E.
User Guide for MINPACK-1.
Argonne National Laboratory, ANL-80-74.

[2] POWELL, M.J.D.
A Hybrid Method for Nonlinear Algebraic Equations.
In: 'Numerical Methods for Nonlinear Algebraic Equations', Rabinowitz, P. (ed.).
Gordon and Breach, 1970.

5. Parameters

Note: this routine uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the **parameter IREVCM.** Between intermediate exits and re-entries, **all parameters other than FVEC and FJAC must remain unchanged.**

1: IREVCM – INTEGER. *Input/Output*

On initial entry: IREVCM must have the value 0.

On intermediate exit: IREVCM specifies what action the user must take before re-entering C05PDF **with** IREVCM **unchanged.** The value of IREVCM should be interpreted as follows:

IREVCM = 1

indicates the start of a new iteration. No action is required by the user but X and FVEC are available for printing.

IREVCM = 2

indicates that before re-entry to C05PDF, FVEC must contain the function value $f_i(x)$.

IREVCM = 3

indicates that before re-entry to C05PDF, FJAC(i,j) must contain the value of $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i,j = 1,2,...,n$.

On final exit: IREVCM = 0, and the algorithm has terminated.

Constraint: IREVCM = 0, 1, 2 or 3.

2: N – INTEGER. *Input*

On initial entry: the number of equations, n.

Constraint: N > 0.

3: X(N) – ***real*** array. *Input/Output*

On initial entry: X(j) must be set to a guess at the jth component of the solution, for $j = 1,2,...,n$.

On intermediate exit: X contains the current point.

On final exit: the final estimate of the solution vector.

4: FVEC(N) – ***real*** array. *Input/Output*

On initial entry: FVEC must be set to the values of the functions evaluated at the initial point X.

On intermediate re-entry: if IREVCM ≠ 2, FVEC must not be changed. If IREVCM = 2, FVEC must be set to the values of the functions computed at the current point X.

On final exit: the function values at the final point, X.

5: FJAC(LDFJAC,N) – ***real*** array. *Input/Output*

On initial entry: FJAC must be set to the values of the Jacobian evaluated at the initial point X.

On intermediate re-entry: if IREVCM ≠ 3, FJAC must not be changed. If IREVCM = 3, FJAC must be set to the value of the Jacobian computed at the current point X.

On final exit: the orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

6: LDFJAC – INTEGER. *Input*

On initial entry: the first dimension of the array FJAC as declared in the (sub)program from which C05PDF is called.

Constraint: LDFJAC ≥ N.

7: XTOL – ***real***. *Input*

On initial entry: the accuracy in X to which the solution is required.

Suggested value: the square root of the ***machine precision***.

Constraint: XTOL ≥ 0.0.

8: DIAG(N) – ***real*** array. *Input/Output*

On initial entry: if MODE = 2 (see below), DIAG must contain multiplicative scale factors for the variables.

Constraint: DIAG(i) > 0.0 for $i = 1,2,...,n$.

On intermediate exit: the scale factors actually used (computed internally if MODE ≠ 2).

9: MODE – INTEGER. *Input*

On initial entry: indicates whether or not the user has provided scaling factors in DIAG. If MODE = 2 the scale factors must be supplied in DIAG. Otherwise, the variables will be scaled internally.

10: FACTOR – ***real***. *Input*

On initial entry: a quantity to be used in determining the initial step bound. In most cases, FACTOR should lie between 0.1 and 100.0. (The step bound is FACTOR×$\|$DIAG×X$\|_2$ if this is non-zero; otherwise the bound is FACTOR.)

Suggested value: FACTOR = 100.0.

Constraint: FACTOR > 0.0.

11: R(LR) – ***real*** array. *Output*

On final exit: the upper triangular matrix R produced by the QR factorization of the final approximate Jacobian, stored row-wise.

12: LR – INTEGER. *Input*

On initial entry: the dimension of the array R as declared in the (sub)program from which C05PDF is called.

Constraint: LR ≥ N×(N+1)/2.

13: QTF(N) – ***real*** array. *Output*

On final exit: the vector $Q^T f$.

14: W(N,4) – ***real*** array. *Workspace*

15: IFAIL – INTEGER. *Input/Output*

On initial entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On final exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, N ≤ 0,
or XTOL < 0.0,
or FACTOR ≤ 0.0,
or LDFJAC < N,
or LR < N×(N+1)/2,
or MODE = 2 and DIAG(i) ≤ 0.0 for some i, i = 1,2,...,N.

IFAIL = 2

On entry, IREVCM < 0 or IREVCM > 3.

IFAIL = 3

No further improvement in the approximate solution X is possible; XTOL is too small.

IFAIL = 4

The iteration is not making good progress, as measured by the improvement from the last 5 Jacobian evaluations.

IFAIL = 5

The iteration is not making good progress, as measured by the improvement from the last 10 iterations.

The values IFAIL = 4 and IFAIL = 5 may indicate that the system does not have a zero, or that the solution is very close to the origin (see Section 7). Otherwise, rerunning C05PDF from a different starting point may avoid the region of difficulty.

7. Accuracy

If $\hat{x}$ is the true solution and D denotes the diagonal matrix whose entries are defined by the array DIAG then C05PDF tries to ensure that

$$\|D(x-\hat{x})\|_2 \leq \text{XTOL} \times \|D\hat{x}\|_2.$$

If this condition is satisfied with XTOL = 10^{-k} then the larger components of Dx have k significant decimal digits. There is a danger that the smaller components of Dx may have large relative errors, but the fast rate of convergence of C05PDF usually avoids this possibility.

If XTOL is less than ***machine precision*** and the above test is satisfied with the ***machine precision*** in place of XTOL, then the routine exits with IFAIL = 3.

Note that this convergence test is based purely on relative error, and may not indicate convergence if the solution is very close to the origin.

The test assumes that the functions and the Jacobian are coded consistently and that the functions are reasonably well behaved. If these conditions are not satisfied then C05PDF may incorrectly indicate convergence. The coding of the Jacobian can be checked using C05ZAF. If the Jacobian is coded correctly, then the validity of the answer can be checked by rerunning C05PDF with a tighter tolerance.

8. Further Comments

The time required by C05PDF to solve a given problem depends on n, the behaviour of the functions, the accuracy requested and the starting point. The number of arithmetic operations executed by C05PDF is about $11.5 \times n^2$ to process each evaluation of the functions and about $1.3 \times n^3$ to process each evaluation of the Jacobian. The timing of C05PDF is strongly influenced by the time spent in the evaluation of the functions and the Jacobian.

Ideally the problem should be scaled so that at the solution the function values are of comparable magnitude.

9. Example

To determine the values $x_1,\ldots,x_9$ which satisfy the tridiagonal equations:

$$(3-2x_1)x_1 - 2x_2 = -1$$
$$-x_{i-1} + (3-2x_i)x_i - 2x_{i+1} = -1, \qquad i = 2,3,\ldots,8$$
$$-x_8 + (3-2x_9)x_9 = -1.$$

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05PDF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, LDFJAC, LR
      PARAMETER        (N=9,LDFJAC=N,LR=(N*(N+1))/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      real             ZERO, ONE, TWO, THREE, FOUR
      PARAMETER        (ZERO=0.0e0,ONE=1.0e0,TWO=2.0e0,THREE=3.0e0,
     +                 FOUR=4.0e0)
```

```
*     .. Local Scalars ..
      real              FACTOR, FNORM, XTOL
      INTEGER           ICOUNT, IFAIL, IREVCM, J, K, MODE
*     .. Local Arrays ..
      real              DIAG(N), FJAC(LDFJAC,N), FVEC(N), QTF(N), R(LR),
     +                  W(N,4), X(N)
*     .. External Functions ..
      real              F06EJF, X02AJF
      EXTERNAL          F06EJF, X02AJF
*     .. External Subroutines ..
      EXTERNAL          C05PDF
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05PDF Example Program Results'
*     The following starting values provide a rough solution.
      DO 20 J = 1, N
         X(J) = -1.0e0
   20 CONTINUE
      XTOL = SQRT(X02AJF())
      DO 40 J = 1, N
         DIAG(J) = 1.0e0
   40 CONTINUE
      MODE = 2
      FACTOR = 100.0e0
      ICOUNT = 0
      IFAIL = 1
      IREVCM = 0
*
   60 CALL C05PDF(IREVCM,N,X,FVEC,FJAC,LDFJAC,XTOL,DIAG,MODE,FACTOR,R,
     +            LR,QTF,W,IFAIL)
*
      IF (IREVCM.EQ.1) THEN
         ICOUNT = ICOUNT + 1
*        Insert print statements here to monitor progess if desired
         GO TO 60
      ELSE IF (IREVCM.EQ.2) THEN
*        Evaluate functions at current point
         DO 80 K = 1, N
            FVEC(K) = (THREE-TWO*X(K))*X(K) + ONE
            IF (K.GT.1) FVEC(K) = FVEC(K) - X(K-1)
            IF (K.LT.N) FVEC(K) = FVEC(K) - TWO*X(K+1)
   80    CONTINUE
         GO TO 60
      ELSE IF (IREVCM.EQ.3) THEN
*        Evaluate Jacobian at current point
         DO 120 K = 1, N
            DO 100 J = 1, N
               FJAC(K,J) = ZERO
  100       CONTINUE
            FJAC(K,K) = THREE - FOUR*X(K)
            IF (K.NE.1) FJAC(K,K-1) = -ONE
            IF (K.NE.N) FJAC(K,K+1) = -TWO
  120    CONTINUE
         GO TO 60
      END IF
*
      WRITE (NOUT,*)
      IF (IFAIL.EQ.0) THEN
         FNORM = F06EJF(N,FVEC,1)
         WRITE (NOUT,99999) 'Final 2 norm of the residuals after',
     +     ICOUNT, ' iterations is ', FNORM
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final approximate solution'
         WRITE (NOUT,99998) (X(J),J=1,N)
```

```
      ELSE
         WRITE (NOUT,99999) 'IFAIL =', IFAIL
         IF (IFAIL.GT.2) THEN
            WRITE (NOUT,*) 'Approximate solution'
            WRITE (NOUT,99998) (X(J),J=1,N)
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,I4,A,e12.4)
99998 FORMAT (5X,3F12.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C05PDF Example Program Results

Final 2 norm of the residuals after  11 iterations is   0.1193E-07

Final approximate solution
         -0.5707     -0.6816     -0.7017
         -0.7042     -0.7014     -0.6919
         -0.6658     -0.5960     -0.4164
```

C05ZAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C05ZAF checks the user-provided gradients of a set of non-linear functions in several variables, for consistency with the functions themselves. The routine must be called twice.

2. Specification

```
      SUBROUTINE C05ZAF (M, N, X, FVEC, FJAC, LDFJAC, XP, FVECP, MODE,
     1                   ERR)
      INTEGER          M, N, LDFJAC, MODE
      real             X(N), FVEC(M), FJAC(LDFJAC,N), XP(N), FVECP(M),
     1                 ERR(M)
```

3. Description

C05ZAF is based upon the MINPACK routine CHKDER (Moré *et al.* [1]). It checks the *i*th gradient for consistency with the *i*th function by computing a forward-difference approximation along a suitably chosen direction and comparing this approximation with the user-supplied gradient along the same direction. The principal characteristic of C05ZAF is its invariance under changes in scale of the variables or functions.

4. References

[1] MORÉ, J.J., GARBOW, B.S. and HILLSTROM, K.E.
User Guide for MINPACK-1.
Argonne National Laboratory, ANL-80-74.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number of functions.

2: N – INTEGER. *Input*

On entry: the number of variables. For use with C05PBF and C05PCF, M = N.

3: X(N) – ***real*** array. *Input*

On entry: the components of a point x, at which the consistency check is to be made. (See Section 8.)

4: FVEC(M) – ***real*** array. *Input*

On entry: when MODE = 2, FVEC must contain the functions evaluated at x.

5: FJAC(LDFJAC,N) – ***real*** array. *Input*

On entry: when MODE = 2, FJAC must contain the user-supplied gradients. (The *i*th row of FJAC must contain the gradient of the *i*th function evaluated at the point x.)

6: LDFJAC – INTEGER. *Input*

On entry: the first dimension of the array FJAC as declared in the (sub)program from which C05ZAF is called.

Constraint: LDFJAC $\geq$ M.

7: XP(N) – ***real*** array. *Output*

On exit: when MODE = 1, XP is set to a neighbouring point to X.

8: FVECP(M) – ***real*** array. *Input*

On entry: when MODE = 2, FVECP must contain the functions evaluated at XP.

9: MODE – INTEGER. *Input*

On entry: the value 1 on the first call and the value 2 on the second call of C05ZAF.

10: ERR(M) – ***real*** array. *Output*

On exit: when MODE = 2, ERR contains measures of correctness of the respective gradients. If there is no loss of significance (see Section 8), then if ERR(i) is 1.0 the ith user-supplied gradient is correct, whilst if ERR(i) is 0.0 the ith gradient is incorrect. For values of ERR(i) between 0.0 and 1.0 the categorisation is less certain. In general, a value of ERR(i) > 0.5 indicates that the ith gradient is probably correct.

6. Error Indicators and Warnings

None.

7. Accuracy

See below.

8. Further Comments

The time required by C05ZAF increases with M and N.

C05ZAF does not perform reliably if cancellation or rounding errors cause a severe loss of significance in the evaluation of a function. Therefore, none of the components of x should be unusually small (in particular, zero) or any other value which may cause loss of significance. The relative differences between corresponding elements of FVECP and FVEC should be at least two orders of magnitude greater than the ***machine precision***.

9. Example

This example checks the Jacobian matrix for a problem with 15 functions of 3 variables. The results indicate that the first 7 gradients are probably incorrect (this is caused by a deliberate error in the code to calculate the Jacobian).

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C05ZAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          M, N, LDFJAC
      PARAMETER        (M=15,N=3,LDFJAC=M)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, MODE
*     .. Local Arrays ..
      real             ERR(M), FJAC(LDFJAC,N), FVEC(M), FVECP(M), X(N),
     +                 XP(N)
*     .. External Subroutines ..
      EXTERNAL         C05ZAF, FCN
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C05ZAF Example Program Results'
      X(1) = 9.2e-1
      X(2) = 1.3e-1
      X(3) = 5.4e-1
      MODE = 1
*
      CALL C05ZAF(M,N,X,FVEC,FJAC,LDFJAC,XP,FVECP,MODE,ERR)
*
      CALL FCN(M,N,X,FVEC,FJAC,LDFJAC,1)
      CALL FCN(M,N,X,FVEC,FJAC,LDFJAC,2)
      CALL FCN(M,N,XP,FVECP,FJAC,LDFJAC,1)
*
      MODE = 2
*
      CALL C05ZAF(M,N,X,FVEC,FJAC,LDFJAC,XP,FVECP,MODE,ERR)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) '     FVEC at X = ', (X(I),I=1,N)
      WRITE (NOUT,*)
      WRITE (NOUT,99998) (FVEC(I),I=1,M)
      WRITE (NOUT,*)
      WRITE (NOUT,99999) '     FVECP at XP = ', (XP(I),I=1,N)
      WRITE (NOUT,*)
      WRITE (NOUT,99998) (FVECP(I),I=1,M)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '     ERR'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) (ERR(I),I=1,M)
      STOP
*
99999 FORMAT (1X,A,3F12.7)
99998 FORMAT (5X,3F12.4)
      END
*
      SUBROUTINE FCN(M,N,X,FVEC,FJAC,LDFJAC,IFLAG)
*     .. Parameters ..
      INTEGER          M1
      PARAMETER        (M1=15)
*     .. Scalar Arguments ..
      INTEGER          IFLAG, LDFJAC, M, N
*     .. Array Arguments ..
      real             FJAC(LDFJAC,N), FVEC(M), X(N)
*     .. Local Scalars ..
      real             TMP1, TMP2, TMP3, TMP4
      INTEGER          I
*     .. Local Arrays ..
      real             Y(M1)
*     .. Data statements ..
      DATA             Y/1.4e-1, 1.8e-1, 2.2e-1, 2.5e-1, 2.9e-1, 3.2e-1,
     +                 3.5e-1, 3.9e-1, 3.7e-1, 5.8e-1, 7.3e-1, 9.6e-1,
     +                 1.34e0, 2.1e0, 4.39e0/
*     .. Executable Statements ..
      IF (IFLAG.NE.2) THEN
         DO 20 I = 1, M
            TMP1 = I
            TMP2 = M + 1 - I
            TMP3 = TMP1
            IF (I.GT.(M+1)/2) TMP3 = TMP2
            FVEC(I) = Y(I) - (X(1)+TMP1/(X(2)*TMP2+X(3)*TMP3))
   20    CONTINUE
      ELSE
         DO 40 I = 1, M
            TMP1 = I
            TMP2 = M + 1 - I
*
```

```
*           Error introduced into next statement for illustration.
*           Corrected statement should read    TMP3 = TMP1 .
*
            TMP3 = TMP2
            IF (I.GT.(M+1)/2) TMP3 = TMP2
            TMP4 = (X(2)*TMP2+X(3)*TMP3)**2
            FJAC(I,1) = -1.0e0
            FJAC(I,2) = TMP1*TMP2/TMP4
            FJAC(I,3) = TMP1*TMP3/TMP4
   40     CONTINUE
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 C05ZAF Example Program Results

      FVEC at X =    0.9200000   0.1300000   0.5400000

          -1.1816     -1.4297     -1.6063
          -1.7453     -1.8407     -1.9216
          -1.9841     -2.0225     -2.4690
          -2.8276     -3.4736     -4.4376
          -6.0477     -9.2678    -18.9181

      FVECP at XP =    0.9200000   0.1300000   0.5400000

          -1.1816     -1.4297     -1.6063
          -1.7453     -1.8407     -1.9216
          -1.9841     -2.0225     -2.4690
          -2.8276     -3.4736     -4.4376
          -6.0477     -9.2678    -18.9181

      ERR

           0.1120      0.0976      0.0949
           0.0979      0.1053      0.1197
           0.1498      1.0000      0.9950
           1.0000      1.0000      1.0000
           0.9917      1.0000      0.9710
```

Chapter C06 – Summation of Series

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
C06BAF	10	Acceleration of convergence of sequence, Shanks' transformation and epsilon algorithm
C06DBF	6	Sum of a Chebyshev series
C06EAF	8	Single 1-D real discrete Fourier transform, no extra workspace
C06EBF	8	Single 1-D Hermitian discrete Fourier transform, no extra workspace
C06ECF	8	Single 1-D complex discrete Fourier transform, no extra workspace
C06EKF	11	Circular convolution or correlation of two real vectors, no extra workspace
C06FAF	8	Single 1-D real discrete Fourier transform, extra workspace for greater speed
C06FBF	8	Single 1-D Hermitian discrete Fourier transform, extra workspace for greater speed
C06FCF	8	Single 1-D complex discrete Fourier transform, extra workspace for greater speed
C06FFF	11	1-D complex discrete Fourier transform of multi-dimensional data
C06FJF	11	Multi-dimensional complex discrete Fourier transform of multi-dimensional data
C06FKF	11	Circular convolution or correlation of two real vectors, extra workspace for greater speed
C06FPF	12	Multiple 1-D real discrete Fourier transforms
C06FQF	12	Multiple 1-D Hermitian discrete Fourier transforms
C06FRF	12	Multiple 1-D complex discrete Fourier transforms
C06FUF	13	2-D complex discrete Fourier transform
C06FXF	17	3-D complex discrete Fourier transform
C06GBF	8	Complex conjugate of Hermitian sequence
C06GCF	8	Complex conjugate of complex sequence
C06GQF	12	Complex conjugate of multiple Hermitian sequences
C06GSF	12	Convert Hermitian sequences to general complex sequences
C06HAF	13	Discrete sine transform
C06HBF	13	Discrete cosine transform
C06HCF	13	Discrete quarter-wave sine transform
C06HDF	13	Discrete quarter-wave cosine transform
C06LAF	12	Inverse Laplace transform, Crump's method
C06LBF	14	Inverse Laplace transform, modified Weeks' method
C06LCF	14	Evaluate inverse Laplace transform as computed by C06LBF

Chapter C06

Summation of Series

Contents

1 Scope of the Chapter

This chapter is concerned with the following tasks:

(1) Calculating the **discrete Fourier transform** of a sequence of real or complex data values, and applying it to calculate convolutions and correlations.
(2) Calculating the **inverse Laplace transform** of a user-supplied function.
(3) Direct summation of orthogonal series.
(4) Acceleration of convergence of a sequence of real values.

2 Background to the Problems

2.1 Discrete Fourier Transforms

2.1.1 Complex transforms

Most of the routines in this chapter calculate the finite **discrete Fourier transform** (DFT) of a sequence of n complex numbers z_j, for $j = 0, 1, \ldots, n-1$. The transform is defined by:

$$\hat{z}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \exp\left(-i\frac{2\pi jk}{n}\right) \tag{1}$$

for $k = 0, 1, \ldots, n-1$. Note that equation (1) makes sense for all integral k and with this extension $\hat{z}_k$ is periodic with period n, i.e., $\hat{z}_k = \hat{z}_{k\pm n}$, and in particular $\hat{z}_{-k} = \hat{z}_{n-k}$.

If we write $z_j = x_j + iy_j$ and $\hat{z}_k = a_k + ib_k$, then the definition of $\hat{z}_k$ may be written in terms of sines and cosines as:

$$a_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \left(x_j \cos\left(\frac{2\pi jk}{n}\right) + y_j \sin\left(\frac{2\pi jk}{n}\right)\right)$$

$$b_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \left(y_j \cos\left(\frac{2\pi jk}{n}\right) - x_j \sin\left(\frac{2\pi jk}{n}\right)\right).$$

The original data values z_j may conversely be recovered from the transform $\hat{z}_k$ by an **inverse discrete Fourier transform**:

$$z_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \hat{z}_k \exp\left(+i\frac{2\pi jk}{n}\right) \tag{2}$$

for $j = 0, 1, \ldots, n-1$. If we take the complex conjugate of (2), we find that the sequence $\bar{z}_j$ is the DFT of the sequence $\bar{\hat{z}}_k$. Hence the inverse DFT of the sequence $\hat{z}_k$ may be obtained by: taking the complex conjugates of the $\hat{z}_k$; performing a DFT; and taking the complex conjugates of the result.

The definition (1) of a one-dimensional transform can easily be extended to multi-dimensional transforms. For example, in two dimensions we have:

$$\hat{z}_{k_1k_2} = \frac{1}{\sqrt{n_1 n_2}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} z_{j_1j_2} \exp\left(-i\frac{2\pi j_1 k_1}{n_1}\right) \exp\left(-i\frac{2\pi j_2 k_2}{n_2}\right).$$

Note. Definitions of the discrete Fourier transform vary. Sometimes (2) is used as the definition of the DFT, and (1) as the definition of the inverse. Also the scale-factor of $\frac{1}{\sqrt{n}}$ may be omitted in the definition of the DFT, and replaced by $\frac{1}{n}$ in the definition of the inverse.

2.1.2 Real transforms

If the original sequence is purely real valued, i.e., $z_j = x_j$, then

$$\hat{z}_k = a_k + ib_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \exp\left(-i\frac{2\pi jk}{n}\right)$$

and $\hat{z}_{n-k}$ is the complex conjugate of $\hat{z}_k$. Thus the DFT of a real sequence is a particular type of complex sequence, called a **Hermitian** sequence, or **half-complex** or **conjugate symmetric**, with the properties:

$$a_{n-k} = a_k \qquad b_{n-k} = -b_k \qquad b_0 = 0$$

and, if n is even, $b_{n/2} = 0$.

Thus a Hermitian sequence of n complex data values can be represented by only n, rather than $2n$, independent real values. This can obviously lead to economies in storage, the following scheme being used in this chapter: the real parts a_k for $0 \leq k \leq n/2$ are stored in normal order in the first $n/2+1$ locations of an array X of length n; the corresponding non-zero imaginary parts are stored in reverse order in the remaining locations of X. In other words, if X is declared with bounds $(0{:}n-1)$ in the user's (sub)program, the real and imaginary parts of $\hat{z}_k$ are stored as follows:

	if $n = 2s$	if $n = 2s-1$
X(0)	a_0	a_0
X(1)	a_1	a_1
X(2)	a_2	a_2
⋮	⋮	⋮
X($s-1$)	a_{s-1}	a_{s-1}
X(s)	a_s	b_{s-1}
X($s+1$)	b_{s-1}	b_{s-2}
⋮	⋮	⋮
X($n-2$)	b_2	b_2
X($n-1$)	b_1	b_1

Hence

$$x_j = \frac{1}{\sqrt{n}}\left(a_0 + 2\sum_{k=1}^{n/2-1}\left(a_k \cos\left(\frac{2\pi jk}{n}\right) - b_k \sin\left(\frac{2\pi jk}{n}\right)\right) + a_{n/2}\right)$$

where $a_{n/2} = 0$ if n is odd.

2.1.3 Real symmetric transforms

In many applications the sequence x_j will not only be real, but may also possess additional symmetries which we may exploit to reduce further the computing time and storage requirements. For example, if the sequence x_j is **odd** ($x_j = -x_{n-j}$), then the discrete Fourier transform of x_j contains only sine terms. Rather than compute the transform of an odd sequence, we define the **sine transform** of a real sequence by

$$\hat{x}_k = \sqrt{\frac{2}{n}}\sum_{j=1}^{n-1} x_j \sin\left(\frac{\pi jk}{n}\right),$$

which could have been computed using the Fourier transform of a real odd sequence of length $2n$. In this case the x_j are arbitrary, and the symmetry only becomes apparent when the sequence is extended. Similarly we define the **cosine transform** of a real sequence by

$$\hat{x}_k = \sqrt{\frac{2}{n}}\left\{\frac{1}{2}x_0 + \sum_{j=1}^{n-1} x_j \cos\left(\frac{\pi jk}{n}\right) + \frac{1}{2}(-1)^k x_n\right\}$$

which could have been computed using the Fourier transform of a real **even** sequence of length $2n$.

In addition to these 'half-wave' symmetries described above, sequences arise in practice with 'quarter-wave' symmetries. We define the **quarter-wave sine transform** by

$$\hat{x}_k = \frac{1}{\sqrt{n}}\left\{\sum_{j=1}^{n-1} x_j \sin\left(\frac{\pi j(2k-1)}{2n}\right) + \frac{1}{2}(-1)^{k-1} x_n\right\}$$

which could have been computed using the Fourier transform of a real sequence of length $4n$ of the form

$$(0, x_1, \ldots, x_n, x_{n-1}, \ldots, x_1, 0, -x_1, \ldots, -x_n, -x_{n-1}, \ldots, -x_1).$$

Similarly we may define the **quarter-wave cosine transform** by

$$\hat{x}_k = \frac{1}{\sqrt{n}} \left\{ \frac{1}{2} x_0 + \sum_{j=1}^{n-1} x_j \cos\left(\frac{\pi j(2k-1)}{2n} \right) \right\}$$

which could have been computed using the Fourier transform of a real sequence of length $4n$ of the form

$$(x_0, x_1, \ldots, x_{n-1}, 0, -x_{n-1}, \ldots, -x_0, -x_1, \ldots, -x_{n-1}, 0, x_{n-1}, \ldots, x_1).$$

2.1.4 Fourier integral transforms

The usual application of the discrete Fourier transform is that of obtaining an approximation of the **Fourier integral transform**

$$F(s) = \int_{-\infty}^{\infty} f(t) \exp(-i2\pi st)\, dt$$

when $f(t)$ is negligible outside some region $(0, c)$. Dividing the region into n equal intervals we have

$$F(s) \cong \frac{c}{n} \sum_{j=0}^{n-1} f_j \exp(-i2\pi sjc/n)$$

and so

$$F_k \cong \frac{c}{n} \sum_{j=0}^{n-1} f_j \exp(-i2\pi jk/n)$$

for $k = 0, 1, \ldots, n-1$, where $f_j = f(jc/n)$ and $F_k = F(k/c)$.

Hence the discrete Fourier transform gives an approximation to the Fourier integral transform in the region $s = 0$ to $s = n/c$.

If the function $f(t)$ is defined over some more general interval (a, b), then the integral transform can still be approximated by the discrete transform provided a shift is applied to move the point a to the origin.

2.1.5 Convolutions and correlations

One of the most important applications of the discrete Fourier transform is to the computation of the discrete **convolution** or **correlation** of two vectors x and y defined (as in Brigham [1]) by:

$$\text{convolution}: \; z_k = \sum_{j=0}^{n-1} x_j y_{k-j}$$

$$\text{correlation}: \; w_k = \sum_{j=0}^{n-1} \bar{x}_j y_{k+j}$$

(Here x and y are assumed to be periodic with period n.)

Under certain circumstances (see Brigham [1]) these can be used as approximations to the convolution or correlation integrals defined by:

$$z(s) = \int_{-\infty}^{\infty} x(t) y(s-t)\, dt$$

and

$$w(s) = \int_{-\infty}^{\infty} \bar{x}(t) y(s+t)\, dt, \quad -\infty < s < \infty.$$

For more general advice on the use of Fourier transforms, see Hamming [5]; more detailed information on the fast Fourier transform algorithm can be found in Gentleman and Sande [4] and Brigham [1].

2.1.6 Applications to solving partial differential equations

A further application of the fast Fourier transform, and in particular of the Fourier transforms of symmetric sequences, is in the solution of elliptic partial differential equations. If an equation is discretised using finite differences, then it is possible to reduce the problem of solving the resulting large system of linear equations to that of solving a number of tridiagonal systems of linear equations. This is accomplished by uncoupling the equations using Fourier transforms, where the nature of the boundary conditions determines the choice of transforms – see Section 3.3. Full details of the Fourier method for the solution of partial differential equations may be found in Swarztrauber [7], [8].

2.2 Inverse Laplace Transforms

Let $f(t)$ be a real function of t, with $f(t) = 0$ for $t < 0$, and be piecewise continuous and of exponential order α, i.e.,

$$|f(t)| \leq Me^{\alpha t}$$

for large t, where α is the minimal such exponent.

The Laplace transform of $f(t)$ is given by

$$F(s) = \int_0^\infty e^{-st} f(t)\, dt, \quad t > 0$$

where $F(s)$ is defined for $\mathrm{Re}(s) > \alpha$.

The inverse transform is defined by the Bromwich integral

$$f(t) = \frac{1}{2\pi i} \int_{a-i\infty}^{a+i\infty} e^{st} F(s)\, ds, \quad t > 0$$

The integration is performed along the line $s = a$ in the complex plane, where $a > \alpha$. This is equivalent to saying that the line $s = a$ lies to the right of all singularities of $F(s)$. For this reason, the value of α is crucial to the correct evaluation of the inverse. It is not essential to know α exactly, but an upper bound must be known.

The problem of determining an inverse Laplace transform may be classified according to whether (a) $F(s)$ is known for real values only, or (b) $F(s)$ is known in functional form and can therefore be calculated for complex values of s. Problem (a) is very ill defined and no routines are provided. Two methods are provided for problem (b).

2.3 Direct Summation of Orthogonal Series

For any series of functions ϕ_i which satisfy a recurrence

$$\phi_{r+1}(x) + \alpha_r(x)\phi_r(x) + \beta_r(x)\phi_{r-1}(x) = 0$$

the sum

$$\sum_{r=0}^{n} a_r \phi_r(x)$$

is given by

$$\sum_{r=0}^{n} a_r \phi_r(x) = b_0(x)\phi_0(x) + b_1(x)\{\phi_1(x) + \alpha_0(x)\phi_0(x)\}$$

where

$$b_r(x) + \alpha_r(x)b_{r+1}(x) + \beta_{r+1}(x)b_{r+2}(x) = a_r b_{n+1}(x) = b_{n+2}(x) = 0.$$

This may be used to compute the sum of the series. For further reading, see Hamming [5].

2.4 Acceleration of Convergence

This device has applications in a large number of fields, such as summation of series, calculation of integrals with oscillatory integrands (including, for example, Hankel transforms), and root-finding. The mathematical description is as follows. Given a sequence of values $\{s_n\}$, $n = m, m+1, m+2, \ldots, m+2l$ then, except in certain singular cases, parameters, a, b_i, c_i may be determined such that

$$s_n = a + \sum_{i=1}^{l} b_i c_i^n.$$

If the sequence $\{s_n\}$ converges, then a may be taken as an estimate of the limit. The method will also find a pseudo-limit of certain divergent sequences – see Shanks [6] for details.

To use the method to sum a series, the terms s_n of the sequence should be the partial sums of the series, e.g. $s_n = \sum_{k=1}^{n} t_k$, where t_k is the kth term of the series. The algorithm can also be used to some advantage to evaluate integrals with oscillatory integrands; one approach is to write the integral (in this case over a semi-infinite interval) as:

$$\int_0^\infty f(x)\,dx = \int_0^{a_1} f(x)\,dx + \int_{a_1}^{a_2} f(x)\,dx + \int_{a_2}^{a_3} f(x)\,dx + \ldots$$

and to consider the sequence of values

$$s_1 = \int_0^{a_1} f(x)\,dx;\ s_2 = \int_0^{a_2} f(x)\,dx = s_1 + \int_{a_1}^{a_2} f(x)\,dx, \text{etc},$$

where the integrals are evaluated using standard quadrature methods. In choosing the values of the a_k, it is worth bearing in mind that C06BAF converges much more rapidly for sequences whose values oscillate about a limit. The a_k should thus be chosen to be (close to) the zeros of $f(x)$, so that successive contributions to the integral are of opposite sign. As an example, consider the case where $f(x) = M(x)\sin x$ and $M(x) > 0$: convergence will be much improved if $a_k = k\pi$ rather than $a_k = 2k\pi$.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

3.1 One-dimensional Fourier Transforms

The choice of routine is determined first of all by whether the data values constitute a real, Hermitian or general complex sequence. It is wasteful of time and storage to use an inappropriate routine.

Three groups, each of three routines, are provided

	Group 1	Group 2	Group 3
Real sequences	C06EAF	C06FAF	C06FPF
Hermitian sequences	C06EBF	C06FBF	C06FQF
General complex sequences	C06ECF	C06FCF	C06FRF

Group 1 routines each compute a single transform of length n, without requiring any extra working storage. Group 2 routines also compute a single transform of length n, but require one additional ***real*** work-array of length n. For some values of n — when n has unpaired prime factors — Group 1 routines are particularly slow and the Group 2 routines are much more efficient. The Group 1 and Group 2 routines impose some restrictions on the value of n, namely that no prime factor of n may exceed 19 and the total number of prime factors (including repetitions) may not exceed 20 (though the latter restriction only becomes relevant when $n > 10^6$).

Group 3 routines are all designed to perform several transforms in a single call, all with the same value of n. They are designed to be much faster than the Group 1 and Group 2 routines on vector-processing machines. They do however require more working storage. Even on scalar processors, they may be somewhat faster than repeated calls to Group 1 or Group 2 routines because of reduced overheads and

because they pre-compute and store the required values of trigonometric functions. Group 3 routines impose no practical restrictions on the value of n; however the fast Fourier transform algorithm ceases to be 'fast' if applied to values of n which cannot be expressed as a product of small prime factors. All the above routines are particularly efficient if the only prime factors of n are 2, 3 or 5.

If extensive use is to be made of these routines, users who are concerned about efficiency are advised to conduct their own timing tests.

To compute inverse discrete Fourier transforms the above routines should be used in conjunction with the utility routines C06GBF, C06GCF and C06GQF which form the complex conjugate of a Hermitian or general sequence of complex data values.

3.2 Half- and Quarter-wave Transforms

Four routines are provided for computing fast Fourier transforms (FFTs) of real symmetric sequences. C06HAF computes multiple Fourier sine transforms, C06HBF computes multiple Fourier cosine transforms, C06HCF computes multiple quarter-wave Fourier sine transforms and C06HDF computes multiple quarter-wave Fourier cosine transforms. These four routines are designed to be particularly efficient on vector processors.

3.3 Application to Elliptic Partial Differential Equations

As described in Section 2.1, Fourier transforms may be used in the solution of elliptic partial differential equations.

C06HAF may be used to solve equations where the solution is specified along the boundary.

C06HBF may be used to solve equations where the derivative of the solution is specified along the boundary.

C06HCF may be used to solve equations where the solution is specified on the lower boundary, and the derivative of the solution is specified on the upper boundary.

C06HDF may be used to solve equations where the derivative of the solution is specified on the lower boundary, and the solution is specified on the upper boundary.

For equations with periodic boundary conditions the full-range Fourier transforms computed by C06FPF and C06FQF are appropriate.

3.4 Multi-dimensional Fourier Transforms

The following routines compute multi-dimensional discrete Fourier transforms of complex data:

C06FUF 2 dimensions

C06FXF 3 dimensions

C06FJF any number of dimensions

C06FUF and C06FXF should be used in preference to C06FJF for 2- and 3-dimensional transforms, as they are easier to use and are likely to be more efficient, especially on vector processors.

3.5 Convolution and Correlation

C06EKF and C06FKF each compute either the discrete convolution or the discrete correlation of two real vectors. The distinction between these two routines is the same as that between the C06E- and C06F- routines described in Section 3.1.

3.6 Inverse Laplace Transforms

Two methods are provided: Weeks' method and Crump's method. Both require the function $F(s)$ to be evaluated for complex values of s. If in doubt which method to use, try Weeks' method first: when it is suitable, it is usually much faster.

Typically the inversion of a Laplace transform becomes harder as t increases so that all numerical methods tend to have a limit on the range of t for which the inverse $f(t)$ can be computed. C06LAF is useful for small and moderate values of t.

It is often convenient or necessary to scale a problem so that α is close to 0. For this purpose it is useful to remember that the inverse of $F(s+k)$ is $\exp(-kt)f(t)$. The method used by C06LAF is not so satisfactory when $f(t)$ is close to zero, in which case a term may be added to $F(s)$, e.g., $k/s + F(s)$ has the inverse $k + f(t)$.

Singularities in the inverse function $f(t)$ generally cause numerical methods to perform less well. The positions of singularities can often be identified by examination of $F(s)$. If $F(s)$ contains a term of the form $\exp(-ks)/s$ then a finite discontinuity may be expected in the inverse at $t = k$. C06LAF, for example, is capable of estimating a discontinuous inverse but, as the approximation used is continuous, Gibbs' phenomena (overshoots around the discontinuity) result. If possible, such singularities of $F(s)$ should be removed before computing the inverse.

3.7 Direct Summation of Orthogonal Series

Only one routine is available, C06DBF, which sums a finite Chebyshev series

$$\sum_{j=0}^{n} c_j T_j(x), \quad \sum_{j=0}^{n} c_j T_{2j}(x) \text{ or } \sum_{j=0}^{n} c_j T_{2j+1}(x)$$

depending on the choice of a parameter.

3.8 Acceleration of Convergence

Only one routine is available, C06BAF.

4 Index

real sequence, space-saving	C06EAF
real sequence, time-saving	C06FAF
half- and quarter-wave transforms	
multiple Fourier sine transforms	C06HAF
multiple Fourier cosine transforms	C05HBF
multiple quarter-wave sine transforms	C06HCF
multiple quarter-wave cosine transforms	C06HDF
Inverse Laplace Transform	
Crump's method	C06LAF
Weeks' method	
compute coefficients of solution	C06LBF
evaluate solution	C06LCF
Summation of Chebyshev series	C06DBF

5 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Davies S B and Martin B (1979) Numerical inversion of the Laplace transform: A survey and comparison of methods *J. Comput. Phys.* **33** 1–32

[3] Fox L and Parker I B (1968) *Chebyshev Polynomials in Numerical Analysis* Oxford University Press

[4] Gentleman W S and Sande G (1966) Fast Fourier transforms for fun and profit *Proc. Joint Computer Conference, AFIPS* **29** 563–578

[5] Hamming R W (1962) *Numerical Methods for Scientists and Engineers* McGraw-Hill

[6] Shanks D (1955) Nonlinear transformations of divergent and slowly convergent sequences *J. Math. Phys.* **34** 1–42

[7] Swarztrauber P N (1977) The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle *SIAM Rev.* **19 (3)** 490–501

[8] Swarztrauber P N (1984) Fast Poisson solvers *Studies in Numerical Analysis* (ed G H Golub) Mathematical Association of America

[9] Swarztrauber P N (1986) Symmetric FFT's *Math. Comput.* **47 (175)** 323–346

[10] Wynn P (1956) On a device for computing the $e_m(S_n)$ transformation *Math. Tables Aids Comput.* **10** 91–96

C06BAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06BAF accelerates the convergence of a given convergent sequence to its limit.

2. Specification

```
SUBROUTINE C06BAF (SEQN, NCALL, RESULT, ABSERR, WORK, IWORK, IFAIL)
INTEGER          NCALL, IWORK, IFAIL
real             SEQN, RESULT, ABSERR, WORK(IWORK)
```

3. Description

The routine performs Shanks' transformation on a given sequence of real values by means of the Epsilon algorithm of Wynn [2]. A (possibly unreliable) estimate of the absolute error is also given.

The routine must be called repetitively, once for each new term in the sequence.

4. References

[1] SHANKS, D.
Nonlinear Transformations of Divergent and Slowly Convergent Sequences.
J. Math. Phys., 34, pp. 1-42, 1955.

[2] WYNN, P.
On a Device for Computing the $e_m(S_n)$ Transformation.
Math. Tables Aids Comp. 10, pp. 91-96, 1956.

5. Parameters

1: SEQN – ***real***. *Input*

On entry: the next term of the sequence to be considered.

2: NCALL – INTEGER. *Input/Output*

On entry: on the first call NCALL must be set to 0. Thereafter NCALL **must not** be changed between calls.

On exit: the number of terms in the sequence that have been considered.

3: RESULT – ***real***. *Output*

On exit: the estimate of the limit of the sequence. For the first two calls, RESULT = SEQN.

4: ABSERR – ***real***. *Output*

On exit: an estimate of the absolute error in RESULT. For the first three calls, ABSERR is set to a large machine-dependent constant.

5: WORK(IWORK) – ***real*** array. *Workspace*

Used as workspace, but **must not** be changed between calls.

6: IWORK – INTEGER. *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which C06BAF is called.

Suggested value: (maximum number of terms in the sequence) + 6. See Section 8.2.

Constraint: IWORK ≥ 7.

7: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NCALL < 0.

IFAIL = 2

On entry, IWORK < 7.

7. Accuracy

The accuracy of the absolute error estimate ABSERR varies considerably with the type of sequence to which the routine is applied. In general it is better when applied to oscillating sequences than to monotonic sequences where it may be a severe underestimate.

8. Further Comments

8.1. Timing

The time taken by the routine is approximately proportional to the final value of NCALL.

8.2. Choice of IWORK

For long sequences, a 'window' of the last n values can be used instead of all the terms of the sequence. Tests on a variety of problems indicate that a suitable value is $n = 50$; this implies a value for IWORK of 56. Users are advised to experiment with other values for their own specific problems.

8.3. Convergence

The routine will induce convergence in some divergent sequences. See Shanks [1] for more details.

9. Example

The example program attempts to sum the infinite series

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^2} = \frac{\pi^2}{12}$$

by considering the sequence of partial sums

$$\sum_{n=1}^{1}, \sum_{n=1}^{2}, \sum_{n=1}^{3}, \ldots, \sum_{n=1}^{10}$$

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06BAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          IWORK
      PARAMETER        (IWORK=16)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ABSERR, ANS, ERROR, PI, R, RESULT, SEQN, SIG
      INTEGER          I, IFAIL, NCALL
```

```
*     .. Local Arrays ..
      real                 WORK(IWORK)
*     .. External Functions ..
      real                 X01AAF
      EXTERNAL             X01AAF
*     .. External Subroutines ..
      EXTERNAL             C06BAF
*     .. Intrinsic Functions ..
      INTRINSIC            real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06BAF Example Program Results'
      WRITE (NOUT,*)
      PI = X01AAF(0.0e0)
      ANS = PI**2/12.0e0
      NCALL = 0
      SIG = 1.0e0
      SEQN = 0.0e0
      WRITE (NOUT,*)
     + '                                          Estimated       Actual'
      WRITE (NOUT,*)
     + '    I       SEQN        RESULT          abs error        error'
      WRITE (NOUT,*)
      DO 20 I = 1, 10
         R = real(I)
         SEQN = SEQN + SIG/(R**2)
         IFAIL = 1
*
         CALL C06BAF(SEQN,NCALL,RESULT,ABSERR,WORK,IWORK,IFAIL)
*
         IF (IFAIL.NE.0) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'C06BAF fails. IFAIL=', IFAIL
            STOP
         END IF
         ERROR = RESULT - ANS
         SIG = -SIG
         WRITE (NOUT,99998) I, SEQN, RESULT, ABSERR, ERROR
   20 CONTINUE
      STOP
*
99999 FORMAT (1X,A,I2)
99998 FORMAT (1X,I4,2F12.4,3X,2e14.2)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 C06BAF Example Program Results

                                    Estimated       Actual
   I       SEQN        RESULT       abs error        error

   1      1.0000      1.0000        0.13+155       0.18E+00
   2      0.7500      0.7500        0.13+155      -0.72E-01
   3      0.8611      0.8269        0.13+155       0.45E-02
   4      0.7986      0.8211        0.26E+00      -0.14E-02
   5      0.8386      0.8226        0.78E-01       0.12E-03
   6      0.8108      0.8224        0.60E-02      -0.33E-04
   7      0.8312      0.8225        0.15E-02       0.35E-05
   8      0.8156      0.8225        0.16E-03      -0.85E-06
   9      0.8280      0.8225        0.37E-04       0.10E-06
  10      0.8180      0.8225        0.45E-05      -0.23E-07
```

C06DBF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06DBF returns the value of the sum of a Chebyshev series through the routine name.

2. Specification

```
real FUNCTION C06DBF (X, C, N, S)
INTEGER          N, S
real             X, C(N)
```

3. Description

This routine evaluates the sum of a Chebyshev series of one of three forms according to the value of the parameter S:

$$\text{S} = 1 : 0.5c_1 + \sum_{j=2}^{n} c_j T_{j-1}(x),$$

$$\text{S} = 2 : 0.5c_1 + \sum_{j=2}^{n} c_j T_{2j-2}(x),$$

$$\text{S} = 3 : \sum_{j=1}^{n} c_j T_{2j-1}(x)$$

where x lies in the range $-1.0 \leq x \leq 1.0$. Here $T_r(x)$ is the Chebyshev polynomial of order r in x, defined by $\cos(ry)$ where $\cos y = x$.

The method used is based upon a three-term recurrence relation; for details see Clenshaw [1].

4. References

[1] CLENSHAW, C.W.
Chebyshev Series for Mathematical Functions.
NPL Mathematical Tables, Vol. 5, HMSO, London, 1962.

5. Parameters

1: X – ***real***. *Input*

On entry: the argument x of the series.

Constraint: $-1.0 \leq \text{X} \leq 1.0$.

2: C(N) – ***real*** array. *Input*

On entry: C(j) must contain the coefficient c_j of the Chebyshev series, for $j = 1,2,...,n$.

3: N – INTEGER. *Input*

On entry: the number of terms, n, in the series.

4: S – INTEGER. *Input*

On entry: S must have the value 1, 2 or 3 according to whether the series is general, even or odd respectively (see Section 3). For all other values of S, the routine behaves as though S = 2.

6. Error Indicators and Warnings

None.

7. Accuracy

There may be a loss of significant figures due to cancellation between terms. However, provided that n is not too large, the routine yields results which differ little from the best attainable for a given word length.

8. Further Comments

The time taken by the routine increases with n.

This routine has been prepared in the present form to complement a number of integral equation solving routines which use Chebyshev series methods, e.g. D05AAF and D05ABF.

9. Example

This program evaluates

$$0.5 + T_1(x) + 0.5T_2(x) + 0.25T_3(x)$$

at the point $x = 0.5$.

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06DBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             CALC, X
*     .. Local Arrays ..
      real             C(4)
*     .. External Functions ..
      real             C06DBF
      EXTERNAL         C06DBF
*     .. Data statements ..
      DATA             C/1.0e0, 1.0e0, 0.5e0, 0.25e0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06DBF Example Program Results'
      X = 0.5e0
      CALC = C06DBF(X,C,4,1)
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Sum =', CALC
      STOP
*
99999 FORMAT (1X,A,F8.4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C06DBF Example Program Results

Sum =  0.5000
```

C06EAF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06EAF calculates the discrete Fourier transform of a sequence of n real data values. (No extra workspace required.)

2 Specification

```
SUBROUTINE C06EAF(X, N, IFAIL)
INTEGER          N, IFAIL
real             X(N)
```

3 Description

Given a sequence of n real data values x_j, for $j = 0, 1, \ldots, n-1$, this routine calculates their discrete Fourier transform defined by:

$$\hat{z}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \times \exp\left(-i\frac{2\pi jk}{n}\right), \quad k = 0, 1, \ldots, n-1.$$

(Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.) The transformed values $\hat{z}_k$ are complex, but they form a Hermitian sequence (i.e., $\hat{z}_{n-k}$ is the complex conjugate of $\hat{z}_k$), so they are completely determined by n real numbers (see also the Chapter Introduction).

To compute the inverse discrete Fourier transform defined by:

$$\hat{w}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \times \exp\left(+i\frac{2\pi jk}{n}\right),$$

this routine should be followed by a call of C06GBF to form the complex conjugates of the $\hat{z}_k$.

The routine uses the fast Fourier transform (FFT) algorithm (Brigham [1]). There are some restrictions on the value of n (see Section 5).

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

5 Parameters

1: X(N) — ***real*** array *Input/Output*

On entry: if X is declared with bounds (0:N−1) in the (sub)program from which C06EAF is called, then X(j) must contain x_j, for $j = 0, 1, \ldots, n-1$.

On exit: the discrete Fourier transform stored in Hermitian form. If the components of the transform $\hat{z}_k$ are written as $a_k + ib_k$, and if X is declared with bounds (0:N−1) in the (sub)program from which C06EAF is called, then for $0 \leq k \leq n/2$, a_k is contained in X(k), and for $1 \leq k \leq (n-1)/2$, b_k is contained in X($n-k$). (See also Section 2.1.2 of the Chapter Introduction and the Example Program.)

2: N — INTEGER *Input*

On entry: the number of data values, n. The largest prime factor of N must not exceed 19, and the total number of prime factors of N, counting repetitions, must not exceed 20.

Constraint: N > 1.

3: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

At least one of the prime factors of N is greater than 19.

IFAIL = 2

N has more than 20 prime factors.

IFAIL = 3

N $\leq$ 1.

IFAIL = 4

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $n \times \log n$, but also depends on the factorization of n. The routine is somewhat faster than average if the only prime factors of n are 2, 3 or 5; and fastest of all if n is a power of 2.

On the other hand, the routine is particularly slow if n has several unpaired prime factors, i.e., if the 'square-free' part of n has several factors. For such values of n, routine C06FAF (which requires an additional n elements of workspace) is considerably faster.

9 Example

This program reads in a sequence of real data values, and prints their discrete Fourier transform (as computed by C06EAF), after expanding it from Hermitian form into a full complex sequence.

It then performs an inverse transform using C06GBF and C06EBF, and prints the sequence so obtained alongside the original data values.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06EAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=20)
      INTEGER          NIN, NOUT
```

```
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, N, N2, NJ
*     .. Local Arrays ..
      real             A(0:NMAX-1), B(0:NMAX-1), X(0:NMAX-1),
     +                 XX(0:NMAX-1)
*     .. External Subroutines ..
      EXTERNAL         C06EAF, C06EBF, C06GBF
*     .. Intrinsic Functions ..
      INTRINSIC        MOD
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06EAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=120) N
      IF (N.GT.1 .AND. N.LE.NMAX) THEN
         DO 40 J = 0, N - 1
            READ (NIN,*) X(J)
            XX(J) = X(J)
   40    CONTINUE
         IFAIL = 0
*
         CALL C06EAF(X,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of discrete Fourier transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '          Real      Imag'
         WRITE (NOUT,*)
         A(0) = X(0)
         B(0) = 0.0e0
         N2 = (N-1)/2
         DO 60 J = 1, N2
            NJ = N - J
            A(J) = X(J)
            A(NJ) = X(J)
            B(J) = X(NJ)
            B(NJ) = -X(NJ)
   60    CONTINUE
         IF (MOD(N,2).EQ.0) THEN
            A(N2+1) = X(N2+1)
            B(N2+1) = 0.0e0
         END IF
         DO 80 J = 0, N - 1
            WRITE (NOUT,99999) J, A(J), B(J)
   80    CONTINUE
*
         CALL C06GBF(X,N,IFAIL)
         CALL C06EBF(X,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '        Original  Restored'
         WRITE (NOUT,*)
         DO 100 J = 0, N - 1
            WRITE (NOUT,99999) J, XX(J), X(J)
```

```
  100    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
  120 STOP
*
99999 FORMAT (1X,I5,2F10.5)
      END
```

9.2 Program Data

```
C06EAF Example Program Data
    7
  0.34907
  0.54890
  0.74776
  0.94459
  1.13850
  1.32850
  1.51370
```

9.3 Program Results

```
C06EAF Example Program Results

Components of discrete Fourier transform

          Real      Imag

    0   2.48361   0.00000
    1  -0.26599   0.53090
    2  -0.25768   0.20298
    3  -0.25636   0.05806
    4  -0.25636  -0.05806
    5  -0.25768  -0.20298
    6  -0.26599  -0.53090

Original sequence as restored by inverse transform

        Original  Restored

    0   0.34907   0.34907
    1   0.54890   0.54890
    2   0.74776   0.74776
    3   0.94459   0.94459
    4   1.13850   1.13850
    5   1.32850   1.32850
    6   1.51370   1.51370
```

C06EBF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06EBF calculates the discrete Fourier transform of a Hermitian sequence of n complex data values. (No extra workspace required.)

2 Specification

```
SUBROUTINE C06EBF(X, N, IFAIL)
INTEGER           N, IFAIL
real              X(N)
```

3 Description

Given a Hermitian sequence of n complex data values z_j (i.e., a sequence such that z_0 is real and z_{n-j} is the complex conjugate of z_j, for $j = 1, 2, \ldots, n-1$) this routine calculates their discrete Fourier transform defined by:

$$\hat{x}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \times \exp\left(-i\frac{2\pi jk}{n}\right), \quad k = 0, 1, \ldots, n-1.$$

(Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.) The transformed values $\hat{x}_k$ are purely real (see also the Chapter Introduction).

To compute the inverse discrete Fourier transform defined by:

$$\hat{y}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \times \exp\left(+i\frac{2\pi jk}{n}\right),$$

this routine should be preceded by a call of C06GBF to form the complex conjugates of the z_j.

The routine uses the fast Fourier transform (FFT) algorithm (Brigham [1]). There are some restrictions on the value of n (see Section 5).

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

5 Parameters

1: X(N) — ***real*** array *Input/Output*

On entry: the sequence to be transformed stored in Hermitian form. If the data values z_j are written as $x_j + iy_j$, and if X is declared with bounds (0:N−1) in the subroutine from which C06EBF is called, then for $0 \le j \le n/2$, x_j is contained in X(j), and for $1 \le j \le (n-1)/2$, y_j is contained in X($n-j$). (See also Section 2.1.2 of the Chapter Introduction and the Example Program.)

On exit: the components of the discrete Fourier transform $\hat{x}_k$. If X is declared with bounds (0:N−1) in the (sub)program from which C06EBF is called, then $\hat{x}_k$ is stored in X(k), for $k = 0, 1, \ldots, n-1$.

2: N — INTEGER *Input*

On entry: the number of data values, n. The largest prime factor of N must not exceed 19, and the total number of prime factors of N, counting repetitions, must not exceed 20.

Constraint: N > 1.

3: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

At least one of the prime factors of N is greater than 19.

IFAIL = 2

N has more than 20 prime factors.

IFAIL = 3

N $\leq$ 1.

IFAIL = 4

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $n \times \log n$, but also depends on the factorization of n. The routine is somewhat faster than average if the only prime factors of n are 2, 3 or 5; and fastest of all if n is a power of 2.

On the other hand, the routine is particularly slow if n has several unpaired prime factors, i.e., if the 'square-free' part of n has several factors. For such values of n, routine C06FBF (which requires an additional n elements of workspace) is considerably faster.

9 Example

This program reads in a sequence of real data values which is assumed to be a Hermitian sequence of complex data values stored in Hermitian form. The input sequence is expanded into a full complex sequence and printed alongside the original sequence. The discrete Fourier transform (as computed by C06EBF) is printed out.

The program then performs an inverse transform using C06EAF and C06GBF, and prints the sequence so obtained alongside the original data values.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06EBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
```

```
      PARAMETER        (NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, N, N2, NJ
*     .. Local Arrays ..
      real             U(0:NMAX-1), V(0:NMAX-1), X(0:NMAX-1),
     +                 XX(0:NMAX-1)
*     .. External Subroutines ..
      EXTERNAL         C06EAF, C06EBF, C06GBF
*     .. Intrinsic Functions ..
      INTRINSIC        MOD
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06EBF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=140) N
      IF (N.GT.1 .AND. N.LE.NMAX) THEN
         DO 40 J = 0, N - 1
            READ (NIN,*) X(J)
            XX(J) = X(J)
   40    CONTINUE
         U(0) = X(0)
         V(0) = 0.0e0
         N2 = (N-1)/2
         DO 60 J = 1, N2
            NJ = N - J
            U(J) = X(J)
            U(NJ) = X(J)
            V(J) = X(NJ)
            V(NJ) = -X(NJ)
   60    CONTINUE
         IF (MOD(N,2).EQ.0) THEN
            U(N2+1) = X(N2+1)
            V(N2+1) = 0.0e0
         END IF
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence and corresponding complex sequence'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '         Data           Real      Imag'
         WRITE (NOUT,*)
         DO 80 J = 0, N - 1
            WRITE (NOUT,99999) J, X(J), '     ', U(J), V(J)
   80    CONTINUE
         IFAIL = 0
*
         CALL C06EBF(X,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of discrete Fourier transform'
         WRITE (NOUT,*)
         DO 100 J = 0, N - 1
            WRITE (NOUT,99999) J, X(J)
  100    CONTINUE
*
         CALL C06EAF(X,N,IFAIL)
         CALL C06GBF(X,N,IFAIL)
```

```
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '        Original  Restored'
         WRITE (NOUT,*)
         DO 120 J = 0, N - 1
            WRITE (NOUT,99998) J, XX(J), X(J)
  120    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
  140 STOP
*
99999 FORMAT (1X,I5,F10.5,A,2F10.5)
99998 FORMAT (1X,I5,2F10.5)
      END
```

9.2 Program Data

```
C06EBF Example Program Data
    7
  0.34907
  0.54890
  0.74776
  0.94459
  1.13850
  1.32850
  1.51370
```

9.3 Program Results

```
C06EBF Example Program Results

Original sequence and corresponding complex sequence

          Data           Real      Imag

     0   0.34907        0.34907   0.00000
     1   0.54890        0.54890   1.51370
     2   0.74776        0.74776   1.32850
     3   0.94459        0.94459   1.13850
     4   1.13850        0.94459  -1.13850
     5   1.32850        0.74776  -1.32850
     6   1.51370        0.54890  -1.51370

Components of discrete Fourier transform

     0   1.82616
     1   1.86862
     2  -0.01750
     3   0.50200
     4  -0.59873
     5  -0.03144
     6  -2.62557
```

```
Original sequence as restored by inverse transform

        Original  Restored

   0    0.34907   0.34907
   1    0.54890   0.54890
   2    0.74776   0.74776
   3    0.94459   0.94459
   4    1.13850   1.13850
   5    1.32850   1.32850
   6    1.51370   1.51370
```

C06ECF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06ECF calculates the discrete Fourier transform of a sequence of n complex data values. (No extra workspace required.)

2 Specification

```
SUBROUTINE C06ECF(X, Y, N, IFAIL)
INTEGER           N, IFAIL
real              X(N), Y(N)
```

3 Description

Given a sequence of n complex data values z_j, for $j = 0, 1, \ldots, n-1$, this routine calculates their discrete Fourier transform defined by:

$$\hat{z}_k = a_k + ib_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \times \exp\left(-i\frac{2\pi jk}{n}\right), \quad k = 0, 1, \ldots, n-1.$$

(Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.)

To compute the inverse discrete Fourier transform defined by:

$$\hat{w}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \times \exp\left(+i\frac{2\pi jk}{n}\right),$$

this routine should be preceded and followed by calls of C06GCF to form the complex conjugates of the z_j and the $\hat{z}_k$.

The routine uses the fast Fourier transform (FFT) algorithm (Brigham [1]). There are some restrictions on the value of n (see Section 5).

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

5 Parameters

1: X(N) — ***real*** array *Input/Output*

On entry: if X is declared with bounds (0:N−1) in the (sub)program from which C06ECF is called, then X(j) must contain x_j, the real part of z_j, for $j = 0, 1, \ldots, n-1$.

On exit: the real parts a_k of the components of the discrete Fourier transform. If X is declared with bounds (0:N−1) in the (sub)program from which C06ECF is called, then a_k is contained in X(k), for $k = 0, 1, \ldots, n-1$.

2: Y(N) — ***real*** array *Input/Output*

On entry: if Y is declared with bounds (0:N−1) in the (sub)program from which C06ECF is called, then Y(j) must contain y_j, the imaginary part of z_j, for $j = 0, 1, \ldots, n-1$.

On exit: the imaginary parts b_k of the components of the discrete Fourier transform. If Y is declared with bounds (0:N−1) in the (sub)program from which C06ECF is called, then b_k is contained in Y(k), for $k = 0, 1, \ldots, n-1$.

3: N — INTEGER *Input*

On entry: the number of data values, n. The largest prime factor of N must not exceed 19, and the total number of prime factors of N, counting repetitions, must not exceed 20.

Constraint: N > 1.

4: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

At least one of the prime factors of N is greater than 19.

IFAIL = 2

N has more than 20 prime factors.

IFAIL = 3

N ≤ 1.

IFAIL = 4

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $n \times \log n$, but also depends on the factorization of n. The routine is somewhat faster than average if the only prime factors of n are 2, 3 or 5; and fastest of all if n is a power of 2.

On the other hand, the routine is particularly slow if n has several unpaired prime factors, i.e., if the 'square-free' part of n has several factors. For such values of n, routine C06FCF (which requires an additional n ***real*** elements of workspace) is considerably faster.

9 Example

This program reads in a sequence of complex data values and prints their discrete Fourier transform.

It then performs an inverse transform using C06GCF and C06ECF, and prints the sequence so obtained alongside the original data values.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06ECF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, N
*     .. Local Arrays ..
      real             X(0:NMAX-1), XX(0:NMAX-1), Y(0:NMAX-1),
     +                 YY(0:NMAX-1)
*     .. External Subroutines ..
      EXTERNAL         C06ECF, C06GCF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06ECF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=100) N
      IF (N.GT.1 .AND. N.LE.NMAX) THEN
         DO 40 J = 0, N - 1
            READ (NIN,*) X(J), Y(J)
            XX(J) = X(J)
            YY(J) = Y(J)
   40    CONTINUE
         IFAIL = 0
*
         CALL C06ECF(X,Y,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of discrete Fourier transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '          Real      Imag'
         WRITE (NOUT,*)
         DO 60 J = 0, N - 1
            WRITE (NOUT,99999) J, X(J), Y(J)
   60    CONTINUE
*
         CALL C06GCF(Y,N,IFAIL)
         CALL C06ECF(X,Y,N,IFAIL)
         CALL C06GCF(Y,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '             Original                  Restored'
         WRITE (NOUT,*)
     +     '          Real      Imag           Real      Imag'
         WRITE (NOUT,*)
         DO 80 J = 0, N - 1
            WRITE (NOUT,99999) J, XX(J), YY(J), X(J), Y(J)
   80    CONTINUE
         GO TO 20
```

```
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
  100 STOP
*
99999 FORMAT (1X,I5,2F10.5,5X,2F10.5)
      END
```

9.2 Program Data

```
C06ECF Example Program Data
    7
  0.34907  -0.37168
  0.54890  -0.35669
  0.74776  -0.31175
  0.94459  -0.23702
  1.13850  -0.13274
  1.32850   0.00074
  1.51370   0.16298
```

9.3 Program Results

```
C06ECF Example Program Results

Components of discrete Fourier transform

          Real      Imag

    0   2.48361  -0.47100
    1  -0.55180   0.49684
    2  -0.36711   0.09756
    3  -0.28767  -0.05865
    4  -0.22506  -0.17477
    5  -0.14825  -0.30840
    6   0.01983  -0.56496

Original sequence as restored by inverse transform

              Original                Restored
          Real      Imag          Real      Imag

    0   0.34907  -0.37168       0.34907  -0.37168
    1   0.54890  -0.35669       0.54890  -0.35669
    2   0.74776  -0.31175       0.74776  -0.31175
    3   0.94459  -0.23702       0.94459  -0.23702
    4   1.13850  -0.13274       1.13850  -0.13274
    5   1.32850   0.00074       1.32850   0.00074
    6   1.51370   0.16298       1.51370   0.16298
```

C06EKF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06EKF calculates the circular convolution or correlation of two real vectors of period n. No extra workspace is required.

2. Specification

```
SUBROUTINE C06EKF (JOB, X, Y, N, IFAIL)
INTEGER          JOB, N, IFAIL
real             X(N), Y(N)
```

3. Description

This routine computes:

if JOB = 1, the discrete **convolution** of x and y, defined by:

$$z_k = \sum_{j=0}^{n-1} x_j y_{k-j} = \sum_{j=0}^{n-1} x_{k-j} y_j;$$

if JOB = 2, the discrete **correlation** of x and y defined by:

$$w_k = \sum_{j=0}^{n-1} x_j y_{k+j}.$$

Here x and y are real vectors, assumed to be periodic, with period n, i.e. $x_j = x_{j\pm n} = x_{j\pm 2n} = \ldots$; z and w are then also periodic with period n.

Note: this usage of the terms 'convolution' and 'correlation' is taken from Brigham [1]. The term 'convolution' is sometimes used to denote both these computations.

If $\hat{x}$, $\hat{y}$, $\hat{z}$ and $\hat{w}$ are the discrete Fourier transforms of these sequences,

i.e. $\hat{x}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \times \exp\left(-i\frac{2\pi jk}{n}\right)$, etc.,

then $\hat{z}_k = \sqrt{n}.\hat{x}_k\hat{y}_k$

and $\hat{w}_k = \sqrt{n}.\bar{\hat{x}}_k\hat{y}_k$

(the bar denoting complex conjugate).

This routine calls the same auxiliary routines as C06EAF and C06EBF to compute discrete Fourier transforms, and there are some restrictions on the value of n.

4. References

[1] BRIGHAM, E.O.
The Fast Fourier Transform.
Prentice-Hall, 1973.

5. Parameters

1: JOB – INTEGER. *Input*

On entry: the computation to be performed:

if JOB = 1, $z_k = \sum_{j=0}^{n-1} x_j y_{k-j}$ (convolution);

if JOB = 2, $w_k = \sum_{j=0}^{n-1} x_j y_{k+j}$ (correlation).

Constraint: JOB = 1 or 2.

2: X(N) – ***real*** array. *Input/Output*

On entry: the elements of one period of the vector x. If X is declared with bounds (0:N–1) in the (sub)program from which C06EKF is called, then X(j) must contain x_j, for $j = 0,1,...,n-1$.

On exit: the corresponding elements of the discrete convolution or correlation.

3: Y(N) – ***real*** array. *Input/Output*

On entry: the elements of one period of the vector y. If Y is declared with bounds (0:N–1) in the (sub)program from which C06EKF is called, then Y(j) must contain y_j, for $j = 0,1,...,n-1$.

On exit: the discrete Fourier transform of the convolution or correlation returned in the array X; the transform is stored in Hermitian form, exactly as described in the document C06EAF.

4: N – INTEGER. *Input*

On entry: the number of values, n, in one period of the vectors X and Y. The largest prime factor of N must not exceed 19, and the total number of prime factors of N, counting repetitions, must not exceed 20.

Constraint: N > 1.

5: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

At least one of the prime factors of N is greater than 19.

IFAIL = 2

N has more than 20 prime factors.

IFAIL = 3

N ≤ 1.

IFAIL = 4

JOB ≠ 1 or 2.

7. Accuracy

The results should be accurate to within a small multiple of the ***machine precision***.

8. Further Comments

The time taken by the routine is approximately proportional to $n \times \log n$, but also depends on the factorization of n. The routine is faster than average if the only prime factors are 2, 3 or 5; and fastest of all if n is a power of 2.

The routine is particularly slow if n has several unpaired prime factors, i.e. if the 'square free' part of n has several factors. For such values of n, routine C06FKF is considerably faster (but requires an additional workspace of n elements).

9. Example

This program reads in the elements of one period of two real vectors x and y and prints their discrete convolution and correlation (as computed by C06EKF). In realistic computations the number of data values would be much larger.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06EKF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=64)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, N
*     .. Local Arrays ..
      real             XA(0:NMAX-1), XB(0:NMAX-1), YA(0:NMAX-1),
     +                 YB(0:NMAX-1)
*     .. External Subroutines ..
      EXTERNAL         C06EKF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06EKF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=80) N
      IF (N.GT.1 .AND. N.LE.NMAX) THEN
         DO 40 J = 0, N - 1
            READ (NIN,*) XA(J), YA(J)
            XB(J) = XA(J)
            YB(J) = YA(J)
   40    CONTINUE
         IFAIL = 0
*
         CALL C06EKF(1,XA,YA,N,IFAIL)
         CALL C06EKF(2,XB,YB,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) '        Convolution  Correlation'
         WRITE (NOUT,*)
         DO 60 J = 0, N - 1
            WRITE (NOUT,99999) J, XA(J), XB(J)
   60    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
   80 STOP
*
99999 FORMAT (1X,I5,2F13.5)
      END
```

9.2. Program Data

```
C06EKF Example Program Data
    9
      1.00      0.50
      1.00      0.50
      1.00      0.50
      1.00      0.50
      1.00      0.00
      0.00      0.00
      0.00      0.00
      0.00      0.00
      0.00      0.00
```

9.3. Program Results

```
C06EKF Example Program Results

         Convolution  Correlation

 0         0.50000      2.00000
 1         1.00000      1.50000
 2         1.50000      1.00000
 3         2.00000      0.50000
 4         2.00000      0.00000
 5         1.50000      0.50000
 6         1.00000      1.00000
 7         0.50000      1.50000
 8         0.00000      2.00000
```

C06FAF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06FAF calculates the discrete Fourier transform of a sequence of n real data values (using a work array for extra speed).

2 Specification

```
SUBROUTINE C06FAF(X, N, WORK, IFAIL)
INTEGER          N, IFAIL
real             X(N), WORK(N)
```

3 Description

Given a sequence of n real data values x_j, for $j = 0, 1, \ldots, n-1$, this routine calculates their discrete Fourier transform defined by:

$$\hat{z}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \times \exp\left(-i\frac{2\pi jk}{n}\right), \quad k = 0, 1, \ldots, n-1.$$

(Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.) The transformed values $\hat{z}_k$ are complex, but they form a Hermitian sequence (i.e., $\hat{z}_{n-k}$ is the complex conjugate of $\hat{z}_k$), so they are completely determined by n real numbers (see also the Chapter Introduction).

To compute the inverse discrete Fourier transform defined by:

$$\hat{w}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \times \exp\left(+i\frac{2\pi jk}{n}\right),$$

this routine should be followed by a call of C06GBF to form the complex conjugates of the $\hat{z}_k$.

The routine uses the fast Fourier transform (FFT) algorithm in Brigham [1]. There are some restrictions on the value of n (see Section 5).

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

5 Parameters

1: X(N) — ***real*** array *Input/Output*

On entry: if X is declared with bounds (0:N−1) in the (sub)program from which C06FAF is called, then X(j) must contain x_j, for $j = 0, 1, \ldots, n-1$.

On exit: the discrete Fourier transform stored in Hermitian form. If the components of the transform $\hat{z}_k$ are written as $a_k + ib_k$, and if X is declared with bounds (0:N−1) in the (sub)program from which C06FAF is called, then for $0 \leq k \leq n/2$, a_k is contained in X(k), and for $1 \leq k \leq (n-1)/2$, b_k is contained in X($n-k$). (See also Section 2.1.2 of the Chapter Introduction and the Example Program.)

2: N — INTEGER *Input*

On entry: the number of data values, n. The largest prime factor of N must not exceed 19, and the total number of prime factors of N, counting repetitions, must not exceed 20.

Constraint: N > 1.

3: WORK(N) — ***real*** array *Workspace*

4: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

At least one of the prime factors of N is greater than 19.

IFAIL = 2

N has more than 20 prime factors.

IFAIL = 3

N $\leq$ 1.

IFAIL = 4

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $n \times \log n$, but also depends on the factorization of n. The routine is somewhat faster than average if the only prime factors of n are 2, 3 or 5; and fastest of all if n is a power of 2.

9 Example

This program reads in a sequence of real data values and prints their discrete Fourier transform (as computed by C06FAF), after expanding it from Hermitian form into a full complex sequence.

It then performs an inverse transform, using C06GBF and C06FBF, and prints the sequence so obtained alongside the original data values.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
```

```
      INTEGER          IFAIL, J, N, N2, NJ
*     .. Local Arrays ..
      real             A(0:NMAX-1), B(0:NMAX-1), WORK(NMAX),
     +                 X(0:NMAX-1), XX(0:NMAX-1)
*     .. External Subroutines ..
      EXTERNAL         C06FAF, C06FBF, C06GBF
*     .. Intrinsic Functions ..
      INTRINSIC        MOD
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=120) N
      IF (N.GT.1 .AND. N.LE.NMAX) THEN
         DO 40 J = 0, N - 1
            READ (NIN,*) X(J)
            XX(J) = X(J)
   40    CONTINUE
         IFAIL = 0
*
         CALL C06FAF(X,N,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of discrete Fourier transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '          Real      Imag'
         WRITE (NOUT,*)
         A(0) = X(0)
         B(0) = 0.0e0
         N2 = (N-1)/2
         DO 60 J = 1, N2
            NJ = N - J
            A(J) = X(J)
            A(NJ) = X(J)
            B(J) = X(NJ)
            B(NJ) = -X(NJ)
   60    CONTINUE
         IF (MOD(N,2).EQ.0) THEN
            A(N2+1) = X(N2+1)
            B(N2+1) = 0.0e0
         END IF
         DO 80 J = 0, N - 1
            WRITE (NOUT,99999) J, A(J), B(J)
   80    CONTINUE
*
         CALL C06GBF(X,N,IFAIL)
         CALL C06FBF(X,N,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '        Original  Restored'
         WRITE (NOUT,*)
         DO 100 J = 0, N - 1
            WRITE (NOUT,99999) J, XX(J), X(J)
  100    CONTINUE
         GO TO 20
```

```
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
  120 STOP
*
99999 FORMAT (1X,I5,2F10.5)
      END
```

9.2 Program Data

```
C06FAF Example Program Data
    7
  0.34907
  0.54890
  0.74776
  0.94459
  1.13850
  1.32850
  1.51370
```

9.3 Program Results

```
C06FAF Example Program Results

Components of discrete Fourier transform

          Real      Imag

    0   2.48361   0.00000
    1  -0.26599   0.53090
    2  -0.25768   0.20298
    3  -0.25636   0.05806
    4  -0.25636  -0.05806
    5  -0.25768  -0.20298
    6  -0.26599  -0.53090

Original sequence as restored by inverse transform

        Original  Restored

    0   0.34907   0.34907
    1   0.54890   0.54890
    2   0.74776   0.74776
    3   0.94459   0.94459
    4   1.13850   1.13850
    5   1.32850   1.32850
    6   1.51370   1.51370
```

C06FBF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06FBF calculates the discrete Fourier transform of a Hermitian sequence of n complex data values (using a work array for extra speed).

2 Specification

```
SUBROUTINE C06FBF(X, N, WORK, IFAIL)
INTEGER           N, IFAIL
real              X(N), WORK(N)
```

3 Description

Given a Hermitian sequence of n complex data values z_j (i.e., a sequence such that z_0 is real and z_{n-j} is the complex conjugate of z_j, for $j = 1, 2, \ldots, n-1$), this routine calculates their discrete Fourier transform defined by:

$$\hat{x}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \times \exp\left(-i\frac{2\pi jk}{n}\right), \quad k = 0, 1, \ldots, n-1.$$

(Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.) The transformed values $\hat{x}_k$ are purely real (see also the Chapter Introduction).

To compute the inverse discrete Fourier transform defined by:

$$\hat{y}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \times \exp\left(+i\frac{2\pi jk}{n}\right),$$

this routine should be preceded by a call of C06GBF to form the complex conjugates of the z_j.

The routine uses the fast Fourier transform (FFT) algorithm in Brigham [1]. There are some restrictions on the value of n (see Section 5).

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

5 Parameters

1: X(N) — ***real*** array *Input/Output*

On entry: the sequence to be transformed stored in Hermitian form. If the data values z_j are written as $x_j + iy_j$, and if X is declared with bounds (0:N− 1)in the (sub)program from which C06FBF is called, then for $0 \le j \le n/2$, x_j is contained in X(j), and for $1 \le j \le (n-1)/2$, y_j is contained in X($n-j$). (See also Section 2.1.2 of the Chapter Introduction and the Example Program.)

On exit: the components of the discrete Fourier transform $\hat{x}_k$. If X is declared with bounds (0:N−1) in the (sub)program from which C06FBF is called, then $\hat{x}_k$ is stored in X(k) for $k = 0, 1, \ldots, n-1$.

2: N — INTEGER *Input*

On entry: the number of data values, n. The largest prime factor of N must not exceed 19, and the total number of prime factors of N, counting repetitions, must not exceed 20.

Constraint: N > 1.

3: WORK(N) — ***real*** array *Workspace*

4: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

At least one of the prime factors of N is greater than 19.

IFAIL = 2

N has more than 20 prime factors.

IFAIL = 3

N $\leq$ 1.

IFAIL = 4

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $n \times \log n$, but also depends on the factorization of n. The routine is somewhat faster than average if the only prime factors of n are 2, 3 or 5; and fastest of all if n is a power of 2.

9 Example

This program reads in a sequence of real data values which is assumed to be a Hermitian sequence of complex data values stored in Hermitian form. The input sequence is expanded into a full complex sequence and printed alongside the original sequence. The discrete Fourier transform (as computed by C06FBF) is printed out.

The program then performs an inverse transform using C06FAF and C06GBF, and prints the sequence so obtained alongside the original data values.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=20)
      INTEGER          NIN, NOUT
```

```
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, N, N2, NJ
*     .. Local Arrays ..
      real             U(0:NMAX-1), V(0:NMAX-1), WORK(NMAX),
     +                 X(0:NMAX-1), XX(0:NMAX-1)
*     .. External Subroutines ..
      EXTERNAL         C06FAF, C06FBF, C06GBF
*     .. Intrinsic Functions ..
      INTRINSIC        MOD
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FBF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=140) N
      IF (N.GT.1 .AND. N.LE.NMAX) THEN
         DO 40 J = 0, N - 1
            READ (NIN,*) X(J)
            XX(J) = X(J)
   40    CONTINUE
         U(0) = X(0)
         V(0) = 0.0e0
         N2 = (N-1)/2
         DO 60 J = 1, N2
            NJ = N - J
            U(J) = X(J)
            U(NJ) = X(J)
            V(J) = X(NJ)
            V(NJ) = -X(NJ)
   60    CONTINUE
         IF (MOD(N,2).EQ.0) THEN
            U(N2+1) = X(N2+1)
            V(N2+1) = 0.0e0
         END IF
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence and corresponding complex sequence'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '         Data           Real      Imag'
         WRITE (NOUT,*)
         DO 80 J = 0, N - 1
            WRITE (NOUT,99999) J, X(J), '     ', U(J), V(J)
   80    CONTINUE
         IFAIL = 0
*
         CALL C06FBF(X,N,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of discrete Fourier transform'
         WRITE (NOUT,*)
         DO 100 J = 0, N - 1
            WRITE (NOUT,99999) J, X(J)
  100    CONTINUE
*
         CALL C06FAF(X,N,WORK,IFAIL)
         CALL C06GBF(X,N,IFAIL)
*
         WRITE (NOUT,*)
```

```
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '        Original  Restored'
         WRITE (NOUT,*)
         DO 120 J = 0, N - 1
            WRITE (NOUT,99998) J, XX(J), X(J)
  120    CONTINUE
         GO TO 20
       ELSE
         WRITE (NOUT,*) 'Invalid value of N'
       END IF
  140  STOP
*
99999 FORMAT (1X,I5,F10.5,A,2F10.5)
99998 FORMAT (1X,I5,2F10.5)
      END
```

9.2 Program Data

```
C06FBF Example Program Data
   7
 0.34907
 0.54890
 0.74776
 0.94459
 1.13850
 1.32850
 1.51370
```

9.3 Program Results

```
C06FBF Example Program Results

Original sequence and corresponding complex sequence

         Data           Real      Imag

    0   0.34907        0.34907   0.00000
    1   0.54890        0.54890   1.51370
    2   0.74776        0.74776   1.32850
    3   0.94459        0.94459   1.13850
    4   1.13850        0.94459  -1.13850
    5   1.32850        0.74776  -1.32850
    6   1.51370        0.54890  -1.51370

Components of discrete Fourier transform

    0   1.82616
    1   1.86862
    2  -0.01750
    3   0.50200
    4  -0.59873
    5  -0.03144
    6  -2.62557
```

```
Original sequence as restored by inverse transform

        Original  Restored

   0    0.34907   0.34907
   1    0.54890   0.54890
   2    0.74776   0.74776
   3    0.94459   0.94459
   4    1.13850   1.13850
   5    1.32850   1.32850
   6    1.51370   1.51370
```

C06FCF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06FCF calculates the discrete Fourier transform of a sequence of n complex data values (using a work array for extra speed).

2 Specification

```
SUBROUTINE C06FCF(X, Y, N, WORK, IFAIL)
INTEGER           N, IFAIL
real              X(N), Y(N), WORK(N)
```

3 Description

Given a sequence of n complex data values z_j, for $j = 0, 1, \ldots, n-1$, this routine calculates their discrete Fourier transform defined by:

$$\hat{z}_k = a_k + ib_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \times \exp\left(-i\frac{2\pi jk}{n}\right), \quad k = 0, 1, \ldots, n-1.$$

(Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.)

To compute the inverse discrete Fourier transform defined by:

$$\hat{w}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \times \exp\left(+i\frac{2\pi jk}{n}\right),$$

this routine should be preceded and followed by calls of C06GCF to form the complex conjugates of the z_j and the $\hat{z}_k$.

The routine uses the fast Fourier transform (FFT) algorithm in Brigham [1]. There are some restrictions on the value of n (see Section 5).

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

5 Parameters

1: X(N) — ***real*** array *Input/Output*

On entry: if X is declared with bounds (0:N−1) in the (sub)program from which C06FCF is called, then X(j) must contain x_j, the real part of z_j, for $j = 0, 1, \ldots, n-1$.

On exit: the real parts a_k of the components of the discrete Fourier transform. If X is declared with bounds (0:N−1) in the (sub)program from which C06FCF is called, then for $0 \leq k \leq n-1$, a_k is contained in X(k).

2: Y(N) — ***real*** array *Input/Output*

On entry: if Y is declared with bounds (0:N−1) in the (sub)program from which C06FCF is called, then Y(j) must contain y_j, the imaginary part of z_j, for $j = 0, 1, \ldots, n-1$.

On exit: the imaginary parts b_k of the components of the discrete Fourier transform. If Y is declared with bounds (0:N−1) in the (sub)program from which C06FCF is called, then for $0 \leq k \leq n-1$, b_k is contained in Y(k).

3: N — INTEGER *Input*

On entry: the number of data values, n. The largest prime factor of N must not exceed 19, and the total number of prime factors of N, counting repetitions, must not exceed 20.

Constraint: N > 1.

4: WORK(N) — ***real*** array *Workspace*

5: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

At least one of the prime factors of N is greater than 19.

IFAIL = 2

N has more than 20 prime factors.

IFAIL = 3

N ≤ 1.

IFAIL = 4

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $n \times \log n$, but also depends on the factorization of n. The routine is somewhat faster than average if the only prime factors of n are 2, 3 or 5; and fastest of all if n is a power of 2.

9 Example

This program reads in a sequence of complex data values and prints their discrete Fourier transform (as computed by C06FCF).

It then performs an inverse transform, using C06GCF and C06FCF, and prints the sequence so obtained alongside the original data values.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, N
*     .. Local Arrays ..
      real             WORK(NMAX), X(0:NMAX-1), XX(0:NMAX-1),
     +                 Y(0:NMAX-1), YY(0:NMAX-1)
*     .. External Subroutines ..
      EXTERNAL         C06FCF, C06GCF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FCF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=100) N
      IF (N.GT.1 .AND. N.LE.NMAX) THEN
         DO 40 J = 0, N - 1
            READ (NIN,*) X(J), Y(J)
            XX(J) = X(J)
            YY(J) = Y(J)
   40    CONTINUE
         IFAIL = 0
*
         CALL C06FCF(X,Y,N,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of discrete Fourier transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '          Real      Imag'
         WRITE (NOUT,*)
         DO 60 J = 0, N - 1
            WRITE (NOUT,99999) J, X(J), Y(J)
   60    CONTINUE
*
         CALL C06GCF(Y,N,IFAIL)
         CALL C06FCF(X,Y,N,WORK,IFAIL)
         CALL C06GCF(Y,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '             Original                  Restored'
         WRITE (NOUT,*)
     +     '          Real      Imag           Real      Imag'
         WRITE (NOUT,*)
         DO 80 J = 0, N - 1
            WRITE (NOUT,99999) J, XX(J), YY(J), X(J), Y(J)
   80    CONTINUE
         GO TO 20
```

```
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
  100 STOP
*
99999 FORMAT (1X,I5,2F10.5,5X,2F10.5)
      END
```

9.2 Program Data

```
C06FCF Example Program Data
    7
  0.34907  -0.37168
  0.54890  -0.35669
  0.74776  -0.31175
  0.94459  -0.23702
  1.13850  -0.13274
  1.32850   0.00074
  1.51370   0.16298
```

9.3 Program Results

```
C06FCF Example Program Results

Components of discrete Fourier transform

          Real      Imag

    0   2.48361  -0.47100
    1  -0.55180   0.49684
    2  -0.36711   0.09756
    3  -0.28767  -0.05865
    4  -0.22506  -0.17477
    5  -0.14825  -0.30840
    6   0.01983  -0.56496

Original sequence as restored by inverse transform

              Original                Restored
            Real      Imag          Real      Imag

    0   0.34907  -0.37168       0.34907  -0.37168
    1   0.54890  -0.35669       0.54890  -0.35669
    2   0.74776  -0.31175       0.74776  -0.31175
    3   0.94459  -0.23702       0.94459  -0.23702
    4   1.13850  -0.13274       1.13850  -0.13274
    5   1.32850   0.00074       1.32850   0.00074
    6   1.51370   0.16298       1.51370   0.16298
```

C06FFF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06FFF computes the discrete Fourier transform of one variable in a multivariate sequence of complex data values.

2. Specification

```
SUBROUTINE C06FFF (NDIM, L, ND, N, X, Y, WORK, LWORK, IFAIL)
INTEGER          NDIM, L, ND(NDIM), N, LWORK, IFAIL
real             X(N), Y(N), WORK(LWORK)
```

3. Description

This routine computes the discrete Fourier transform of one variable (the lth say) in a multivariate sequence of complex data values $z_{j_1 j_2 \ldots j_m}$, where $j_1 = 0,1,\ldots,n_1-1$, $j_2 = 0,1,\ldots,n_2-1$, and so on. Thus the individual dimensions are $n_1, n_2, \ldots, n_m$, and the total number of data values is $n = n_1 \times n_2 \times \ldots \times n_m$.

The routine computes n/n_l one-dimensional transforms defined by:

$$\hat{z}_{j_1 \ldots k_l \ldots j_m} = \frac{1}{\sqrt{n_l}} \sum_{j_l=0}^{n_l-1} z_{j_1 \ldots j_l \ldots j_m} \times \exp\left(-\frac{2\pi i j_l k_l}{n_l}\right)$$

where $k_l = 0,1,\ldots,n_l-1$.

(Note the scale factor of $\frac{1}{\sqrt{n_l}}$ in this definition.)

To compute the inverse discrete Fourier transforms, defined with $\exp\left(+\frac{2\pi i j_l k_l}{n_l}\right)$ in the above formula instead of $\exp\left(-\frac{2\pi i j_l k_l}{n_l}\right)$, this routine should be preceded and followed by calls of C06GCF to form the complex conjugates of the data values and the transform.

The data values must be supplied in a pair of one-dimensional arrays (real and imaginary parts separately), in accordance with the Fortran convention for storing multi-dimensional data (i.e. with the first subscript j_1 varying most rapidly).

This routine calls C06FCF to perform one-dimensional discrete Fourier transforms by the Fast Fourier Transform algorithm in Brigham [1], and hence there are some restrictions on the values of n_l (See Section 5).

4. References

[1] BRIGHAM, E.O.
The Fast Fourier Transform.
Prentice-Hall, 1973.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions (or variables) in the multivariate data, m.

Constraint: NDIM ≥ 1.

2: L – INTEGER. *Input*

On entry: the index of the variable (or dimension) on which the discrete Fourier transform is to be performed, l.

Constraint: $1 \leq L \leq NDIM$.

3: ND(NDIM) – INTEGER array. *Input*

On entry: ND(i) must contain n_i (the dimension of the ith variable), for $i = 1,2,...,m$. The largest prime factor of ND(l) must not exceed 19, and the total number of prime factors of ND(l), counting repetitions, must not exceed 20.

Constraint: ND(i) ≥ 1 for all i.

4: N – INTEGER. *Input*

On entry: the total number of data values, n.

Constraint: N = ND(1)×ND(2)×...×ND(NDIM).

5: X(N) – ***real*** array. *Input/Output*

On entry: X($1+j_1+n_1j_2+n_1n_2j_3+...$) must contain the real part of the complex data value $z_{j_1j_2...j_m}$, for $0 \leq j_1 < n_1, 0 \leq j_2 < n_2,...$; i.e. the values are stored in consecutive elements of the array according to the Fortran convention for storing multi-dimensional arrays.

On exit: the real parts of the corresponding elements of the computed transform.

6: Y(N) – ***real*** array. *Input/Output*

On entry: the imaginary parts of the complex data values, stored in the same way as the real parts in the array X.

On exit: the imaginary parts of the corresponding elements of the computed transform.

7: WORK(LWORK) – ***real*** array. *Workspace*

8: LWORK – INTEGER. *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which C06FFF is called.

Constraint: LWORK $\geq$ 3×ND(L).

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

NDIM < 1.

IFAIL = 2

N ≠ ND(1)×ND(2)×...×ND(NDIM).

IFAIL = 3

L < 1 or L > NDIM.

IFAIL = 10×L + 1

At least one of the prime factors of ND(L) is greater than 19.

IFAIL = 10×L + 2

ND(L) has more than 20 prime factors.

IFAIL = 10×L + 3

ND(L) < 1.

IFAIL = 10×L + 4

LWORK < 3×ND(L).

7. Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8. Further Comments

The time taken by the routine is approximately proportional to $n \times \log n_l$, but also depends on the factorization of n_l. The routine is somewhat faster than average if the only prime factors of n_l are 2, 3 or 5; and fastest of all if n_l is a power of 2.

9. Example

This program reads in a bivariate sequence of complex data values and prints the discrete Fourier transform of the second variable. It then performs an inverse transform and prints the sequence so obtained, which may be compared with the original data values.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FFF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM, NMAX, LWORK
      PARAMETER        (NDIM=2,NMAX=96,LWORK=96)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, L, N
*     .. Local Arrays ..
      real             WORK(LWORK), X(NMAX), Y(NMAX)
      INTEGER          ND(NDIM)
*     .. External Subroutines ..
      EXTERNAL         C06FFF, C06GCF, READXY, WRITXY
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FFF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=40) ND(1), ND(2), L
      N = ND(1)*ND(2)
      IF (N.GE.1 .AND. N.LE.NMAX) THEN
         CALL READXY(NIN,X,Y,ND(1),ND(2))
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data'
         CALL WRITXY(NOUT,X,Y,ND(1),ND(2))
         IFAIL = 0
*
*        Compute transform
         CALL C06FFF(NDIM,L,ND,N,X,Y,WORK,LWORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Discrete Fourier transform of variable ', L
         CALL WRITXY(NOUT,X,Y,ND(1),ND(2))
*
```

```
*        Compute inverse transform
         CALL C06GCF(Y,N,IFAIL)
         CALL C06FFF(NDIM,L,ND,N,X,Y,WORK,LWORK,IFAIL)
         CALL C06GCF(Y,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         CALL WRITXY(NOUT,X,Y,ND(1),ND(2))
         GO TO 20
       ELSE
         WRITE (NOUT,*) 'Invalid value of N'
       END IF
    40 STOP
*
99999 FORMAT (1X,A,I1)
      END
*
      SUBROUTINE READXY(NIN,X,Y,N1,N2)
*     Read 2-dimensional complex data
*     .. Scalar Arguments ..
      INTEGER           N1, N2, NIN
*     .. Array Arguments ..
      real              X(N1,N2), Y(N1,N2)
*     .. Local Scalars ..
      INTEGER           I, J
*     .. Executable Statements ..
      DO 20 I = 1, N1
         READ (NIN,*) (X(I,J),J=1,N2)
         READ (NIN,*) (Y(I,J),J=1,N2)
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE WRITXY(NOUT,X,Y,N1,N2)
*     Print 2-dimensional complex data
*     .. Scalar Arguments ..
      INTEGER           N1, N2, NOUT
*     .. Array Arguments ..
      real              X(N1,N2), Y(N1,N2)
*     .. Local Scalars ..
      INTEGER           I, J
*     .. Executable Statements ..
      DO 20 I = 1, N1
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Real ', (X(I,J),J=1,N2)
         WRITE (NOUT,99999) 'Imag ', (Y(I,J),J=1,N2)
   20 CONTINUE
      RETURN
*
99999 FORMAT (1X,A,7F10.3,/(6X,7F10.3))
      END
```

9.2. Program Data

```
C06FFF Example Program Data
    3    5    2
     1.000     0.999     0.987     0.936     0.802
     0.000    -0.040    -0.159    -0.352    -0.597
     0.994     0.989     0.963     0.891     0.731
    -0.111    -0.151    -0.268    -0.454    -0.682
     0.903     0.885     0.823     0.694     0.467
    -0.430    -0.466    -0.568    -0.720    -0.884
```

9.3. Program Results

```
C06FFF Example Program Results

Original data

Real      1.000     0.999     0.987     0.936     0.802
Imag      0.000    -0.040    -0.159    -0.352    -0.597

Real      0.994     0.989     0.963     0.891     0.731
Imag     -0.111    -0.151    -0.268    -0.454    -0.682

Real      0.903     0.885     0.823     0.694     0.467
Imag     -0.430    -0.466    -0.568    -0.720    -0.884

Discrete Fourier transform of variable 2

Real      2.113     0.288     0.126    -0.003    -0.287
Imag     -0.513     0.000     0.130     0.190     0.194

Real      2.043     0.286     0.139     0.018    -0.263
Imag     -0.745    -0.032     0.115     0.189     0.225

Real      1.687     0.260     0.170     0.079    -0.176
Imag     -1.372    -0.125     0.063     0.173     0.299

Original sequence as restored by inverse transform

Real      1.000     0.999     0.987     0.936     0.802
Imag      0.000    -0.040    -0.159    -0.352    -0.597

Real      0.994     0.989     0.963     0.891     0.731
Imag     -0.111    -0.151    -0.268    -0.454    -0.682

Real      0.903     0.885     0.823     0.694     0.467
Imag     -0.430    -0.466    -0.568    -0.720    -0.884
```

C06FJF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06FJF computes the multi-dimensional discrete Fourier transform of a multivariate sequence of complex data values.

2. Specification

```
SUBROUTINE C06FJF (NDIM, ND, N, X, Y, WORK, LWORK, IFAIL)
INTEGER          NDIM, ND(NDIM), N, LWORK, IFAIL
real             X(N), Y(N), WORK(LWORK)
```

3. Description

This routine computes the multi-dimensional discrete Fourier transform of a multi-dimensional sequence of complex data values $z_{j_1 j_2 \cdots j_m}$, where $j_1 = 0,1,\ldots,n_1-1$, $j_2 = 0,1,\ldots,n_2-1$, and so on. Thus the individual dimensions are $n_1, n_2, \ldots, n_m$, and the total number of data values $n = n_1 \times n_2 \times \cdots \times n_m$.

The discrete Fourier transform is here defined (e.g. for $m = 2$) by:

$$\hat{z}_{k_1,k_2} = \frac{1}{\sqrt{n}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} z_{j_1,j_2} \times \exp\left(-2\pi i\left(\frac{j_1 k_1}{n_1} + \frac{j_2 k_2}{n_2}\right)\right),$$

where $k_1 = 0,1,\ldots,n_1-1$, $k_2 = 0,1,\ldots,n_2-1$.

The extension to higher dimensions is obvious. (Note the scale factor of $\frac{1}{\sqrt{n}}$ in this definition.)

To compute the inverse discrete Fourier transform, defined with $\exp(+2\pi i(\ldots))$ in the above formula instead of $\exp(-2\pi i(\ldots))$, this routine should be preceded and followed by calls of C06GCF to form the complex conjugates of the data values and the transform.

The data values must be supplied in a pair of one-dimensional arrays (real and imaginary parts separately), in accordance with the Fortran convention for storing multi-dimensional data (i.e. with the first subscript j_1 varying most rapidly).

This routine calls C06FCF to perform one-dimensional discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in Brigham [1], and hence there are some restrictions on the values of the n_i (see Section 5).

4. References

[1] BRIGHAM, E.O.
The Fast Fourier Transform.
Prentice-Hall, 1973.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions (or variables), m, in the multivariate data.

Constraint: NDIM $\geq$ 1.

2: ND(NDIM) – INTEGER array. *Input*

On entry: ND(i) must contain n_i (the dimension of the ith variable), for $i = 1,2,\ldots,m$. The largest prime factor of each ND(i) must not exceed 19, and the total number of prime factors of ND(i), counting repetitions, must not exceed 20.

Constraint: ND(i) $\geq$ 1.

3: N – INTEGER. *Input*

On entry: the total number of data values, n.

Constraint: N = ND(1)×ND(2)×···×ND(NDIM).

4: X(N) – ***real*** array. *Input/Output*

On entry: X($1+j_1+n_1j_2+n_1n_2j_3+$...) must contain the real part of the complex data value $z_{j_1j_2\cdots j_m}$, for $0 \le j_1 \le n_1-1$, $0 \le j_2 \le n_2-1$,...; i.e. the values are stored in consecutive elements of the array according to the Fortran convention for storing multi-dimensional arrays.

On exit: the real parts of the corresponding elements of the computed transform.

5: Y(N) – ***real*** array. *Input/Output*

On entry: the imaginary parts of the complex data values, stored in the same way as the real parts in the array X.

On exit: the imaginary parts of the corresponding elements of the computed transform.

6: WORK(LWORK) – ***real*** array. *Workspace*

7: LWORK – INTEGER. *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which C06FJF is called.

Constraint: LWORK ≥ 3×max{ND(i)}.

8: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

NDIM < 1.

IFAIL = 2

N ≠ ND(1)×ND(2)×···×ND(NDIM).

IFAIL = 10×L + 1

At least one of the prime factors of ND(L) is greater than 19.

IFAIL = 10×L + 2

ND(L) has more than 20 prime factors.

IFAIL = 10×L + 3

ND(L) < 1.

IFAIL = 10×L + 4

LWORK < 3×ND(L).

7. Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8. Further Comments

The time taken by the routine is approximately proportional to $n \times \log n$, but also depends on the factorization of the individual dimensions ND(i). The routine is somewhat faster than average if their only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

9. Example

This program reads in a bivariate sequence of complex data values and prints the two-dimensional Fourier transform. It then performs an inverse transform and prints the sequence so obtained, which may be compared to the original data values.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FJF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM, NMAX, LWORK
      PARAMETER        (NDIM=2,NMAX=96,LWORK=96)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, N
*     .. Local Arrays ..
      real             WORK(LWORK), X(NMAX), Y(NMAX)
      INTEGER          ND(NDIM)
*     .. External Subroutines ..
      EXTERNAL         C06FJF, C06GCF, READXY, WRITXY
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FJF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=40) ND(1), ND(2)
      N = ND(1)*ND(2)
      IF (N.GE.1 .AND. N.LE.NMAX) THEN
         CALL READXY(NIN,X,Y,ND(1),ND(2))
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         CALL WRITXY(NOUT,X,Y,ND(1),ND(2))
         IFAIL = 0
*
*        Compute transform
         CALL C06FJF(NDIM,ND,N,X,Y,WORK,LWORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of discrete Fourier transform'
         CALL WRITXY(NOUT,X,Y,ND(1),ND(2))
*
*        Compute inverse transform
         CALL C06GCF(Y,N,IFAIL)
         CALL C06FJF(NDIM,ND,N,X,Y,WORK,LWORK,IFAIL)
         CALL C06GCF(Y,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         CALL WRITXY(NOUT,X,Y,ND(1),ND(2))
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
   40 STOP
      END
*
```

```
      SUBROUTINE READXY(NIN,X,Y,N1,N2)
*     Read 2-dimensional complex data
*     .. Scalar Arguments ..
      INTEGER           N1, N2, NIN
*     .. Array Arguments ..
      real              X(N1,N2), Y(N1,N2)
*     .. Local Scalars ..
      INTEGER           I, J
*     .. Executable Statements ..
      DO 20 I = 1, N1
         READ (NIN,*) (X(I,J),J=1,N2)
         READ (NIN,*) (Y(I,J),J=1,N2)
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE WRITXY(NOUT,X,Y,N1,N2)
*     Print 2-dimensional complex data
*     .. Scalar Arguments ..
      INTEGER           N1, N2, NOUT
*     .. Array Arguments ..
      real              X(N1,N2), Y(N1,N2)
*     .. Local Scalars ..
      INTEGER           I, J
*     .. Executable Statements ..
      DO 20 I = 1, N1
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Real ', (X(I,J),J=1,N2)
         WRITE (NOUT,99999) 'Imag ', (Y(I,J),J=1,N2)
   20 CONTINUE
      RETURN
*
99999 FORMAT (1X,A,7F10.3,/(6X,7F10.3))
      END
```

9.2. Program Data

```
C06FJF Example Program Data
    3    5
     1.000     0.999     0.987     0.936     0.802
     0.000    -0.040    -0.159    -0.352    -0.597
     0.994     0.989     0.963     0.891     0.731
    -0.111    -0.151    -0.268    -0.454    -0.682
     0.903     0.885     0.823     0.694     0.467
    -0.430    -0.466    -0.568    -0.720    -0.884
```

9.3. Program Results

```
 C06FJF Example Program Results

 Original data values

 Real      1.000     0.999     0.987     0.936     0.802
 Imag      0.000    -0.040    -0.159    -0.352    -0.597

 Real      0.994     0.989     0.963     0.891     0.731
 Imag     -0.111    -0.151    -0.268    -0.454    -0.682

 Real      0.903     0.885     0.823     0.694     0.467
 Imag     -0.430    -0.466    -0.568    -0.720    -0.884
```

```
Components of discrete Fourier transform

Real      3.373     0.481     0.251     0.054    -0.419
Imag     -1.519    -0.091     0.178     0.319     0.415

Real      0.457     0.055     0.009    -0.022    -0.076
Imag      0.137     0.032     0.039     0.036     0.004

Real     -0.170    -0.037    -0.042    -0.038    -0.002
Imag      0.493     0.058     0.008    -0.025    -0.083

Original sequence as restored by inverse transform

Real      1.000     0.999     0.987     0.936     0.802
Imag      0.000    -0.040    -0.159    -0.352    -0.597

Real      0.994     0.989     0.963     0.891     0.731
Imag     -0.111    -0.151    -0.268    -0.454    -0.682

Real      0.903     0.885     0.823     0.694     0.467
Imag     -0.430    -0.466    -0.568    -0.720    -0.884
```

C06FKF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06FKF calculates the circular convolution or correlation of two real vectors of period n (using a work array for extra speed).

2. Specification

```
SUBROUTINE C06FKF (JOB, X, Y, N, WORK, IFAIL)
INTEGER          JOB, N, IFAIL
real             X(N), Y(N), WORK(N)
```

3. Description

This routine computes:

if JOB = 1, the discrete **convolution** of x and y, defined by:

$$z_k = \sum_{j=0}^{n-1} x_j y_{k-j} = \sum_{j=0}^{n-1} x_{k-j} y_j;$$

if JOB = 2, the discrete **correlation** of x and y defined by:

$$w_k = \sum_{j=0}^{n-1} x_j y_{k+j}.$$

Here x and y are real vectors, assumed to be periodic, with period n, i.e. $x_j = x_{j\pm n} = x_{j\pm 2n} = \cdots$; z and w are then also periodic with period n.

Note: this usage of the terms 'convolution' and 'correlation' is taken from Brigham [1]. The term 'convolution' is sometimes used to denote both these computations.

If $\hat{x}$, $\hat{y}$, $\hat{z}$ and $\hat{w}$ are the discrete Fourier transforms of these sequences,

i.e. $\hat{x}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \times \exp\left(-i\frac{2\pi jk}{n}\right)$, etc.,

then $\hat{z}_k = \sqrt{n}.\hat{x}_k\hat{y}_k$

and $\hat{w}_k = \sqrt{n}.\overline{\hat{x}}_k\hat{y}_k$

(the bar denoting complex conjugate).

This routine calls the same auxiliary routines as C06FAF and C06FBF to compute discrete Fourier transforms, and there are some restrictions on the value of n.

4. References

[1] BRIGHAM, E.O.
The Fast Fourier Transform.
Prentice-Hall, 1973.

5. Parameters

1: JOB – INTEGER. *Input*

On entry: the computation to be performed:

if JOB = 1, $z_k = \sum_{j=0}^{n-1} x_j y_{k-j}$ (convolution);

if JOB = 2, $w_k = \sum_{j=0}^{n-1} x_j y_{k+j}$ (correlation).

Constraint: JOB = 1 or 2.

2: X(N) – ***real*** array. *Input/Output*

On entry: the elements of one period of the vector x. If X is declared with bounds (0:N–1) in the (sub)program from which C06FKF is called, then X(j) must contains x_j, for $j = 0,1,...,n-1$.

On exit: the corresponding elements of the discrete convolution or correlation.

3: Y(N) – ***real*** array. *Input/Output*

On entry: the elements of one period of the vector y. If Y is declared with bounds (0:N–1) in the (sub)program from which C06FKF is called, then Y(j) must contain y_j, for $j = 0,1,...,n-1$.

On exit: the discrete Fourier transform of the convolution or correlation returned in the array X; the transform is stored in Hermitian form, exactly as described in the document for C06FAF.

4: N – INTEGER. *Input*

On entry: the number of values, n, in one period of the vectors X and Y. The largest prime factor of N must not exceed 19 and the total number of prime factors of N, counting repetitions, must not exceed 20.

Constraint: N > 1.

5: WORK(N) – ***real*** array. *Workspace*

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

At least one of the prime factors of N is greater than 19.

IFAIL = 2

N has more than 20 prime factors.

IFAIL = 3

N ≤ 1.

IFAIL = 4

JOB ≠ 1 or 2.

7. Accuracy

The results should be accurate to within a small multiple of the ***machine precision***.

8. Further Comments

The time taken by the routine is approximately proportional to $n \times \log n$, but also depends on the factorization of n. The routine is somewhat faster than average if the only prime factors of n are 2, 3 or 5; and fastest of all if n is a power of 2.

9. Example

This program reads in the elements of one period of two real vectors x and y, and prints their discrete convolution and correlation (as computed by C06FKF). In realistic computations the number of data values would be much larger.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FKF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=64)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, N
*     .. Local Arrays ..
      real             WORK(NMAX), XA(NMAX), XB(NMAX), YA(NMAX),
     +                 YB(NMAX)
*     .. External Subroutines ..
      EXTERNAL         C06FKF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FKF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=80) N
      WRITE (NOUT,*)
      IF (N.GT.1 .AND. N.LE.NMAX) THEN
         DO 40 J = 1, N
            READ (NIN,*) XA(J), YA(J)
            XB(J) = XA(J)
            YB(J) = YA(J)
   40    CONTINUE
         IFAIL = 0
*
         CALL C06FKF(1,XA,YA,N,WORK,IFAIL)
         CALL C06FKF(2,XB,YB,N,WORK,IFAIL)
*
         WRITE (NOUT,*) '        Convolution  Correlation'
         WRITE (NOUT,*)
         DO 60 J = 1, N
            WRITE (NOUT,99999) J - 1, XA(J), XB(J)
   60    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
   80 STOP
*
99999 FORMAT (1X,I5,2F13.5)
      END
```

9.2. Program Data

```
C06FKF Example Program Data
    9
      1.00      0.50
      1.00      0.50
      1.00      0.50
      1.00      0.50
      1.00      0.00
      0.00      0.00
      0.00      0.00
      0.00      0.00
      0.00      0.00
```

9.3. Program Results

```
C06FKF Example Program Results

        Convolution  Correlation

 0        0.50000      2.00000
 1        1.00000      1.50000
 2        1.50000      1.00000
 3        2.00000      0.50000
 4        2.00000      0.00000
 5        1.50000      0.50000
 6        1.00000      1.00000
 7        0.50000      1.50000
 8        0.00000      2.00000
```

C06FPF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06FPF computes the discrete Fourier transforms of m sequences, each containing n real data values. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
SUBROUTINE C06FPF(M, N, X, INIT, TRIG, WORK, IFAIL)
INTEGER           M, N, IFAIL
real              X(M*N), TRIG(2*N), WORK(M*N)
CHARACTER*1       INIT
```

3 Description

Given m sequences of n real data values x_j^p, for $j = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$, this routine simultaneously calculates the Fourier transforms of all the sequences defined by:

$$\hat{z}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j^p \times \exp\left(-i\frac{2\pi jk}{n}\right), \quad k = 0, 1, \ldots, n-1; \quad p = 1, 2, \ldots, m.$$

(Note the scale factor $\frac{1}{\sqrt{n}}$ in this definition.)

The transformed values $\hat{z}_k^p$ are complex, but for each value of p the $\hat{z}_k^p$ form a Hermitian sequence (i.e., $\hat{z}_{n-k}^p$ is the complex conjugate of $\hat{z}_k^p$), so they are completely determined by mn real numbers (see also the Chapter Introduction).

The discrete Fourier transform is sometimes defined using a positive sign in the exponential term:

$$\hat{z}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j^p \times \exp\left(+i\frac{2\pi jk}{n}\right).$$

To compute this form, this routine should be followed by a call to C06GQF to form the complex conjugates of the $\hat{z}_k^p$.

The routine uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, which is described in Temperton [2]. Special coding is provided for the factors 2, 3, 4, 5 and 6. This routine is designed to be particularly efficient on vector processors, and it becomes especially fast as M, the number of transforms to be computed in parallel, increases.

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Temperton C (1983) Fast mixed-radix real Fourier transforms *J. Comput. Phys.* **52** 340–350

5 Parameters

1: M — INTEGER *Input*

On entry: the number of sequences to be transformed, m.

Constraint: M $\geq$ 1.

2: N — INTEGER *Input*

On entry: the number of real values in each sequence, n.

Constraint: N ≥ 1.

3: X(M*N) — ***real*** array *Input/Output*

On entry: the data must be stored in X as if in a two-dimensional array of dimension (1:M,0:N−1); each of the m sequences is stored in a **row** of the array. In other words, if the data values of the pth sequence to be transformed are denoted by x_j^p, for $j = 0, 1, \ldots, n-1$, then the mn elements of the array X must contain the values

$$x_0^1, x_0^2, \ldots, x_0^m,\ x_1^1, x_1^2, \ldots, x_1^m, \ldots,\ x_{n-1}^1, x_{n-1}^2, \ldots, x_{n-1}^m.$$

On exit: the m discrete Fourier transforms stored as if in a two-dimensional array of dimension (1:M,0:N−1). Each of the m transforms is stored in a **row** of the array in Hermitian form, overwriting the corresponding original sequence. If the n components of the discrete Fourier transform $\hat{z}_k^p$ are written as $a_k^p + ib_k^p$, then for $0 \leq k \leq n/2$, a_k^p is contained in X(p,k), and for $1 \leq k \leq (n-1)/2$, b_k^p is contained in X$(p, n-k)$. (See also Section 2.1.2 of the Chapter Introduction.)

4: INIT — CHARACTER*1 *Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the array TRIG, then INIT must be set equal to 'I' (**I**nitial call).

If INIT contains 'S' (**S**ubsequent call), then the routine assumes that trigonometric coefficients for the specified value of n are supplied in the array TRIG, having been calculated in a previous call to one of C06FPF, C06FQF or C06FRF.

If INIT contains 'R' (**R**estart) then the routine assumes that trigonometric coefficients for the particular value of n are supplied in the array TRIG, but does not check that C06FPF, C06FQF or C06FRF have previously been called. This option allows the TRIG array to be stored in an external file, read in and re-used without the need for a call with INIT equal to 'I'. The routine carries out a simple test to check that the current value of n is consistent with the array TRIG.

Constraint: INIT = 'I', 'S' or 'R'.

5: TRIG(2*N) — ***real*** array *Input/Output*

On entry: if INIT = 'S' or 'R', TRIG must contain the required coefficients calculated in a previous call of the routine. Otherwise TRIG need not be set.

On exit: TRIG contains the required coefficients (computed by the routine if INIT = 'I').

6: WORK(M*N) — ***real*** array *Workspace*

7: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

IFAIL = 1

On entry, M < 1.

IFAIL = 2

N < 1.

IFAIL = 3

INIT is not one of 'I', 'S' or 'R'.

IFAIL = 4

INIT = 'S', but none of C06FPF, C06FQF or C06FRF have previously been called.

IFAIL = 5

INIT = 'S' or 'R', but the array TRIG and the current value of N are inconsistent.

IFAIL = 6

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $nm \times \log n$, but also depends on the factors of n. The routine is fastest if the only prime factors of n are 2, 3 and 5, and is particularly slow if n is a large prime, or has large prime factors.

9 Example

This program reads in sequences of real data values and prints their discrete Fourier transforms (as computed by C06FPF). The Fourier transforms are expanded into full complex form using C06GSF and printed. Inverse transforms are then calculated by calling C06GQF followed by C06FQF showing that the original sequences are restored.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FPF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=5,NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, J, M, N
*     .. Local Arrays ..
      real             TRIG(2*NMAX), U(NMAX*MMAX), V(NMAX*MMAX),
     +                 WORK(2*MMAX*NMAX), X(NMAX*MMAX)
*     .. External Subroutines ..
      EXTERNAL         C06FPF, C06FQF, C06GQF, C06GSF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FPF Example Program Results'
*     Skip heading in data file
```

```
      READ (NIN,*)
   20 READ (NIN,*,END=140) M, N
      IF (M.LE.MMAX .AND. N.LE.NMAX) THEN
         DO 40 J = 1, M
            READ (NIN,*) (X(I*M+J),I=0,N-1)
   40    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         WRITE (NOUT,*)
         DO 60 J = 1, M
            WRITE (NOUT,99999) '     ', (X(I*M+J),I=0,N-1)
   60    CONTINUE
         IFAIL = 0
*
         CALL C06FPF(M,N,X,'Initial',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Discrete Fourier transforms in Hermitian format'
         WRITE (NOUT,*)
         DO 80 J = 1, M
            WRITE (NOUT,99999) '     ', (X(I*M+J),I=0,N-1)
   80    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Fourier transforms in full complex form'
*
         CALL C06GSF(M,N,X,U,V,IFAIL)
*
         DO 100 J = 1, M
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Real ', (U(I*M+J),I=0,N-1)
            WRITE (NOUT,99999) 'Imag ', (V(I*M+J),I=0,N-1)
  100    CONTINUE
*
         CALL C06GQF(M,N,X,IFAIL)
         CALL C06FQF(M,N,X,'Subsequent',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data as restored by inverse transform'
         WRITE (NOUT,*)
         DO 120 J = 1, M
            WRITE (NOUT,99999) '     ', (X(I*M+J),I=0,N-1)
  120    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of M or N'
      END IF
  140 STOP
*
99999 FORMAT (1X,A,6F10.4)
      END
```

9.2 Program Data

```
C06FPF Example Program Data
     3     6
     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```

9.3 Program Results

```
C06FPF Example Program Results

Original data values

          0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
          0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
          0.9172    0.0644    0.6037    0.6430    0.0428    0.4815

Discrete Fourier transforms in Hermitian format

          1.0737   -0.1041    0.1126   -0.1467   -0.3738   -0.0044
          1.3961   -0.0365    0.0780   -0.1521   -0.0607    0.4666
          1.1237    0.0914    0.3936    0.1530    0.3458   -0.0508

Fourier transforms in full complex form

Real      1.0737   -0.1041    0.1126   -0.1467    0.1126   -0.1041
Imag      0.0000   -0.0044   -0.3738    0.0000    0.3738    0.0044

Real      1.3961   -0.0365    0.0780   -0.1521    0.0780   -0.0365
Imag      0.0000    0.4666   -0.0607    0.0000    0.0607   -0.4666

Real      1.1237    0.0914    0.3936    0.1530    0.3936    0.0914
Imag      0.0000   -0.0508    0.3458    0.0000   -0.3458    0.0508

Original data as restored by inverse transform

          0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
          0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
          0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```

C06FQF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06FQF computes the discrete Fourier transforms of m Hermitian sequences, each containing n complex data values. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
SUBROUTINE C06FQF(M, N, X, INIT, TRIG, WORK, IFAIL)
INTEGER           M, N, IFAIL
real              X(M*N), TRIG(2*N), WORK(M*N)
CHARACTER*1       INIT
```

3 Description

Given m Hermitian sequences of n complex data values z_j^p, for $j = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$, this routine simultaneously calculates the Fourier transforms of all the sequences defined by:

$$\hat{x}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j^p \times \exp\left(-i\frac{2\pi jk}{n}\right), \quad k = 0, 1, \ldots, n-1; \quad p = 1, 2, \ldots, m.$$

(Note the scale factor $\frac{1}{\sqrt{n}}$ in this definition.)

The transformed values are purely real (see also the Chapter Introduction).

The discrete Fourier transform is sometimes defined using a positive sign in the exponential term

$$\hat{x}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j^p \times \exp\left(+i\frac{2\pi jk}{n}\right).$$

To compute this form, this routine should be preceded by a call to C06GQF to form the complex conjugates of the $\hat{z}_j^p$.

The routine uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, which is described in Temperton [2]. Special code is included for the factors 2, 3, 4, 5 and 6. This routine is designed to be particularly efficient on vector processors, and it becomes especially fast as m, the number of transforms to be computed in parallel, increases.

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Temperton C (1983) Fast mixed-radix real Fourier transforms *J. Comput. Phys.* **52** 340–350

5 Parameters

1: M — INTEGER *Input*

On entry: the number of sequences to be transformed, m.

Constraint: M $\geq$ 1.

2: N — INTEGER *Input*

On entry: the number of data values in each sequence, n.

Constraint: N $\geq$ 1.

3: X(M*N) — ***real*** array *Input/Output*

On entry: the data must be stored in X as if in a two-dimensional array of dimension (1:M,0:N−1); each of the m sequences is stored in a **row** of the array in Hermitian form. If the n data values z_j^p are written as $x_j^p + iy_j^p$, then for $0 \leq j \leq n/2$, x_j^p is contained in X(p, j), and for $1 \leq j \leq (n-1)/2$, y_j^p is contained in X$(p, n-j)$. (See also Section 2.1.2 of the Chapter Introduction.)

On exit: the components of the m discrete Fourier transforms, stored as if in a two-dimensional array of dimension (1:M,0:N−1). Each of the m transforms is stored as a **row** of the array, overwriting the corresponding original sequence. If the n components of the discrete Fourier transform are denoted by $\hat{x}_k^p$, for $k = 0, 1, \ldots, n-1$, then the mn elements of the array X contain the values

$$\hat{x}_0^1, \hat{x}_0^2, \ldots, \hat{x}_0^m,\ \hat{x}_1^1, \hat{x}_1^2, \ldots,\ \hat{x}_1^m, \ldots,\ \hat{x}_{n-1}^1, \hat{x}_{n-1}^2, \ldots, \hat{x}_{n-1}^m.$$

4: INIT — CHARACTER*1 *Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the array TRIG, then INIT must be set equal to 'I' (**I**nitial call).

If INIT contains 'S' (**S**ubsequent call), then the routine assumes that trigonometric coefficients for the specified value of n are supplied in the array TRIG, having been calculated in a previous call to one of C06FPF, C06FQF or C06FRF.

If INIT contains 'R' (**R**estart), then the routine assumes that trigonometric coefficients for the particular value of N are supplied in the array TRIG, but does not check that C06FPF, C06FQF or C06FRF have previously been called. This option allows the TRIG array to be stored in an external file, read in and re-used without the need for a call with INIT equal to 'I'. The routine carries out a simple test to check that the current value of n is compatible with the array TRIG.

Constraint: INIT = 'I', 'S' or 'R'.

5: TRIG(2*N) — ***real*** array *Input/Output*

On entry: if INIT = 'S' or 'R, TRIG must contain the required coefficients calculated in a previous call of the routine. Otherwise TRIG need not be set.

On exit: TRIG contains the required coefficients (computed by the routine if INIT = 'I').

6: WORK(M*N) — ***real*** array *Workspace*

7: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

IFAIL = 1

On entry, M < 1.

IFAIL = 2

On entry, N < 1.

IFAIL = 3

On entry, INIT is not one of 'I', 'S' or 'R'.

IFAIL = 4

Not used at this Mark.

IFAIL = 5

On entry, INIT = 'S' or 'R', but the array TRIG and the current value of n are inconsistent.

IFAIL = 6

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $nm \times \log n$, but also depends on the factors of n. The routine is fastest if the only prime factors of n are 2, 3 and 5, and is particularly slow if n is a large prime, or has large prime factors.

9 Example

This program reads in sequences of real data values which are assumed to be Hermitian sequences of complex data stored in Hermitian form. The sequences are expanded into full complex form using C06GSF and printed. The discrete Fourier transforms are then computed (using C06FQF) and printed out. Inverse transforms are then calculated by calling C06FPF followed by C06GQF showing that the original sequences are restored.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FQF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=5,NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, J, M, N
*     .. Local Arrays ..
      real             TRIG(2*NMAX), U(MMAX*NMAX), V(MMAX*NMAX),
     +                 WORK(2*NMAX*MMAX), X(MMAX*NMAX)
*     .. External Subroutines ..
      EXTERNAL         C06FPF, C06FQF, C06GQF, C06GSF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FQF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=140) M, N
      IF (M.LE.MMAX .AND. N.LE.NMAX) THEN
         DO 40 J = 1, M
            READ (NIN,*) (X(I*M+J),I=0,N-1)
```

```
   40    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         WRITE (NOUT,*)
         DO 60 J = 1, M
            WRITE (NOUT,99999) '     ', (X(I*M+J),I=0,N-1)
   60    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data written in full complex form'
         IFAIL = 0
*
         CALL C06GSF(M,N,X,U,V,IFAIL)
*
         DO 80 J = 1, M
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Real ', (U(I*M+J),I=0,N-1)
            WRITE (NOUT,99999) 'Imag ', (V(I*M+J),I=0,N-1)
   80    CONTINUE
*
         CALL C06FQF(M,N,X,'Initial',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Discrete Fourier transforms (real values)'
         WRITE (NOUT,*)
         DO 100 J = 1, M
            WRITE (NOUT,99999) '     ', (X(I*M+J),I=0,N-1)
  100    CONTINUE
*
         CALL C06FPF(M,N,X,'Subsequent',TRIG,WORK,IFAIL)
         CALL C06GQF(M,N,X,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data as restored by inverse transform'
         WRITE (NOUT,*)
         DO 120 J = 1, M
            WRITE (NOUT,99999) '     ', (X(I*M+J),I=0,N-1)
  120    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of M or N'
      END IF
  140 STOP
*
99999 FORMAT (1X,A,6F10.4)
      END
```

9.2 Program Data

```
C06FQF Example Program Data
     3     6
     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```

9.3 Program Results

```
C06FQF Example Program Results

Original data values

        0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
        0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
        0.9172    0.0644    0.6037    0.6430    0.0428    0.4815

Original data written in full complex form

Real    0.3854    0.6772    0.1138    0.6751    0.1138    0.6772
Imag    0.0000    0.1424    0.6362    0.0000   -0.6362   -0.1424

Real    0.5417    0.2983    0.1181    0.7255    0.1181    0.2983
Imag    0.0000    0.8723    0.8638    0.0000   -0.8638   -0.8723

Real    0.9172    0.0644    0.6037    0.6430    0.6037    0.0644
Imag    0.0000    0.4815    0.0428    0.0000   -0.0428   -0.4815

Discrete Fourier transforms (real values)

        1.0788    0.6623   -0.2391   -0.5783    0.4592   -0.4388
        0.8573    1.2261    0.3533   -0.2222    0.3413   -1.2291
        1.1825    0.2625    0.6744    0.5523    0.0540   -0.4790

Original data as restored by inverse transform

        0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
        0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
        0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```

C06FRF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06FRF computes the discrete Fourier transforms of m sequences, each containing n complex data values. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
SUBROUTINE C06FRF(M, N, X, Y, INIT, TRIG, WORK, IFAIL)
INTEGER           M, N, IFAIL
real              X(M*N), Y(M*N), TRIG(2*N), WORK(2*M*N)
CHARACTER*1       INIT
```

3 Description

Given m sequences of n complex data values z_j^p, for $j = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$, this routine simultaneously calculates the Fourier transforms of all the sequences defined by:

$$\hat{z}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j^p \times \exp\left(-i\frac{2\pi jk}{n}\right), \quad k = 0, 1, \ldots, n-1; \quad p = 1, 2, \ldots, m.$$

(Note the scale factor $\frac{1}{\sqrt{n}}$ in this definition.)

The discrete Fourier transform is sometimes defined using a positive sign in the exponential term

$$\hat{z}_k^p = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j^p \times \exp\left(+i\frac{2\pi jk}{n}\right).$$

To compute this form, this routine should be preceded and followed by a call of C06GCF to form the complex conjugates of the z_j^p and the $\hat{z}_k^p$.

The routine uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, which is described in Temperton [2]. Special code is provided for the factors 2, 3, 4, 5 and 6. This routine is designed to be particularly efficient on vector processors, and it becomes especially fast as m, the number of transforms to be computed in parallel, increases.

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Temperton C (1983) Self-sorting mixed-radix fast Fourier transforms *J. Comput. Phys.* **52** 1–23

5 Parameters

1: M — INTEGER *Input*

On entry: the number of sequences to be transformed, m.

Constraint: M $\geq$ 1.

2: N — INTEGER *Input*

On entry: the number of complex values in each sequence, n.

Constraint: N $\geq$ 1.

3: X(M*N) — ***real*** array *Input/Output*

4: Y(M*N) — ***real*** array *Input/Output*

On entry: the real and imaginary parts of the complex data must be stored in X and Y respectively as if in a two-dimensional array of dimension (1:M,0:N−1); each of the m sequences is stored in a **row** of each array. In other words, if the real parts of the pth sequence to be transformed are denoted by x_j^p, for $j = 0, 1, \ldots, n-1$, then the mn elements of the array X must contain the values

$$x_0^1, x_0^2, \ldots, x_0^m,\ x_1^1, x_1^2, \ldots, x_1^m, \ldots,\ x_{n-1}^1, x_{n-1}^2, \ldots, x_{n-1}^m.$$

On exit: X and Y are overwritten by the real and imaginary parts of the complex transforms.

5: INIT — CHARACTER*1 *Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the array TRIG, then INIT must be set equal to 'I' (Initial call).

If INIT contains 'S' (Subsequent call), then the routine assumes that trigonometric coefficients for the specified value of n are supplied in the array TRIG, having been calculated in a previous call to one of C06FPF, C06FQF or C06FRF.

If INIT contains 'R' (Restart) then the routine assumes that trigonometric coefficients for the particular value of n are supplied in the array TRIG, but does not check that C06FPF, C06FQF or C06FRF have previously been called. This option allows the TRIG array to be stored in an external file, read in and re-used without the need for a call with INIT equal to 'I'. The routine carries out a simple test to check that the current value of n is compatible with the array TRIG.

Constraint: INIT = 'I', 'S' or 'R'.

6: TRIG(2*N) — ***real*** array *Input/Output*

On entry: if INIT = 'S', or 'R', TRIG must contain the required coefficients calculated in a previous call of the routine. Otherwise TRIG need not be set.

On exit: TRIG contains the required coefficients (computed by the routine if INIT = 'I').

7: WORK(2*M*N) — ***real*** array *Workspace*

8: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

IFAIL = 1

On entry, M < 1.

IFAIL = 2

On entry, N < 1.

IFAIL = 3

On entry, INIT is not one of 'I', 'S' or 'R'.

IFAIL = 4

Not used at this Mark.

IFAIL = 5

On entry, INIT = 'S' or 'R', but the array TRIG and the current value of n are inconsistent.

IFAIL = 6

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $nm \times \log n$, but also depends on the factors of n. The routine is fastest if the only prime factors of n are 2, 3 and 5, and is particularly slow if n is a large prime, or has large prime factors.

9 Example

This program reads in sequences of complex data values and prints their discrete Fourier transforms (as computed by C06FRF). Inverse transforms are then calculated using C06GCF and C06FRF and printed out, showing that the original sequences are restored.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FRF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=5,NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, J, M, N
*     .. Local Arrays ..
      real             TRIG(2*NMAX), WORK(2*MMAX*NMAX), X(MMAX*NMAX),
     +                 Y(MMAX*NMAX)
*     .. External Subroutines ..
      EXTERNAL         C06FRF, C06GCF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FRF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=120) M, N
      IF (M.LE.MMAX .AND. N.LE.NMAX) THEN
         DO 40 J = 1, M
            READ (NIN,*) (X(I*M+J),I=0,N-1)
            READ (NIN,*) (Y(I*M+J),I=0,N-1)
   40    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         DO 60 J = 1, M
```

```
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Real ', (X(I*M+J),I=0,N-1)
            WRITE (NOUT,99999) 'Imag ', (Y(I*M+J),I=0,N-1)
   60     CONTINUE
          IFAIL = 0
*
          CALL C06FRF(M,N,X,Y,'Initial',TRIG,WORK,IFAIL)
*
          WRITE (NOUT,*)
          WRITE (NOUT,*) 'Discrete Fourier transforms'
          DO 80 J = 1, M
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Real ', (X(I*M+J),I=0,N-1)
            WRITE (NOUT,99999) 'Imag ', (Y(I*M+J),I=0,N-1)
   80     CONTINUE
*
          CALL C06GCF(Y,M*N,IFAIL)
          CALL C06FRF(M,N,X,Y,'Subsequent',TRIG,WORK,IFAIL)
          CALL C06GCF(Y,M*N,IFAIL)
*
          WRITE (NOUT,*)
          WRITE (NOUT,*) 'Original data as restored by inverse transform'
          DO 100 J = 1, M
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Real ', (X(I*M+J),I=0,N-1)
            WRITE (NOUT,99999) 'Imag ', (Y(I*M+J),I=0,N-1)
  100     CONTINUE
          GO TO 20
        ELSE
          WRITE (NOUT,*) 'Invalid value of M or N'
        END IF
  120 STOP
*
99999 FORMAT (1X,A,6F10.4)
      END
```

9.2 Program Data

```
C06FRF Example Program Data
     3     6
     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
     0.9089    0.3118    0.3465    0.6198    0.2668    0.1614
     0.1156    0.0685    0.2060    0.8630    0.6967    0.2792
     0.6214    0.8681    0.7060    0.8652    0.9190    0.3355
```

9.3 Program Results

```
C06FRF Example Program Results

Original data values

Real      0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
Imag      0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
```

```
Real    0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
Imag    0.9089    0.3118    0.3465    0.6198    0.2668    0.1614

Real    0.1156    0.0685    0.2060    0.8630    0.6967    0.2792
Imag    0.6214    0.8681    0.7060    0.8652    0.9190    0.3355

Discrete Fourier transforms

Real    1.0737   -0.5706    0.1733   -0.1467    0.0518    0.3625
Imag    1.3961   -0.0409   -0.2958   -0.1521    0.4517   -0.0321

Real    1.1237    0.1728    0.4185    0.1530    0.3686    0.0101
Imag    1.0677    0.0386    0.7481    0.1752    0.0565    0.1403

Real    0.9100   -0.3054    0.4079   -0.0785   -0.1193   -0.5314
Imag    1.7617    0.0624   -0.0695    0.0725    0.1285   -0.4335

Original data as restored by inverse transform

Real    0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
Imag    0.5417    0.2983    0.1181    0.7255    0.8638    0.8723

Real    0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
Imag    0.9089    0.3118    0.3465    0.6198    0.2668    0.1614

Real    0.1156    0.0685    0.2060    0.8630    0.6967    0.2792
Imag    0.6214    0.8681    0.7060    0.8652    0.9190    0.3355
```

C06FUF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06FUF computes the two-dimensional discrete Fourier transform of a bivariate sequence of complex data values. This routine is designed to be particularly efficient on vector processors.

2. Specification

```
SUBROUTINE C06FUF (M, N, X, Y, INIT, TRIGM, TRIGN, WORK, IFAIL)
INTEGER          M, N, IFAIL
real             X(M*N), Y(M*N), TRIGM(2*M), TRIGN(2*N),
1                WORK(2*M*N)
CHARACTER*1      INIT
```

3. Description

This routine computes the two-dimensional discrete Fourier transform of a bivariate sequence of complex data values $z_{j_1 j_2}$, where $j_1 = 0,1,...,m-1$, $j_2 = 0,1,...,n-1$.

The discrete Fourier transform is here defined by:

$$\hat{z}_{k_1 k_2} = \frac{1}{\sqrt{mn}} \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} z_{j_1 j_2} \times \exp\left(-2\pi i\left(\frac{j_1 k_1}{m} + \frac{j_2 k_2}{n}\right)\right),$$

where $k_1 = 0,1,...,m-1$, $k_2 = 0,1,...,n-1$.

(Note the scale factor of $\frac{1}{\sqrt{mn}}$ in this definition.)

To compute the inverse discrete Fourier transform, defined with $\exp(+2\pi i(...))$ in the above formula instead of $\exp(-2\pi i(...))$, this routine should be preceded and followed by calls of C06GCF to form the complex conjugates of the data values and the transform.

This routine calls C06FRF to perform multiple one-dimensional discrete Fourier transforms by the fast Fourier transform (FFT) algorithm in Brigham [1]. It is designed to be particularly efficient on vector processors.

4. References

[1] BRIGHAM, E.O.
The Fast Fourier Transform.
Prentice-Hall, 1973.

[2] TEMPERTON, C.
Self-sorting Mixed-radix Fast Fourier Transforms.
J. Comput. Phys., 52, pp. 1-23, 1983.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number of rows, m, of the arrays X and Y.

Constraint: M ≥ 1.

2: N – INTEGER. *Input*

On entry: the number of columns, n, of the arrays X and Y.

Constraint: N ≥ 1.

3: X(M*N) – ***real*** array. *Input/Output*
4: Y(M*N) – ***real*** array. *Input/Output*

On entry: the real and imaginary parts of the complex data values must be stored in arrays X and Y respectively. If X and Y are regarded as two-dimensional arrays of dimension (0:M−1,0:N−1), then $X(j_1,j_2)$ and $Y(j_1,j_2)$ must contain the real and imaginary parts of z_{j_1,j_2}.

On exit: the real and imaginary parts respectively of the corresponding elements of the computed transform.

5: INIT – CHARACTER*1. *Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the arrays TRIGM and TRIGN, then INIT must be set equal to 'I' or 'i', (Initial call).

If INIT contains 'S' or 's', (Subsequent call), then the routine assumes that trigonometric coefficients for the specified values of *m* and *n* are supplied in the arrays TRIGM and TRIGN, having been calculated in a previous call to the routine.

If INIT contains 'R' or 'r', (Restart), then the routine assumes that trigonometric coefficients for the particular values of *m* and *n* are supplied in the arrays TRIGM and TRIGN, but does not check that the routine has previously been called. This option allows the TRIGM and TRIGN arrays to be stored in an external file, read in and re-used without the need for a call with INIT equal to 'I' or 'i'. The routine carries out a simple test to check that the current values of *m* and *n* are compatible with the arrays TRIGM and TRIGN.

Constraint: INIT = 'I', 'i', 'S', 's', 'R' or 'r'.

6: TRIGM(2*M) – ***real*** array. *Input/Output*
7: TRIGN(2*N) – ***real*** array. *Input/Output*

On entry: if INIT = 'S', 's', 'R' or 'r', TRIGM and TRIGN must contain the required coefficients calculated in a previous call of the routine. Otherwise TRIGM and TRIGN need not be set.

If $m = n$ the same array may be supplied for TRIGM and TRIGN.

On exit: TRIGM and TRIGN contain the required coefficients (computed by the routine if INIT = 'I' or 'i').

8: WORK(2*M*N) – ***real*** array. *Workspace*

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M < 1.

IFAIL = 2

On entry, N < 1.

IFAIL = 3

On entry, INIT is not one of 'I', 'i', 'S', 's', 'R' or 'r'.

IFAIL = 4

On entry, INIT = 'S' or 's', but C06FUF has not previously been called.

IFAIL = 5

On entry, INIT = 'S', 's', 'R' or 'r', but at least one of the arrays TRIGM and TRIGN is inconsistent with the current value of M or N.

7. Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8. Further Comments

The time taken by the routine is approximately proportional to $mn \times \log(mn)$, but also depends on the factorization of the individual dimensions m and n. The routine is somewhat faster than average if their only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

9. Example

This program reads in a bivariate sequence of complex data values and prints the two-dimensional Fourier transform. It then performs an inverse transform and prints the sequence so obtained, which may be compared to the original data values.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FUF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX, MNMAX
      PARAMETER        (MMAX=96,NMAX=96,MNMAX=MMAX*NMAX)
*     .. Local Scalars ..
      INTEGER          IFAIL, M, N
*     .. Local Arrays ..
      real             TRIGM(2*MMAX), TRIGN(2*NMAX), WORK(2*MNMAX),
     +                 X(MNMAX), Y(MNMAX)
*     .. External Subroutines ..
      EXTERNAL         C06FUF, C06GCF, READXY, WRITXY
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FUF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=40) M, N
      IF (M*N.GE.1 .AND. M*N.LE.MNMAX) THEN
         CALL READXY(NIN,X,Y,M,N)
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         CALL WRITXY(NOUT,X,Y,M,N)
         IFAIL = 0
*
*        -- Compute transform
         CALL C06FUF(M,N,X,Y,'Initial',TRIGM,TRIGN,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of discrete Fourier transform'
         CALL WRITXY(NOUT,X,Y,M,N)
*
*        -- Compute inverse transform
         CALL C06GCF(Y,M*N,IFAIL)
         CALL C06FUF(M,N,X,Y,'Subsequent',TRIGM,TRIGN,WORK,IFAIL)
         CALL C06GCF(Y,M*N,IFAIL)
```

```
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         CALL WRITXY(NOUT,X,Y,M,N)
         GO TO 20
       ELSE
         WRITE (NOUT,*) ' ** Invalid value of M or N'
       END IF
    40 STOP
       END
*
       SUBROUTINE READXY(NIN,X,Y,N1,N2)
*      Read 2-dimensional complex data
*      .. Scalar Arguments ..
       INTEGER           N1, N2, NIN
*      .. Array Arguments ..
       real              X(N1,N2), Y(N1,N2)
*      .. Local Scalars ..
       INTEGER           I, J
*      .. Executable Statements ..
       DO 20 I = 1, N1
         READ (NIN,*) (X(I,J),J=1,N2)
         READ (NIN,*) (Y(I,J),J=1,N2)
    20 CONTINUE
       RETURN
       END
*
       SUBROUTINE WRITXY(NOUT,X,Y,N1,N2)
*      Print 2-dimensional complex data
*      .. Scalar Arguments ..
       INTEGER           N1, N2, NOUT
*      .. Array Arguments ..
       real              X(N1,N2), Y(N1,N2)
*      .. Local Scalars ..
       INTEGER           I, J
*      .. Executable Statements ..
       DO 20 I = 1, N1
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Real ', (X(I,J),J=1,N2)
         WRITE (NOUT,99999) 'Imag ', (Y(I,J),J=1,N2)
    20 CONTINUE
       RETURN
*
99999 FORMAT (1X,A,7F10.3,/(6X,7F10.3))
       END
```

9.2. Program Data

```
C06FUF Example Program Data
3  5  : Number of rows, M, and columns, N, in X and Y
     1.000     0.999     0.987     0.936     0.802  :  X(0,J), J=0,...,N-1
     0.000    -0.040    -0.159    -0.352    -0.597  :  Y(0,J), J=0,...,N-1
     0.994     0.989     0.963     0.891     0.731  :  X(1,J), J=0,...,N-1
    -0.111    -0.151    -0.268    -0.454    -0.682  :  Y(1,J), J=0,...,N-1
     0.903     0.885     0.823     0.694     0.467  :  X(2,J), J=0,...,N-1
    -0.430    -0.466    -0.568    -0.720    -0.884  :  Y(2,J), J=0,...,N-1
```

9.3. Program Results

```
C06FUF Example Program Results

Original data values

Real      1.000     0.999     0.987     0.936     0.802
Imag      0.000    -0.040    -0.159    -0.352    -0.597

Real      0.994     0.989     0.963     0.891     0.731
Imag     -0.111    -0.151    -0.268    -0.454    -0.682

Real      0.903     0.885     0.823     0.694     0.467
Imag     -0.430    -0.466    -0.568    -0.720    -0.884

Components of discrete Fourier transform

Real      3.373     0.481     0.251     0.054    -0.419
Imag     -1.519    -0.091     0.178     0.319     0.415

Real      0.457     0.055     0.009    -0.022    -0.076
Imag      0.137     0.032     0.039     0.036     0.004

Real     -0.170    -0.037    -0.042    -0.038    -0.002
Imag      0.493     0.058     0.008    -0.025    -0.083

Original sequence as restored by inverse transform

Real      1.000     0.999     0.987     0.936     0.802
Imag      0.000    -0.040    -0.159    -0.352    -0.597

Real      0.994     0.989     0.963     0.891     0.731
Imag     -0.111    -0.151    -0.268    -0.454    -0.682

Real      0.903     0.885     0.823     0.694     0.467
Imag     -0.430    -0.466    -0.568    -0.720    -0.884
```

C06FXF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06FXF computes the three-dimensional discrete Fourier transform of a trivariate sequence of complex data values. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
      SUBROUTINE C06FXF(N1, N2, N3, X, Y, INIT, TRIGN1, TRIGN2, TRIGN3,
     1                  WORK, IFAIL)
      INTEGER           N1, N2, N3, IFAIL
      real              X(N1*N2*N3), Y(N1*N2*N3), TRIGN1(2*N1),
     1                  TRIGN2(2*N2), TRIGN3(2*N3), WORK(2*N1*N2*N3)
      CHARACTER*1       INIT
```

3 Description

This routine computes the three-dimensional discrete Fourier transform of a trivariate sequence of complex data values $z_{j_1 j_2 j_3}$, where $j_1 = 0, 1, \ldots, n_1 - 1$, $j_2 = 0, 1, \ldots, n_2 - 1$, $j_3 = 0, 1, \ldots, n_3 - 1$.

The discrete Fourier transform is here defined by:

$$\hat{z}_{k_1 k_2 k_3} = \frac{1}{\sqrt{n_1 n_2 n_3}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} z_{j_1 j_2 j_3} \times \exp\left(-2\pi i \left(\frac{j_1 k_1}{n_1} + \frac{j_2 k_2}{n_2} + \frac{j_3 k_3}{n_3}\right)\right),$$

where $k_1 = 0, 1, \ldots, n_1 - 1$, $k_2 = 0, 1, \ldots, n_2 - 1$, $k_3 = 0, 1, \ldots, n_3 - 1$.

(Note the scale factor of $\frac{1}{\sqrt{n_1 n_2 n_3}}$ in this definition.)

To compute the inverse discrete Fourier transform, defined with $\exp(+2\pi i(\ldots))$ in the above formula instead of $\exp(-2\pi i(\ldots))$, this routine should be preceded and followed by calls of C06GCF to form the complex conjugates of the data values and the transform.

This routine calls C06FRF to perform multiple one-dimensional discrete Fourier transforms by the fast Fourier transform (FFT) algorithm (Brigham [1]). It is designed to be particularly efficient on vector processors.

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Temperton C (1983) Self-sorting mixed-radix fast Fourier transforms *J. Comput. Phys.* **52** 1–23

5 Parameters

1: N1 — INTEGER *Input*

On entry: the first dimension of the transform, n_1.

Constraint: N1 ≥ 1.

2: N2 — INTEGER *Input*

On entry: the second dimension of the transform, n_2.

Constraint: N2 ≥ 1.

3: N3 — INTEGER *Input*

On entry: the third dimension of the transform, n_3.

Constraint: N3 ≥ 1.

4: X(N1*N2*N3) — ***real*** array *Input/Output*

5: Y(N1*N2*N3) — ***real*** array *Input/Output*

On entry: the real and imaginary parts of the complex data values must be stored in arrays X and Y respectively. If X and Y are regarded as three-dimensional arrays of dimension (0:N1−1, 0:N2−1, 0:N3−1), then X(j_1, j_2, j_3) and Y(j_1, j_2, j_3) must contain the real and imaginary parts of $z_{j_1 j_2 j_3}$.

On exit: the real and imaginary parts respectively of the corresponding elements of the computed transform.

6: INIT — CHARACTER*1 *Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the arrays TRIGN1, TRIGN2 and TRIGN3, then INIT must be set equal to 'I', (Initial call).

If INIT = 'S', (Subsequent call), then the routine assumes that trigonometric coefficients for the specified values of n_1, n_2 and n_3 are supplied in the arrays TRIGN1, TRIGN2 and TRIGN3, having been calculated in a previous call to the routine.

If INIT = 'R', (Restart), then the routine assumes that trigonometric coefficients for the specified values of n_1, n_2 and n_3 are supplied in the arrays TRIGN1, TRIGN2 and TRIGN3, but does not check that the routine has previously been called. This option allows the TRIGN1, TRIGN2 and TRIGN3 arrays to be stored in an external file, read in and re-used without the need for a call with INIT equal to 'I'. The routine carries out a simple test to check that the current values of n_1, n_2 and n_3 are compatible with the arrays TRIGN1, TRIGN2 and TRIGN3.

Constraint: INIT = 'I', 'S' or 'R'.

7: TRIGN1(2*N1) — ***real*** array *Input/Output*

8: TRIGN2(2*N2) — ***real*** array *Input/Output*

9: TRIGN3(2*N3) — ***real*** array *Input/Output*

On entry: if INIT = 'S' or 'R', TRIGN1, TRIGN2 and TRIGN3 must contain the required coefficients calculated in a previous call of the routine. Otherwise TRIGN1, TRIGN2 and TRIGN3 need not be set. If $n_i = n_j$ the same array may be supplied for TRIGNi and TRIGNj, for $i, j = 1, 2, 3$.

On exit: TRIGN1, TRIGN2 and TRIGN3 contain the required coefficients (computed by the routine if INIT = 'I').

10: WORK(2*N1*N2*N3) — ***real*** array *Workspace*

11: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

IFAIL = 1

On entry, N1 < 1.

IFAIL = 2

On entry, N2 < 1.

IFAIL = 3

On entry, N3 < 1.

IFAIL = 4

On entry, INIT is not one of 'I', 'S' or 'R'.

IFAIL = 5

Not used at this Mark.

IFAIL = 6

On entry, INIT = 'S' or 'R', but at least one of the arrays TRIGN1, TRIGN2 and TRIGN3 is inconsistent with the current value of N1, N2 or N3.

IFAIL = 7

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $n_1 n_2 n_3 \times \log(n_1 n_2 n_3)$, but also depends on the factorization of the individual dimensions n_1, n_2 and n_3. The routine is somewhat faster than average if their only prime factors are 2, 3 or 5; and fastest of all if they are powers of 2.

9 Example

This program reads in a trivariate sequence of complex data values and prints the three-dimensional Fourier transform. It then performs an inverse transform and prints the sequence so obtained, which may be compared to the original data values.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06FXF Example Program Text
*     Mark 17 Release. NAG Copyright 1995.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          N1MAX, N2MAX, N3MAX, NMAX
      PARAMETER        (N1MAX=16,N2MAX=16,N3MAX=16,
     +                 NMAX=N1MAX*N2MAX*N3MAX)
*     .. Local Scalars ..
      INTEGER          IFAIL, N, N1, N2, N3
*     .. Local Arrays ..
      real             TRIGN1(2*N1MAX), TRIGN2(2*N2MAX),
     +                 TRIGN3(2*N3MAX), WORK(2*NMAX), X(NMAX), Y(NMAX)
```

```
*     .. External Subroutines ..
      EXTERNAL         C06FXF, C06GCF, READXY, WRITXY
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06FXF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=40) N1, N2, N3
      N = N1*N2*N3
      IF (N.GE.1 .AND. N.LE.NMAX) THEN
         CALL READXY(NIN,X,Y,N1,N2,N3)
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         CALL WRITXY(NOUT,X,Y,N1,N2,N3)
         IFAIL = 0
*
*        -- Compute transform
         CALL C06FXF(N1,N2,N3,X,Y,'Initial',TRIGN1,TRIGN2,TRIGN3,WORK,
     +              IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Components of discrete Fourier transform'
         CALL WRITXY(NOUT,X,Y,N1,N2,N3)
*
*        -- Compute inverse transform
         CALL C06GCF(Y,N,IFAIL)
         CALL C06FXF(N1,N2,N3,X,Y,'Subsequent',TRIGN1,TRIGN2,TRIGN3,
     +              WORK,IFAIL)
         CALL C06GCF(Y,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Original sequence as restored by inverse transform'
         CALL WRITXY(NOUT,X,Y,N1,N2,N3)
         GO TO 20
      ELSE
         WRITE (NOUT,*) ' ** Invalid value of n1, n2 or n3'
      END IF
   40 STOP
      END
*
      SUBROUTINE READXY(NIN,X,Y,N1,N2,N3)
*     Read 3-dimensional complex data
*     .. Scalar Arguments ..
      INTEGER          N1, N2, N3, NIN
*     .. Array Arguments ..
      real             X(N1,N2,N3), Y(N1,N2,N3)
*     .. Local Scalars ..
      INTEGER          I, J, K
*     .. Executable Statements ..
      DO 40 I = 1, N1
         DO 20 J = 1, N2
            READ (NIN,*) (X(I,J,K),K=1,N3)
            READ (NIN,*) (Y(I,J,K),K=1,N3)
   20    CONTINUE
   40 CONTINUE
      RETURN
      END
*
```

```
      SUBROUTINE WRITXY(NOUT,X,Y,N1,N2,N3)
*     Print 3-dimensional complex data
*     .. Scalar Arguments ..
      INTEGER           N1, N2, N3, NOUT
*     .. Array Arguments ..
      real              X(N1,N2,N3), Y(N1,N2,N3)
*     .. Local Scalars ..
      INTEGER           I, J, K
*     .. Executable Statements ..
      DO 40 I = 1, N1
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'z(i,j,k) for i =', I
         DO 20 J = 1, N2
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Real ', (X(I,J,K),K=1,N3)
            WRITE (NOUT,99999) 'Imag ', (Y(I,J,K),K=1,N3)
   20    CONTINUE
   40 CONTINUE
      RETURN
*
99999 FORMAT (1X,A,7F10.3,/(6X,7F10.3))
99998 FORMAT (1X,A,I6)
      END
```

9.2 Program Data

```
C06FXF Example Program Data
2 3 4  : values of N1, N2, N3
     1.000     0.999     0.987     0.936   :  X(0,0,J), J=0,...,N3-1
     0.000    -0.040    -0.159    -0.352   :  Y(0,0,J), J=0,...,N3-1
     0.994     0.989     0.963     0.891   :  X(0,1,J), J=0,...,N3-1
    -0.111    -0.151    -0.268    -0.454   :  Y(0,1,J), J=0,...,N3-1
     0.903     0.885     0.823     0.694   :  X(0,2,J), J=0,...,N3-1
    -0.430    -0.466    -0.568    -0.720   :  Y(0,2,J), J=0,...,N3-1
     0.500     0.499     0.487     0.436   :  X(1,0,J), J=0,...,N3-1
     0.500     0.040     0.159     0.352   :  Y(1,0,J), J=0,...,N3-1
     0.494     0.489     0.463     0.391   :  X(1,1,J), J=0,...,N3-1
     0.111     0.151     0.268     0.454   :  Y(1,1,J), J=0,...,N3-1
     0.403     0.385     0.323     0.194   :  X(1,2,J), J=0,...,N3-1
     0.430     0.466     0.568     0.720   :  Y(1,2,J), J=0,...,N3-1
```

9.3 Program Results

```
C06FXF Example Program Results

Original data values

z(i,j,k) for i =     1

Real      1.000     0.999     0.987     0.936
Imag      0.000    -0.040    -0.159    -0.352

Real      0.994     0.989     0.963     0.891
Imag     -0.111    -0.151    -0.268    -0.454

Real      0.903     0.885     0.823     0.694
Imag     -0.430    -0.466    -0.568    -0.720
```

```
z(i,j,k) for i =     2

Real      0.500     0.499     0.487     0.436
Imag      0.500     0.040     0.159     0.352

Real      0.494     0.489     0.463     0.391
Imag      0.111     0.151     0.268     0.454

Real      0.403     0.385     0.323     0.194
Imag      0.430     0.466     0.568     0.720

Components of discrete Fourier transform

z(i,j,k) for i =     1

Real      3.292     0.051     0.113     0.051
Imag      0.102    -0.042     0.102     0.246

Real      0.143     0.016    -0.024    -0.050
Imag     -0.086     0.153     0.127     0.086

Real      0.143    -0.050    -0.024     0.016
Imag      0.290     0.118     0.077     0.051

z(i,j,k) for i =     2

Real      1.225     0.355     0.000    -0.355
Imag     -1.620     0.083     0.162     0.083

Real      0.424     0.020     0.013    -0.007
Imag      0.320    -0.115    -0.091    -0.080

Real     -0.424     0.007    -0.013    -0.020
Imag      0.320    -0.080    -0.091    -0.115

Original sequence as restored by inverse transform

z(i,j,k) for i =     1

Real      1.000     0.999     0.987     0.936
Imag      0.000    -0.040    -0.159    -0.352

Real      0.994     0.989     0.963     0.891
Imag     -0.111    -0.151    -0.268    -0.454

Real      0.903     0.885     0.823     0.694
Imag     -0.430    -0.466    -0.568    -0.720

z(i,j,k) for i =     2

Real      0.500     0.499     0.487     0.436
Imag      0.500     0.040     0.159     0.352

Real      0.494     0.489     0.463     0.391
Imag      0.111     0.151     0.268     0.454
```

Real	0.403	0.385	0.323	0.194
Imag	0.430	0.466	0.568	0.720

C06GBF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06GBF forms the complex conjugate of a Hermitian sequence of n data values.

2. Specification

```
SUBROUTINE C06GBF (X, N, IFAIL)
INTEGER          N, IFAIL
real             X(N)
```

3. Description

This is a utility routine for use in conjunction with C06EAF, C06EBF, C06FAF or C06FBF to calculate inverse discrete Fourier transforms (see the Chapter Introduction).

4. References

None.

5. Parameters

1: X(N) – ***real*** array. *Input/Output*

On entry: if the data values z_j are written as $x_j + iy_j$ and if X is declared with bounds (0:N–1) in the (sub)program from which C06GBF is called, then for $0 \leq j \leq n/2$, X(j) must contain x_j ($= x_{n-j}$), while for $n/2 < j \leq n-1$, X(j) must contain $-y_j$ ($= y_{n-j}$). In other words, X must contain the Hermitian sequence in Hermitian form. (See also Section 2.1.2 of the Chapter Introduction).

On exit: the imaginary parts y_j are negated. The real parts x_j are not referenced.

2: N – INTEGER. *Input*

On entry: the number of data values, n.

Constraint: N $\geq$ 1.

3: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

N < 1.

7. Accuracy

Exact.

8. Further Comments

The time taken by the routine is negligible.

9. Example

This program reads in a sequence of real data values, calls C06EAF followed by C06GBF to compute their inverse discrete Fourier transform, and prints this after expanding it from Hermitian form into a full complex sequence.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06GBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, N, N2, NJ
*     .. Local Arrays ..
      real             A(0:NMAX-1), B(0:NMAX-1), X(0:NMAX-1)
*     .. External Subroutines ..
      EXTERNAL         C06EAF, C06GBF
*     .. Intrinsic Functions ..
      INTRINSIC        MOD
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06GBF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=100) N
      IF (N.GT.1 .AND. N.LT.NMAX) THEN
         DO 40 J = 0, N - 1
            READ (NIN,*) X(J)
   40    CONTINUE
         IFAIL = 0
*
         CALL C06EAF(X,N,IFAIL)
         CALL C06GBF(X,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Components of inverse discrete Fourier transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '            Real      Imag'
         WRITE (NOUT,*)
         A(0) = X(0)
         B(0) = 0.0e0
         N2 = (N-1)/2
         DO 60 J = 1, N2
            NJ = N - J
            A(J) = X(J)
            A(NJ) = X(J)
            B(J) = X(NJ)
            B(NJ) = -X(NJ)
   60    CONTINUE
         IF (MOD(N,2).EQ.0) THEN
            A(N2+1) = X(N2+1)
            B(N2+1) = 0.0e0
         END IF
         DO 80 J = 0, N - 1
            WRITE (NOUT,99999) J, A(J), B(J)
   80    CONTINUE
```

```
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
  100 STOP
*
99999 FORMAT (1X,I6,2F10.5)
      END
```

9.2. Program Data

```
C06GBF Example Program Data
    7
  0.34907
  0.54890
  0.74776
  0.94459
  1.13850
  1.32850
  1.51370
```

9.3. Program Results

```
 C06GBF Example Program Results

 Components of inverse discrete Fourier transform

            Real       Imag

      0   2.48361   0.00000
      1  -0.26599  -0.53090
      2  -0.25768  -0.20298
      3  -0.25636  -0.05806
      4  -0.25636   0.05806
      5  -0.25768   0.20298
      6  -0.26599   0.53090
```

C06GCF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06GCF forms the complex conjugate of a sequence of n data values.

2. Specification

```
SUBROUTINE C06GCF (Y, N, IFAIL)
INTEGER          N, IFAIL
real             Y(N)
```

3. Description

This is a utility routine for use in conjunction with C06ECF or C06FCF to calculate inverse discrete Fourier transforms (see the Chapter Introduction).

4. References

None.

5. Parameters

1: Y(N) – ***real*** array. *Input/Output*

On entry: if Y is declared with bounds (0:N–1) in the (sub)program which C06GCF is called, then Y(j) must contain the imaginary part of the jth data value, for $0 \leq j \leq n-1$.

On exit: these values are negated.

2: N – INTEGER. *Input*

On entry: the number of data values, n.

Constraint: $N \geq 1$.

3: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

$N < 1$.

7. Accuracy

Exact.

8. Further Comments

The time taken by the routine is negligible.

9. Example

This program reads in a sequence of complex data values and prints their inverse discrete Fourier transform as computed by calling C06GCF, followed by C06ECF and C06GCF again.

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06GCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          IFAIL, J, N
*     .. Local Arrays ..
      real             X(0:NMAX-1), Y(0:NMAX-1)
*     .. External Subroutines ..
      EXTERNAL         C06ECF, C06GCF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06GCF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=80) N
      IF (N.GT.1 .AND. N.LE.NMAX) THEN
         DO 40 J = 0, N - 1
            READ (NIN,*) X(J), Y(J)
   40    CONTINUE
         IFAIL = 0
*
         CALL C06GCF(Y,N,IFAIL)
         CALL C06ECF(X,Y,N,IFAIL)
         CALL C06GCF(Y,N,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Components of inverse discrete Fourier transform'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '           Real      Imag'
         WRITE (NOUT,*)
         DO 60 J = 0, N - 1
            WRITE (NOUT,99999) J, X(J), Y(J)
   60    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of N'
      END IF
   80 STOP
*
99999 FORMAT (1X,I6,2F10.5)
      END
```

9.2. Program Data

```
C06GCF Example Program Data
    7
  0.34907  -0.37168
  0.54890  -0.35669
  0.74776  -0.31175
  0.94459  -0.23702
  1.13850  -0.13274
  1.32850   0.00074
  1.51370   0.16298
```

9.3. Program Results

```
C06GCF Example Program Results

Components of inverse discrete Fourier transform

          Real      Imag

0    2.48361  -0.47100
1    0.01983  -0.56496
2   -0.14825  -0.30840
3   -0.22506  -0.17477
4   -0.28767  -0.05865
5   -0.36711   0.09756
6   -0.55180   0.49684
```

C06GQF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06GQF forms the complex conjugates of m Hermitian sequences, each containing n data values.

2. Specification

```
SUBROUTINE C06GQF (M, N, X, IFAIL)
INTEGER          M, N, IFAIL
real             X(M*N)
```

3. Description

This is a utility routine for use in conjunction with C06FPF and C06FQF to calculate inverse discrete Fourier transforms (see the Chapter Introduction).

4. References

None.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number of Hermitian sequences to be conjugated, m.

Constraint: M ≥ 1.

2: N – INTEGER. *Input*

On entry: the number of data values in each Hermitian sequence, n.

Constraint: N ≥ 1.

3: X(M*N) – ***real*** array. *Input/Output*

On entry: the data must be stored in array X as if in a two-dimensional array of dimension (1:M,0:N−1); each of the m sequences is stored in a **row** of the array in Hermitian form. If the n data values z_j^p are written as $x_j^p + iy_j^p$, then for $0 \le j \le n/2$, x_j^p is contained in X(p,j), and for $1 \le j \le (n-1)/2$, y_j^p is contained in X(p,$n-j$). (See also Section 2.1.2 of the Chapter Introduction.)

On exit: the imaginary parts y_j^p are negated. The real parts x_j^p are not referenced.

4: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M < 1.

IFAIL = 2

On entry, N < 1.

7. Accuracy

Exact.

8. Further Comments

None.

9. Example

This program reads in sequences of real data values which are assumed to be Hermitian sequences of complex data stored in Hermitian form. The sequences are expanded into full complex form using C06GSF and printed. The sequences are then conjugated (using C06GQF) and the conjugated sequences are expanded into complex form using C06GSF and printed out.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06GQF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=5,NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, J, M, N
*     .. Local Arrays ..
      real             U(MMAX*NMAX), V(MMAX*NMAX), X(MMAX*NMAX)
*     .. External Subroutines ..
      EXTERNAL         C06GQF, C06GSF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06GQF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=140) M, N
      IF (M.LE.MMAX .AND. N.LE.NMAX) THEN
         DO 40 J = 1, M
            READ (NIN,*) (X(I*M+J),I=0,N-1)
   40    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         WRITE (NOUT,*)
         DO 60 J = 1, M
            WRITE (NOUT,99999) '     ', (X(I*M+J),I=0,N-1)
   60    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data written in full complex form'
         IFAIL = 0
*
         CALL C06GSF(M,N,X,U,V,IFAIL)
*
         DO 80 J = 1, M
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Real ', (U(I*M+J),I=0,N-1)
            WRITE (NOUT,99999) 'Imag ', (V(I*M+J),I=0,N-1)
   80    CONTINUE
*
         CALL C06GQF(M,N,X,IFAIL)
*
```

```
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Conjugated data values'
         WRITE (NOUT,*)
         DO 100 J = 1, M
            WRITE (NOUT,99999) '     ', (X(I*M+J),I=0,N-1)
  100    CONTINUE
*
         CALL C06GSF(M,N,X,U,V,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Conjugated data written in full complex form'
*
         CALL C06GSF(M,N,X,U,V,IFAIL)
*
         DO 120 J = 1, M
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Real ', (U(I*M+J),I=0,N-1)
            WRITE (NOUT,99999) 'Imag ', (V(I*M+J),I=0,N-1)
  120    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of M or N'
      END IF
  140 STOP
*
99999 FORMAT (1X,A,6F10.4)
      END
```

9.2. Program Data

```
C06GQF Example Program Data
     3     6
     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```

9.3. Program Results

```
 C06GQF Example Program Results

 Original data values

          0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
          0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
          0.9172    0.0644    0.6037    0.6430    0.0428    0.4815

 Original data written in full complex form

 Real     0.3854    0.6772    0.1138    0.6751    0.1138    0.6772
 Imag     0.0000    0.1424    0.6362    0.0000   -0.6362   -0.1424

 Real     0.5417    0.2983    0.1181    0.7255    0.1181    0.2983
 Imag     0.0000    0.8723    0.8638    0.0000   -0.8638   -0.8723

 Real     0.9172    0.0644    0.6037    0.6430    0.6037    0.0644
 Imag     0.0000    0.4815    0.0428    0.0000   -0.0428   -0.4815

 Conjugated data values

          0.3854    0.6772    0.1138    0.6751   -0.6362   -0.1424
          0.5417    0.2983    0.1181    0.7255   -0.8638   -0.8723
          0.9172    0.0644    0.6037    0.6430   -0.0428   -0.4815
```

```
Conjugated data written in full complex form

Real     0.3854    0.6772    0.1138    0.6751    0.1138    0.6772
Imag     0.0000   -0.1424   -0.6362    0.0000    0.6362    0.1424

Real     0.5417    0.2983    0.1181    0.7255    0.1181    0.2983
Imag     0.0000   -0.8723   -0.8638    0.0000    0.8638    0.8723

Real     0.9172    0.0644    0.6037    0.6430    0.6037    0.0644
Imag     0.0000   -0.4815   -0.0428    0.0000    0.0428    0.4815
```

C06GSF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06GSF takes m Hermitian sequences, each containing n data values, and forms the real and imaginary parts of the m corresponding complex sequences.

2. Specification

```
SUBROUTINE C06GSF (M, N, X, U, V, IFAIL)
INTEGER          M, N, IFAIL
real             X(M*N), U(M*N), V(M*N)
```

3. Description

This is a utility routine for use in conjunction with C06FPF and C06FQF (see the Chapter Introduction).

4. References

None.

5. Parameters

1: M – INTEGER. *Input*

On entry: the number of Hermitian sequences, m, to be converted into complex form.

Constraint: M ≥ 1.

2: N – INTEGER. *Input*

On entry: the number of data values, n, in each sequence.

Constraint: N ≥ 1.

3: X(M*N) – ***real*** array. *Input*

On entry: the data must be stored in X as if in a two-dimensional array of dimension (1:M,0:N−1); each of the m sequences is stored in a **row** of the array in Hermitian form. If the n data values z_j^p are written as $x_j^p + iy_j^p$, then for $0 \le j \le n/2$, x_j^p is contained in X(p,j), and for $1 \le j \le (n-1)/2$, y_j^p is contained in X(p,$n-j$). (See also Section 2.1.2 of the Chapter Introduction.)

4: U(M*N) – ***real*** array. *Output*

5: V(M*N) – ***real*** array. *Output*

On exit: the real and imaginary parts of the m sequences of length n, are stored in U and V respectively, as if in two-dimensional arrays of dimension (1:M,0:N−1); each of the m sequences is stored as if in a **row** of each array. In other words, if the real parts of the pth sequence are denoted by x_j^p, for $j = 0,1,\ldots,n-1$ then the mn elements of the array U contain the values

$$x_0^1, x_0^2, \ldots, x_0^m,\ x_1^1, x_1^2, \ldots, x_1^m,\ \ldots,\ x_{n-1}^1, x_{n-1}^2, \ldots, x_{n-1}^m.$$

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M < 1.

IFAIL = 2

On entry, N < 1.

7. Accuracy

Exact.

8. Further Comments

None.

9. Example

This program reads in sequences of real data values which are assumed to be Hermitian sequences of complex data stored in Hermitian form. The sequences are then expanded into full complex form using C06GSF and printed.

9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06GSF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=5,NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, J, M, N
*     .. Local Arrays ..
      real             U(MMAX*NMAX), V(MMAX*NMAX), X(MMAX*NMAX)
*     .. External Subroutines ..
      EXTERNAL         C06GSF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06GSF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=100) M, N
      IF (M.LE.MMAX .AND. N.LE.NMAX) THEN
         DO 40 J = 1, M
            READ (NIN,*) (X(I*M+J),I=0,N-1)
   40    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         WRITE (NOUT,*)
         DO 60 J = 1, M
            WRITE (NOUT,99999) '     ', (X(I*M+J),I=0,N-1)
   60    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data written in full complex form'
         IFAIL = 0
*
         CALL C06GSF(M,N,X,U,V,IFAIL)
*
```

```
         DO 80 J = 1, M
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Real ', (U(I*M+J),I=0,N-1)
            WRITE (NOUT,99999) 'Imag ', (V(I*M+J),I=0,N-1)
   80    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of M or N'
      END IF
  100 STOP
*
99999 FORMAT (1X,A,6F10.4)
      END
```

9.2. Program Data

```
C06GSF Example Program Data
     3     6
     0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
     0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
     0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```

9.3. Program Results

```
 C06GSF Example Program Results

 Original data values

          0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
          0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
          0.9172    0.0644    0.6037    0.6430    0.0428    0.4815

 Original data written in full complex form

 Real      0.3854    0.6772    0.1138    0.6751    0.1138    0.6772
 Imag      0.0000    0.1424    0.6362    0.0000   -0.6362   -0.1424

 Real      0.5417    0.2983    0.1181    0.7255    0.1181    0.2983
 Imag      0.0000    0.8723    0.8638    0.0000   -0.8638   -0.8723

 Real      0.9172    0.0644    0.6037    0.6430    0.6037    0.0644
 Imag      0.0000    0.4815    0.0428    0.0000   -0.0428   -0.4815
```

C06HAF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06HAF computes the discrete Fourier sine transforms of m sequences of real data values. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
SUBROUTINE C06HAF(M, N, X, INIT, TRIG, WORK, IFAIL)
INTEGER           M, N, IFAIL
real              X(M*N), TRIG(2*N), WORK(M*N)
CHARACTER*1       INIT
```

3 Description

Given m sequences of $n-1$ real data values x_j^p, for $j = 1, 2, \ldots, n-1$; $p = 1, 2, \ldots, m$, this routine simultaneously calculates the Fourier sine transforms of all the sequences defined by:

$$\hat{x}_k^p = \sqrt{\frac{2}{n}} \sum_{j=1}^{n-1} x_j^p \times \sin\left(jk\frac{\pi}{n}\right), \quad k = 1, 2, \ldots, n-1; \quad p = 1, 2, \ldots, m.$$

(Note the scale factor $\sqrt{\frac{2}{n}}$ in this definition.)

The Fourier sine transform defined above is its own inverse, and two consecutive calls of this routine with the same data will restore the original data.

The transform calculated by this routine can be used to solve Poisson's equation when the solution is specified at both left and right boundaries (Swarztrauber [2]). (See the Chapter Introduction.)

The routine uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, described in Temperton [4], together with pre- and post-processing stages described in Swarztrauber [3]. Special coding is provided for the factors 2, 3, 4, 5 and 6. This routine is designed to be particularly efficient on vector processors, and it becomes especially fast as m, the number of transforms to be computed in parallel, increases.

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Swarztrauber P N (1977) The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle *SIAM Rev.* **19 (3)** 490–501

[3] Swarztrauber P N (1982) Vectorizing the FFT's *Parallel Computation* (ed G Rodrique) Academic Press 51–83

[4] Temperton C (1983) Fast mixed-radix real Fourier transforms *J. Comput. Phys.* **52** 340–350

5 Parameters

1: M — INTEGER *Input*

On entry: the number of sequences to be transformed, m.

Constraint: M ≥ 1.

2: N — INTEGER *Input*

On entry: one more than the number of real values in each sequence, i.e., the number of values in each sequence is $n - 1$.

Constraint: N ≥ 1.

3: X(M*N) — ***real*** array *Input/Output*

On entry: the data must be stored in X as if in a two-dimensional array of dimension (1:M,1:N); each of the m sequences is stored in a **row** of the array. In other words, if the $n - 1$ data values of the pth sequence to be transformed are denoted by x_j^p, for $j = 1, 2, \ldots, n-1$; $p = 1, 2, \ldots, m$, then the first $m(n-1)$ elements of the array X must contain the values

$$x_1^1, x_1^2, \ldots, x_1^m,\ x_2^1, x_2^2, \ldots, x_2^m, \ldots,\ x_{n-1}^1, x_{n-1}^2, \ldots, x_{n-1}^m.$$

The nth element of each row x_n^p, for $p = 1, 2, \ldots, m$, is required as workspace. These m elements may contain arbitrary values on entry, and are set to zero by the routine.

On exit: the m Fourier transforms stored as if in a two-dimensional array of dimension (1:M,1:N). Each of the m transforms is stored in a **row** of the array, overwriting the corresponding original sequence. If the $n - 1$ components of the pth Fourier sine transform are denoted by $\hat{x}_k^p$, for $k = 1, 2, \ldots, n-1$; $p = 1, 2, \ldots, m$, then the mn elements of the array X contain the values

$$\hat{x}_1^1, \hat{x}_1^2, \ldots, \hat{x}_1^m,\ \hat{x}_2^1, \hat{x}_2^2, \ldots, \hat{x}_2^m, \ldots,\ \hat{x}_{n-1}^1, \hat{x}_{n-1}^2, \ldots, \hat{x}_{n-1}^m, 0, 0, \ldots, 0\ (m \text{ times}).$$

If $n = 1$, the m elements of X are set to zero.

4: INIT — CHARACTER*1 *Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the array TRIG, then INIT must be set equal to 'I' (**I**nitial call).

If INIT contains 'S' (**S**ubsequent call), then the routine assumes that trigonometric coefficients for the specified value of n are supplied in the array TRIG, having been calculated in a previous call to one of C06HAF, C06HBF, C06HCF or C06HDF.

If INIT contains 'R' (**R**estart), then the routine assumes that trigonometric coefficients for the particular value of n are supplied in the array TRIG, but does not check that C06HAF, C06HBF, C06HCF or C06HDF have previously been called. This option allows the TRIG array to be stored in an external file, read in and re-used without the need for a call with INIT equal to 'I'. The routine carries out a simple test to check that the current value of n is consistent with the array TRIG.

Constraint: INIT =, 'I', 'S' or 'R'.

5: TRIG(2*N) — ***real*** array *Input/Output*

On entry: if INIT = 'S' or 'R', TRIG must contain the required coefficients calculated in a previous call of the routine. Otherwise TRIG need not be set.

On exit: TRIG contains the required coefficients (computed by the routine if INIT = 'I').

6: WORK(M*N) — ***real*** array *Workspace*

7: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

IFAIL = 1

On entry, M < 1.

IFAIL = 2

On entry, N < 1.

IFAIL = 3

On entry, INIT is not one of 'I', 'S' or 'R'.

IFAIL = 4

Not used at this Mark.

IFAIL = 5

On entry, INIT = 'S' or 'R', but the array TRIG and the current value of N are inconsistent.

IFAIL = 6

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $nm \times \log n$, but also depends on the factors of n. The routine is fastest if the only prime factors of n are 2, 3 and 5, and is particularly slow if n is a large prime, or has large prime factors.

9 Example

This program reads in sequences of real data values and prints their Fourier sine transforms (as computed by C06HAF). It then calls C06HAF again and prints the results which may be compared with the original sequence.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06HAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=5,NMAX=20)
*     .. Local Scalars ..
```

```
      INTEGER          I, IFAIL, J, M, N
*     .. Local Arrays ..
      real             TRIG(2*NMAX), WORK(MMAX*NMAX), X(NMAX*MMAX)
*     .. External Subroutines ..
      EXTERNAL         C06HAF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06HAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=120) M, N
      IF (M.LE.MMAX .AND. N.LE.NMAX) THEN
         DO 40 J = 1, M
            READ (NIN,*) (X((I-1)*M+J),I=1,N-1)
   40    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         WRITE (NOUT,*)
         DO 60 J = 1, M
            WRITE (NOUT,99999) (X((I-1)*M+J),I=1,N-1)
   60    CONTINUE
         IFAIL = 0
*
*        -- Compute transform
         CALL C06HAF(M,N,X,'Initial',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Discrete Fourier sine transforms'
         WRITE (NOUT,*)
         DO 80 J = 1, M
            WRITE (NOUT,99999) (X((I-1)*M+J),I=1,N-1)
   80    CONTINUE
*
*        -- Compute inverse transform
         CALL C06HAF(M,N,X,'Subsequent',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data as restored by inverse transform'
         WRITE (NOUT,*)
         DO 100 J = 1, M
            WRITE (NOUT,99999) (X((I-1)*M+J),I=1,N-1)
  100    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of M or N'
      END IF
  120 STOP
*
99999 FORMAT (6X,6F10.4)
      END
```

9.2 Program Data

```
C06HAF Example Program Data
3  6 : Number of sequences, M,   (number of values in each sequence)+1, N
 0.6772  0.1138  0.6751  0.6362  0.1424  : X, sequence 1
 0.2983  0.1181  0.7255  0.8638  0.8723  : X, sequence 2
 0.0644  0.6037  0.6430  0.0428  0.4815  : X, sequence 3
```

9.3 Program Results

```
C06HAF Example Program Results

Original data values

        0.6772    0.1138    0.6751    0.6362    0.1424
        0.2983    0.1181    0.7255    0.8638    0.8723
        0.0644    0.6037    0.6430    0.0428    0.4815

Discrete Fourier sine transforms

        1.0014    0.0062    0.0834    0.5286    0.2514
        1.2477   -0.6599    0.2570    0.0859    0.2658
        0.8521    0.0719   -0.0561   -0.4890    0.2056

Original data as restored by inverse transform

        0.6772    0.1138    0.6751    0.6362    0.1424
        0.2983    0.1181    0.7255    0.8638    0.8723
        0.0644    0.6037    0.6430    0.0428    0.4815
```

C06HBF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06HBF computes the discrete Fourier cosine transforms of m sequences of real data values. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
SUBROUTINE C06HBF(M, N, X, INIT, TRIG, WORK, IFAIL)
INTEGER           M, N, IFAIL
real              X(M*(N+1)), TRIG(2*N), WORK(M*N)
CHARACTER*1       INIT
```

3 Description

Given m sequences of $n+1$ real data values x_j^p, for $j = 0, 1, \ldots, n$; $p = 1, 2, \ldots, m$, this routine simultaneously calculates the Fourier cosine transforms of all the sequences defined by:

$$\hat{x}_k^p = \sqrt{\frac{2}{n}} \left\{ \frac{1}{2}x_0^p + \sum_{j=1}^{n-1} x_j^p \times \cos\left(jk\frac{\pi}{n}\right) + \frac{1}{2}(-1)^k x_n^p \right\}, \quad k = 0, 1, \ldots, n; \quad p = 1, 2, \ldots, m.$$

(Note the scale factor $\sqrt{\frac{2}{n}}$ in this definition.)

The Fourier cosine transform is its own inverse and two calls of this routine with the same data will restore the original data.

The transform calculated by this routine can be used to solve Poisson's equation when the derivative of the solution is specified at both left and right boundaries (Swarztrauber [2]). (See the Chapter Introduction.)

The routine uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, described in Temperton [4], together with pre- and post-processing stages described in Swarztrauber [3]. Special coding is provided for the factors 2, 3, 4, 5 and 6. This routine is designed to be particularly efficient on vector processors, and it becomes especially fast as m, the number of transforms to be computed in parallel, increases.

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Swarztrauber P N (1977) The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle *SIAM Rev.* **19 (3)** 490–501

[3] Swarztrauber P N (1982) Vectorizing the FFT's *Parallel Computation* (ed G Rodrique) Academic Press 51–83

[4] Temperton C (1983) Fast mixed-radix real Fourier transforms *J. Comput. Phys.* **52** 340–350

5 Parameters

1: M — INTEGER *Input*

On entry: the number of sequences to be transformed, m.

Constraint: M $\geq$ 1.

2: N — INTEGER *Input*

On entry: one less than the number of real values in each sequence, i.e., the number of values in each sequence is $n + 1$.

Constraint: N ≥ 1.

3: X(M*(N+1)) — ***real*** array *Input/Output*

On entry: the data must be stored in X as if in a two-dimensional array of dimension (1:M,0:N); each of the m sequences is stored in a **row** of the array. In other words, if the $(n + 1)$ data values of the pth sequence to be transformed are denoted by x_j^p, for $j = 0, 1, \ldots, n$; $p = 1, 2, \ldots, m$, then the $m(n + 1)$ elements of the array X must contain the values

$$x_0^1, x_0^2, \ldots, x_0^m,\ x_1^1, x_1^2, \ldots, x_1^m, \ldots,\ x_n^1, x_n^2, \ldots, x_n^m.$$

On exit: the m Fourier cosine transforms stored as if in a two-dimensional array of dimension (1:M,0:N). Each of the m transforms is stored in a **row** of the array, overwriting the corresponding original data. If the $(n + 1)$ components of the pth Fourier cosine transform are denoted by $\hat{x}_k^p$, for $k = 0, 1, \ldots, n$; $p = 1, 2, \ldots, m$, then the $m(n + 1)$ elements of the array X contain the values

$$\hat{x}_0^1, \hat{x}_0^2, \ldots, \hat{x}_0^m,\ \hat{x}_1^1, \hat{x}_1^2, \ldots, \hat{x}_1^m, \ldots,\ \hat{x}_n^1, \hat{x}_n^2, \ldots, \hat{x}_n^m.$$

4: INIT — CHARACTER*1 *Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the array TRIG, then INIT must be set equal to 'I' (**I**nitial call).

If INIT contains 'S' (**S**ubsequent call), then the routine assumes that trigonometric coefficients for the specified value of n are supplied in the array TRIG, having been calculated in a previous call to one of C06HAF, C06HBF, C06HCF or C06HDF.

If INIT contains 'R' (**R**estart), then the routine assumes that trigonometric coefficients for the particular value of n are supplied in the array TRIG, but does not check that C06HAF, C06HBF, C06HCF or C06HDF have previously been called. This option allows the TRIG array to be stored in an external file, read in and re-used without the need for a call with INIT equal to 'I'. The routine carries out a simple test to check that the current value of n is consistent with the array TRIG.

Constraint: INIT = 'I', 'S' or 'R'.

5: TRIG(2*N) — ***real*** array *Input/Output*

On entry: if INIT = 'S' or 'R', TRIG must contain the required coefficients calculated in a previous call of the routine. Otherwise TRIG need not be set.

On exit: TRIG contains the required coefficients (computed by the routine if INIT = 'I'.

6: WORK(M*N) — ***real*** array *Workspace*

7: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

IFAIL = 1

On entry, M < 1.

IFAIL = 2

On entry, N < 1.

IFAIL = 3

On entry, INIT is not one of 'I', 'S' or 'R'.

IFAIL = 4

Not used at this Mark.

IFAIL = 5

On entry, INIT = 'S' or 'R', but the array TRIG and the current value of n are inconsistent.

IFAIL = 6

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $nm \times \log n$, but also depends on the factors of n. The routine is fastest if the only prime factors of n are 2, 3 and 5, and is particularly slow if n is a large prime, or has large prime factors.

9 Example

This program reads in sequences of real data values and prints their Fourier cosine transforms (as computed by C06HBF). It then calls the routine again and prints the results which may be compared with the original sequence.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06HBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=5,NMAX=20)
*     .. Local Scalars ..
```

```
      INTEGER          I, IFAIL, J, M, N
*     .. Local Arrays ..
      real             TRIG(2*NMAX), WORK(MMAX*NMAX), X((NMAX+1)*MMAX)
*     .. External Subroutines ..
      EXTERNAL         C06HBF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06HBF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=120) M, N
      IF (M.LE.MMAX .AND. N.LE.NMAX) THEN
         DO 40 J = 1, M
            READ (NIN,*) (X(I*M+J),I=0,N)
   40    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         WRITE (NOUT,*)
         DO 60 J = 1, M
            WRITE (NOUT,99999) (X(I*M+J),I=0,N)
   60    CONTINUE
         IFAIL = 0
*
*        -- Compute transform
         CALL C06HBF(M,N,X,'Initial',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Discrete Fourier cosine transforms'
         WRITE (NOUT,*)
         DO 80 J = 1, M
            WRITE (NOUT,99999) (X(I*M+J),I=0,N)
   80    CONTINUE
*
*        -- Compute inverse transform
         CALL C06HBF(M,N,X,'Subsequent',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data as restored by inverse transform'
         WRITE (NOUT,*)
         DO 100 J = 1, M
            WRITE (NOUT,99999) (X(I*M+J),I=0,N)
  100    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of M or N'
      END IF
  120 STOP
*
99999 FORMAT (6X,7F10.4)
      END
```

9.2 Program Data

```
C06HBF Example Program Data
3  6 : Number of sequences, M,   (number of values in each sequence)-1, N
 0.3854  0.6772  0.1138  0.6751  0.6362  0.1424  0.9562 : X, sequence 1
 0.5417  0.2983  0.1181  0.7255  0.8638  0.8723  0.4936 : X, sequence 2
 0.9172  0.0644  0.6037  0.6430  0.0428  0.4815  0.2057 : X, sequence 3
```

9.3 Program Results

```
C06HBF Example Program Results

Original data values

    0.3854    0.6772    0.1138    0.6751    0.6362    0.1424    0.9562
    0.5417    0.2983    0.1181    0.7255    0.8638    0.8723    0.4936
    0.9172    0.0644    0.6037    0.6430    0.0428    0.4815    0.2057

Discrete Fourier cosine transforms

    1.6833   -0.0482    0.0176    0.1368    0.3240   -0.5830   -0.0427
    1.9605   -0.4884   -0.0655    0.4444    0.0964    0.0856   -0.2289
    1.3838    0.1588   -0.0761   -0.1184    0.3512    0.5759    0.0110

Original data as restored by inverse transform

    0.3854    0.6772    0.1138    0.6751    0.6362    0.1424    0.9562
    0.5417    0.2983    0.1181    0.7255    0.8638    0.8723    0.4936
    0.9172    0.0644    0.6037    0.6430    0.0428    0.4815    0.2057
```

C06HCF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06HCF computes the discrete quarter-wave Fourier sine transforms of m sequences of real data values. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
SUBROUTINE C06HCF(DIRECT, M, N, X, INIT, TRIG, WORK, IFAIL)
INTEGER           M, N, IFAIL
real              X(M*N), TRIG(2*N), WORK(M*N)
CHARACTER*1       DIRECT, INIT
```

3 Description

Given m sequences of n real data values x_j^p, for $j = 1, 2, \ldots, n$; $p = 1, 2, \ldots, m$, this routine simultaneously calculates the quarter-wave Fourier sine transforms of all the sequences defined by:

$$\hat{x}_k^p = \frac{1}{\sqrt{n}} \left\{ \sum_{j=1}^{n-1} x_j^p \times \sin\left(j(2k-1)\frac{\pi}{2n}\right) + \frac{1}{2}(-1)^{k-1} x_n^p \right\}, \quad \text{if DIRECT = 'F',}$$

or its inverse

$$x_k^p = \frac{2}{\sqrt{n}} \sum_{j=1}^{n} \hat{x}_j^p \times \sin\left((2j-1)k\frac{\pi}{2n}\right), \quad \text{if DIRECT = 'B',}$$

for $k = 1, 2, \ldots, n$; $p = 1, 2, \ldots, m$.

(Note the scale factor $\frac{1}{\sqrt{n}}$ in this definition.)

A call of the routine with DIRECT = 'F' followed by a call with DIRECT = 'B' will restore the original data.

The transform calculated by this routine can be used to solve Poisson's equation when the solution is specified at the left boundary, and the derivative of the solution is specified at the right boundary (Swarztrauber [2]). (See the Chapter Introduction.)

The routine uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, described in Temperton [4], together with pre- and post-processing stages described in Swarztrauber [3]. Special coding is provided for the factors 2, 3, 4, 5 and 6. This routine is designed to be particularly efficient on vector processors, and it becomes especially fast as m, the number of transforms to be computed in parallel, increases.

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Swarztrauber P N (1977) The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle *SIAM Rev.* **19 (3)** 490–501

[3] Swarztrauber P N (1982) Vectorizing the FFT's *Parallel Computation* (ed G Rodrique) Academic Press 51–83

[4] Temperton C (1983) Fast mixed-radix real Fourier transforms *J. Comput. Phys.* **52** 340–350

5 Parameters

1: DIRECT — CHARACTER*1 *Input*

On entry: if the **F**orward transform as defined in Section 3 is to be computed, then DIRECT must be set equal to 'F'. If the **B**ackward transform is to be computed, that is the inverse, then DIRECT must be set equal to 'B'.

Constraint: DIRECT = 'F' or 'B'.

2: M — INTEGER *Input*

On entry: the number of sequences to be transformed, m.

Constraint: M ≥ 1.

3: N — INTEGER *Input*

On entry: the number of real values in each sequence, n.

Constraint: N ≥ 1.

4: X(M*N) — ***real*** array *Input/Output*

On entry: the data must be stored in X as if in a two-dimensional array of dimension (1:M,1:N); each of the m sequences is stored in a **row** of the array. In other words, if the data values of the p th sequence to be transformed are denoted by x_j^p, for $j = 1,2,\ldots,n$; $p = 1,2,\ldots,m$, then the mn elements of the array X must contain the values

$$x_1^1, x_1^2, \ldots, x_1^m,\ x_2^1, x_2^2, \ldots, x_2^m, \ldots,\ x_n^1, x_n^2, \ldots, x_n^m.$$

On exit: the m quarter-wave sine transforms stored as if in a two-dimensional array of dimension (1:M,1:N). Each of the m transforms is stored in a **row** of the array, overwriting the corresponding original sequence. If the n components of the pth quarter-wave sine transform are denoted by $\hat{x}_k^p$, for $k = 1,2,\ldots,n$; $p = 1,2,\ldots,m$, then the mn elements of the array X contain the values

$$\hat{x}_1^1, \hat{x}_1^2, \ldots, \hat{x}_1^m,\ \hat{x}_2^1, \hat{x}_2^2, \ldots, \hat{x}_2^m, \ldots,\ \hat{x}_n^1, \hat{x}_n^2, \ldots, \hat{x}_n^m.$$

5: INIT — CHARACTER*1 *Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the array TRIG, then INIT must be set equal to 'I' (**I**nitial call).

If INIT contains 'S' (**S**ubsequent call), then the routine assumes that trigonometric coefficients for the specified value of n are supplied in the array TRIG, having been calculated in a previous call to one of C06HAF, C06HBF, C06HCF or C06HDF.

If INIT contains 'R' (**R**estart), then the routine assumes that trigonometric coefficients for the particular value of n are supplied in the array TRIG, but does not check that routines C06HAF, C06HBF, C06HCF or C06HDF have previously been called. This option allows the TRIG array to be stored in an external file, read in and re-used without the need for a call with INIT equal to 'I'. The routine carries out a simple test to check that the current value of n is consistent with the array TRIG.

Constraint: INIT = 'I', 'S' or 'R'.

6: TRIG(2*N) — ***real*** array *Input/Output*

On entry: if INIT = 'S' or 'R', TRIG must contain the required coefficients calculated in a previous call of the routine. Otherwise TRIG need not be set.

On exit: TRIG contains the required coefficients (computed by the routine if INIT = 'I').

7: WORK(M*N) — ***real*** array *Workspace*

8: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

IFAIL = 1

On entry, M < 1.

IFAIL = 2

On entry, N < 1.

IFAIL = 3

On entry, INIT is not one of 'I', 'S' or 'R'.

IFAIL = 4

Not used at this Mark.

IFAIL = 5

On entry, INIT =, 'S' or 'R', but the array TRIG and the current value of N are inconsistent.

IFAIL = 6

On entry, DIRECT is not one of 'F' or 'B'.

IFAIL = 7

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $nm \times \log n$, but also depends on the factors of n. The routine is fastest if the only prime factors of n are 2, 3 and 5, and is particularly slow if n is a large prime, or has large prime factors.

9 Example

This program reads in sequences of real data values and prints their quarter-wave sine transforms as computed by C06HCF with DIRECT = or 'F'. It then calls the routine again with DIRECT = 'B' and prints the results which may be compared with the original data.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06HCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=5,NMAX=20)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, J, M, N
*     .. Local Arrays ..
      real             TRIG(2*NMAX), WORK(MMAX*NMAX), X(NMAX*MMAX)
*     .. External Subroutines ..
      EXTERNAL         C06HCF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06HCF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=120) M, N
      IF (M.LE.MMAX .AND. N.LE.NMAX) THEN
         DO 40 J = 1, M
            READ (NIN,*) (X(I*M+J),I=0,N-1)
   40    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         WRITE (NOUT,*)
         DO 60 J = 1, M
            WRITE (NOUT,99999) (X(I*M+J),I=0,N-1)
   60    CONTINUE
         IFAIL = 0
*
*        -- Compute transform
         CALL C06HCF('Forward',M,N,X,'Initial',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Discrete quarter-wave Fourier sine transforms'
         WRITE (NOUT,*)
         DO 80 J = 1, M
            WRITE (NOUT,99999) (X(I*M+J),I=0,N-1)
   80    CONTINUE
*
*        -- Compute inverse transform
         CALL C06HCF('Backward',M,N,X,'Subsequent',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data as restored by inverse transform'
         WRITE (NOUT,*)
         DO 100 J = 1, M
            WRITE (NOUT,99999) (X(I*M+J),I=0,N-1)
  100    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of M or N'
      END IF
  120 STOP
```

```
*
99999 FORMAT (6X,7F10.4)
      END
```

9.2 Program Data

```
C06HCF Example Program Data
3  6 : Number of sequences, M, and number of values in each sequence, N
 0.3854  0.6772  0.1138  0.6751  0.6362  0.1424  : X, sequence 1
 0.5417  0.2983  0.1181  0.7255  0.8638  0.8723  : X, sequence 2
 0.9172  0.0644  0.6037  0.6430  0.0428  0.4815  : X, sequence 3
```

9.3 Program Results

```
C06HCF Example Program Results

Original data values

        0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
        0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
        0.9172    0.0644    0.6037    0.6430    0.0428    0.4815

Discrete quarter-wave Fourier sine transforms

        0.7304    0.2078    0.1150    0.2577   -0.2869   -0.0815
        0.9274   -0.1152    0.2532    0.2883   -0.0026   -0.0635
        0.6268    0.3547    0.0760    0.3078    0.4987   -0.0507

Original data as restored by inverse transform

        0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
        0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
        0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```

C06HDF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

C06HDF computes the discrete quarter-wave Fourier cosine transforms of m sequences of real data values. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
SUBROUTINE C06HDF(DIRECT, M, N, X, INIT, TRIG, WORK, IFAIL)
INTEGER           M, N, IFAIL
real              X(M*N), TRIG(2*N), WORK(M*N)
CHARACTER*1       DIRECT, INIT
```

3 Description

Given m sequences of n real data values x_j^p, for $j = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$, this routine simultaneously calculates the quarter-wave Fourier cosine transforms of all the sequences defined by:

$$\hat{x}_k^p = \frac{1}{\sqrt{n}} \left\{ \frac{1}{2} x_0^p + \sum_{j=1}^{n-1} x_j^p \times \cos\left(j(2k-1)\frac{\pi}{2n} \right) \right\}, \quad \text{if DIRECT = 'F' or 'f',}$$

or its inverse

$$x_k^p = \frac{2}{\sqrt{n}} \sum_{j=0}^{n-1} \hat{x}_j^p \times \cos\left((2j-1)k\frac{\pi}{2n} \right), \quad \text{if DIRECT = 'B' } \textit{or} \text{ 'b',}$$

for $k = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$.

(Note the scale factor $\frac{1}{\sqrt{n}}$ in this definition.)

A call of the routine with DIRECT = 'F' followed by a call with DIRECT = 'B' will restore the original data.

The transform calculated by this routine can be used to solve Poisson's equation when the derivative of the solution is specified at the left boundary, and the solution is specified at the right boundary (Swarztrauber [2]). (See the Chapter Introduction.)

The routine uses a variant of the fast Fourier transform (FFT) algorithm (Brigham [1]) known as the Stockham self-sorting algorithm, described in Temperton [4], together with pre- and post-processing stages described in Swarztrauber [3]. Special coding is provided for the factors 2, 3, 4, 5 and 6. This routine is designed to be particularly efficient on vector processors, and it becomes especially fast as m, the number of transforms to be computed in parallel, increases.

4 References

[1] Brigham E O (1973) *The Fast Fourier Transform* Prentice-Hall

[2] Swarztrauber P N (1977) The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle *SIAM Rev.* **19 (3)** 490–501

[3] Swarztrauber P N (1982) Vectorizing the FFT's *Parallel Computation* (ed G Rodrique) Academic Press 51–83

[4] Temperton C (1983) Fast mixed-radix real Fourier transforms *J. Comput. Phys.* **52** 340–350

5 Parameters

1: DIRECT — CHARACTER*1 *Input*

On entry: if the **F**orward transform as defined in Section 3 is to be computed, then DIRECT must be set equal to 'F'. If the **B**ackward transform is to be computed, that is the inverse, then DIRECT must be set equal to 'B'.

Constraint: DIRECT = 'F' or 'B'.

2: M — INTEGER *Input*

On entry: the number of sequences to be transformed, m.

Constraint: $M \geq 1$.

3: N — INTEGER *Input*

On entry: the number of real values in each sequence, n.

Constraint: $N \geq 1$.

4: X(M*N) — ***real*** array *Input/Output*

On entry: the data must be stored in X as if in a two-dimensional array of dimension (1:M,0:N−1); each of the m sequences is stored in a **row** of the array. In other words, if the data values of the pth sequence to be transformed are denoted by x_j^p, for $j = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$, then the mn elements of the array X must contain the values

$$x_0^1, x_0^2, \ldots, x_0^m,\ x_1^1, x_1^2, \ldots, x_1^m, \ldots,\ x_{n-1}^1, x_{n-1}^2, \ldots, x_{n-1}^m.$$

On exit: the m quarter-wave cosine transforms stored as if in a two-dimensional array of dimension (1:M,0:N−1). Each of the m transforms is stored in a **row** of the array, overwriting the corresponding original sequence. If the n components of the pth quarter-wave cosine transform are denoted by $\hat{x}_k^p$, for $k = 0, 1, \ldots, n-1$; $p = 1, 2, \ldots, m$, then the mn elements of the array X contain the values

$$\hat{x}_0^1, \hat{x}_0^2, \ldots, \hat{x}_0^m,\ \hat{x}_1^1, \hat{x}_1^2, \ldots, \hat{x}_1^m, \ldots,\ \hat{x}_{n-1}^1, \hat{x}_{n-1}^2, \ldots, \hat{x}_{n-1}^m.$$

5: INIT — CHARACTER*1 *Input*

On entry: if the trigonometric coefficients required to compute the transforms are to be calculated by the routine and stored in the array TRIG, then INIT must be set equal to 'I' (**I**nitial call).

If INIT contains 'S' (**S**ubsequent call), then the routine assumes that trigonometric coefficients for the specified value of n are supplied in the array TRIG, having been calculated in a previous call to one of C06HAF, C06HBF, C06HCF or C06HDF.

If INIT contains 'R' (**R**estart), then the routine assumes that trigonometric coefficients for the particular value of n are supplied in the array TRIG, but does not check that C06HAF, C06HBF, C06HCF or C06HDF have previously been called. This option allows the TRIG array to be stored in an external file, read in and re-used without the need for a call with INIT equal to 'I'. The routine carries out a simple test to check that the current value of n is consistent with the array TRIG.

Constraint: INIT = 'I', 'S' or 'R'.

6: TRIG(2*N) — ***real*** array *Input/Output*

On entry: if INIT = 'S' or 'R', TRIG must contain the required coefficients calculated in a previous call of the routine. Otherwise TRIG need not be set.

On exit: TRIG contains the required coefficients (computed by the routine if INIT = 'I').

7: WORK(M*N) — ***real*** array *Workspace*

8: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

IFAIL = 1

On entry, M < 1.

IFAIL = 2

On entry, N < 1.

IFAIL = 3

On entry, INIT is not one of 'I', 'S' or 'R'.

IFAIL = 4

Not used at this Mark.

IFAIL = 5

On entry, INIT =, 'S' or 'R', but the array TRIG and the current value of N are inconsistent.

IFAIL = 6

On entry, DIRECT is not one of 'F' or 'B'.

IFAIL = 7

An unexpected error has occurred in an internal call. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

Some indication of accuracy can be obtained by performing a subsequent inverse transform and comparing the results with the original sequence (in exact arithmetic they would be identical).

8 Further Comments

The time taken by the routine is approximately proportional to $nm \times \log n$, but also depends on the factors of n. The routine is fastest if the only prime factors of n are 2, 3 and 5, and is particularly slow if n is a large prime, or has large prime factors.

9 Example

This program reads in sequences of real data values and prints their quarter-wave cosine transforms as computed by C06HDF with DIRECT = 'F'. It then calls the routine again with DIRECT = or 'B' and prints the results which may be compared with the original data.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06HDF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=5,NMAX=20)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, J, M, N
*     .. Local Arrays ..
      real             TRIG(2*NMAX), WORK(MMAX*NMAX), X(NMAX*MMAX)
*     .. External Subroutines ..
      EXTERNAL         C06HDF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06HDF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
   20 READ (NIN,*,END=120) M, N
      IF (M.LE.MMAX .AND. N.LE.NMAX) THEN
         DO 40 J = 1, M
            READ (NIN,*) (X(I*M+J),I=0,N-1)
   40    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data values'
         WRITE (NOUT,*)
         DO 60 J = 1, M
            WRITE (NOUT,99999) (X(I*M+J),I=0,N-1)
   60    CONTINUE
         IFAIL = 0
*
*        -- Compute transform
         CALL C06HDF('Forward',M,N,X,'Initial',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     'Discrete quarter-wave Fourier cosine transforms'
         WRITE (NOUT,*)
         DO 80 J = 1, M
            WRITE (NOUT,99999) (X(I*M+J),I=0,N-1)
   80    CONTINUE
*
*        -- Compute inverse transform
         CALL C06HDF('Backward',M,N,X,'Subsequent',TRIG,WORK,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Original data as restored by inverse transform'
         WRITE (NOUT,*)
         DO 100 J = 1, M
            WRITE (NOUT,99999) (X(I*M+J),I=0,N-1)
  100    CONTINUE
         GO TO 20
      ELSE
         WRITE (NOUT,*) 'Invalid value of M or N'
      END IF
```

```
  120 STOP
*
99999 FORMAT (6X,7F10.4)
      END
```

9.2 Program Data

```
C06HDF Example Program Data
3  6 : Number of sequences, M, and number of values in each sequence, N
 0.3854  0.6772  0.1138  0.6751  0.6362  0.1424 : X, sequence 1
 0.5417  0.2983  0.1181  0.7255  0.8638  0.8723 : X, sequence 2
 0.9172  0.0644  0.6037  0.6430  0.0428  0.4815 : X, sequence 3
```

9.3 Program Results

```
C06HDF Example Program Results

Original data values

        0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
        0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
        0.9172    0.0644    0.6037    0.6430    0.0428    0.4815

Discrete quarter-wave Fourier cosine transforms

        0.7257   -0.2216    0.1011    0.2355   -0.1406   -0.2282
        0.7479   -0.6172    0.4112    0.0791    0.1331   -0.0906
        0.6713   -0.1363   -0.0064   -0.0285    0.4758    0.1475

Original data as restored by inverse transform

        0.3854    0.6772    0.1138    0.6751    0.6362    0.1424
        0.5417    0.2983    0.1181    0.7255    0.8638    0.8723
        0.9172    0.0644    0.6037    0.6430    0.0428    0.4815
```

C06LAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06LAF estimates values of the inverse Laplace transform of a given function using a Fourier series approximation. Real and imaginary parts of the function, and a bound on the exponential order of the inverse, are required.

2. Specification

```
      SUBROUTINE C06LAF (FUN, N, T, VALINV, ERREST, RELERR, ALPHAB, TFAC,
     1                   MXTERM, NTERMS, NA, ALOW, AHIGH, NFEVAL, WORK,
     2                   IFAIL)
      INTEGER          N, MXTERM, NTERMS, NA, NFEVAL, IFAIL
      real             T(N), VALINV(N), ERREST(N), RELERR, ALPHAB, TFAC,
     1                 ALOW, AHIGH, WORK(4*MXTERM+2)
      EXTERNAL         FUN
```

3. Description

Given a function $F(p)$ defined for complex values of p, this routine estimates values of its inverse Laplace transform by Crump's method [2]. (For a definition of the Laplace transform and its inverse, see the Chapter Introduction.)

Crump's method applies the epsilon algorithm (Wynn [3]) to the summation in Durbin's Fourier series approximation [1]

$$f(t_j) \simeq \frac{e^{at_j}}{\tau}\left[\tfrac{1}{2}F(a)-\sum_{k=1}^{\infty}\left\{\mathrm{Re}\left(F\left(a+\frac{k\pi i}{\tau}\right)\right)\cos\frac{k\pi t_j}{\tau} - \mathrm{Im}\left(F\left(a+\frac{k\pi i}{\tau}\right)\right)\sin\frac{k\pi t_j}{\tau}\right\}\right],$$

for $j = 1,2,...,n$, by choosing a such that a prescribed relative error should be achieved. The method is modified slightly if $t = 0.0$ so that an estimate of $f(0.0)$ can be obtained when it has a finite value. τ is calculated as $t_{fac} \times \max(0.01, t_j)$, where $t_{fac} > 0.5$. The user specifies t_{fac} and α_b, an upper bound on the exponential order α of the inverse function $f(t)$. α has two alternative interpretations:

(i) α is the smallest number such that

$$|f(t)| \leq m \times \exp(\alpha t) \text{ for large } t,$$

(ii) α is the real part of the singularity of $F(p)$ with largest real part.

The method depends critically on the value of α. See Section 8 for further details. The routine calculates at least two different values of the parameter a, such that $a > \alpha_b$, in an attempt to achieve the requested relative error and provide error estimates. The values of t_j, for $j = 1,2,...,n$, must be supplied in monotonically increasing order. The routine calculates the values of the inverse function $f(t_j)$ in decreasing order of j.

4. References

[1] DURBIN, F.
Numerical Inversion of Laplace Transforms: an Efficient Improvement to Dubner and Abate's Method.
Comput. J., 17, pp. 371-376, 1974.

[2] CRUMP, K.S.
Numerical Inversion of Laplace Transforms Using a Fourier Series Approximation.
J. Assoc. Comput. Mach., 23, pp. 89-96, 1976.

[3] WYNN, P.
On a Device for Computing the $e_m(S_n)$ Transformation.
Math. Tables Aids Comp. 10, pp. 91-96, 1956.

5. Parameters

1: FUN – SUBROUTINE, supplied by the user. *External Procedure*

FUN must evaluate the real and imaginary parts of the function $F(p)$ for a given value of p.

Its specification is:

```
SUBROUTINE FUN(PR, PI, FR, FI)
real          PR, PI, FR, FI
```

1: PR – ***real***. *Input*
2: PI – ***real***. *Input*

On entry: the real and imaginary parts of the argument p.

3: FR – ***real***. *Output*
4: FI – ***real***. *Output*

On exit: the real and imaginary parts of the value $F(p)$.

FUN must be declared as EXTERNAL in the (sub)program from which C06LAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: N – INTEGER. *Input*

On entry: the number of points, n, at which the value of the inverse Laplace transform is required.

Constraint: N ≥ 1.

3: T(N) – ***real*** array. *Input*

On entry: each T(j) must specify a point at which the inverse Laplace transform is required, for $j = 1,2,...,n$.

Constraint: $0.0 \leq T(1) < T(2) < ... < T(n)$.

4: VALINV(N) – ***real*** array. *Output*

On exit: an estimate of the value of the inverse Laplace transform at $t = T(j)$, for $j = 1,2,...,n$.

5: ERREST(N) – ***real*** array. *Output*

On exit: an estimate of the error in VALINV(j). This is usually an estimate of relative error but, if VALINV(j) < RELERR, ERREST(j) estimates the absolute error. ERREST(j) is unreliable when VALINV(j) is small but slightly greater than RELERR.

6: RELERR – ***real***. *Input*

On entry: the required relative error in the values of the inverse Laplace transform. If the absolute value of the inverse is less than RELERR, then absolute accuracy is used instead. RELERR must be in the range 0.0 ≤ RELERR < 1.0. If RELERR is set too small or to 0.0, then the routine uses a value sufficiently larger than ***machine precision***.

7: ALPHAB – ***real***. *Input*

On entry: α_b, an upper bound for α (see Section 3). Usually, α_b should be specified equal to, or slightly larger than, the value of α. If $\alpha_b < \alpha$ then the prescribed accuracy may not be achieved or completely incorrect results may be obtained. If α_b is too large the routine will be inefficient and convergence may not be achieved.

Note: it is as important to specify α_b correctly as it is to specify the correct function for inversion.

8: TFAC – ***real***. *Input*

On entry: t_{fac}, a factor to be used in calculating the parameter τ. Larger values (e.g. 5.0) may be specified for difficult problems, but these may require very large values of MXTERM.

Suggested value: TFAC = 0.8.

Constraint: TFAC > 0.5.

9: MXTERM – INTEGER. *Input*

On entry: the maximum number of (complex) terms to be used in the evaluation of the Fourier series.

Suggested value: MXTERM ≥ 100, except for very simple problems.

Constraint: MXTERM ≥ 1.

10: NTERMS – INTEGER. *Output*

On exit: the number of (complex) terms actually used.

11: NA – INTEGER. *Output*

On exit: the number of values of a used by the routine. See Section 8.

12: ALOW – ***real***. *Output*

On exit: the smallest value of a used in the algorithm. This may be used for checking the value of ALPHAB – see Section 8.

13: AHIGH – ***real***. *Output*

On exit: the largest value of a used in the algorithm. This may be used for checking the value of ALPHAB – see Section 8.

14: NFEVAL – INTEGER. *Output*

On exit: the number of calls to FUN made by the routine.

15: WORK(4*MXTERM+2) – ***real*** array. *Workspace*

16: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, N < 1,
or MXTERM < 1,
or RELERR < 0.0,
or RELERR ≥ 1.0,
or TFAC ≤ 0.5.

IFAIL = 2

On entry, T(1) < 0.0,
or T(1),T(2),...,T(N) are not in strictly increasing order.

IFAIL = 3

T(N) is too large for this value of ALPHAB. If necessary, scale the problem as described in Section 8.

IFAIL = 4

The required accuracy cannot be obtained. It is possible that ALPHAB is less than α. Alternatively, the problem may be especially difficult. Try increasing TFAC, ALPHAB or both.

IFAIL = 5

Convergence failure in the epsilon algorithm. Some values of VALINV(j) may be calculated to the desired accuracy; this may be determined by examining the values of ERREST(j). Try reducing the range of T or increasing MXTERM. If IFAIL = 5 still results, try reducing TFAC.

IFAIL = 6

All values of VALINV(j) have been calculated but not all are to the requested accuracy; the values of ERREST(j) should be examined carefully. Try reducing the range of t, or increasing TFAC, ALPHAB or both.

7. Accuracy

The error estimates are often very close to the true error but, because the error control depends on an asymptotic formula, the required error may not always be met. There are two principal causes of this: Gibbs' phenomena, and zero or small values of the inverse Laplace transform.

Gibbs' phenomena (see the Chapter Introduction) are exhibited near $t = 0.0$ (due to the method) and around discontinuities in the inverse Laplace transform $f(t)$. If there is a discontinuity at $t = c$ then the method converges such that $f(c) \rightarrow (f(c-)+f(c+))/2$.

Apparent loss of accuracy, when $f(t)$ is small, may not be serious. Crump's method keeps control of relative error so that good approximations to small function values may appear to be very inaccurate. If $|f(t)|$ is estimated to be less than RELERR then this routine switches to absolute error estimation. However, when $|f(t)|$ is slightly larger than RELERR the relative error estimates are likely to cause IFAIL = 6. If this is found inconvenient it can sometimes be avoided by adding k/p to the function $F(p)$, which shifts the inverse to $k+f(t)$.

Loss of accuracy may also occur for highly oscillatory functions.

More serious loss of accuracy can occur if α is unknown and is incorrectly estimated. See Section 8.

8. Further Comments

8.1. Timing

The value of n is less important in general than the value of NTERMS. Unless the subroutine FUN is very inexpensive to compute, the timing is proportional to NA×NTERMS. For simple problems NA = 2 but in difficult problems NA may be somewhat larger.

8.2. Precautions

The user is referred to the Chapter Introduction for advice on simplifying problems with particular difficulties, e.g. where the inverse is known to be a step function.

The method does not work well for large values of t when α is positive. It is advisable, especially if IFAIL = 3 is obtained, to scale the problem if $|\alpha|$ is much greater than 1.0. See the Chapter Introduction.

The range of values of t specified for a particular call should not be greater than about 10 units. This is because the method uses parameters based on the value T(n) and these tend to be less appropriate as t becomes smaller. However, as the timing of the routine is not especially dependent on n, it is usually far more efficient to evaluate the inverse for ranges of t than to make separate calls to the routine for each value of t.

The most important parameter to specify correctly is ALPHAB, an upper bound for α. If, on entry, ALPHAB is sufficently smaller than α then completely incorrect results will be obtained with IFAIL = 0. Unless α is known theoretically it is strongly advised that the user should test any estimated value used. This may be done by specifying a single value of t (i.e T(n), n = 1) with two sets of suitable values of TFAC, RELERR and MXTERM, and examining the resulting values of ALOW and AHIGH. The value of T(1) should be chosen very carefully and the following points should be borne in mind:

(i) T(1) should be small but not too close to 0.0 because of Gibbs' phenomenon (see Section 7),

(ii) the larger the value of T(1), the smaller the range of values of a that will be used in the algorithm,

(iii) T(1) should ideally not be chosen such that f(T(1)) = 0.0 or a very small value. For suitable problems T(1) might be chosen as, say, 0.1 or 1.0 depending on these factors. The routine calculates ALOW from the formula

$$\text{ALOW} = \text{ALPHAB} - \frac{\ln(0.1\times\text{RELERR})}{2\times\tau}.$$

Additional values of a are computed by adding $1/\tau$ to the previous value. As τ = TFAC×T(n), it will be seen that large values of TFAC and RELERR will test for a close to ALPHAB. Small values of TFAC and RELERR will test for a large. If the result of both tests is IFAIL = 0, with comparable values for the inverse, then this gives some credibility to the chosen value of ALPHAB. The user should note that this test could be more computationally expensive than the calculation of the inverse itself. The example program (see Section 9) illustrates how such a test may be performed.

9. Example

The example program estimates the inverse Laplace transform of the function $F(p) = 1/(p+1/2)$. The true inverse of $F(p)$ is $\exp(-t/2)$. Two preliminary calls to the routine are made to verify that the chosen value of ALPHAB is suitable. For these tests the single value T(1) = 1.0 is used. To test values of a close to ALPHAB, the values TFAC = 5.0 and RELERR = 0.01 are chosen. To test larger a, the values TFAC = 0.8 and RELERR = 1.0E−3 are used. Because the values of the computed inverse are similar and IFAIL = 0 in each case, these tests show that there is unlikely to be a singularity of $F(p)$ in the region $-0.04 \leq \mathrm{Re}\, p \leq 6.51$.

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06LAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX, MXTERM
      PARAMETER        (NMAX=20,MXTERM=200)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             AHIGH, ALOW, ALPHAB, RELERR, TFAC
      INTEGER          I, IFAIL, N, NA, NFEVAL, NTERMS
*     .. Local Arrays ..
      real             ERREST(NMAX), T(NMAX), TRUREL(NMAX),
     +                 TRURES(NMAX), VALINV(NMAX), WORK(4*MXTERM+2)
*     .. External Subroutines ..
      EXTERNAL         C06LAF, FUN
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, EXP, real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06LAF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '(results may be machine-dependent)'
      ALPHAB = -0.5e0
      T(1) = 1.0e0
*
*     Test for values of a close to ALPHAB.
*
      RELERR = 0.01e0
      TFAC = 7.5e0
      WRITE (NOUT,*)
      WRITE (NOUT,99997) 'Test with T(1) =', T(1)
      WRITE (NOUT,*)
      WRITE (NOUT,99999) '  MXTERM =', MXTERM, '  TFAC =', TFAC,
     +  '  ALPHAB =', ALPHAB, '  RELERR =', RELERR
      IFAIL = -1
*
      CALL C06LAF(FUN,1,T,VALINV,ERREST,RELERR,ALPHAB,TFAC,MXTERM,
     +            NTERMS,NA,ALOW,AHIGH,NFEVAL,WORK,IFAIL)
*
      IF (IFAIL.GT.0 .AND. IFAIL.LT.5) GO TO 60
      WRITE (NOUT,*)
      WRITE (NOUT,*) '   T        Result        exp(-T/2)   ',
     +  'Relative error  Error estimate'
      TRURES(1) = EXP(-T(1)/2.0e0)
      TRUREL(1) = ABS((VALINV(1)-TRURES(1))/TRURES(1))
      WRITE (NOUT,99998) T(1), VALINV(1), TRURES(1), TRUREL(1),
     +  ERREST(1)
      WRITE (NOUT,*)
      WRITE (NOUT,99996) ' NTERMS =', NTERMS, '  NFEVAL =', NFEVAL,
     +  '  ALOW =', ALOW, '  AHIGH =', AHIGH, '  IFAIL =', IFAIL
*
*     Test for larger values of a.
*
      RELERR = 1.0e-3
      TFAC = 0.8e0
      WRITE (NOUT,*)
      WRITE (NOUT,99997) 'Test with T(1) =', T(1)
      WRITE (NOUT,*)
      WRITE (NOUT,99999) '  MXTERM =', MXTERM, '  TFAC =', TFAC,
     +  '  ALPHAB =', ALPHAB, '  RELERR =', RELERR
      IFAIL = -1
*
```

```
      CALL C06LAF(FUN,1,T,VALINV,ERREST,RELERR,ALPHAB,TFAC,MXTERM,
     +            NTERMS,NA,ALOW,AHIGH,NFEVAL,WORK,IFAIL)
*
      IF (IFAIL.GT.0 .AND. IFAIL.LT.5) GO TO 60
      WRITE (NOUT,*)
      WRITE (NOUT,*) '   T        Result        exp(-T/2)   ',
     +  'Relative error  Error estimate'
      TRURES(1) = EXP(-T(1)/2.0e0)
      TRUREL(1) = ABS((VALINV(1)-TRURES(1))/TRURES(1))
      WRITE (NOUT,99998) T(1), VALINV(1), TRURES(1), TRUREL(1),
     +  ERREST(1)
      WRITE (NOUT,*)
      WRITE (NOUT,99996) ' NTERMS =', NTERMS, '  NFEVAL =', NFEVAL,
     +  '  ALOW =', ALOW, '  AHIGH =', AHIGH, '  IFAIL =', IFAIL
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Compute inverse'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) '  MXTERM =', MXTERM, '  TFAC =', TFAC,
     +  '  ALPHAB =', ALPHAB, '  RELERR =', RELERR
      WRITE (NOUT,*)
      WRITE (NOUT,*) '   T        Result        exp(-T/2)   ',
     +  'Relative error  Error estimate'
      N = 5
      DO 20 I = 1, N
         T(I) = real(I)
   20 CONTINUE
      IFAIL = -1
*
      CALL C06LAF(FUN,N,T,VALINV,ERREST,RELERR,ALPHAB,TFAC,MXTERM,
     +            NTERMS,NA,ALOW,AHIGH,NFEVAL,WORK,IFAIL)
*
      IF (IFAIL.GT.0 .AND. IFAIL.LT.5) GO TO 60
      DO 40 I = 1, N
         TRURES(I) = EXP(-T(I)/2.0e0)
         TRUREL(I) = ABS((VALINV(I)-TRURES(I))/TRURES(I))
   40 CONTINUE
      WRITE (NOUT,99998) (T(I),VALINV(I),TRURES(I),TRUREL(I),ERREST(I),
     +  I=1,N)
   60 WRITE (NOUT,*)
      WRITE (NOUT,99996) ' NTERMS =', NTERMS, '  NFEVAL =', NFEVAL,
     +  '  ALOW =', ALOW, '  AHIGH =', AHIGH, '  IFAIL =', IFAIL
*
99999 FORMAT (1X,A,I4,A,F6.2,A,F6.2,A,1P,e8.1)
99998 FORMAT (1X,F4.1,7X,F6.3,9X,F6.3,8X,e8.1,8X,e8.1)
99997 FORMAT (1X,A,F4.1)
99996 FORMAT (1X,A,I4,A,I4,A,F7.2,A,F7.2,A,I2)
      END
*
      SUBROUTINE FUN(PR,PI,FR,FI)
*     Function to be inverted
*     .. Scalar Arguments ..
      real             FI, FR, PI, PR
*     .. External Subroutines ..
      EXTERNAL         A02ACF
*     .. Executable Statements ..
      CALL A02ACF(1.0e0,0.0e0,PR+0.5e0,PI,FR,FI)
*
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C06LAF Example Program Results

(results may be machine-dependent)

Test with T(1) = 1.0

  MXTERM = 200  TFAC =  7.50  ALPHAB = -0.50  RELERR = 1.0E-02

    T        Result        exp(-T/2)   Relative error  Error estimate
 1.0         0.607          0.607         0.1E-02         0.4E-02

 NTERMS =  18  NFEVAL =  36  ALOW =  -0.04  AHIGH =   0.09  IFAIL = 0

Test with T(1) = 1.0

  MXTERM = 200  TFAC =  0.80  ALPHAB = -0.50  RELERR = 1.0E-03

    T        Result        exp(-T/2)   Relative error  Error estimate
 1.0         0.607          0.607         0.2E-04         0.8E-04

 NTERMS =  13  NFEVAL =  28  ALOW =   5.26  AHIGH =   6.51  IFAIL = 0

Compute inverse

  MXTERM = 200  TFAC =  0.80  ALPHAB = -0.50  RELERR = 1.0E-03

    T        Result        exp(-T/2)   Relative error  Error estimate
 1.0         0.607          0.607         0.5E-04         0.3E-03
 2.0         0.368          0.368         0.7E-05         0.9E-04
 3.0         0.223          0.223         0.2E-04         0.8E-04
 4.0         0.135          0.135         0.1E-04         0.8E-04
 5.0         0.082          0.082         0.2E-04         0.8E-04

 NTERMS =  23  NFEVAL =  43  ALOW =   0.65  AHIGH =   0.90  IFAIL = 0
```

C06LBF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06LBF computes the inverse Laplace transform $f(t)$ of a user-supplied function $F(s)$, defined for complex s. The routine uses a modification of Weeks' method which is suitable when $f(t)$ has continuous derivatives of all orders. The routine returns the coefficients of an expansion which approximates $f(t)$ and can be evaluated for given values of t by subsequent calls of C06LCF.

2. Specification

```
SUBROUTINE C06LBF (F, SIGMA0, SIGMA, B, EPSTOL, MMAX, M,
1                  ACOEF, ERRVEC, IFAIL)
INTEGER          MMAX, M, IFAIL
real             SIGMA0, SIGMA, B, EPSTOL, ACOEF(MMAX), ERRVEC(8)
complex          F
EXTERNAL         F
```

3. Description

Given a function $f(t)$ of a real variable t, its Laplace transform $F(s)$ is a function of a complex variable s, defined by:

$$F(s) = \int_0^\infty e^{-st} f(t)\, dt, \qquad \mathrm{Re}\, s > \sigma_0.$$

Then $f(t)$ is the inverse Laplace transform of $F(s)$. The value σ_0 is referred to as the abscissa of convergence of the Laplace transform; it is the rightmost real part of the singularities of $F(s)$.

This routine, along with its companion C06LCF, attempts to solve the following problem:

given a function $F(s)$, compute values of its inverse Laplace transform $f(t)$ for specified values of t.

The method is a modification of Weeks' method (see Garbow *et al.* [1]), which approximates $f(t)$ by a truncated Laguerre expansion:

$$\tilde{f}(t) = e^{\sigma t} \sum_{i=0}^{m-1} a_i\, e^{-bt/2}\, L_i(bt), \qquad \sigma > \sigma_0, \quad b > 0$$

where $L_i(x)$ is the Laguerre polynomial of degree i. This routine computes the coefficients a_i of the above Laguerre expansion; the expansion can then be evaluated for specified t by calling C06LCF. The user must supply the value of σ_0, and also suitable values for σ and b: see Section 8 for guidance.

The method is only suitable when $f(t)$ has continuous derivatives of all orders. For such functions the approximation $\tilde{f}(t)$ is usually good and inexpensive. The routine will fail with an error exit if the method is not suitable for the supplied function $F(s)$.

The routine is designed to satisfy an accuracy criterion of the form:

$$\left| \frac{f(t) - \tilde{f}(t)}{e^{\sigma t}} \right| < \varepsilon_{tol}, \qquad \text{for all } t$$

where ε_{tol} is a user-supplied bound. The error measure on the left hand side is referred to as the **pseudo-relative error**, or **pseudo-error** for short. Note that if $\sigma > 0$ and t is large, the absolute error in $\tilde{f}(t)$ may be very large.

C06LBF is derived from the subroutine MODUL1 in [2].

4. References

[1] GARBOW B.S., GIUNTA G., LYNESS J.N. and MURLI A.
Software for an implementation of Weeks' method for the inverse Laplace transform problem.
A.C.M. Trans. Math. Software, 14, pp. 163-170, 1988.

[2] GARBOW B.S., GIUNTA G., LYNESS J.N. and MURLI A.
Algorithm 662: A Fortran software package for the numerical inversion of the Laplace transform based on Weeks' method.
A.C.M. Trans. Math. Software, 14, pp. 171-176, 1988.

5. Parameters

1: F – ***complex*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the Laplace transform function $F(s)$ for a given complex value of s.

Its specification is:

```
complex FUNCTION F (S)
complex          S
```

1: S – ***complex***. *Input*

On entry: the value of s for which $F(s)$ must be evaluated. The real part of S is greater than σ_0.

F must be declared as EXTERNAL in the (sub)program from which C06LBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: SIGMA0 – ***real***. *Input*

On entry: the abscissa of convergence of the Laplace transform, σ_0.

3: SIGMA – ***real***. *Input/Output*

On entry: the parameter σ of the Laguerre expansion. If on entry SIGMA $\leq \sigma_0$, SIGMA is reset to $\sigma_0 + 0.7$.

On exit: the value actually used for σ, as just described.

4: B – ***real***. *Input/Output*

On entry: the parameter b of the Laguerre expansion. If on entry B $< 2(\sigma-\sigma_0)$, B is reset to $2.5(\sigma-\sigma_0)$.

On exit: the value actually used for b, as just described.

5: EPSTOL – ***real***. *Input*

On entry: the required relative pseudo-accuracy, that is, an upper bound on $|f(t)-\tilde{f}(t)|e^{-\sigma t}$.

6: MMAX – INTEGER. *Input*

On entry: an upper bound on the number of Laguerre expansion coefficients to be computed. The number of coefficients actually computed is always a power of 2, so MMAX should be a power of 2; if MMAX is not a power of 2 then the maximum number of coefficients calculated will be the largest power of 2 less than MMAX.

Suggested value: MMAX = 1024 is sufficient for all but a few exceptional cases.

Constraint: MMAX ≥ 8.

7: M – INTEGER. *Output*

On exit: the number of Laguerre expansion coefficients actually computed. The number of calls to F is M/2 + 2.

8: ACOEF(MMAX) – ***real*** array. *Output*

On exit: the first M elements contain the computed Laguerre expansion coefficients, a_i.

9: ERRVEC(8) – ***real*** array. *Output*

On exit: an 8-component vector of diagnostic information:

ERRVEC(1) = overall estimate of the pseudo-error $|f(t)-\tilde{f}(t)|e^{-\sigma t}$;
= ERRVEC(2) + ERRVEC(3) + ERRVEC(4);

ERRVEC(2) = estimate of the discretisation pseudo-error;

ERRVEC(3) = estimate of the truncation pseudo-error;

ERRVEC(4) = estimate of the condition pseudo-error on the basis of minimal noise levels in function values;

ERRVEC(5) = K, coefficient of a heuristic decay function for the expansion coefficients;

ERRVEC(6) = R, base of the decay function for the expansion coefficients;

ERRVEC(7) = logarithm of the largest expansion coefficient; and

ERRVEC(8) = logarithm of the smallest nonzero expansion coefficient.

The values K and R returned in ERRVEC(5) and ERRVEC (6) define a decay function KR^{-i} constructed by the routine for the purposes of error estimation. It satisfies

$$|a_i| < KR^{-i}, \qquad \text{for } i = 1,2,...,m.$$

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, MMAX < 8.

IFAIL = 2

The estimated pseudo-error bounds are slightly larger than EPSTOL. Note, however, that the actual errors in the final results may be smaller than EPSTOL as bounds independent of the value of t are pessimistic.

IFAIL = 3

Computation was terminated early because the estimate of rounding error was greater than EPSTOL. Increasing EPSTOL may help.

IFAIL = 4

The decay rate of the coefficients is too small. Increasing MMAX may help.

IFAIL = 5

The decay rate of the coefficients is too small. In addition the rounding error is such that the required accuracy cannot be obtained. Increasing MMAX or EPSTOL may help.

IFAIL = 6

The behaviour of the coefficients does not enable reasonable prediction of error bounds. Check the value of SIGMA0. In this case, ERRVEC(i) is set to –1.0, for i = 1 to 5.

When IFAIL ≥ 3, changing SIGMA or B may help. If not, the method should be abandoned.

7. Accuracy

The error estimate returned in ERRVEC(1) has been found in practice to be a highly reliable bound on the pseudo-error $|f(t)-\tilde{f}(t)|e^{-\sigma t}$.

8. Further Comments

8.1 The Role of σ_0

Nearly all techniques for inversion of the Laplace transform require the user to supply the value of σ_0, the convergence abscissa, or else an upper bound on σ_0. For this routine, one of the reasons for having to supply σ_0 is that the parameter σ must be greater than σ_0; otherwise the series for $\tilde{f}(t)$ will not converge.

If you do not know the value of σ_0, you must be prepared for significant preliminary effort, either in experimenting with the method and obtaining chaotic results, or in attempting to locate the rightmost singularity of $F(s)$.

The value of σ_0 is also relevant in defining a natural accuracy criterion. For large t, $f(t)$ is of uniform numerical order $ke^{\sigma_0 t}$, so a **natural** measure of relative accuracy of the approximation $\tilde{f}(t)$ is:

$$\varepsilon_{nat}(t) = (\tilde{f}(t) - f(t))/e^{\sigma_0 t} .$$

The routine uses the supplied value of σ_0 only in determining the values of σ and b (see below); thereafter it bases its computation entirely on σ and b.

8.2 Choice of σ

Even when the value of σ_0 is known, choosing a value for σ is not easy. Briefly, the series for $\tilde{f}(t)$ converges slowly when $\sigma - \sigma_0$ is small, and faster when $\sigma - \sigma_0$ is larger. However the natural accuracy measure satisfies

$$|\varepsilon_{nat}(t)| < \varepsilon_{tol}\ e^{(\sigma-\sigma_0)t}$$

and this degrades exponentially with t, the exponential constant being $\sigma - \sigma_0$.

Hence, if you require meaningful results over a large range of values of t, you should choose $\sigma - \sigma_0$ small, in which case the series for $\tilde{f}(t)$ converges slowly; while for a smaller range of values of t, you can allow $\sigma - \sigma_0$ to be larger and obtain faster convergence.

The default value for σ used by the routine is $\sigma_0 + 0.7$. There is no theoretical justification for this.

8.3 Choice of b

The simplest advice for choosing b is to set $b/2 \geq \sigma - \sigma_0$. The default value used by the routine is $2.5(\sigma - \sigma_0)$.

A more refined choice is to set

$$b/2 \geq \min_j |\sigma - s_j||$$

where s_j are the singularities of $F(s)$.

9. Example

To compute values of the inverse Laplace transform of the function

$$F(s) = \frac{3}{s^2 - 9}.$$

The exact answer is

$$f(t) = \sinh 3t.$$

The program first calls C06LBF to compute the coefficients of the Laguerre expansion, and then calls C06LCF to evaluate the expansion at $t = 1, 2, 3, 4, 5$.

9.1. Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     C06LBF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          MMAX
      PARAMETER        (MMAX=512)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             B, EPSTOL, EXACT, FINV, PSERR, SIGMA, SIGMA0, T
      INTEGER          IFAIL, J, M
*     .. Local Arrays ..
      real             ACOEF(MMAX), ERRVEC(8)
*     .. External Subroutines ..
      EXTERNAL         C06LBF, C06LCF
*     .. External Functions ..
      complex          F
      EXTERNAL         F
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, EXP, real, SINH
*     .. Executable Statements ..
      WRITE (NOUT,*) 'C06LBF Example Program Results'
      SIGMA0 = 3.0e0
      EPSTOL = 0.00001e0
      SIGMA = 0.0e0
      B = 0.0e0
      IFAIL = 0
*
*     Compute inverse transform
      CALL C06LBF(F,SIGMA0,SIGMA,B,EPSTOL,MMAX,M,ACOEF,ERRVEC,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'No. of coefficients returned by C06LBF =', M
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  '                    Computed          Exact       Pseudo'
      WRITE (NOUT,*)
     +  '         t             f(t)            f(t)        error'
      WRITE (NOUT,*)
*
*     Evaluate inverse transform for different values of t
      DO 20 J = 0, 5
         T = real(J)
*
         CALL C06LCF(T,SIGMA,B,M,ACOEF,ERRVEC,FINV,IFAIL)
*
         EXACT = SINH(3.0e0*T)
         PSERR = ABS(FINV-EXACT)/EXP(SIGMA*T)
         WRITE (NOUT,99998) T, FINV, EXACT, PSERR
   20 CONTINUE
      STOP
*
```

```
99999 FORMAT (1X,A,I6)
99998 FORMAT (1X,1P,e10.2,2e15.4,e12.1)
      END
*
      complex  FUNCTION F(S)
*     .. Scalar Arguments ..
      complex             S
*     .. Executable Statements ..
      F = 3.0e0/(S**2-9.0e0)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
C06LBF Example Program Results

No. of coefficients returned by C06LBF =    64

                 Computed          Exact       Pseudo
        t            f(t)           f(t)        error

  0.00E+00     1.5129E-09     0.0000E+00     1.5E-09
  1.00E+00     1.0018E+01     1.0018E+01     1.7E-09
  2.00E+00     2.0171E+02     2.0171E+02     1.2E-10
  3.00E+00     4.0515E+03     4.0515E+03     9.8E-10
  4.00E+00     8.1377E+04     8.1377E+04     3.0E-10
  5.00E+00     1.6345E+06     1.6345E+06     1.7E-09
```

C06LCF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details. The routine name may be precision-dependent.

1. Purpose

C06LCF evaluates an inverse Laplace transform at a given point, using the expansion coefficients computed by C06LBF.

2. Specification

```
SUBROUTINE C06LCF (T, SIGMA, B, M, ACOEF, ERRVEC, FINV, IFAIL)
INTEGER          M, IFAIL
real             T, SIGMA, B, ACOEF(M), ERRVEC(8), FINV
```

3. Description

This routine is designed to be used following a call to C06LBF, which computes an inverse Laplace transform by representing it as a Laguerre expansion of the form:

$$\tilde{f}(t) = e^{\sigma t} \sum_{i=0}^{m-1} a_i \; e^{-bt/2} \; L_i(bt), \qquad \sigma > \sigma_0, \quad b > 0$$

where $L_i(x)$ is the Laguerre polynomial of degree i.

This routine simply evaluates the above expansion for a specified value of t.

C06LCF is derived from the subroutine MODUL2 in [1].

4. References

[1] GARBOW B.S., GIUNTA G., LYNESS J.N. and MURLI A.
Algorithm 662: A Fortran software package for the numerical inversion of the Laplace transform based on Weeks' method.
A.C.M. Trans. Math. Software, 14, pp. 171-176, 1988.

5. Parameters

1: T – ***real***. *Input*

On entry: the value t for which the inverse Laplace transform $f(t)$ must be evaluated.

2: SIGMA – ***real***. *Input*
3: B – ***real***. *Input*
4: M – INTEGER. *Input*
5: ACOEF(M) – ***real*** array. *Input*
6: ERRVEC(8) – ***real*** array. *Input*

On entry: SIGMA, B, M, ACOEF and ERRVEC must be unchanged from the previous call of C06LBF.

7: FINV – ***real***. *Output*

On exit: the approximation to the inverse Laplace transform at t.

8: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

The approximation to $f(t)$ is too large to be representable: FINV is set to 0.0.

IFAIL = 2

The approximation to $f(t)$ is too small to be representable: FINV is set to 0.0.

7. Accuracy

The error estimate returned by C06LBF in ERRVEC(1) has been found in practice to be a highly reliable bound on the pseudo-error $|f(t)-\tilde{f}(t)|e^{-\sigma t}$.

8. Further Comments

The routine is primarily designed to evaluate $\tilde{f}(t)$ when $t > 0$. When $t \leq 0$, the result approximates the analytic continuation of $f(t)$; the approximation becomes progressively poorer as t becomes more negative.

9. Example

See example for C06LBF.