NAG Fortran Library

Introductory Guide

Mark 19

This Introductory Guide serves as an extensive introduction to the NAG Fortran Library, Mark 19. For each chapter of the Library, it gives background advice on the subject area covered, recommendations on the choice and use of routines and a summary of the purpose of each routine. For a detailed description of the use of each routine, refer to the main NAG Fortran Library Manual, Mark 19.

NAG Fortran Library Introductory Guide, Mark 19

July 1999 ISBN 1-85206-170-7

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

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

Nihon NAG KK
Nagashima Building 2F
2-24-3 Higashi
Shibuya-ku
Tokyo
Japan

Tel: +81 (0)3 5485 2901
Fax: +81 (0)3 5485 2903

NAG GmbH
Schleißheimerstraße 5
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.

NAG Fortran Library Introductory Guide, Mark 19

Contents

G08	Nonparametric Statistics
G10	Smoothing in Statistics
G11	Contingency Table Analysis
G12	Survival Analysis
G13	Time Series Analysis
H	Operations Research
M01	Sorting
P01	Error Trapping
S	Approximations of Special Functions
X01	Mathematical Constants
X02	Machine Constants
X03	Innerproducts
X04	Input/Output Utilities
X05	Date and Time Utilities

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')

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

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

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

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)

Introduction

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 (Chapter H and Chapter 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).

Chapter F06 (Linear Algebra Support Routines) contains all the Basic Linear Algebra Subprograms, BLAS, with NAG-style names as well as with the actual BLAS names, e.g., F06AAF (SROTG/DROTG). The names in brackets are the equivalent single and double precision BLAS names respectively. Chapter F07 (Linear Equations (LAPACK)) and Chapter F08 (Least-squares and Eigenvalue Problems (LAPACK)) contain routines derived from the LAPACK project. Like the BLAS, these routines have NAG-style names as well as LAPACK names, e.g., F07ADF (SGETRF/DGETRF). Details regarding these alternate names can be found in the relevant Chapter Introductions.

In order to take full advantage of machine-specific versions of BLAS and LAPACK routines provided by some computer hardware vendors, you are encouraged to use the BLAS and LAPACK names (e.g., SROTG/DROTG and SGETRF/DGETRF) rather than the corresponding NAG-style names (e.g., F06AAF and F07ADF) wherever possible in your programs.

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 in Volume 1 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 (Chapter F06, Chapter X01, Chapter X02) do not have individual routine documents; instead, all the routines are described together in the Chapter Introduction. Another exception is Chapter A00, which contains neither a Chapter Introduction nor any routine documents. It does however contain a user-callable support routine that identifies which version of the Library is available at your site (see Section 1.7).

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 ability to display mathematics and symbols has now reached a stage whereby it is possible to produce a satisfactory full HTML version of the Library documentation that will provide ready access to users via standard Web browsers. This HTML version will replace the current hypertext version (TextWare), but will retain many of the features of that product. The aim is to have an HTML version of Mark 19 of the Fortran Library documentation available for distribution with the Library software. It will also be accessible via the NAG Web site. Future releases may take advantage of technology that is currently being developed (e.g., MathML).

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., 16, 17, 18. The current Mark is 19.

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., 19A, 19B, and so on (Mark 19 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 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 most 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: FLSOL19D
                Mark: 19
*** 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 may 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 concerned with (b).

2.2 Programming Advice

The NAG Fortran Library and its documentation are designed on the assumption that you 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 ± 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 Chapter Introduction or the F08 Chapter Introduction for further details.

Some routines in Chapter F06 output an error message if an illegal input parameter is detected, then terminate program execution immediately. See the F06 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.

2.6 Thread Safety

Some implementations of the Library facilitate the use of threads; that is, you can call routines from the Library from within a multi-threaded application. You should note however that Mark 19 is not fully thread safe. See the document 'Thread Safety' for more detailed guidance on using the Library in a multi-threaded context. You may also need to refer to the Users' Note for details of whether your implementation of the Library has been compiled in a manner that facilitates the use of threads.

2.7 Calling the Library from Other Languages

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

As part of its Library service, NAG provides a C Header Files service which comprises a set of header files indicating the match between C and Fortran data types for various compilers, documentation and examples. The documentation and examples are available from the NAG Web site.

The Dynamic Link Library (DLL) version can be called in a straightforward manner from Visual Basic. Guidance on this is provided as part of the NAG Fortran Library DLLs. Further details can be found on the NAG Web site.

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:

Library Contents	–	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 upper case 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 ≥ 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 uses only the first N columns.

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

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

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 your 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 to you in this form. They may also be obtained directly from the NAG Web site.

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 online 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 compile and run it. You may of course need to refer back to the relevant documentation in the case of difficulties, for advice on assessment of results, and so on.

As you become familiar with the Library, some of steps (a) to (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 or available on the NAG Web site;
- 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 Support from NAG

NAG places considerable emphasis on providing high quality user support. In addition to comprehensive documentation we offer a variety of services to support our users.

(a) NAG Response Centres

The Response Centres are available to answer 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 telephone (answering machine) at all times. You can find the contact details for your local Response Centre in the Support Documentation supplied with this product.

However, general queries concerning this library should be directed initially to any local advisory service your site may provide.

(b) NAG Web Sites

The NAG web sites provide a valuable resource for product information, technical documentation and demonstrations, as well as articles of more general interest. The sites can be accessed at:

www.nag.co.uk or www.nag.com

(c) Training Courses

NAG organises workshops and training courses at various locations throughout the world. Information about forthcoming courses is posted on the NAG web sites. If you have a particular training requirement please contact us.

As well as offering these services to users, NAG values feedback to ensure that we continue to develop products that meet your needs. We welcome your comments.

5 Background to NAG

Various aspects of the design and development of the NAG Library, and NAG's technical policies and organisation are given in references [2], [3], [4], and [5].

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 19 News

1 Introduction

At Mark 19 of the Fortran Library new functionality has been introduced in addition to improvements in existing areas. The Library now contains 1155 documented routines, of which 62 are new at this Mark. These extend the areas of fast Fourier transforms (FFTs), optimization, eigenvalue problems (LAPACK), sparse linear algebra, statistics, operations research (OR) and sorting as summarized below.

The most significant additions to the FFT chapter (Chapter C06) are as follows:

- new routines for complex Fourier transforms using complex data type arrays;
- new routines for sine and cosine transforms.

Coverage in the optimization chapter (Chapter E04) has been extended with the addition of a routine to solve sparse nonlinear programming problems.

New routines for solving eigenproblems (Chapter F08) are included for:

- computing all the eigenvalues (and optionally all the eigenvectors) of real symmetric and complex Hermitian matrices;
- reducing real and complex rectangular band matrices to upper bidiagonal form;
- computing a split Cholesky factorization of real symmetric positive-definite and complex Hermitian positive-definite band matrices;
- reducing real symmetric-definite and complex Hermitian-definite banded generalized eigenproblems to standard form.

Coverage in the sparse linear algebra chapter (Chapter F11) has been extended to provide iterative methods and preconditioners for complex symmetric and non-Hermitian linear systems of equations.

Two of the new routines are in the statistics chapters (Chapter G01 to Chapter G13). They include facilities (in the stated chapters) for:

- conditional logistic analysis for case-control studies and survival analysis (G11);
- computing the risk sets in the analysis of survival data (G12).

Coverage in the OR chapter (Chapter H) has been extended to provide solvers for dense and sparse integer quadratic programming problems.

A new routine for sorting a vector of complex numbers into the order specified by a vector of ranks is included in Chapter M01.

2 New Routines

The 62 new user-callable routines included in the NAG Fortran Library at Mark 19 are as follows.

C06PAF	Single one-dimensional real and Hermitian complex discrete Fourier transform, using complex data format for Hermitian sequences
C06PCF	Single one-dimensional complex discrete Fourier transform, complex data format
C06PFF	One-dimensional complex discrete Fourier transform of multi-dimensional data (using complex data type)
C06PJF	Multi-dimensional complex discrete Fourier transform of multi-dimensional data (using complex data type)
C06PKF	Circular convolution or correlation of two complex vectors
C06PPF	Multiple one-dimensional real and Hermitian complex discrete Fourier transforms, using complex data format for Hermitian sequences
C06PQF	Multiple one-dimensional real and Hermitian complex discrete Fourier transforms, using complex data format for Hermitian sequences and sequences stored as columns
C06PRF	Multiple one-dimensional complex discrete Fourier transforms using complex data format
C06PSF	Multiple one-dimensional complex discrete Fourier transforms using complex data format and sequences stored as columns

C06PUF	Two-dimensional complex discrete Fourier transform, complex data format
C06PXF	Three-dimensional complex discrete Fourier transform, complex data format
C06RAF	Discrete sine transform (easy-to-use)
C06RBF	Discrete cosine transform (easy-to-use)
C06RCF	Discrete quarter-wave sine transform (easy-to-use)
C06RDF	Discrete quarter-wave cosine transform (easy-to-use)
E04UGF	NLP problem (sparse)
E04UHF	Read optional parameter values for E04UGF from external file
E04UJF	Supply optional parameter values to E04UGF
F08FCF	(SSYEVD/DSYEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, using divide and conquer
F08FQF	(CHEEVD/ZHEEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, using divide and conquer
F08GCF	(SSPEVD/DSPEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, packed storage, using divide and conquer
F08GQF	(CHPEVD/ZHPEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, packed storage, using divide and conquer
F08HCF	(SSBEVD/DSBEVD) All eigenvalues and optionally all eigenvectors of real symmetric band matrix, using divide and conquer
F08HQF	(CHBEVD/ZHBEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian band matrix, using divide and conquer
F08JCF	(SSTEVD/DSTEVD) All eigenvalues and optionally all eigenvectors of real symmetric tridiagonal matrix, using divide and conquer
F08LEF	(SGBBRD/DGBBRD) Reduction of real rectangular band matrix to upper bidiagonal form
F08LSF	(CGBBRD/ZGBBRD) Reduction of complex rectangular band matrix to upper bidiagonal form
F08UEF	(SSBGST/DSBGST) Reduction of real symmetric-definite banded generalized eigenproblem $Ax = \lambda Bx$ to standard form $Cy = \lambda y$, such that C has the same bandwidth as A
F08UFF	(SPBSTF/DPBSTF) Computes a split Cholesky factorization of real symmetric positive-definite band matrix A
F08USF	(CHBGST/ZHBGST) Reduction of complex Hermitian-definite banded generalized eigenproblem $Ax = \lambda Bx$ to standard form $Cy = \lambda y$, such that C has the same bandwidth as A
F08UTF	(CPBSTF/ZPBSTF) Computes a split Cholesky factorization of complex Hermitian positive-definite band matrix A
F11BDF	Real sparse nonsymmetric linear systems, set-up for F11BEF
F11BEF	Real sparse nonsymmetric linear systems, preconditioned RGMRES, CGS, Bi-CGSTAB or TFQMR method
F11BFF	Real sparse nonsymmetric linear systems, diagnostic for F11BEF
F11BRF	Complex sparse non-Hermitian linear systems, set-up for F11BSF
F11BSF	Complex sparse non-Hermitian linear systems, preconditioned RGMRES, CGS, Bi-CGSTAB or TFQMR method
F11BTF	Complex sparse non-Hermitian linear systems, diagnostic for F11BSF
F11DNF	Complex sparse non-Hermitian linear systems, incomplete LU factorization
F11DPF	Solution of complex linear system involving incomplete LU preconditioning matrix generated by F11DNF
F11DQF	Solution of complex sparse non-Hermitian linear system, RGMRES, CGS Bi-CGSTAB or TFQMR method, preconditioner computed by F11DNF (Black Box)
F11DRF	Solution of linear system involving preconditioning matrix generated by applying SSOR to complex sparse non-Hermitian matrix
F11DSF	Solution of complex sparse non-Hermitian linear system, RGMRES, CGS, Bi-CGSTAB or TFQMR method, Jacobi or SSOR preconditioner (Black Box)
F11JNF	Complex sparse Hermitian matrix, incomplete Cholesky factorization
F11JPF	Solution of complex linear system involving incomplete Cholesky preconditioning matrix generated by F11JNF
F11JQF	Solution of complex sparse Hermitian linear system, conjugate gradient/Lanczos method, preconditioner computed by F11JNF (Black Box)

F11JRF	Solution of linear system involving preconditioning matrix generated by applying SSOR to complex sparse Hermitian matrix
F11JSF	Solution of complex sparse Hermitian linear system, conjugate gradient/Lanczos method, Jacobi or SSOR preconditioner (Black Box)
F11XNF	Complex sparse non-Hermitian matrix vector multiply
F11XSF	Complex sparse Hermitian matrix vector multiply
F11ZNF	Complex sparse non-Hermitian matrix reorder routine
F11ZPF	Complex sparse Hermitian matrix reorder routine
G11CAF	Returns parameter estimates for the conditional analysis of stratified data
G12ZAF	Creates the risk sets associated with the Cox proportional hazards model for fixed covariates
H02CBF	Integer QP problem (dense)
H02CCF	Read optional parameter values for H02CBF from external file
H02CDF	Supply optional parameter values to H02CBF
H02CEF	Integer LP or QP problem (sparse)
H02CFF	Read optional parameter values for H02CEF from external file
H02CGF	Supply optional parameter values to H02CEF
M01EDF	Rearrange a vector according to given ranks, complex numbers
X04ACF	Open unit number for reading, writing or appending, and associate unit with named file
X04ADF	Close file associated with given unit number

3 Withdrawn Routines

The following routines have been withdrawn from the NAG Fortran Library at Mark 19. Warning of their withdrawal was included in the Mark 18 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
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)

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 advice on how to change a call to the old routine into a call to its recommended replacement.

The following routines will be withdrawn at Mark 20.

Routine Scheduled for Withdrawal	Recommended Replacement
E01SEF	E01SGF
E01SFF	E01SHF

The following routines have been superseded, but will not be withdrawn from the Library until Mark 21 at the earliest.

Superseded routine	**Recommended Replacement**
F11BAF	F11BDF
F11BBF	F11BEF
F11BCF	F11BFF

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 Fortran 77 context the constructs that prohibit thread safety are, potentially, DATA, SAVE, COMMON and EQUIVALENCE. This is because such constructs define data that will be shared by different threads, perhaps leading to unwanted interactions between them; for example, the possibility that one thread may be modifying the contents of a COMMON block at the same time as another thread is reading it. You are therefore advised to avoid the use of such constructs wherever possible within multi-threaded applications.

At Mark 19 of the NAG Library the use of unsafe constructs has been eliminated from the majority of routines in the Library, making them thread safe. However, there are some routines where complete removal of these constructs would seriously affect their interface design and usability. In such cases it makes more sense to keep the routines unchanged and give clear warnings in the documentation that care should be taken when calling such routines in a multi-threaded context. It should be noted that it is safe to call any NAG routine in one thread (only) of a multi-threaded application.

Some Library routines require you to supply a routine and to pass the name of the routine as an argument in the call to the Library routine. It is often the case that you need to supply your routine with more information than can be given via the interface argument list. In such circumstances it is usual to define a COMMON block containing the required data in the supplied routine (and also in the calling program). It is safe to do this only if no data referenced in the defined COMMON block is updated within the supplied routine (thus avoiding the possibility of simultaneous modification by different threads). Where separate calls are made to a Library routine by different threads and these calls require different data sets to be passed through COMMON blocks to user-supplied routines, these routines and the COMMON blocks defined within them should have different names.

You are also advised to check whether the Library routines you intend to call have equivalent reverse communication interfaces, which are designed specifically for problems where user-supplied routine interfaces are not flexible enough for a given problem; their use should eliminate the need to provide data through COMMON blocks.

The Library contains routines for setting the current error and advisory message unit numbers (X04AAF and X04ABF). These routines use the SAVE statement to retain the values of the current unit numbers between calls. It is therefore not advisable for different threads of a multi-threaded program to set the message unit numbers to different values. A consequence of this is that error or advisory messages output simultaneously may become garbled, and in any event there is no indication of which thread produces which message. You are therefore advised always to select the 'soft failure' mechanism without any error message (IFAIL = +1, see Section 2.3 of Essential Intorducation) on entry to each NAG routine called from a multi-threaded application; it is then essential that the value of IFAIL is tested on return to the application.

A related problem is that of multiple threads writing to or reading from files. You are advised to make different threads use different unit numbers for opening files and to give these files different names (perhaps by appending an index number to the file basename). The only alternative to this is for you to protect each write to a file or unit number; for example, by putting each WRITE statement in a critical region.

You are also advised to refer to the Users' Note for details of whether the Library has been compiled in a manner that facilitates the use of multiple threads. Please note however that at Mark 19 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	D02CJF	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
D02PCF	D02PDF	D02PVF	D02PWF	D02PXF	D02PYF
D02PZF	D02QFF	D02QGF	D02QXF	D02QYF	D02QZF
D02RAF	D02SAF	D02XJF	D02XKF	D02ZAF	D03PCF
D03PDF	D03PEF	D03PFF	D03PHF	D03PJF	D03PKF
D03PLF	D03PPF	D03PRF	D03PSF	D03PUF	D03PVF
D03PWF	D03PXF	D03PZF	D03RAF	D03RBF	D05BDF
D05BEF	E02GBF	E04DGF	E04DJF	E04DKF	E04MFF
E04MGF	E04MHF	E04MZF	E04NCF	E04NDF	E04NEF
E04NFF	E04NGF	E04NHF	E04NKF	E04NLF	E04NMF
E04UCF	E04UDF	E04UEF	E04UFF	E04UGF	E04UHF
E04UJF	E04UNF	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	G03FCF	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	H02CBF	H02CCF	H02CDF
H02CEF	H02CFF	H02CGF	X04AAF	X04ABF	

NAG Fortran Library, Mark 19 Library Contents

Chapter A00 – Library Identification

A00AAF	Prints details of the NAG Fortran Library implementation

Chapter A02 – Complex Arithmetic

A02AAF	Square root of complex number
A02ABF	Modulus of 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 system of nonlinear equations using function values only (reverse communication)
C05PBF	Solution of system of nonlinear equations using first derivatives (easy-to-use)
C05PCF	Solution of system of nonlinear equations using first derivatives (comprehensive)
C05PDF	Solution of system of nonlinear equations using first derivatives (reverse communication)
C05ZAF	Check user's routine for calculating first 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 one-dimensional real discrete Fourier transform, no extra workspace
C06EBF	Single one-dimensional Hermitian discrete Fourier transform, no extra workspace
C06ECF	Single one-dimensional complex discrete Fourier transform, no extra workspace
C06EKF	Circular convolution or correlation of two real vectors, no extra workspace
C06FAF	Single one-dimensional real discrete Fourier transform, extra workspace for greater speed
C06FBF	Single one-dimensional Hermitian discrete Fourier transform, extra workspace for greater speed
C06FCF	Single one-dimensional complex discrete Fourier transform, extra workspace for greater speed
C06FFF	One-dimensional 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 one-dimensional real discrete Fourier transforms
C06FQF	Multiple one-dimensional Hermitian discrete Fourier transforms
C06FRF	Multiple one-dimensional complex discrete Fourier transforms
C06FUF	Two-dimensional complex discrete Fourier transform
C06FXF	Three-dimensional 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
C06PAF	Single one-dimensional real and Hermitian complex discrete Fourier transform, using complex data format for Hermitian sequences
C06PCF	Single one-dimensional complex discrete Fourier transform, complex data format
C06PFF	One-dimensional complex discrete Fourier transform of multi-dimensional data (using complex data type)
C06PJF	Multi-dimensional complex discrete Fourier transform of multi-dimensional data (using complex data type)
C06PKF	Circular convolution or correlation of two complex vectors
C06PPF	Multiple one-dimensional real and Hermitian complex discrete Fourier transforms, using complex data format for Hermitian sequences
C06PQF	Multiple one-dimensional real and Hermitian complex discrete Fourier transforms, using complex data format for Hermitian sequences and sequences stored as columns
C06PRF	Multiple one-dimensional complex discrete Fourier transforms using complex data format
C06PSF	Multiple one-dimensional complex discrete Fourier transforms using complex data format and sequences stored as columns
C06PUF	Two-dimensional complex discrete Fourier transform, complex data format
C06PXF	Three-dimensional complex discrete Fourier transform, complex data format
C06RAF	Discrete sine transform (easy-to-use)
C06RBF	Discrete cosine transform (easy-to-use)
C06RCF	Discrete quarter-wave sine transform (easy-to-use)
C06RDF	Discrete quarter-wave cosine transform (easy-to-use)

Chapter D01 – Quadrature

D01AHF	One-dimensional quadrature, adaptive, finite interval, strategy due to Patterson, suitable for well-behaved integrands
D01AJF	One-dimensional quadrature, adaptive, finite interval, strategy due to Piessens and de Doncker, allowing for badly-behaved integrands
D01AKF	One-dimensional quadrature, adaptive, finite interval, method suitable for oscillating functions
D01ALF	One-dimensional quadrature, adaptive, finite interval, allowing for singularities at user-specified break-points
D01AMF	One-dimensional quadrature, adaptive, infinite or semi-infinite interval
D01ANF	One-dimensional quadrature, adaptive, finite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01APF	One-dimensional quadrature, adaptive, finite interval, weight function with end-point singularities of algebraico-logarithmic type
D01AQF	One-dimensional quadrature, adaptive, finite interval, weight function $1/(x - c)$, Cauchy principal value (Hilbert transform)
D01ARF	One-dimensional quadrature, non-adaptive, finite interval with provision for indefinite integrals
D01ASF	One-dimensional quadrature, adaptive, semi-infinite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01ATF	One-dimensional quadrature, adaptive, finite interval, variant of D01AJF efficient on vector machines
D01AUF	One-dimensional quadrature, adaptive, finite interval, variant of D01AKF efficient on vector machines
D01BAF	One-dimensional 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	One-dimensional quadrature, non-adaptive, finite interval
D01DAF	Two-dimensional 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	One-dimensional 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 first-order linear equations
D02KAF	Second-order Sturm–Liouville problem, regular system, finite range, eigenvalue only
D02KDF	Second-order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue only, user-specified break-points
D02KEF	Second-order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue and eigenfunction, user-specified break-points
D02LAF	Second-order ODEs, IVP, Runge–Kutta–Nystrom method
D02LXF	Second-order ODEs, IVP, set-up for D02LAF
D02LYF	Second-order ODEs, IVP, diagnostics for D02LAF
D02LZF	Second-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, two-dimensional arbitrary domain
D03EBF	Elliptic PDE, solution of finite difference equations by SIP, five-point two-dimensional molecule, iterate to convergence
D03ECF	Elliptic PDE, solution of finite difference equations by SIP for seven-point three-dimensional 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, three-dimensional Cartesian co-ordinates
D03MAF	Triangulation of 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 two-dimensional molecule, one iteration
D03UBF	Elliptic PDE, solution of finite difference equations by SIP, seven-point three-dimensional 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, second kind, split kernel
D05ABF	Linear non-singular Fredholm integral equation, second kind, smooth kernel
D05BAF	Nonlinear Volterra convolution equation, second kind
D05BDF	Nonlinear convolution Volterra–Abel equation, second kind, weakly singular
D05BEF	Nonlinear convolution Volterra–Abel equation, first 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 first 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 fitted bicubic spline at a vector of points
E02DFF	Evaluation of 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 two-dimensional 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 first derivative
E04CCF	Unconstrained minimum, simplex algorithm, function of several variables using function values only (comprehensive)
E04DGF	Unconstrained minimum, preconditioned conjugate gradient algorithm, function of several variables using first 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 first derivatives (comprehensive)
E04GDF	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using first derivatives (comprehensive)
E04GYF	Unconstrained minimum of a sum of squares, combined Gauss–Newton and quasi-Newton algorithm, using first derivatives (easy-to-use)

E04GZF	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using first derivatives (easy-to-use)
E04HCF	Check user's routine for calculating first derivatives of function
E04HDF	Check user's routine for calculating second derivatives of function
E04HEF	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm, using second derivatives (comprehensive)
E04HYF	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm, using second derivatives (easy-to-use)
E04JYF	Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using function values only (easy-to-use)
E04KDF	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (comprehensive)
E04KYF	Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using first derivatives (easy-to-use)
E04KZF	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (easy-to-use)
E04LBF	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first and second derivatives (comprehensive)
E04LYF	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first and second derivatives (easy-to-use)
E04MFF	LP problem (dense)
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 first 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 first derivatives (reverse communication, comprehensive)
E04UGF	NLP problem (sparse)
E04UHF	Read optional parameter values for E04UGF from external file
E04UJF	Supply optional parameter values to E04UGF
E04UNF	Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)
E04UQF	Read optional parameter values for E04UNF from external file
E04URF	Supply optional parameter values to E04UNF
E04XAF	Estimate (using numerical differentiation) gradient and/or Hessian of a function
E04YAF	Check user's routine for calculating Jacobian of first 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 first 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 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

Chapter F02 – Eigenvalues and Eigenvectors

Chapter F03 – Determinants

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 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 problem
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)
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 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	(SROTG/DROTG) Generate real plane rotation
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	(SDOT/DDOT) Dot product of two real vectors
F06ECF	(SAXPY/DAXPY) Add scalar times real vector to real vector
F06EDF	(SSCAL/DSCAL) Multiply real vector by scalar
F06EFF	(SCOPY/DCOPY) Copy real vector
F06EGF	(SSWAP/DSWAP) Swap two real vectors
F06EJF	(SNRM2/DNRM2) Compute Euclidean norm of real vector
F06EKF	(SASUM/DASUM) Sum absolute values of real vector elements
F06EPF	(SROT/DROT) Apply real plane rotation
F06ERF	(SDOTI/DDOTI) Dot product of two real sparse vectors
F06ETF	(SAXPYI/DAXPYI) Add scalar times real sparse vector to real sparse vector
F06EUF	(SGTHR/DGTHR) Gather real sparse vector
F06EVF	(SGTHRZ/DGTHRZ) Gather and set to zero real sparse vector
F06EWF	(SSCTR/DSCTR) Scatter real sparse vector
F06EXF	(SROTI/DROTI) Apply plane rotation to two real sparse vectors
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	(CDOTU/ZDOTU) Dot product of two complex vectors, unconjugated
F06GBF	(CDOTC/ZDOTC) Dot product of two complex vectors, conjugated
F06GCF	(CAXPY/ZAXPY) Add scalar times complex vector to complex vector
F06GDF	(CSCAL/ZSCAL) Multiply complex vector by complex scalar
F06GFF	(CCOPY/ZCOPY) Copy complex vector
F06GGF	(CSWAP/ZSWAP) Swap two complex vectors
F06GRF	(CDOTUI/ZDOTUI) Dot product of two complex sparse vector, unconjugated
F06GSF	(CDOTCI/ZDOTCI) Dot product of two complex sparse vector, conjugated
F06GTF	(CAXPYI/ZAXPYI) Add scalar times complex sparse vector to complex sparse vector
F06GUF	(CGTHR/ZGTHR) Gather complex sparse vector
F06GVF	(CGTHRZ/ZGTHRZ) Gather and set to zero complex sparse vector
F06GWF	(CSCTR/ZSCTR) Scatter complex sparse vector

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	(CSSCAL/ZDSCAL) Multiply complex vector by real scalar
F06JJF	(SCNRM2/DZNRM2) Compute Euclidean norm of complex vector
F06JKF	(SCASUM/DZASUM) Sum absolute values of complex vector elements
F06JLF	(ISAMAX/IDAMAX) Index, real vector element with largest absolute value
F06JMF	(ICAMAX/IZAMAX) Index, complex vector element with largest absolute value
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	(SGEMV/DGEMV) Matrix-vector product, real rectangular matrix
F06PBF	(SGBMV/DGBMV) Matrix-vector product, real rectangular band matrix
F06PCF	(SSYMV/DSYMV) Matrix-vector product, real symmetric matrix
F06PDF	(SSBMV/DSBMV) Matrix-vector product, real symmetric band matrix
F06PEF	(SSPMV/DSPMV) Matrix-vector product, real symmetric packed matrix
F06PFF	(STRMV/DTRMV) Matrix-vector product, real triangular matrix
F06PGF	(STBMV/DTBMV) Matrix-vector product, real triangular band matrix
F06PHF	(STPMV/DTPMV) Matrix-vector product, real triangular packed matrix
F06PJF	(STRSV/DTRSV) System of equations, real triangular matrix
F06PKF	(STBSV/DTBSV) System of equations, real triangular band matrix
F06PLF	(STPSV/DTPSV) System of equations, real triangular packed matrix
F06PMF	(SGER/DGER) Rank-1 update, real rectangular matrix
F06PPF	(SSYR/DSYR) Rank-1 update, real symmetric matrix
F06PQF	(SSPR/DSPR) Rank-1 update, real symmetric packed matrix
F06PRF	(SSYR2/DSYR2) Rank-2 update, real symmetric matrix
F06PSF	(SSPR2/DSPR2) Rank-2 update, real symmetric packed matrix
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 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	(CGEMV/ZGEMV) Matrix-vector product, complex rectangular matrix
F06SBF	(CGBMV/ZGBMV) Matrix-vector product, complex rectangular band matrix
F06SCF	(CHEMV/ZHEMV) Matrix-vector product, complex Hermitian matrix
F06SDF	(CHBMV/ZHBMV) Matrix-vector product, complex Hermitian band matrix
F06SEF	(CHPMV/ZHPMV) Matrix-vector product, complex Hermitian packed matrix
F06SFF	(CTRMV/ZTRMV) Matrix-vector product, complex triangular matrix
F06SGF	(CTBMV/ZTBMV) Matrix-vector product, complex triangular band matrix
F06SHF	(CTPMV/ZTPMV) Matrix-vector product, complex triangular packed matrix
F06SJF	(CTRSV/ZTRSV) System of equations, complex triangular matrix
F06SKF	(CTBSV/ZTBSV) System of equations, complex triangular band matrix
F06SLF	(CTPSV/ZTPSV) System of equations, complex triangular packed matrix
F06SMF	(CGERU/ZGERU) Rank-1 update, complex rectangular matrix, unconjugated vector
F06SNF	(CGERC/ZGERC) Rank-1 update, complex rectangular matrix, conjugated vector
F06SPF	(CHER/ZHER) Rank-1 update, complex Hermitian matrix
F06SQF	(CHPR/ZHPR) Rank-1 update, complex Hermitian packed matrix
F06SRF	(CHER2/ZHER2) Rank-2 update, complex Hermitian matrix
F06SSF	(CHPR2/ZHPR2) Rank-2 update, complex Hermitian packed matrix
F06TFF	Matrix copy, complex rectangular or trapezoidal matrix
F06THF	Matrix initialisation, complex rectangular matrix
F06TMF	Unitary similarity transformation of 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	$QRxk$ 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 (SGEMM/DGEMM) Matrix-matrix product, two real rectangular matrices
F06YCF (SSYMM/DSYMM) Matrix-matrix product, one real symmetric matrix, one real rectangular matrix
F06YFF (STRMM/DTRMM) Matrix-matrix product, one real triangular matrix, one real rectangular matrix
F06YJF (STRSM/DTRSM) Solves system of equations with multiple right-hand sides, real triangular coefficient matrix
F06YPF (SSYRK/DSYRK) Rank-k update of real symmetric matrix
F06YRF (SSYR2K/DSYR2K) Rank-$2k$ update of real symmetric matrix
F06ZAF (CGEMM/ZGEMM) Matrix-matrix product, two complex rectangular matrices
F06ZCF (CHEMM/ZHEMM) Matrix-matrix product, one complex Hermitian matrix, one complex rectangular matrix
F06ZFF (CTRMM/ZTRMM) Matrix-matrix product, one complex triangular matrix, one complex rectangular matrix
F06ZJF (CTRSM/ZTRSM) Solves system of equations with multiple right-hand sides, complex triangular coefficient matrix
F06ZPF (CHERK/ZHERK) Rank-k update of complex Hermitian matrix
F06ZRF (CHER2K/ZHER2K) Rank-$2k$ update of complex Hermitian matrix
F06ZTF (CSYMM/ZSYMM) Matrix-matrix product, one complex symmetric matrix, one complex rectangular matrix
F06ZUF (CSYRK/ZSYRK) Rank-k update of complex symmetric matrix
F06ZWF (CSYR2K/ZHER2K) Rank-$2k$ update of complex symmetric matrix

Chapter F07 – Linear Equations (LAPACK)

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

F07FGF	(SPOCON/DPOCON) Estimate condition number of real symmetric positive-definite matrix, matrix already factorized by F07FDF
F07FHF	(SPORFS/DPORFS) Refined solution with error bounds of real symmetric positive-definite system of linear equations, multiple right-hand sides
F07FJF	(SPOTRI/DPOTRI) Inverse of real symmetric positive-definite matrix, matrix already factorized by F07FDF
F07FRF	(CPOTRF/ZPOTRF) Cholesky factorization of complex Hermitian positive-definite matrix
F07FSF	(CPOTRS/ZPOTRS) Solution of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07FRF
F07FUF	(CPOCON/ZPOCON) Estimate condition number of complex Hermitian positive-definite matrix, matrix already factorized by F07FRF
F07FVF	(CPORFS/ZPORFS) Refined solution with error bounds of complex Hermitian positive-definite system of linear equations, multiple right-hand sides
F07FWF	(CPOTRI/ZPOTRI) Inverse of complex Hermitian positive-definite matrix, matrix already factorized by F07FRF
F07GDF	(SPPTRF/DPPTRF) Cholesky factorization of real symmetric positive-definite matrix, packed storage
F07GEF	(SPPTRS/DPPTRS) Solution of real symmetric positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07GDF, packed storage
F07GGF	(SPPCON/DPPCON) Estimate condition number of real symmetric positive-definite matrix, matrix already factorized by F07GDF, packed storage
F07GHF	(SPPRFS/DPPRFS) Refined solution with error bounds of real symmetric positive-definite system of linear equations, multiple right-hand sides, packed storage
F07GJF	(SPPTRI/DPPTRI) Inverse of real symmetric positive-definite matrix, matrix already factorized by F07GDF, packed storage
F07GRF	(CPPTRF/ZPPTRF) Cholesky factorization of complex Hermitian positive-definite matrix, packed storage
F07GSF	(CPPTRS/ZPPTRS) Solution of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07GRF, packed storage
F07GUF	(CPPCON/ZPPCON) Estimate condition number of complex Hermitian positive-definite matrix, matrix already factorized by F07GRF, packed storage
F07GVF	(CPPRFS/ZPPRFS) Refined solution with error bounds of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, packed storage
F07GWF	(CPPTRI/ZPPTRI) Inverse of complex Hermitian positive-definite matrix, matrix already factorized by F07GRF, packed storage
F07HDF	(SPBTRF/DPBTRF) Cholesky factorization of real symmetric positive-definite band matrix
F07HEF	(SPBTRS/DPBTRS) Solution of real symmetric positive-definite band system of linear equations, multiple right-hand sides, matrix already factorized by F07HDF
F07HGF	(SPBCON/DPBCON) Estimate condition number of real symmetric positive-definite band matrix, matrix already factorized by F07HDF
F07HHF	(SPBRFS/DPBRFS) Refined solution with error bounds of real symmetric positive-definite band system of linear equations, multiple right-hand sides
F07HRF	(CPBTRF/ZPBTRF) Cholesky factorization of complex Hermitian positive-definite band matrix
F07HSF	(CPBTRS/ZPBTRS) Solution of complex Hermitian positive-definite band system of linear equations, multiple right-hand sides, matrix already factorized by F07HRF
F07HUF	(CPBCON/ZPBCON) Estimate condition number of complex Hermitian positive-definite band matrix, matrix already factorized by F07HRF
F07HVF	(CPBRFS/ZPBRFS) Refined solution with error bounds of complex Hermitian positive-definite band system of linear equations, multiple right-hand sides
F07MDF	(SSYTRF/DSYTRF) Bunch–Kaufman factorization of real symmetric indefinite matrix
F07MEF	(SSYTRS/DSYTRS) Solution of real symmetric indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07MDF
F07MGF	(SSYCON/DSYCON) Estimate condition number of real symmetric indefinite matrix, matrix already factorized by F07MDF
F07MHF	(SSYRFS/DSYRFS) Refined solution with error bounds of real symmetric indefinite system of linear equations, multiple right-hand sides

F07MJF	(SSYTRI/DSYTRI) Inverse of real symmetric indefinite matrix, matrix already factorized by F07MDF
F07MRF	(CHETRF/ZHETRF) Bunch–Kaufman factorization of complex Hermitian indefinite matrix
F07MSF	(CHETRS/ZHETRS) Solution of complex Hermitian indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07MRF
F07MUF	(CHECON/ZHECON) Estimate condition number of complex Hermitian indefinite matrix, matrix already factorized by F07MRF
F07MVF	(CHERFS/ZHERFS) Refined solution with error bounds of complex Hermitian indefinite system of linear equations, multiple right-hand sides
F07MWF	(CHETRI/ZHETRI) Inverse of complex Hermitian indefinite matrix, matrix already factorized by F07MRF
F07NRF	(CSYTRF/ZSYTRF) Bunch–Kaufman factorization of complex symmetric matrix
F07NSF	(CSYTRS/ZSYTRS) Solution of complex symmetric system of linear equations, multiple right-hand sides, matrix already factorized by F07NRF
F07NUF	(CSYCON/ZSYCON) Estimate condition number of complex symmetric matrix, matrix already factorized by F07NRF
F07NVF	(CSYRFS/ZSYRFS) Refined solution with error bounds of complex symmetric system of linear equations, multiple right-hand sides
F07NWF	(CSYTRI/ZSYTRI) Inverse of complex symmetric matrix, matrix already factorized by F07NRF
F07PDF	(SSPTRF/DSPTRF) Bunch–Kaufman factorization of real symmetric indefinite matrix, packed storage
F07PEF	(SSPTRS/DSPTRS) Solution of real symmetric indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07PDF, packed storage
F07PGF	(SSPCON/DSPCON) Estimate condition number of real symmetric indefinite matrix, matrix already factorized by F07PDF, packed storage
F07PHF	(SSPRFS/DSPRFS) Refined solution with error bounds of real symmetric indefinite system of linear equations, multiple right-hand sides, packed storage
F07PJF	(SSPTRI/DSPTRI) Inverse of real symmetric indefinite matrix, matrix already factorized by F07PDF, packed storage
F07PRF	(CHPTRF/ZHPTRF) Bunch–Kaufman factorization of complex Hermitian indefinite matrix, packed storage
F07PSF	(CHPTRS/ZHPTRS) Solution of complex Hermitian indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07PRF, packed storage
F07PUF	(CHPCON/ZHPCON) Estimate condition number of complex Hermitian indefinite matrix, matrix already factorized by F07PRF, packed storage
F07PVF	(CHPRFS/ZHPRFS) Refined solution with error bounds of complex Hermitian indefinite system of linear equations, multiple right-hand sides, packed storage
F07PWF	(CHPTRI/ZHPTRI) Inverse of complex Hermitian indefinite matrix, matrix already factorized by F07PRF, packed storage
F07QRF	(CSPTRF/ZSPTRF) Bunch–Kaufman factorization of complex symmetric matrix, packed storage
F07QSF	(CSPTRS/ZSPTRS) Solution of complex symmetric system of linear equations, multiple right-hand sides, matrix already factorized by F07QRF, packed storage
F07QUF	(CSPCON/ZSPCON) Estimate condition number of complex symmetric matrix, matrix already factorized by F07QRF, packed storage
F07QVF	(CSPRFS/ZSPRFS) Refined solution with error bounds of complex symmetric system of linear equations, multiple right-hand sides, packed storage
F07QWF	(CSPTRI/ZSPTRI) Inverse of complex symmetric matrix, matrix already factorized by F07QRF, packed storage
F07TEF	(STRTRS/DTRTRS) Solution of real triangular system of linear equations, multiple right-hand sides
F07TGF	(STRCON/DTRCON) Estimate condition number of real triangular matrix
F07THF	(STRRFS/DTRRFS) Error bounds for solution of real triangular system of linear equations, multiple right-hand sides
F07TJF	(STRTRI/DTRTRI) Inverse of real triangular matrix
F07TSF	(CTRTRS/ZTRTRS) Solution of complex triangular system of linear equations, multiple right-hand sides

F07TUF (CTRCON/ZTRCON) Estimate condition number of complex triangular matrix
F07TVF (CTRRFS/ZTRRFS) Error bounds for solution of complex triangular system of linear equations, multiple right-hand sides
F07TWF (CTRTRI/ZTRTRI) Inverse of complex triangular matrix
F07UEF (STPTRS/DTPTRS) Solution of real triangular system of linear equations, multiple right-hand sides, packed storage
F07UGF (STPCON/DTPCON) Estimate condition number of real triangular matrix, packed storage
F07UHF (STPRFS/DTPRFS) Error bounds for solution of real triangular system of linear equations, multiple right-hand sides, packed storage
F07UJF (STPTRI/DTPTRI) Inverse of real triangular matrix, packed storage
F07USF (CTPTRS/ZTPTRS) Solution of complex triangular system of linear equations, multiple right-hand sides, packed storage
F07UUF (CTPCON/ZTPCON) Estimate condition number of complex triangular matrix, packed storage
F07UVF (CTPRFS/ZTPRFS) Error bounds for solution of complex triangular system of linear equations, multiple right-hand sides, packed storage
F07UWF (CTPTRI/ZTPTRI) Inverse of complex triangular matrix, packed storage
F07VEF (STBTRS/DTBTRS) Solution of real band triangular system of linear equations, multiple right-hand sides
F07VGF (STBCON/DTBCON) Estimate condition number of real band triangular matrix
F07VHF (STBRFS/DTBRFS) Error bounds for solution of real band triangular system of linear equations, multiple right-hand sides
F07VSF (CTBTRS/ZTBTRS) Solution of complex band triangular system of linear equations, multiple right-hand sides
F07VUF (CTBCON/ZTBCON) Estimate condition number of complex band triangular matrix
F07VVF (CTBRFS/ZTBRFS) Error bounds for solution of complex band triangular system of linear equations, multiple right-hand sides

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

F08AEF (SGEQRF/DGEQRF) QR factorization of real general rectangular matrix
F08AFF (SORGQR/DORGQR) Form all or part of orthogonal Q from QR factorization determined by F08AEF or F08BEF
F08AGF (SORMQR/DORMQR) Apply orthogonal transformation determined by F08AEF or F08BEF
F08AHF (SGELQF/DGELQF) LQ factorization of real general rectangular matrix
F08AJF (SORGLQ/DORGLQ) Form all or part of orthogonal Q from LQ factorization determined by F08AHF
F08AKF (SORMLQ/DORMLQ) Apply orthogonal transformation determined by F08AHF
F08ASF (CGEQRF/ZGEQRF) QR factorization of complex general rectangular matrix
F08ATF (CUNGQR/ZUNGQR) Form all or part of unitary Q from QR factorization determined by F08ASF or F08BSF
F08AUF (CUNMQR/ZUNMQR) Apply unitary transformation determined by F08ASF or F08BSF
F08AVF (CGELQF/ZGELQF) LQ factorization of complex general rectangular matrix
F08AWF (CUNGLQ/ZUNGLQ) Form all or part of unitary Q from LQ factorization determined by F08AVF
F08AXF (CUNMLQ/ZUNMLQ) Apply unitary transformation determined by F08AVF
F08BEF (SGEQPF/DGEQPF) QR factorization of real general rectangular matrix with column pivoting
F08BSF (CGEQPF/ZGEQPF) QR factorization of complex general rectangular matrix with column pivoting
F08FCF (SSYEVD/DSYEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, using divide and conquer
F08FEF (SSYTRD/DSYTRD) Orthogonal reduction of real symmetric matrix to symmetric tridiagonal form
F08FFF (SORGTR/DORGTR) Generate orthogonal transformation matrix from reduction to tridiagonal form determined by F08FEF
F08FGF (SORMTR/DORMTR) Apply orthogonal transformation determined by F08FEF
F08FQF (CHEEVD/ZHEEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, using divide and conquer
F08FSF (CHETRD/ZHETRD) Unitary reduction of complex Hermitian matrix to real symmetric tridiagonal form

F08FTF	(CUNGTR/ZUNGTR) Generate unitary transformation matrix from reduction to tridiagonal form determined by F08FSF
F08FUF	(CUNMTR/ZUNMTR) Apply unitary transformation matrix determined by F08FSF
F08GCF	(SSPEVD/DSPEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, packed storage, using divide and conquer
F08GEF	(SSPTRD/DSPTRD) Orthogonal reduction of real symmetric matrix to symmetric tridiagonal form, packed storage
F08GFF	(SOPGTR/DOPGTR) Generate orthogonal transformation matrix from reduction to tridiagonal form determined by F08GEF
F08GGF	(SOPMTR/DOPMTR) Apply orthogonal transformation determined by F08GEF
F08GQF	(CHPEVD/ZHPEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, packed storage, using divide and conquer
F08GSF	(CHPTRD/ZHPTRD) Unitary reduction of complex Hermitian matrix to real symmetric tridiagonal form, packed storage
F08GTF	(CUPGTR/ZUPGTR) Generate unitary transformation matrix from reduction to tridiagonal form determined by F08GSF
F08GUF	(CUPMTR/ZUPMTR) Apply unitary transformation matrix determined by F08GSF
F08HCF	(SSBEVD/DSBEVD) All eigenvalues and optionally all eigenvectors of real symmetric band matrix, using divide and conquer
F08HEF	(SSBTRD/DSBTRD) Orthogonal reduction of real symmetric band matrix to symmetric tridiagonal form
F08HQF	(CHBEVD/ZHBEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian band matrix, using divide and conquer
F08HSF	(CHBTRD/ZHBTRD) Unitary reduction of complex Hermitian band matrix to real symmetric tridiagonal form
F08JCF	(SSTEVD/DSTEVD) All eigenvalues and optionally all eigenvectors of real symmetric tridiagonal matrix, using divide and conquer
F08JEF	(SSTEQR/DSTEQR) All eigenvalues and eigenvectors of real symmetric tridiagonal matrix, reduced from real symmetric matrix using implicit QL or QR
F08JFF	(SSTERF/DSTERF) All eigenvalues of real symmetric tridiagonal matrix, root-free variant of QL or QR
F08JGF	(SPTEQR/DPTEQR) All eigenvalues and eigenvectors of real symmetric positive-definite tridiagonal matrix, reduced from real symmetric positive-definite matrix
F08JJF	(SSTEBZ/DSTEBZ) Selected eigenvalues of real symmetric tridiagonal matrix by bisection
F08JKF	(SSTEIN/DSTEIN) Selected eigenvectors of real symmetric tridiagonal matrix by inverse iteration, storing eigenvectors in real array
F08JSF	(CSTEQR/ZSTEQR) All eigenvalues and eigenvectors of real symmetric tridiagonal matrix, reduced from complex Hermitian matrix, using implicit QL or QR
F08JUF	(CPTEQR/ZPTEQR) All eigenvalues and eigenvectors of real symmetric positive-definite tridiagonal matrix, reduced from complex Hermitian positive-definite matrix
F08JXF	(CSTEIN/ZSTEIN) Selected eigenvectors of real symmetric tridiagonal matrix by inverse iteration, storing eigenvectors in complex array
F08KEF	(SGEBRD/DGEBRD) Orthogonal reduction of real general rectangular matrix to bidiagonal form
F08KFF	(SORGBR/DORGBR) Generate orthogonal transformation matrices from reduction to bidiagonal form determined by F08KEF
F08KGF	(SORMBR/DORMBR) Apply orthogonal transformations from reduction to bidiagonal form determined by F08KEF
F08KSF	(CGEBRD/ZGEBRD) Unitary reduction of complex general rectangular matrix to bidiagonal form
F08KTF	(CUNGBR/ZUNGBR) Generate unitary transformation matrices from reduction to bidiagonal form determined by F08KSF
F08KUF	(CUNMBR/ZUNMBR) Apply unitary transformations from reduction to bidiagonal form determined by F08KSF
F08LEF	(SGBBRD/DGBBRD) Reduction of real rectangular band matrix to upper bidiagonal form
F08LSF	(CGBBRD/ZGBBRD) Reduction of complex rectangular band matrix to upper bidiagonal form
F08MEF	(SBDSQR/DBDSQR) SVD of real bidiagonal matrix reduced from real general matrix
F08MSF	(CBDSQR/ZBDSQR) SVD of real bidiagonal matrix reduced from complex general matrix

F08NEF	(SGEHRD/DGEHRD) Orthogonal reduction of real general matrix to upper Hessenberg form
F08NFF	(SORGHR/DORGHR) Generate orthogonal transformation matrix from reduction to Hessenberg form determined by F08NEF
F08NGF	(SORMHR/DORMHR) Apply orthogonal transformation matrix from reduction to Hessenberg form determined by F08NEF
F08NHF	(SGEBAL/DGEBAL) Balance real general matrix
F08NJF	(SGEBAK/DGEBAK) Transform eigenvectors of real balanced matrix to those of original matrix supplied to F08NHF
F08NSF	(CGEHRD/ZGEHRD) Unitary reduction of complex general matrix to upper Hessenberg form
F08NTF	(CUNGHR/ZUNGHR) Generate unitary transformation matrix from reduction to Hessenberg form determined by F08NSF
F08NUF	(CUNMHR/ZUNMHR) Apply unitary transformation matrix from reduction to Hessenberg form determined by F08NSF
F08NVF	(CGEBAL/ZGEBAL) Balance complex general matrix
F08NWF	(CGEBAK/ZGEBAK) Transform eigenvectors of complex balanced matrix to those of original matrix supplied to F08NVF
F08PEF	(SHSEQR/DHSEQR) Eigenvalues and Schur factorization of real upper Hessenberg matrix reduced from real general matrix
F08PKF	(SHSEIN/DHSEIN) Selected right and/or left eigenvectors of real upper Hessenberg matrix by inverse iteration
F08PSF	(CHSEQR/ZHSEQR) Eigenvalues and Schur factorization of complex upper Hessenberg matrix reduced from complex general matrix
F08PXF	(CHSEIN/ZHSEIN) Selected right and/or left eigenvectors of complex upper Hessenberg matrix by inverse iteration
F08QFF	(STREXC/DTREXC) Reorder Schur factorization of real matrix using orthogonal similarity transformation
F08QGF	(STRSEN/DTRSEN) Reorder Schur factorization of real matrix, form orthonormal basis of right invariant subspace for selected eigenvalues, with estimates of sensitivities
F08QHF	(STRSYL/DTRSYL) Solve real Sylvester matrix equation $AX + XB = C$, A and B are upper quasi-triangular or transposes
F08QKF	(STREVC/DTREVC) Left and right eigenvectors of real upper quasi-triangular matrix
F08QLF	(STRSNA/DTRSNA) Estimates of sensitivities of selected eigenvalues and eigenvectors of real upper quasi-triangular matrix
F08QTF	(CTREXC/ZTREXC) Reorder Schur factorization of complex matrix using unitary similarity transformation
F08QUF	(CTRSEN/ZTRSEN) Reorder Schur factorization of complex matrix, form orthonormal basis of right invariant subspace for selected eigenvalues, with estimates of sensitivities
F08QVF	(CTRSYL/ZTRSYL) Solve complex Sylvester matrix equation $AX + XB = C$, A and B are upper triangular or conjugate-transposes
F08QXF	(CTREVC/ZTREVC) Left and right eigenvectors of complex upper triangular matrix
F08QYF	(CTRSNA/ZTRSNA) Estimates of sensitivities of selected eigenvalues and eigenvectors of complex upper triangular matrix
F08SEF	(SSYGST/DSYGST) Reduction to standard form of real symmetric-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$, B factorized by F07FDF
F08SSF	(CHEGST/ZHEGST) Reduction to standard form of complex Hermitian-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$, B factorized by F07FRF
F08TEF	(SSPGST/DSPGST) 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
F08TSF	(CHPGST/ZHPGST) 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
F08UEF	(SSBGST/DSBGST) Reduction of real symmetric-definite banded generalized eigenproblem $Ax = \lambda Bx$ to standard form $Cy = \lambda y$, such that C has the same bandwidth as A
F08UFF	(SPBSTF/DPBSTF) Computes a split Cholesky factorization of real symmetric positive-definite band matrix A
F08USF	(CHBGST/ZHBGST) Reduction of complex Hermitian-definite banded generalized eigenproblem $Ax = \lambda Bx$ to standard form $Cy = \lambda y$, such that C has the same bandwidth as A
F08UTF	(CPBSTF/ZPBSTF) Computes a split Cholesky factorization of complex Hermitian positive-definite band matrix A

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
F11BDF	Real sparse nonsymmetric linear systems, set-up for F11BEF
F11BEF	Real sparse nonsymmetric linear systems, preconditioned RGMRES, CGS, Bi-CGSTAB or TFQMR method
F11BFF	Real sparse nonsymmetric linear systems, diagnostic for F11BEF
F11BRF	Complex sparse non-Hermitian linear systems, set-up for F11BSF
F11BSF	Complex sparse non-Hermitian linear systems, preconditioned RGMRES, CGS, Bi-CGSTAB or TFQMR method
F11BTF	Complex sparse non-Hermitian linear systems, diagnostic for F11BSF
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 preconditioning matrix generated 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)
F11DNF	Complex sparse non-Hermitian linear systems, incomplete *LU* factorization
F11DPF	Solution of complex linear system involving incomplete *LU* preconditioning matrix generated by F11DNF
F11DQF	Solution of complex sparse non-Hermitian linear system, RGMRES, CGS, Bi-CGSTAB or TFQMR method, preconditioner computed by F11DNF (Black Box)
F11DRF	Solution of linear system involving preconditioning matrix generated by applying SSOR to complex sparse non-Hermitian matrix
F11DSF	Solution of complex sparse non-Hermitian linear system, RGMRES, CGS, Bi-CGSTAB or TFQMR method, Jacobi or SSOR preconditioner (Black Box)
F11GAF	Real sparse symmetric linear systems, set-up for F11GBF
F11GBF	Real sparse symmetric linear systems, preconditioned 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 Cholesky 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)
F11JNF	Complex sparse Hermitian matrix, incomplete Cholesky factorization
F11JPF	Solution of complex linear system involving incomplete Cholesky preconditioning matrix generated by F11JNF
F11JQF	Solution of complex sparse Hermitian linear system, conjugate gradient/Lanczos method, preconditioner computed by F11JNF (Black Box)
F11JRF	Solution of linear system involving preconditioning matrix generated by applying SSOR to complex sparse Hermitian matrix
F11JSF	Solution of complex sparse Hermitian 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
F11XNF	Complex sparse non-Hermitian matrix vector multiply
F11XSF	Complex sparse Hermitian matrix vector multiply
F11ZAF	Real sparse nonsymmetric matrix reorder routine
F11ZBF	Real sparse symmetric matrix reorder routine
F11ZNF	Complex sparse non-Hermitian matrix reorder routine
F11ZPF	Complex sparse Hermitian 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 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, log-normal 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 a 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, one-sample
G07EBF	Robust confidence intervals, two-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
G11CAF	Returns parameter estimates for the conditional analysis of stratified data
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
G12ZAF	Creates the risk sets associated with the Cox proportional hazards model for fixed covariates

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 LP problem (dense)
H02BFF	Interpret MPSX data file defining IP or LP problem, optimize and print solution
H02BUF	Convert MPSX data file defining IP or LP problem to format required by H02BBF or E04MFF
H02BVF	Print IP or LP solutions with user specified names for rows and columns
H02BZF	Integer programming solution, supplies further information on solution obtained by H02BBF
H02CBF	Integer QP problem (dense)
H02CCF	Read optional parameter values for H02CBF from external file
H02CDF	Supply optional parameter values to H02CBF
H02CEF	Integer LP or QP problem (sparse)
H02CFF	Read optional parameter values for H02CEF from external file
H02CGF	Supply optional parameter values to H02CEF
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
M01EDF	Rearrange a vector according to given ranks, complex numbers
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	$\mathrm{arctanh} x$
S11ABF	$\mathrm{arcsinh} x$
S11ACF	$\mathrm{arccosh} x$
S13AAF	Exponential integral $E_1(x)$
S13ACF	Cosine integral $\mathrm{Ci}(x)$
S13ADF	Sine integral $\mathrm{Si}(x)$
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 $\mathrm{erfc}(x)$
S15AEF	Error function $\mathrm{erf}(x)$
S15AFF	Dawson's integral
S15DDF	Scaled complex complement of error function, $\exp(-z^2)\mathrm{erfc}(-iz)$
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 $\mathrm{Ai}(x)$
S17AHF	Airy function $\mathrm{Bi}(x)$
S17AJF	Airy function $\mathrm{Ai}'(x)$
S17AKF	Airy function $\mathrm{Bi}'(x)$
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 $\mathrm{Ai}(z)$ and $\mathrm{Ai}'(z)$, complex z
S17DHF	Airy functions $\mathrm{Bi}(z)$ and $\mathrm{Bi}'(z)$, 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 $e^{-\lvert x\rvert} I_0(x)$
S18CFF	Modified Bessel function $e^{-\lvert x\rvert} I_1(x)$
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 $\mathrm{ber}\ x$
S19ABF	Kelvin function $\mathrm{bei}\ x$

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 permissible 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
X04ACF	Open unit number for reading, writing or appending, and associate unit with named file
X04ADF	Close file associated with given unit number
X04BAF	Write formatted record to external file
X04BBF	Read formatted record from external file
X04CAF	Print real general matrix (easy-to-use)
X04CBF	Print real general matrix (comprehensive)
X04CCF	Print real packed triangular matrix (easy-to-use)
X04CDF	Print real packed triangular matrix (comprehensive)
X04CEF	Print real packed banded matrix (easy-to-use)
X04CFF	Print real packed banded matrix (comprehensive)
X04DAF	Print complex general matrix (easy-to-use)
X04DBF	Print complex general matrix (comprehensive)
X04DCF	Print complex packed triangular matrix (easy-to-use)
X04DDF	Print complex packed triangular matrix (comprehensive)
X04DEF	Print complex packed banded matrix (easy-to-use)
X04DFF	Print complex packed banded matrix (comprehensive)
X04EAF	Print integer matrix (easy-to-use)
X04EBF	Print integer matrix (comprehensive)

Chapter X05 – Date and Time Utilities

X05AAF	Return date and time as an array of integers
X05ABF	Convert array of integers representing date and time to character string
X05ACF	Compare two character strings representing date and time
X05BAF	Return the CPU time

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. Copies of these documents may be obtained from NAG upon request. 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
D02XAF	18	D02PXF and associated D02P routines
D02XBF	18	D02PXF and associated D02P routines
D02XGF	14	D02QZF
D02XHF	14	D02QZF
D02YAF	18	D02PDF and associated D02P routines
D03PAF	17	D03PCF

Withdrawn Routine	Mark of Withdrawal	Recommended Replacement
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	19	E04FYF
E04GAF	8	E04GBF, E04GCF, E04GDF or E04GEF
E04GCF	19	E04GYF
E04HAF	7	E04UCF
E04HBF	16	no longer required
E04HFF	19	E04HYF
E04JAF	19	E04JYF
E04JBF	16	E04UCF
E04KAF	19	E04KYF
E04KBF	16	E04UCF
E04KCF	19	E04KZF
E04LAF	19	E04LYF
E04MBF	18	E04MFF
E04NAF	18	E04NFF
E04UAF	13	E04UCF
E04UPF	19	E04UNF
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)

Withdrawn Routine	Mark of Withdrawal	Recommended Replacement
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)
F01MAF	19	F11JAF
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)

Withdrawn Routine	Mark of Withdrawal	Recommended Replacement
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)
F02BBF	19	F02FCF
F02BCF	19	F02ECF
F02BDF	19	F02GCF
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)

Withdrawn Routine	Mark of Withdrawal	Recommended Replacement
F04LDF	18	F07BEF (SGBTRS/DGBTRS)
F04MAF	19	F11JCF
F04MBF	19	F11GAF, F11GBF and F11GCF (or F11JCF or F11JEF)
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

Withdrawn Routine	Mark of Withdrawal	Recommended Replacement
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

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. Copies of the documents for withdrawn routines may be obtained from NAG upon request.

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 $(2, \mathrm{N}+1)$ 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 $(4 * (\mathrm{N}+1))$.

C02AEF
Withdrawn at Mark 16

```
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 $(2 * (\mathrm{N}+1))$.

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,32*N,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 in length. The integer parameter IRELAB (which can take values 0, 1 or 2) is catered 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 routine 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 or 2) is catered 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 routine 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

Existing programs 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 precise details of a replacement call. Please consult the appropriate routine documents.

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), and 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 has been superseded by E01SGF which gives improved accuracy, facilities for obtaining gradient values and a consistent interface with 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 calls 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
Withdrawn 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 (sub)program. 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(*)
C
      CALL LSFUN1(M,N,XC,FVECC)
C
      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
Withdrawn 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 (sub)program. 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(*)
C
      CALL LSFUN2(M,N,XC,FVECC,FJACC,LJC)
C
      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
Withdrawn 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 (sub)program. 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(*)
C
      CALL LSFUN2(M,N,XC,FVECC,FJACC,LJC)
C
      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
Withdrawn 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 (sub)program. 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(*)
C
      CALL LSFUN2(M,N,XC,FVECC,FJACC,LJC)
C
      RETURN
      END
C
      SUBROUTINE LSHES(M,N,FVECC,XC,B,LB,IUSER,USER)
      INTEGER    M, N, LB, IUSER(*)
      real       FVECC(M), XC(N), B(LB), USER(*)
C
      CALL LSHES2(M,N,FVECC,XC,B,LB)
C
      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
Withdrawn 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 (sub)program. 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(*)
C
      CALL FUNCT1(N,XC,FC)
C
      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 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    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 supplies the arrays IUSER and USER specifically 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 1.0*e*+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. It should also be noted that 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
Withdrawn at Mark 19

```
Old: CALL E04KAF(N,IBOUND,BL,BU,X,F,G,IW,LIW,W,LW,IFAIL)
New: CALL E04KYF(N,IBOUND,FUNCT,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 (sub)program. 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(*)
C
      CALL FUNCT2(N,XC,FC,GC)
C
      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
```

E04KCF
Withdrawn at Mark 19

```
Old: CALL E04KCF(N,IBOUND,BL,BU,X,F,G,IW,LIW,W,LW,IFAIL)
New: CALL E04KZF(N,IBOUND,FUNCT,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 E04KCF. FUNCT must be declared as EXTERNAL in the calling (sub)program. 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(*)
C
      CALL FUNCT2(N,XC,FC,GC)
C
      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.

E04LAF
Withdrawn at Mark 19

```
Old: CALL E04LAF(N,IBOUND,BL,BU,X,F,G,IW,LIW,W,LW,IFAIL)
New: CALL E04LYF(N,IBOUND,FUNCT,HESS,BL,BU,X,F,G,IW,LIW,W,LW,IUSER,USER,IFAIL)
```

FUNCT and HESS appear in the parameter list instead of the fixed-name subroutines FUNCT2 and HESS2 of E04LAF. FUNCT and HESS must both be declared as EXTERNAL in the calling (sub)program. In addition they have an extra two parameters, IUSER and USER, over and above those of FUNCT2 and HESS2. They may be derived from FUNCT2 and HESS2 as follows:

```
      SUBROUTINE FUNCT(N,XC,FC,GC,IUSER,USER)
      INTEGER    N, IUSER(*)
      real       XC(N), FC, GC(N), USER(*)
C
      CALL FUNCT2(N,XC,FC,GC)
C
      RETURN
      END
```

```
      SUBROUTINE HESS(N,XC,HESLC,LH,HESDC,IUSER,USER)
      INTEGER    N, LH, IUSER(*)
      real       XC(N), HESLC(LH), HESDC(N), USER(*)
C
      CALL HESS2(N,XC,HESLC,LH,HESDC)
C
      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.

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 $2\times N$ to $2\times N + 3$. 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 $2\times N$ to $2\times N + 3$. 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
Withdrawn 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 $3\times N + NCLIN + NCNLN$ to $3\times N + NCLIN + 2\times NCNLN$. 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 sorgtr('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 $Q^TAQ = T$.

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
Withdrawn 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 leading 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,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,B,IB,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
Withdrawn 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
Withdrawn 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
Withdrawn 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;

$lwork = \max(4 * (N - 1), 1)$ 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:

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

$lwork = \max(5 \times (\mathrm{N}-1), 2)$ 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:

$lwork = \max(3 \times (\mathrm{M}-1), 1)$ when M = N and WANTB is false;

$lwork = \max(5 \times (\mathrm{M}-1), 1)$ when M = N and WANTB is true;

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

$lwork = \mathrm{M}^2 + 5 \times (\mathrm{M}-1)$ 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:

$lwork = \mathrm{N}^2 + 5 \times (\mathrm{N} - 1)$ when $\mathrm{M} \geq \mathrm{N}$;

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

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 $\mathrm{D1} \times 2.0^{\mathrm{ID}}$ 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 F07ARF (CGETRF/ZGETRF) 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
Withdrawn 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
Withdrawn 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).

F11 – Sparse Linear Algebra

F11BAF
Superseded at Mark 19
Scheduled for withdrawal at Mark 21

```
Old: CALL F11BAF(METHOD,PRECON,NORM,WEIGHT,ITERM,N,M,TOL,MAXITN,
    +            ANORM,SIGMAX,MONIT,LWREQ,IFAIL)
New: CALL F11BDF(METHOD,PRECON,NORM,WEIGHT,ITERM,N,M,TOL,MAXITN,
    +            ANORM,SIGMAX,MONIT,WORK,LWORK,LWREQ,IFAIL)
```

F11BDF contains two additional parameters as follows:

WORK(LWORK) – ***real*** array.

LWORK – INTEGER.

See the routine document for further information.

F11BBF
Superseded at Mark 19
Scheduled for withdrawal at Mark 21

```
Old: CALL F11BBF(IREVCM,U,V,WORK,LWORK,IFAIL)
New: CALL F11BEF(IREVCM,U,V,WGT,WORK,LWORK,IFAIL)
```

WGT must be a one-dimensional ***real*** array of length at least n (the order of the matrix) if weights are to be used in the termination criterion, and 1 otherwise. Note that the call to F11BEF requires the weights to be supplied in WGT($1:n$) rather than WORK($1:n$). The minimum value of the parameter LWORK may also need to be changed.

F11BCF
Superseded at Mark 19
Scheduled for withdrawal at Mark 21

```
Old: CALL F11BCF(ITN,STPLHS,STPRHS,ANORM,SIGMAX,IFAIL)
New: CALL F11BFF(ITN,STPLHS,STPRHS,ANORM,SIGMAX,WORK,LWORK,IFAIL)
```

F11BFF contains two additional parameters as follows:

WORK(LWORK) – ***real*** array.

LWORK – INTEGER.

See the routine document for further information.

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 $(X^T X)^{-1}$; however, $(X^T X)^{-1}$ 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 $X^T X$.

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 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,CO,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,CO,C,IFAIL)
     C      else if ICR = 1 in the call to G13DAF
            CALL G13DMF('R-Correlations',NS,NX,W,NSM,NL,WMEAN,CO,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)
         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
      IFAIL = 0

      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.

P01 – Error Trapping

P01AAF
Withdrawn at Mark 13

Existing programs should be modified to call P01ABF. Please consult the appropriate routine document.

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)

Indexes

Keywords in Context

GAMS Index

Keywords in Context for the NAG Fortran 77 Library

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 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)		
C2	Powers, roots, reciprocals		
		A02AAF	Square root of complex number
C3	Polynomials		
C3a	Orthogonal		
C3a2	Chebyshev, Legendre		
		C06DBF	Sum of a Chebyshev series
		E02AEF	Evaluation of fitted polynomial in one variable from Chebyshev series form (simplified parameter list)
		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
C4	Elementary transcendental functions		
C4a	Trigonometric, inverse trigonometric		
		F06BCF	Recover cosine and sine from given real tangent
		F06CCF	Recover cosine and sine from given complex tangent, real cosine
		F06CDF	Recover cosine and sine from given complex tangent, real sine
		S07AAF	$\tan x$
		S09AAF	$\arcsin x$
		S09ABF	$\arccos x$
C4b	Exponential, logarithmic		
		S01BAF	$\ln(1+x)$
		S01EAF	Complex exponential, e^z
C4c	Hyperbolic, inverse hyperbolic		
		S10AAF	$\tanh x$
		S10ABF	$\sinh x$
		S10ACF	$\cosh x$
		S11AAF	$\mathrm{arctanh} x$
		S11ABF	$\mathrm{arcsinh} x$
		S11ACF	$\mathrm{arccosh} x$
C5	Exponential and logarithmic integrals		
		S13AAF	Exponential integral $E_1(x)$
C6	Cosine and sine integrals		
		S13ACF	Cosine integral $\mathrm{Ci}(x)$
		S13ADF	Sine integral $\mathrm{Si}(x)$
C7	Gamma		
C7a	Gamma, log gamma, reciprocal gamma		
		S14AAF	Gamma function
		S14ABF	Log Gamma function
C7c	Psi function		
		S14ACF	$\psi(x) - \ln x$
		S14ADF	Scaled derivatives of $\psi(x)$
C7e	Incomplete gamma		
		S14BAF	Incomplete Gamma functions $P(a,x)$ and $Q(a,x)$
C8	Error functions		
C8a	Error functions, their inverses, integrals, including the normal distribution function		
		S15ABF	Cumulative normal distribution function $P(x)$
		S15ACF	Complement of cumulative normal distribution function $Q(x)$
		S15ADF	Complement of error function $\mathrm{erfc}(x)$
		S15AEF	Error function $\mathrm{erf}(x)$
		S15DDF	Scaled complex complement of error function, $\exp(-z^2)\mathrm{erfc}(-iz)$

C8b	Fresnel integrals	
	S20ACF	Fresnel integral $S(x)$
	S20ADF	Fresnel integral $C(x)$
C8c	Dawson's integral	
	S15AFF	Dawson's integral
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 $e^{-\lvert x\rvert} I_0(x)$
	S18CFF	Modified Bessel function $e^{-\lvert x\rvert} I_1(x)$
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 $R_J(x,y,z,r)$
D	Linear Algebra	
D1	Elementary vector and matrix operations	
D1a	Elementary vector operations	
D1a1	Set to constant	
	F06DBF	Broadcast scalar into integer vector
	F06EVF	(SGTHRZ/DGTHRZ) Gather and set to zero real sparse vector
	F06FBF	Broadcast scalar into real vector
	F06GVF	(CGTHRZ/ZGTHRZ) Gather and set to zero complex sparse vector
	F06HBF	Broadcast scalar into complex vector
D1a2	Minimum and maximum components	
	F06FLF	Elements of real vector with largest and smallest absolute value
	F06JLF	(ISAMAX/IDAMAX) Index, real vector element with largest absolute value
	F06JMF	(ICAMAX/IZAMAX) Index, complex vector element with largest absolute value
	F06KLF	Last non-negligible element of real vector
D1a3	Norm	
D1a3a	L_1 (sum of magnitudes)	
	F06EKF	(SASUM/DASUM) Sum absolute values of real vector elements
	F06JKF	(SCASUM/DZASUM) Sum absolute values of complex vector elements
D1a3b	L_2 (Euclidean norm)	
	F06BMF	Compute Euclidean norm from scaled form
	F06BNF	Compute square root of $(a^2 + b^2)$, real a and b
	F06EJF	(SNRM2/DNRM2) Compute Euclidean norm of real vector
	F06FJF	Update Euclidean norm of real vector in scaled form

F06SCF (CHEMV/ZHEMV) Matrix-vector product, complex Hermitian matrix
F06SDF (CHBMV/ZHBMV) Matrix-vector product, complex Hermitian band matrix
F06SEF (CHPMV/ZHPMV) Matrix-vector product, complex Hermitian packed matrix
F06SFF (CTRMV/ZTRMV) Matrix-vector product, complex triangular matrix
F06SGF (CTBMV/ZTBMV) Matrix-vector product, complex triangular band matrix
F06SHF (CTPMV/ZTPMV) Matrix-vector product, complex triangular packed matrix
F11XAF Real sparse nonsymmetric matrix vector multiply
F11XEF Real sparse symmetric matrix vector multiply
F11XNF Complex sparse non-Hermitian matrix vector multiply
F11XSF Complex sparse Hermitian matrix vector multiply

D1b5 Addition, subtraction

F01CTF Sum or difference of two real matrices, optional scaling and transposition
F01CWF Sum or difference of two complex matrices, optional scaling and transposition
F06PMF (SGER/DGER) Rank-1 update, real rectangular matrix
F06PPF (SSYR/DSYR) Rank-1 update, real symmetric matrix
F06PQF (SSPR/DSPR) Rank-1 update, real symmetric packed matrix
F06PRF (SSYR2/DSYR2) Rank-2 update, real symmetric matrix
F06PSF (SSPR2/DSPR2) Rank-2 update, real symmetric packed matrix
F06SMF (CGERU/ZGERU) Rank-1 update, complex rectangular matrix, unconjugated vector
F06SNF (CGERC/ZGERC) Rank-1 update, complex rectangular matrix, conjugated vector
F06SPF (CHER/ZHER) Rank-1 update, complex Hermitian matrix
F06SQF (CHPR/ZHPR) Rank-1 update, complex Hermitian packed matrix
F06SRF (CHER2/ZHER2) Rank-2 update, complex Hermitian matrix
F06SSF (CHPR2/ZHPR2) Rank-2 update, complex Hermitian packed matrix
F06YPF (SSYRK/DSYRK) Rank-k update of real symmetric matrix
F06ZPF (CHERK/ZHERK) Rank-k update of complex Hermitian matrix
F06ZRF (CHER2K/ZHER2K) Rank-$2k$ update of complex Hermitian matrix
F06ZUF (CSYRK/ZSYRK) Rank-k update of complex symmetric matrix
F06ZWF (CSYR2K/ZHER2K) Rank-$2k$ update of complex symmetric matrix

D1b6 Multiplication

F01CKF Matrix multiplication
F06FCF Multiply real vector by diagonal matrix
F06YAF (SGEMM/DGEMM) Matrix-matrix product, two real rectangular matrices
F06YCF (SSYMM/DSYMM) Matrix-matrix product, one real symmetric matrix, one real rectangular matrix
F06YFF (STRMM/DTRMM) Matrix-matrix product, one real triangular matrix, one real rectangular matrix
F06YRF (SSYR2K/DSYR2K) Rank-$2k$ update of real symmetric matrix
F06ZAF (CGEMM/ZGEMM) Matrix-matrix product, two complex rectangular matrices
F06ZCF (CHEMM/ZHEMM) Matrix-matrix product, one complex Hermitian matrix, one complex rectangular matrix
F06ZFF (CTRMM/ZTRMM) Matrix-matrix product, one complex triangular matrix, one complex rectangular matrix
F06ZTF (CSYMM/ZSYMM) Matrix-matrix product, one complex symmetric matrix, one complex rectangular matrix

D1b8 Copy

F06QFF Matrix copy, real rectangular or trapezoidal matrix
F06TFF Matrix copy, complex rectangular or trapezoidal matrix

D1b9 Storage mode conversion

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
F11ZAF Real sparse nonsymmetric matrix reorder routine
F11ZBF Real sparse symmetric matrix reorder routine
F11ZPF Complex sparse Hermitian matrix reorder routine
F11ZNF Complex sparse non-Hermitian matrix reorder routine

D1b10 Elementary rotation (Givens transformation)

F06QMF Orthogonal similarity transformation of real symmetric matrix as 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
F06TMF Unitary similarity transformation of Hermitian matrix as 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

F07FDF (SPOTRF/DPOTRF) Cholesky factorization of real symmetric positive-definite matrix

F07FEF (SPOTRS/DPOTRS) Solution of real symmetric positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07FDF

F07FGF (SPOCON/DPOCON) Estimate condition number of real symmetric positive-definite matrix, matrix already factorized by F07FDF

F07FHF (SPORFS/DPORFS) Refined solution with error bounds of real symmetric positive-definite system of linear equations, multiple right-hand sides

F07FJF (SPOTRI/DPOTRI) Inverse of real symmetric positive-definite matrix, matrix already factorized by F07FDF

F07GDF (SPPTRF/DPPTRF) Cholesky factorization of real symmetric positive-definite matrix, packed storage

F07GEF (SPPTRS/DPPTRS) Solution of real symmetric positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07GDF, packed storage

F07GGF (SPPCON/DPPCON) Estimate condition number of real symmetric positive-definite matrix, matrix already factorized by F07GDF, packed storage

F07GHF (SPPRFS/DPPRFS) Refined solution with error bounds of real symmetric positive-definite system of linear equations, multiple right-hand sides, packed storage

F07GJF (SPPTRI/DPPTRI) Inverse of real symmetric positive-definite matrix, matrix already factorized by F07GDF, packed storage

D2b2 Positive-definite banded

F01MCF LDL^T factorization of real symmetric positive-definite variable-bandwidth matrix

F04ACF Solution of real symmetric positive-definite banded simultaneous linear equations with multiple right-hand sides (Black Box)

F04MCF Solution of real symmetric positive-definite variable-bandwidth simultaneous linear equations (coefficient matrix already factorized by F01MCF)

F07HDF (SPBTRF/DPBTRF) Cholesky factorization of real symmetric positive-definite band matrix

F07HEF (SPBTRS/DPBTRS) Solution of real symmetric positive-definite band system of linear equations, multiple right-hand sides, matrix already factorized by F07HDF

F07HGF (SPBCON/DPBCON) Estimate condition number of real symmetric positive-definite band matrix, matrix already factorized by F07HDF

F07HHF (SPBRFS/DPBRFS) Refined solution with error bounds of real symmetric positive-definite band system of linear equations, multiple right-hand sides

F08UFF (SPBSTF/DPBSTF) Computes a split Cholesky factorization of real symmetric positive-definite band matrix A

F08UTF (CPBSTF/ZPBSTF) Computes a split Cholesky factorization of complex Hermitian positive-definite band matrix A

D2b2a Tridiagonal

F04FAF Solution of real symmetric positive-definite tridiagonal simultaneous linear equations, one right-hand side (Black Box)

D2b4 Sparse

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

F11GBF Real sparse symmetric linear systems, preconditioned 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 Cholesky 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)

D2c Complex non-Hermitian matrices

D2c1 General

F04ADF Solution of complex simultaneous linear equations with multiple right-hand sides (Black Box)

F07ARF (CGETRF/ZGETRF) LU factorization of complex m by n matrix

F07ASF (CGETRS/ZGETRS) Solution of complex system of linear equations, multiple right-hand sides, matrix already factorized by F07ARF

F07AUF (CGECON/ZGECON) Estimate condition number of complex matrix, matrix already factorized by F07ARF

F07AVF (CGERFS/ZGERFS) Refined solution with error bounds of complex system of linear equations, multiple right-hand sides

F07AWF (CGETRI/ZGETRI) Inverse of complex matrix, matrix already factorized by F07ARF

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

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

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

F07MVF (CHERFS/ZHERFS) Refined solution with error bounds of complex Hermitian indefinite system of linear equations, multiple right-hand sides
F07MWF (CHETRI/ZHETRI) Inverse of complex Hermitian indefinite matrix, matrix already factorized by F07MRF
F07PRF (CHPTRF/ZHPTRF) Bunch–Kaufman factorization of complex Hermitian indefinite matrix, packed storage
F07PSF (CHPTRS/ZHPTRS) Solution of complex Hermitian indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07PRF, packed storage
F07PUF (CHPCON/ZHPCON) Estimate condition number of complex Hermitian indefinite matrix, matrix already factorized by F07PRF, packed storage
F07PVF (CHPRFS/ZHPRFS) Refined solution with error bounds of complex Hermitian indefinite system of linear equations, multiple right-hand sides, packed storage
F07PWF (CHPTRI/ZHPTRI) Inverse of complex Hermitian indefinite matrix, matrix already factorized by F07PRF, packed storage

D2d1b Positive-definite
F07FRF (CPOTRF/ZPOTRF) Cholesky factorization of complex Hermitian positive-definite matrix
F07FSF (CPOTRS/ZPOTRS) Solution of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07FRF
F07FUF (CPOCON/ZPOCON) Estimate condition number of complex Hermitian positive-definite matrix, matrix already factorized by F07FRF
F07FVF (CPORFS/ZPORFS) Refined solution with error bounds of complex Hermitian positive-definite system of linear equations, multiple right-hand sides
F07FWF (CPOTRI/ZPOTRI) Inverse of complex Hermitian positive-definite matrix, matrix already factorized by F07FRF
F07GRF (CPPTRF/ZPPTRF) Cholesky factorization of complex Hermitian positive-definite matrix, packed storage
F07GSF (CPPTRS/ZPPTRS) Solution of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07GRF, packed storage
F07GUF (CPPCON/ZPPCON) Estimate condition number of complex Hermitian positive-definite matrix, matrix already factorized by F07GRF, packed storage
F07GVF (CPPRFS/ZPPRFS) Refined solution with error bounds of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, packed storage
F07GWF (CPPTRI/ZPPTRI) Inverse of complex Hermitian positive-definite matrix, matrix already factorized by F07GRF, packed storage

D2d2 Positive-definite banded
F07HRF (CPBTRF/ZPBTRF) Cholesky factorization of complex Hermitian positive-definite band matrix
F07HSF (CPBTRS/ZPBTRS) Solution of complex Hermitian positive-definite band system of linear equations, multiple right-hand sides, matrix already factorized by F07HRF
F07HUF (CPBCON/ZPBCON) Estimate condition number of complex Hermitian positive-definite band matrix, matrix already factorized by F07HRF
F07HVF (CPBRFS/ZPBRFS) Refined solution with error bounds of complex Hermitian positive-definite band system of linear equations, multiple right-hand sides

D2d4 Sparse
F11JNF Complex sparse Hermitian matrix, incomplete Cholesky factorization
F11JPF Solution of complex linear system involving incomplete Cholesky preconditioning matrix generated by F11JNF
F11JQF Solution of complex sparse Hermitian linear system, conjugate gradient/Lanczos method, preconditioner computed by F11JNF (Black Box)
F11JRF Solution of linear system involving preconditioning matrix generated by applying SSOR to complex sparse Hermitian matrix
F11JSF Solution of complex sparse Hermitian linear system, conjugate gradient/Lanczos method, Jacobi or SSOR preconditioner (Black Box)

D2e Associated operations (e.g., matrix reorderings)
F11XAF Real sparse nonsymmetric matrix vector multiply
F11XEF Real sparse symmetric matrix vector multiply
F11XNF Complex sparse non-Hermitian matrix vector multiply
F11XSF Complex sparse Hermitian matrix vector multiply
F11ZAF Real sparse nonsymmetric matrix reorder routine
F11ZBF Real sparse symmetric matrix reorder routine
F11ZNF Complex sparse non-Hermitian matrix reorder routine
F11ZPF Complex sparse Hermitian matrix reorder routine

D3 Determinants
D3a Real nonsymmetric matrices
D3a1 General
F03AAF Determinant of real matrix (Black Box)
F03AFF *LU* factorization and determinant of real matrix

D3b Real symmetric matrices
D3b1 General

D3b1b	Positive-definite	
	F03ABF	Determinant of real symmetric positive-definite matrix (Black Box)
	F03AEF	LL^T factorization and determinant of real symmetric positive-definite matrix
D3b2	Positive-definite banded	
	F03ACF	Determinant of real symmetric positive-definite band matrix (Black Box)
D3c	Complex non-Hermitian matrices	
D3c1	General	
	F03ADF	Determinant of complex matrix (Black Box)
D4	Eigenvalues, eigenvectors	
D4a	Ordinary eigenvalue problems ($Ax = \lambda x$)	
D4a1	Real symmetric	
	F02FAF	All eigenvalues and eigenvectors of real symmetric matrix (Black Box)
	F02FCF	Selected eigenvalues and eigenvectors of real symmetric matrix (Black Box)
	F06BPF	Compute eigenvalue of 2 by 2 real symmetric matrix
	F08FCF	(SSYEVD/DSYEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, using divide and conquer
	F08GCF	(SSPEVD/DSPEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, packed storage, using divide and conquer
	F08HCF	(SSBEVD/DSBEVD) All eigenvalues and optionally all eigenvectors of real symmetric band matrix, using divide and conquer
D4a2	Real nonsymmetric	
	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)
D4a3	Complex Hermitian	
	F02HAF	All eigenvalues and eigenvectors of complex Hermitian matrix (Black Box)
	F02HCF	Selected eigenvalues and eigenvectors of complex Hermitian matrix (Black Box)
	F08FQF	(CHEEVD/ZHEEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, using divide and conquer
	F08GQF	(CHPEVD/ZHPEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, packed storage, using divide and conquer
	F08HQF	(CHBEVD/ZHBEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian band matrix, using divide and conquer
D4a4	Complex non-Hermitian	
	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)
D4a5	Tridiagonal	
	F08JCF	(SSTEVD/DSTEVD) All eigenvalues and optionally all eigenvectors of real symmetric tridiagonal matrix, using divide and conquer
	F08JEF	(SSTEQR/DSTEQR) All eigenvalues and eigenvectors of real symmetric tridiagonal matrix, reduced from real symmetric matrix using implicit QL or QR
	F08JFF	(SSTERF/DSTERF) All eigenvalues of real symmetric tridiagonal matrix, root-free variant of QL or QR
	F08JGF	(SPTEQR/DPTEQR) All eigenvalues and eigenvectors of real symmetric positive-definite tridiagonal matrix, reduced from real symmetric positive-definite matrix
	F08JJF	(SSTEBZ/DSTEBZ) Selected eigenvalues of real symmetric tridiagonal matrix by bisection
	F08JKF	(SSTEIN/DSTEIN) Selected eigenvectors of real symmetric tridiagonal matrix by inverse iteration, storing eigenvectors in real array
D4a6	Banded	
	F08HCF	(SSBEVD/DSBEVD) All eigenvalues and optionally all eigenvectors of real symmetric band matrix, using divide and conquer
	F08HQF	(CHBEVD/ZHBEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian band matrix, using divide and conquer
D4a7	Sparse	
	F02FJF	Selected eigenvalues and eigenvectors of sparse symmetric eigenproblem (Black Box)
D4b	Generalized eigenvalue problems (e.g., $Ax = \lambda Bx$)	
D4b1	Real symmetric	
	F02FDF	All eigenvalues and eigenvectors of real symmetric-definite generalized problem (Black Box)
	F02FJF	Selected eigenvalues and eigenvectors of sparse symmetric eigenproblem (Black Box)
D4b2	Real general	
	F02BJF	All eigenvalues and optionally eigenvectors of generalized eigenproblem by QZ algorithm, real matrices (Black Box)
D4b3	Complex Hermitian	
	F02HDF	All eigenvalues and eigenvectors of complex Hermitian-definite generalized problem (Black Box)
D4b4	Complex general	
	F02GJF	All eigenvalues and optionally eigenvectors of generalized complex eigenproblem by QZ algorithm (Black Box)
D4b5	Banded	

F02FHF All eigenvalues of generalized banded real symmetric-definite eigenproblem (Black Box)

F02SDF Eigenvector of generalized real banded eigenproblem by inverse iteration

D4c Associated operations

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

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

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

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

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

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

D4c1 Transform problem

D4c1a Balance matrix

F08NHF (SGEBAL/DGEBAL) Balance real general matrix

F08NVF (CGEBAL/ZGEBAL) Balance complex general matrix

D4c1b Reduce to compact form

D4c1b1 Tridiagonal

F08FEF (SSYTRD/DSYTRD) Orthogonal reduction of real symmetric matrix to symmetric tridiagonal form

F08FFF (SORGTR/DORGTR) Generate orthogonal transformation matrix from reduction to tridiagonal form determined by F08FEF

F08FSF (CHETRD/ZHETRD) Unitary reduction of complex Hermitian matrix to real symmetric tridiagonal form

F08FTF (CUNGTR/ZUNGTR) Generate unitary transformation matrix from reduction to tridiagonal form determined by F08FSF

F08GEF (SSPTRD/DSPTRD) Orthogonal reduction of real symmetric matrix to symmetric tridiagonal form, packed storage

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

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

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

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

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

D4c1b2 Hessenberg

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

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

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

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

D4c1b3 Other

F08LEF (SGBBRD/DGBBRD) Reduction of real rectangular band matrix to upper bidiagonal form

F08LSF (CGBBRD/ZGBBRD) Reduction of complex rectangular band matrix to upper bidiagonal form

D4c1c Standardize problem

F01BVF Reduction to standard form, generalized real symmetric-definite banded eigenproblem

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

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

F08TEF (SSPGST/DSPGST) 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

F08TSF (CHPGST/ZHPGST) 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

F08UEF (SSBGST/DSBGST) Reduction of real symmetric-definite banded generalized eigenproblem $Ax = \lambda Bx$ to standard form $Cy = \lambda y$, such that C has the same bandwidth as A

F08USF (CHBGST/ZHBGST) Reduction of complex Hermitian-definite banded generalized eigenproblem $Ax = \lambda Bx$ to standard form $Cy = \lambda y$, such that C has the same bandwidth as A

D4c2 Compute eigenvalues of matrix in compact form

D4c2a Tridiagonal

F08FCF (SSYEVD/DSYEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, using divide and conquer

F08FQF (CHEEVD/ZHEEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, using divide and conquer

F08GCF (SSPEVD/DSPEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, packed storage, using divide and conquer

F08GQF (CHPEVD/ZHPEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, packed storage, using divide and conquer

F08JCF (SSTEVD/DSTEVD) All eigenvalues and optionally all eigenvectors of real symmetric tridiagonal matrix, using divide and conquer

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

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

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

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

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

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

D4c2b Hessenberg

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

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

D4c3 Form eigenvectors from eigenvalues

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

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

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

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

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

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

D4c4 Back transform eigenvectors

F08FGF (SORMTR/DORMTR) Apply orthogonal transformation determined by F08FEF

F08FUF (CUNMTR/ZUNMTR) Apply unitary transformation matrix determined by F08FSF

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

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

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

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

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

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

D5 QR decomposition, Gram–Schmidt orthogonalization

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

F05AAF Gram–Schmidt orthogonalisation of n vectors of order m

	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
	F06TPF	QR factorization by sequence of plane rotations, rank-1 update of complex upper triangular matrix
	F06TQF	$QRxk$ 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
	F08AEF	(SGEQRF/DGEQRF) QR factorization of real general rectangular matrix
	F08AFF	(SORGQR/DORGQR) Form all or part of orthogonal Q from QR factorization determined by F08AEF or F08BEF
	F08AGF	(SORMQR/DORMQR) Apply orthogonal transformation determined by F08AEF or F08BEF
	F08AHF	(SGELQF/DGELQF) LQ factorization of real general rectangular matrix
	F08AJF	(SORGLQ/DORGLQ) Form all or part of orthogonal Q from LQ factorization determined by F08AHF
	F08AKF	(SORMLQ/DORMLQ) Apply orthogonal transformation determined by F08AHF
	F08ASF	(CGEQRF/ZGEQRF) QR factorization of complex general rectangular matrix
	F08ATF	(CUNGQR/ZUNGQR) Form all or part of unitary Q from QR factorization determined by F08ASF or F08BSF
	F08AUF	(CUNMQR/ZUNMQR) Apply unitary transformation determined by F08ASF or F08BSF
	F08AVF	(CGELQF/ZGELQF) LQ factorization of complex general rectangular matrix
	F08AWF	(CUNGLQ/ZUNGLQ) Form all or part of unitary Q from LQ factorization determined by F08AVF
	F08AXF	(CUNMLQ/ZUNMLQ) Apply unitary transformation determined by F08AVF
	F08BEF	(SGEQPF/DGEQPF) QR factorization of real general rectangular matrix with column pivoting
	F08BSF	(CGEQPF/ZGEQPF) QR factorization of complex general rectangular matrix with column pivoting
D6	Singular value decomposition	
	F02WDF	QR factorization, possibly followed by SVD
	F02WEF	SVD of real matrix (Black Box)
	F02WUF	SVD of real upper triangular matrix (Black Box)
	F02XEF	SVD of complex matrix (Black Box)
	F02XUF	SVD of complex upper triangular matrix (Black Box)
	F08KEF	(SGEBRD/DGEBRD) Orthogonal reduction of real general rectangular matrix to bidiagonal form
	F08KFF	(SORGBR/DORGBR) Generate orthogonal transformation matrices from reduction to bidiagonal form determined by F08KEF
	F08KGF	(SORMBR/DORMBR) Apply orthogonal transformations from reduction to bidiagonal form determined by F08KEF
	F08KSF	(CGEBRD/ZGEBRD) Unitary reduction of complex general rectangular matrix to bidiagonal form
	F08KTF	(CUNGBR/ZUNGBR) Generate unitary transformation matrices from reduction to bidiagonal form determined by F08KSF
	F08KUF	(CUNMBR/ZUNMBR) Apply unitary transformations from reduction to bidiagonal form determined by F08KSF
	F08MEF	(SBDSQR/DBDSQR) SVD of real bidiagonal matrix reduced from real general matrix
	F08MSF	(CBDSQR/ZBDSQR) SVD of real bidiagonal matrix reduced from complex general matrix
D8	Other matrix equations (e.g., $AX + XB = C$)	
	F08QHF	(STRSYL/DTRSYL) Solve real Sylvester matrix equation $AX + XB = C$, A and B are upper quasi-triangular or transposes
	F08QVF	(CTRSYL/ZTRSYL) Solve complex Sylvester matrix equation $AX + XB = C$, A and B are upper triangular or conjugate-transposes
D9	Singular, overdetermined or underdetermined systems of linear equations, generalized inverses	
D9a	Unconstrained	
D9a1	Least squares (L_2) solution	
	F04AMF	Least-squares solution of m real equations in n unknowns, rank $= n$, $m \geq n$ using iterative refinement (Black Box)

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)
F04KLF Complex general Gauss–Markov linear model (including weighted least-squares)
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

D9a2 Chebyshev (L_∞) solution
E02GCF L_∞-approximation by general linear function

D9a3 Least absolute value (L_1) solution
E02GAF L_1-approximation by general linear function

D9b Constrained

D9b1 Least squares (L_2) solution
E04NCF Convex QP problem or linearly-constrained linear least-squares problem (dense)
F04JMF Equality-constrained real linear least-squares problem
F04KMF Equality-constrained complex linear least-squares problem

D9b3 Least absolute value (L_1)
E02GBF L_1-approximation by general linear function subject to linear inequality constraints

D9c Generalized inverses
F01BLF Pseudo-inverse and rank of real m by n matrix ($m \geq n$)

E Interpolation

E1 Univariate data (curve fitting)

E1a Polynomial splines (piecewise polynomials)
E01BAF Interpolating functions, cubic spline interpolant, one variable
E01BEF Interpolating functions, monotonicity-preserving, piecewise cubic Hermite, one variable
E02BAF Least-squares curve cubic spline fit (including interpolation)

E1b Polynomials
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
E02AFF Least-squares polynomial fit, special data points (including interpolation)

E1c Other functions (e.g., rational, trigonometric)
E01RAF Interpolating functions, rational interpolant, one variable

E2 Multivariate data (surface fitting)

E2a Gridded
E01DAF Interpolating functions, fitting bicubic spline, data on rectangular grid

E2b Scattered
E01SAF Interpolating functions, method of Renka and Cline, two variables
E01SEF Interpolating functions, modified Shepard's method, 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

E3 Service routines for interpolation

E3a Evaluation of fitted functions, including quadrature

E3a1 Function evaluation
E01BFF Interpolated values, interpolant computed by E01BEF, function only, one variable
E01RBF Interpolated values, evaluate rational interpolant computed by E01RAF, one variable
E01SBF Interpolated values, evaluate interpolant computed by E01SAF, two variables
E01SFF Interpolated values, evaluate interpolant computed by E01SEF, two variables
E02AEF Evaluation of fitted polynomial in one variable from Chebyshev series form (simplified parameter list)
E02AKF Evaluation of fitted polynomial in one variable from Chebyshev series form
E02BBF Evaluation of fitted cubic spline, function only
E02BCF Evaluation of fitted cubic spline, function and derivatives
E02CBF Evaluation of fitted polynomial in two variables
E02DEF Evaluation of fitted bicubic spline at a vector of points
E02DFF Evaluation of fitted bicubic spline at a mesh of points

E3a2 Derivative evaluation
E01BGF Interpolated values, interpolant computed by E01BEF, function and first derivative, one variable
E02AHF Derivative of fitted polynomial in Chebyshev series form
E02BCF Evaluation of fitted cubic spline, function and derivatives

E3a3 Quadrature
E01BHF Interpolated values, interpolant computed by E01BEF, definite integral, one variable

	E02AJF	Integral of fitted polynomial in Chebyshev series form
	E02BDF	Evaluation of fitted cubic spline, definite integral
E3d	Other	
	E02ZAF	Sort two-dimensional data into panels for fitting bicubic splines
F	Solution of nonlinear equations	
F1	Single equation	
F1a	Polynomial	
F1a1	Real coefficients	
	C02AGF	All zeros of real polynomial, modified Laguerre method
	C02AJF	All zeros of real quadratic
F1a2	Complex coefficients	
	C02AFF	All zeros of complex polynomial, modified Laguerre method
	C02AHF	All zeros of complex quadratic
F1b	Nonpolynomial	
	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)
F2	System of equations	
	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 system of nonlinear equations using function values only (reverse communication)
	C05PBF	Solution of system of nonlinear equations using first derivatives (easy-to-use)
	C05PCF	Solution of system of nonlinear equations using first derivatives (comprehensive)
	C05PDF	Solution of system of nonlinear equations using first derivatives (reverse communication)
F3	Service routines (e.g., check user-supplied derivatives)	
	C05ZAF	Check user's routine for calculating first derivatives
	E04HCF	Check user's routine for calculating first derivatives of function
	E04HDF	Check user's routine for calculating second derivatives of function
G	Optimization (*search also classes K, L8*)	
G1	Unconstrained	
G1a	Univariate	
G1a1	Smooth function	
G1a1a	User provides no derivatives	
	E04ABF	Minimum, function of one variable using function values only
G1a1b	User provides first derivatives	
	E04BBF	Minimum, function of one variable, using first derivative
G1b	Multivariate	
G1b1	Smooth function	
G1b1b	User provides first derivatives	
	E04DGF	Unconstrained minimum, preconditioned conjugate gradient algorithm, function of several variables using first derivatives (comprehensive)
G1b2	General function (no smoothness assumed)	
	E04CCF	Unconstrained minimum, simplex algorithm, function of several variables using function values only (comprehensive)
G2	Constrained	
G2a	Linear programming	
G2a1	Dense matrix of constraints	
	E04MFF	LP problem (dense)
	E04NCF	Convex QP problem or linearly-constrained linear least-squares problem (dense)
	E04NFF	QP problem (dense)
	H02BFF	Interpret MPSX data file defining IP or LP problem, optimize and print solution
	H02CBF	Integer QP problem (dense)
G2a2	Sparse matrix of constraints	
	E04NKF	LP or QP problem (sparse)
	E04UGF	NLP problem (sparse)
	H02CEF	Integer LP or QP problem (sparse)
G2b	Transportation and assignments problem	
	H03ABF	Transportation problem, modified 'stepping stone' method
G2c	Integer programming	
G2c1	Zero/one	
	H02BBF	Integer LP problem (dense)
G2c6	Pure integer programming	
	H02BBF	Integer LP problem (dense)

G2c7 Mixed integer programming
- `H02BBF` Integer LP problem (dense)
- `H02BFF` Interpret MPSX data file defining IP or LP problem, optimize and print solution

G2d Network (*for network reliability search class M*)

G2d1 Shortest path
- `H03ADF` Shortest path problem, Dijkstra's algorithm

G2e Quadratic programming

G2e1 Positive-definite Hessian (i.e., convex problem)
- `E04NCF` Convex QP problem or linearly-constrained linear least-squares problem (dense)
- `E04NFF` QP problem (dense)
- `E04NKF` LP or QP problem (sparse)
- `E04UGF` NLP problem (sparse)
- `H02CBF` Integer QP problem (dense)
- `H02CEF` Integer LP or QP problem (sparse)

G2e2 Indefinite Hessian
- `E04NFF` QP problem (dense)
- `E04NKF` LP or QP problem (sparse)
- `E04UGF` NLP problem (sparse)
- `H02CBF` Integer QP problem (dense)
- `H02CEF` Integer LP or QP problem (sparse)

G2h General nonlinear programming

G2h1 Simple bounds

G2h1a Smooth function

G2h1a1 User provides no derivatives
- `E04JYF` Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using function values only (easy-to-use)
- `E04UCF` Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)
- `E04UFF` Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)
- `E04UNF` Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

G2h1a2 User provides first derivatives
- `E04KDF` Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (comprehensive)
- `E04KYF` Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using first derivatives (easy-to-use)
- `E04KZF` Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (easy-to-use)
- `E04UCF` Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)
- `E04UFF` Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)
- `E04UNF` Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

G2h1a3 User provides first and second derivatives
- `E04LBF` Minimum, function of several variables, modified Newton algorithm, simple bounds, using first and second derivatives (comprehensive)
- `E04LYF` Minimum, function of several variables, modified Newton algorithm, simple bounds, using first and second derivatives (easy-to-use)

G2h2 Linear equality or inequality constraints

G2h2a Smooth function

G2h2a1 User provides no derivatives
- `E04UCF` Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)
- `E04UFF` Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)
- `E04UNF` Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

G2h2a2 User provides first derivatives
- `E04UCF` Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)
- `E04UFF` Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)

E04UNF Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

G2h3 Nonlinear constraints

G2h3a Equality constraints only

G2h3a1 Smooth function and constraints

E04UCF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)

E04UFF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)

E04UNF Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

G2h3b Equality and inequality constraints

G2h3b1 Smooth function and constraints

G2h3b1a User provides no derivatives

E04UCF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)

E04UFF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)

E04UNF Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

G2h3b1b User provides first derivatives of function and constraints

E04UCF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)

E04UFF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)

E04UNF Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

G4 Service routines

G4a Problem input (e.g., matrix generation)

E04MZF Converts MPSX data file defining LP or QP problem to format required by E04NKF

E04UQF Read optional parameter values for E04UNF from external file

H02BUF Convert MPSX data file defining IP or LP problem to format required by H02BBF or E04MFF

G4c Check user-supplied derivatives

E04HCF Check user's routine for calculating first derivatives of function

E04HDF Check user's routine for calculating second derivatives of function

E04YAF Check user's routine for calculating Jacobian of first derivatives

E04YBF Check user's routine for calculating Hessian of a sum of squares

E04ZCF Check user's routines for calculating first derivatives of function and constraints

G4d Find feasible point

E04MFF LP problem (dense)

E04NCF Convex QP problem or linearly-constrained linear least-squares problem (dense)

E04NFF QP problem (dense)

E04NKF LP or QP problem (sparse)

E04UCF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)

E04UFF Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)

E04UGF NLP problem (sparse)

E04UNF Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

H02CBF Integer QP problem (dense)

H02CEF Integer LP or QP problem (sparse)

G4f Other

E04DJF Read optional parameter values for E04DGF from external file

E04DKF Supply optional parameter values to E04DGF

E04MGF Read optional parameter values for E04MFF from external file

E04MHF Supply optional parameter values to E04MFF

E04NDF Read optional parameter values for E04NCF from external file

E04NEF Supply optional parameter values to E04NCF

E04NGF Read optional parameter values for E04NFF from external file

E04NHF Supply optional parameter values to E04NFF

E04NLF Read optional parameter values for E04NKF from external file

E04NMF Supply optional parameter values to E04NKF

E04UDF Read optional parameter values for E04UCF or E04UFF from external file

	E04UEF	Supply optional parameter values to E04UCF or E04UFF
	E04UHF	Read optional parameter values for E04UGF from external file
	E04UJF	Supply optional parameter values to E04UGF
	E04UQF	Read optional parameter values for E04UNF from external file
	E04URF	Supply optional parameter values to E04UNF
	E04XAF	Estimate (using numerical differentiation) gradient and/or Hessian of a function
	H02BVF	Print IP or LP solutions with user specified names for rows and columns
	H02BZF	Integer programming solution, supplies further information on solution obtained by H02BBF
	H02CCF	Read optional parameter values for H02CBF from external file
	H02CDF	Supply optional parameter values to H02CBF
	H02CFF	Read optional parameter values for H02CEF from external file
	H02CGF	Supply optional parameter values to H02CEF
H	Differentiation, integration	
H1	Numerical differentiation	
	D04AAF	Numerical differentiation, derivatives up to order 14, function of one real variable
	E04XAF	Estimate (using numerical differentiation) gradient and/or Hessian of a function
H2	Quadrature (numerical evaluation of definite integrals)	
H2a	One-dimensional integrals	
H2a1	Finite interval (general integrand)	
H2a1a	Integrand available via user-defined procedure	
H2a1a1	Automatic (user need only specify required accuracy)	
	D01AHF	One-dimensional quadrature, adaptive, finite interval, strategy due to Patterson, suitable for well-behaved integrands
	D01AJF	One-dimensional quadrature, adaptive, finite interval, strategy due to Piessens and de Doncker, allowing for badly-behaved integrands
	D01ARF	One-dimensional quadrature, non-adaptive, finite interval with provision for indefinite integrals
	D01ATF	One-dimensional quadrature, adaptive, finite interval, variant of D01AJF efficient on vector machines
	D01BDF	One-dimensional quadrature, non-adaptive, finite interval
H2a1a2	Nonautomatic	
	D01BAF	One-dimensional Gaussian quadrature
H2a1b	Integrand available only on grid	
H2a1b2	Nonautomatic	
	D01GAF	One-dimensional quadrature, integration of function defined by data values, Gill–Miller method
H2a2	Finite interval (specific or special type integrand including weight functions, oscillating and singular integrands, principal value integrals, splines, etc.)	
H2a2a	Integrand available via user-defined procedure	
H2a2a1	Automatic (user need only specify required accuracy)	
	D01AKF	One-dimensional quadrature, adaptive, finite interval, method suitable for oscillating functions
	D01ALF	One-dimensional quadrature, adaptive, finite interval, allowing for singularities at user-specified break-points
	D01ANF	One-dimensional quadrature, adaptive, finite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
	D01APF	One-dimensional quadrature, adaptive, finite interval, weight function with end-point singularities of algebraico-logarithmic type
	D01AQF	One-dimensional quadrature, adaptive, finite interval, weight function $1/(x-c)$, Cauchy principal value (Hilbert transform)
	D01AUF	One-dimensional quadrature, adaptive, finite interval, variant of D01AKF efficient on vector machines
H2a2b	Integrand available only on grid	
H2a2b1	Automatic (user need only specify required accuracy)	
	E02AJF	Integral of fitted polynomial in Chebyshev series form
	E02BDF	Evaluation of fitted cubic spline, definite integral
H2a3	Semi-infinite interval (including e^{-x} weight function)	
H2a3a	Integrand available via user-defined procedure	
H2a3a1	Automatic (user need only specify required accuracy)	
	D01AMF	One-dimensional quadrature, adaptive, infinite or semi-infinite interval
	D01ASF	One-dimensional quadrature, adaptive, semi-infinite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
H2a3a2	Nonautomatic	
	D01BAF	One-dimensional Gaussian quadrature
H2a4	Infinite interval (including e^{-x^2} weight function)	
H2a4a	Integrand available via user-defined procedure	
H2a4a1	Automatic (user need only specify required accuracy)	
	D01AMF	One-dimensional quadrature, adaptive, infinite or semi-infinite interval
H2a4a2	Nonautomatic	
	D01BAF	One-dimensional Gaussian quadrature

H2b Multidimensional integrals
H2b1 One or more hyper-rectangular regions (includes iterated integrals)
H2b1a Integrand available via user-defined procedure
H2b1a1 Automatic (user need only specify required accuracy)

- `D01DAF` Two-dimensional quadrature, finite region
- `D01EAF` Multi-dimensional adaptive quadrature over hyper-rectangle, multiple integrands
- `D01FCF` Multi-dimensional adaptive quadrature over hyper-rectangle
- `D01GBF` Multi-dimensional quadrature over hyper-rectangle, Monte Carlo method

H2b1a2 Nonautomatic

- `D01FBF` Multi-dimensional Gaussian quadrature over hyper-rectangle
- `D01FDF` Multi-dimensional quadrature, Sag–Szekeres method, general product region or n-sphere
- `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

H2b2 n-dimensional quadrature on a nonrectangular region
H2b2a Integrand available via user-defined procedure
H2b2a1 Automatic (user need only specify required accuracy)

- `D01JAF` Multi-dimensional quadrature over an n-sphere, allowing for badly-behaved integrands

H2b2a2 Nonautomatic

- `D01PAF` Multi-dimensional quadrature over an n-simplex

H2c Service routines (e.g., compute weights and nodes for quadrature formulas)

- `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
- `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

I Differential and integral equations
I1 Ordinary differential equations (ODE's)
I1a Initial value problems
I1a1 General, nonstiff or mildly stiff
I1a1a One-step methods (e.g., Runge–Kutta)

- `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)
- `D02LAF` Second-order ODEs, IVP, Runge–Kutta–Nystrom method
- `D02PCF` ODEs, IVP, Runge–Kutta method, integration over range with output
- `D02PDF` ODEs, IVP, Runge–Kutta method, integration over one step

I1a1b Multistep methods (e.g., Adams predictor-corrector)

- `D02CJF` ODEs, IVP, Adams method, until function of solution is zero, intermediate output (simple driver)
- `D02QFF` ODEs, IVP, Adams method with root-finding (forward communication, comprehensive)
- `D02QGF` ODEs, IVP, Adams method with root-finding (reverse communication, comprehensive)

I1a2 Stiff and mixed algebraic- differential equations

- `D02EJF` ODEs, stiff IVP, BDF method, until function of solution is zero, intermediate output (simple driver)
- `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)
- `D03PKF` General system of first-order PDEs, coupled DAEs, method of lines, Keller box discretisation, 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

I1b Multipoint boundary value problems
I1b1 Linear

- `D02GBF` ODEs, boundary value problem, finite difference technique with deferred correction, general linear problem

- **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 first-order linear equations
- **D02TGF** nth-order linear ODEs, boundary value problem, collocation and least-squares

I1b2 Nonlinear

- **D02AGF** ODEs, boundary value problem, shooting and matching technique, allowing interior matching point, general parameters to be determined
- **D02GAF** ODEs, boundary value problem, finite difference technique with deferred correction, simple nonlinear 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
- **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
- **D02TKF** ODEs, general nonlinear boundary value problem, collocation technique

I1b3 Eigenvalue (e.g., Sturm-Liouville)

- **D02AGF** ODEs, boundary value problem, shooting and matching technique, allowing interior matching point, general parameters to be determined
- **D02HBF** ODEs, boundary value problem, shooting and matching, general parameters to be determined
- **D02KAF** Second-order Sturm–Liouville problem, regular system, finite range, eigenvalue only
- **D02KDF** Second-order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue only, user-specified break-points
- **D02KEF** Second-order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue and eigenfunction, user-specified break-points

I1c Service routines (e.g., interpolation of solutions, error handling, test programs)

- **D02LXF** Second-order ODEs, IVP, set-up for D02LAF
- **D02LYF** Second-order ODEs, IVP, diagnostics for D02LAF
- **D02LZF** Second-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
- **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
- **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
- **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
- **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

I2 Partial differential equations

I2a Initial boundary value problems

I2a1 Parabolic

I2a1a One spatial dimension

- **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

	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
	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
	D03PYF	PDEs, spatial interpolation with D03PDF or D03PJF
	D03PZF	PDEs, spatial interpolation with D03PCF, D03PEF, D03PFF, D03PHF, D03PKF, D03PLF, D03PPF, D03PRF or D03PSF
I2a1b	Two or more spatial dimensions	
	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
I2a2	Hyperbolic	
	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
	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
	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
I2b	Elliptic boundary value problems	
I2b1	Linear	
I2b1a	Second order	
I2b1a1	Poisson (Laplace) or Helmholtz equation	
I2b1a1a	Rectangular domain (or topologically rectangular in the coordinate system)	
	D03FAF	Elliptic PDE, Helmholtz equation, three-dimensional Cartesian co-ordinates
I2b1a1b	Nonrectangular domain	
	D03EAF	Elliptic PDE, Laplace's equation, two-dimensional arbitrary domain
I2b1a3	Nonseparable problems	
	D03EEF	Discretize a second-order elliptic PDE on a rectangle
I2b4	Service routines	
	D03EEF	Discretize a second-order elliptic PDE on a rectangle
	D03PYF	PDEs, spatial interpolation with D03PDF or D03PJF
	D03PZF	PDEs, spatial interpolation with D03PCF, D03PEF, D03PFF, D03PHF, D03PKF, D03PLF, D03PPF, D03PRF or D03PSF
I2b4a	Domain triangulation (*search also class P*)	
	D03MAF	Triangulation of plane region
I2b4b	Solution of discretized elliptic equations	
	D03EBF	Elliptic PDE, solution of finite difference equations by SIP, five-point two-dimensional molecule, iterate to convergence
	D03ECF	Elliptic PDE, solution of finite difference equations by SIP for seven-point three-dimensional molecule, iterate to convergence
	D03EDF	Elliptic PDE, solution of finite difference equations by a multigrid technique
	D03UAF	Elliptic PDE, solution of finite difference equations by SIP, five-point two-dimensional molecule, one iteration
	D03UBF	Elliptic PDE, solution of finite difference equations by SIP, seven-point three-dimensional molecule, one iteration
I3	Integral equations	
	D05AAF	Linear non-singular Fredholm integral equation, second kind, split kernel
	D05ABF	Linear non-singular Fredholm integral equation, second kind, smooth kernel
	D05BAF	Nonlinear Volterra convolution equation, second kind
	D05BDF	Nonlinear convolution Volterra–Abel equation, second kind, weakly singular
	D05BEF	Nonlinear convolution Volterra–Abel equation, first kind, weakly singular
	D05BWF	Generate weights for use in solving Volterra equations

E02RBF Evaluation of fitted rational function as computed by E02RAF

K6a2 Derivative evaluation

E02AHF Derivative of fitted polynomial in Chebyshev series form

E02BCF Evaluation of fitted cubic spline, function and derivatives

K6a3 Quadrature

E02AJF Integral of fitted polynomial in Chebyshev series form

E02BDF Evaluation of fitted cubic spline, definite integral

K6d Other

E02ZAF Sort two-dimensional data into panels for fitting bicubic splines

L Statistics, probability

L1 Data summarization

L1a One-dimensional data

L1a1 Raw data

G01AAF Mean, variance, skewness, kurtosis, etc, one variable, from raw data

G01ALF Computes a five-point summary (median, hinges and extremes)

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

L1a3 Grouped data

G01ADF Mean, variance, skewness, kurtosis, etc, one variable, from frequency table

L1b Two dimensional data (*search also class L1c*)

G01ABF Mean, variance, skewness, kurtosis, etc, two variables, from raw data

L1c Multi-dimensional data

L1c1 Raw data

G02BDF Correlation-like coefficients (about zero), all variables, no missing values

G02BKF Correlation-like coefficients (about zero), subset of variables, no missing values

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

L1c1b Covariance, correlation

G02BAF Pearson product-moment correlation coefficients, all variables, no missing values

G02BGF Pearson product-moment correlation coefficients, subset of variables, no missing values

G02BNF Kendall/Spearman non-parametric rank correlation coefficients, no missing values, overwriting input data

G02BQF Kendall/Spearman non-parametric rank correlation coefficients, no missing values, preserving input data

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

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

L1c2 Raw data containing missing values (*search also class L1c1*)

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

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

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

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

G02BPF Kendall/Spearman non-parametric rank correlation coefficients, casewise treatment of missing values, overwriting input data

G02BRF Kendall/Spearman non-parametric rank correlation coefficients, casewise treatment of missing values, preserving input data

L5a2t	t distribution	
	G01FBF	Computes deviates for Student's t-distribution
L5b	Multivariate	
	G01NAF	Cumulants and moments of quadratic forms in Normal variables
	G01NBF	Moments of ratios of quadratic forms in Normal variables, and related statistics
L5b1	Cumulative multivariate distribution functions, probability density functions	
L5b1n	Normal	
	G01HAF	Computes probability for the bivariate Normal distribution
	G01HBF	Computes probabilities for the multivariate Normal distribution
L6	Random number generation	
L6a	Univariate	
	G05EYF	Pseudo-random integer from reference vector
L6a2	Beta, binomial, Boolean	
	G05DZF	Pseudo-random logical (boolean) value
	G05EDF	Set up reference vector for generating pseudo-random integers, binomial distribution
	G05FEF	Generates a vector of pseudo-random numbers from a beta distribution
L6a3	Cauchy, χ^2	
	G05DFF	Pseudo-random real numbers, Cauchy distribution
	G05DHF	Pseudo-random real numbers, χ^2 distribution
L6a5	Exponential, extreme value	
	G05DBF	Pseudo-random real numbers, (negative) exponential distribution
	G05FBF	Generates a vector of random numbers from an (negative) exponential distribution
L6a6	F distribution	
	G05DKF	Pseudo-random real numbers, F-distribution
L6a7	Gamma, general (continuous, discrete), geometric	
	G05EXF	Set up reference vector from supplied cumulative distribution function or probability distribution function
	G05FFF	Generates a vector of pseudo-random numbers from a gamma distribution
L6a8	Halfnormal, hypergeometric	
	G05EFF	Set up reference vector for generating pseudo-random integers, hypergeometric distribution
L6a12	Lambda, logistic, lognormal	
	G05DCF	Pseudo-random real numbers, logistic distribution
	G05DEF	Pseudo-random real numbers, log-normal distribution
L6a14	Negative binomial, normal, normal order statistics	
	G05DDF	Pseudo-random real numbers, Normal distribution
	G05EEF	Set up reference vector for generating pseudo-random integers, negative binomial distribution
	G05FDF	Generates a vector of random numbers from a Normal distribution
L6a16	Pareto, Pascal, permutations, Poisson	
	G05DRF	Pseudo-random integer, Poisson distribution
	G05ECF	Set up reference vector for generating pseudo-random integers, Poisson distribution
	G05EHF	Pseudo-random permutation of an integer vector
L6a19	Samples, stable distribution	
	G05EJF	Pseudo-random sample from an integer vector
L6a20	t distribution, time series, triangular	
	G05DJF	Pseudo-random real numbers, Student's t-distribution
	G05EGF	Set up reference vector for univariate ARMA time series model
	G05EWF	Generate next term from reference vector for ARMA time series model
L6a21	Uniform (continuous, discrete), uniform order statistics	
	G05CAF	Pseudo-random real numbers, uniform distribution over (0,1)
	G05DAF	Pseudo-random real numbers, uniform distribution over (a,b)
	G05DYF	Pseudo-random integer from uniform distribution
	G05EBF	Set up reference vector for generating pseudo-random integers, uniform distribution
	G05FAF	Generates a vector of random numbers from a uniform distribution
L6a22	Von Mises	
	G05FSF	Generates a vector of pseudo-random variates from von Mises distribution
L6a23	Weibull	
	G05DPF	Pseudo-random real numbers, Weibull distribution
L6b	Multivariate	
	G05HDF	Generates a realisation of a multivariate time series from a VARMA model
L6b3	Contingency table, correlation matrix	
	G05GBF	Computes random correlation matrix
L6b14	Normal	
	G05EAF	Set up reference vector for multivariate Normal distribution
	G05EZF	Pseudo-random multivariate Normal vector from reference vector
L6b15	Orthogonal matrix	
	G05GAF	Computes random orthogonal matrix
L6c	Service routines (e.g., seed)	
	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
L7	Analysis of variance (including analysis of covariance)	
L7a	One-way	
L7a1	Parametric	
	G04BBF	Analysis of variance, randomized block or completely randomized 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
L7a2	Nonparametric	
	G08AFF	Kruskal–Wallis one-way analysis of variance on k samples of unequal size
L7b	Two-way (*search also class L7d*)	
	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
	G08AEF	Friedman two-way analysis of variance on k matched samples
	G08ALF	Performs the Cochran Q test on cross-classified binary data
L7c	Three-way (e.g., Latin squares) (search also class L7d)	
	G04BCF	Analysis of variance, general row and column design, treatment means and standard errors
L7d	Multi-way	
L7d1	Balanced complete data (e.g., factorial designs)	
	G04CAF	Analysis of variance, complete factorial design, treatment means and standard errors
L7d2	Balanced incomplete data	
	F04JLF	Real general Gauss–Markov linear model (including weighted least-squares)
L7f	Generate experimental designs	
	G02DAF	Fits a general (multiple) linear regression model
	G02DNF	Computes estimable function of a general linear regression model and its standard error
L7g	Service routines	
	G04EAF	Computes orthogonal polynomials or dummy variables for factor/classification variable
L8	Regression (*search also classes D5, D6, D9, G, K*)	
L8a	Simple linear (i.e., $y = b_0 + b_1x$) (*search also class L8h*)	
L8a1	Ordinary least squares	
L8a1a	Parameter estimation	
L8a1a1	Unweighted data	
	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
L8a2	L_p for p different from 2 (e.g., least absolute value, minimax)	
	E02GAF	L_1-approximation by general linear function
	E02GCF	L_∞-approximation by general linear function
L8b	Polynomial (e.g., $y = b_0 + b_1x + b_2x^2$) (*search also class L8c*)	
L8b1	Ordinary least squares	
L8b1b	Parameter estimation	
L8b1b2	Using orthogonal polynomials	
	E02ADF	Least-squares curve fit, by polynomials, arbitrary data points
L8c	Multiple linear (i.e., $y = b_0 + b_1x_1 + \ldots + b_px_p$)	
	F04JLF	Real general Gauss–Markov linear model (including weighted least-squares)
	F04JMF	Equality-constrained real linear least-squares problem
L8c1	Ordinary least squares	
L8c1a	Variable selection	
	G02ECF	Calculates R^2 and C_P values from residual sums of squares
L8c1a1	Using raw data	
	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
	G02EAF	Computes residual sums of squares for all possible linear regressions for a set of independent variables
	G02EEF	Fits a linear regression model by forward selection
L8c1b	Parameter estimation (*search also class L8c1a*)	
L8c1b1	Using raw data	
	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

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

L8c1b2 Using correlation data
G02CGF Multiple linear regression, from correlation coefficients, with constant term
G02CHF Multiple linear regression, from correlation-like coefficients, without constant term

L8c1c Analysis (*search also classes L8c1a and L8c1b*)
G02FAF Calculates standardized residuals and influence statistics

L8c1d Inference (*search also classes L8c1a and L8c1b*)
G02DNF Computes estimable function of a general linear regression model and its standard error
G02FCF Computes Durbin–Watson test statistic

L8c2 Several regressions
G02DGF Fits a general linear regression model for new dependent variable

L8c4 Robust
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

L8c6 Models based on ranks
G08RAF Regression using ranks, uncensored data
G08RBF Regression using ranks, right-censored data

L8e Nonlinear (i.e., $y = F(X,b)$) (*search also class L8h*)
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

L8e1 Ordinary least squares

L8e1b Parameter estimation (*search also class L8e1a*)
E04YCF Covariance matrix for nonlinear least-squares problem (unconstrained)
G02GAF Fits a generalized linear model with Normal errors

L8e1b1 Unweighted data, user provides no derivatives
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)
E04UNF Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

L8e1b2 Unweighted data, user provides derivatives
E04GBF Unconstrained minimum of a sum of squares, combined Gauss–Newton and quasi-Newton algorithm using first derivatives (comprehensive)
E04GDF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using first derivatives (comprehensive)
E04GYF Unconstrained minimum of a sum of squares, combined Gauss–Newton and quasi-Newton algorithm, using first derivatives (easy-to-use)
E04GZF Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using first derivatives (easy-to-use)
E04UNF Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)

L8g Spline (i.e., piecewise polynomial)
E02BAF Least-squares curve cubic spline fit (including interpolation)
E02BEF Least-squares cubic spline curve fit, automatic knot placement
G10ABF Fit cubic smoothing spline, smoothing parameter given
G10ACF Fit cubic smoothing spline, smoothing parameter estimated

L8h EDA (e.g., smoothing)
G10CAF Compute smoothed data sequence using running median smoothers

L8i Service routines (e.g., matrix manipulation for variable selection)
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
G04EAF Computes orthogonal polynomials or dummy variables for factor/classification variable
G10ZAF Reorder data to give ordered distinct observations

L9 Categorical data analysis
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

G11CAF Returns parameter estimates for the conditional analysis of stratified data
G12ZAF Creates the risk sets associated with the Cox proportional hazards model for fixed covariates

L9b Two-way tables (*search also class L9d*)
G01AFF Two-way contingency table analysis, with χ^2/Fisher's exact test
G11AAF χ^2 statistics for two-way contingency table

L9c Log-linear model
G02GCF Fits a generalized linear model with Poisson 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

L10 Time series analysis (*search also class J*)
L10a Univariate (*search also classes L3a6 and L3a7*)
L10a1 Transformations
L10a1c Filters (*search also class K5*)
L10a1c1 Difference
G13AAF Univariate time series, seasonal and non-seasonal differencing

L10a1c4 Other
G13BBF Multivariate time series, filtering by a transfer function model

L10a2 Time domain analysis
L10a2a Summary statistics
G13AUF Computes quantities needed for range-mean or standard deviation-mean plot

L10a2a1 Autocorrelations and autocovariances
G13ABF Univariate time series, sample autocorrelation function

L10a2a2 Partial autocorrelations
G13ACF Univariate time series, partial autocorrelations from autocorrelations

L10a2b Stationarity analysis (*search also class L10a2a*)
G13AUF Computes quantities needed for range-mean or standard deviation-mean plot

L10a2c Autoregressive models
L10a2c1 Model identification
G13ACF Univariate time series, partial autocorrelations from autocorrelations

L10a2d ARMA and ARIMA models (including Box–Jenkins methods)
L10a2d1 Model identification
G13ADF Univariate time series, preliminary estimation, seasonal ARIMA model

L10a2d2 Parameter estimation
G13AEF Univariate time series, estimation, seasonal ARIMA model (comprehensive)
G13AFF Univariate time series, estimation, seasonal ARIMA model (easy-to-use)
G13ASF Univariate time series, diagnostic checking of residuals, following G13AEF or G13AFF
G13BEF Multivariate time series, estimation of multi-input model

L10a2d3 Forecasting
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

L10a2e State-space analysis (e.g., Kalman filtering)
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

L10a2f Analysis of a locally stationary series
G13DXF Calculates the zeros of a vector autoregressive (or moving average) operator

L10a3 Frequency domain analysis (*search also class J1*)
L10a3a Spectral analysis
L10a3a3 Spectrum estimation using the periodogram
G13CBF Univariate time series, smoothed sample spectrum using spectral smoothing by the trapezium frequency (Daniell) window

L10a3a4 Spectrum estimation using the Fourier transform of the autocorrelation function
G13CAF Univariate time series, smoothed sample spectrum using rectangular, Bartlett, Tukey or Parzen lag window

L10b Two time series (*search also classes L3b3c, L10c, and L10d*)
L10b2 Time domain analysis
L10b2a Summary statistics (e.g., cross-correlations)
G13BCF Multivariate time series, cross-correlations

L10b2b Transfer function models
G13BAF Multivariate time series, filtering (pre-whitening) by an ARIMA model
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

L10b3 Frequency domain analysis (*search also class J1*)

L10b3a Cross-spectral analysis

L10b3a3 Cross-spectrum estimation using the cross-periodogram

- `G13CDF` Multivariate time series, smoothed sample cross spectrum using spectral smoothing by the trapezium frequency (Daniell) window

L10b3a4 Cross-spectrum estimation using the Fourier transform of the cross-correlation or cross-covariance function

- `G13CCF` Multivariate time series, smoothed sample cross spectrum using rectangular, Bartlett, Tukey or Parzen lag window

L10b3a6 Spectral functions

- `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

L10c Multivariate time series (*search also classes J1, L3e3 and L10b*)

- `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

L12 Discriminant analysis

- `G03ACF` Performs canonical variate analysis
- `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)

L13 Covariance structure models

L13a Factor analysis

- `G03BAF` Computes orthogonal rotations for loading matrix, generalized orthomax criterion
- `G03BCF` Computes Procrustes rotations
- `G03CAF` Computes 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)
- `G11SAF` Contingency table, latent variable model for binary data

L13b Principal components analysis

- `G03AAF` Performs principal component analysis

L13c Canonical correlation

- `G03ACF` Performs canonical variate analysis
- `G03ADF` Performs canonical correlation analysis

L14 Cluster analysis

L14a One-way

L14a1 Unconstrained

L14a1a Nested

L14a1a1 Joining (e.g., single link)

- `G03ECF` Hierarchical cluster analysis
- `G03EHF` Constructs dendrogram (for use after G03ECF)
- `G03EJF` Computes cluster indicator variable (for use after G03ECF)

L14a1b Non-nested (e.g., K means)

- `G03EFF` K-means cluster analysis

L14d Service routines (e.g., compute distance matrix)

- `G03EAF` Computes distance matrix

L15 Life testing, survival analysis

- `G11CAF` Returns parameter estimates for the conditional analysis of stratified data
- `G12AAF` Computes Kaplan–Meier (product-limit) estimates of survival probabilities
- `G12BAF` Fits Cox's proportional hazard model

L16 Multidimensional scaling

- `G03FAF` Performs principal co-ordinate analysis, classical metric scaling
- `G03FCF` Performs non-metric (ordinal) multidimensional scaling

M Simulation, stochastic modelling (*search also classes L6 and L10*)

N Data handling (*search also class L2*)

N1 Input, output

- `X04ACF` Open unit number for reading, writing or appending, and associate unit with named file

	X04ADF	Close file associated with given unit number
	X04BAF	Write formatted record to external file
	X04BBF	Read formatted record from external file
	X04CAF	Print real general matrix (easy-to-use)
	X04CBF	Print real general matrix (comprehensive)
	X04CCF	Print real packed triangular matrix (easy-to-use)
	X04CDF	Print real packed triangular matrix (comprehensive)
	X04CEF	Print real packed banded matrix (easy-to-use)
	X04CFF	Print real packed banded matrix (comprehensive)
	X04DAF	Print complex general matrix (easy-to-use)
	X04DBF	Print complex general matrix (comprehensive)
	X04DCF	Print complex packed triangular matrix (easy-to-use)
	X04DDF	Print complex packed triangular matrix (comprehensive)
	X04DEF	Print complex packed banded matrix (easy-to-use)
	X04DFF	Print complex packed banded matrix (comprehensive)
	X04EAF	Print integer matrix (easy-to-use)
	X04EBF	Print integer matrix (comprehensive)
N4	Storage management (e.g., stacks, heaps, trees)	
	F06EUF	(SGTHR/DGTHR) Gather real sparse vector
	F06EVF	(SGTHRZ/DGTHRZ) Gather and set to zero real sparse vector
	F06EWF	(SSCTR/DSCTR) Scatter real sparse vector
	F06GUF	(CGTHR/ZGTHR) Gather complex sparse vector
	F06GVF	(CGTHRZ/ZGTHRZ) Gather and set to zero complex sparse vector
	F06GWF	(CSCTR/ZSCTR) Scatter complex sparse vector
N5	Searching	
N5a	Extreme value	
	F06FLF	Elements of real vector with largest and smallest absolute value
	F06JLF	(ISAMAX/IDAMAX) Index, real vector element with largest absolute value
	F06JMF	(ICAMAX/IZAMAX) Index, complex vector element with largest absolute value
	F06KLF	Last non-negligible element of real vector
N6	Sorting	
N6a	Internal	
N6a1	Passive (i.e., construct pointer array, rank)	
	M01DZF	Rank arbitrary data
N6a1a	Integer	
	M01DBF	Rank a vector, integer numbers
	M01DFF	Rank rows of a matrix, integer numbers
	M01DKF	Rank columns of a matrix, integer numbers
N6a1b	Real	
	G01DHF	Ranks, Normal scores, approximate Normal scores or exponential (Savage) scores
	M01DAF	Rank a vector, real numbers
	M01DEF	Rank rows of a matrix, real numbers
	M01DJF	Rank columns of a matrix, real numbers
N6a1c	Character	
	M01DCF	Rank a vector, character data
N6a2	Active	
N6a2a	Integer	
	M01CBF	Sort a vector, integer numbers
N6a2b	Real	
	M01CAF	Sort a vector, real numbers
N6a2c	Character	
	M01CCF	Sort a vector, character data
N8	Permuting	
	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
	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
	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
	M01EDF	Rearrange a vector according to given ranks, complex numbers
	M01ZAF	Invert a permutation
	M01ZBF	Check validity of a permutation
	M01ZCF	Decompose a permutation into cycles
P	Computational geometry (*search also classes G and Q*)	
	D03MAF	Triangulation of plane region

Q	Graphics (*search also class L3*)		
		G01ARF	Constructs a stem and leaf plot
		G01ASF	Constructs a box and whisker plot
R	Service routines		
		A00AAF	Prints details of the NAG Fortran Library implementation
		X05AAF	Return date and time as an array of integers
		X05ABF	Convert array of integers representing date and time to character string
		X05ACF	Compare two character strings representing date and time
		X05BAF	Return the CPU time
R1	Machine-dependent constants		
		X01AAF	Provides the mathematical constant π
		X01ABF	Provides the mathematical constant γ (Euler's Constant)
		X02AHF	The largest permissible 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
R3	Error handling		
R3b	Set unit number for error messages		
		X04AAF	Return or set unit number for error messages
		X04ABF	Return or set unit number for advisory messages
R3c	Other utilities		
		P01ABF	Return value of error indicator/terminate with error message

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.

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 complex number
A02ABF	2	Modulus of 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

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:

$$f(z) \equiv a_0 z^n + a_1 z^{n-1} + a_2 z^{n-2} + \ldots + a_{n-1} z + a_n, \qquad a_0 \neq 0.$$

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:

$$f(z) \equiv a_0 (z - z_1)^{m_1} (z - z_2)^{m_2} \ldots (z - z_k)^{m_k}, \qquad k \leq n,$$

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

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 system of nonlinear equations using function values only (reverse communication)
C05PBF	9	Solution of system of nonlinear equations using first derivatives (easy-to-use)
C05PCF	9	Solution of system of nonlinear equations using first derivatives (comprehensive)
C05PDF	14	Solution of system of nonlinear equations using first derivatives (reverse communication)
C05ZAF	9	Check user's routine for calculating first 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 $J_{ij}(x) = \left(\frac{\partial f_i}{\partial x_j}\right)$ 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

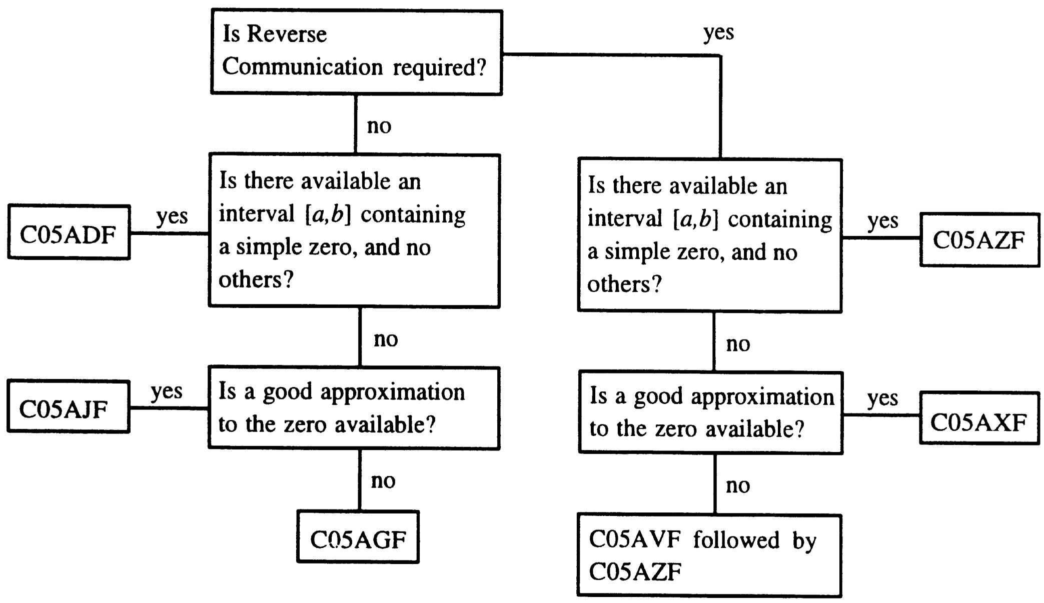
$$J(x)\sigma = -\epsilon,$$

(see Chapter F04).

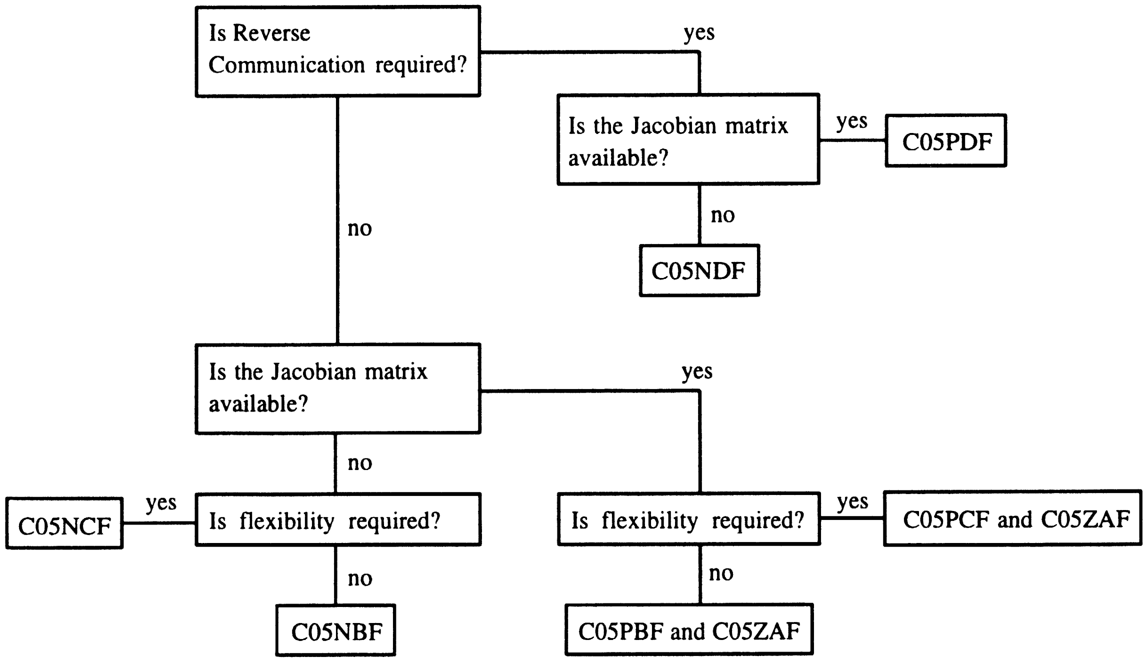
Thus one can estimate the sensitivity of x to any uncertainties in the specification of $f_i(x)$, for $i = 1, 2, \ldots, n$. As noted above, the sophisticated routines C05NCF and C05PCF (or, in reverse communication, C05NDF and C05PDF) provide the QR decomposition of J.

4 Decision Trees

(i) Functions of One Variable



(ii) Functions of Several Variables



5 Index

Zeros of functions of one variable:	
Direct communication:	
binary search followed by Bus and Dekker algorithm	C05AGF
Bus and Dekker algorithm	C05ADF
continuation method	C05AJF
Reverse communication:	
binary search	C05AVF
Bus and Dekker algorithm	C05AZF
continuation method	C05AXF
Zeros of functions of several variables:	
Direct communication:	
easy-to-use	C05NBF
easy-to-use, derivatives required	C05PBF
sophisticated	C05NCF
sophisticated, derivatives required	C05PCF
Reverse Communication:	
sophisticated	C05NDF
sophisticated, derivatives required	C05PDF
Checking Routine:	
Checks user-supplied Jacobian	C05ZAF

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

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 one-dimensional real discrete Fourier transform, no extra workspace
C06EBF	8	Single one-dimensional Hermitian discrete Fourier transform, no extra workspace
C06ECF	8	Single one-dimensional complex discrete Fourier transform, no extra workspace
C06EKF	11	Circular convolution or correlation of two real vectors, no extra workspace
C06FAF	8	Single one-dimensional real discrete Fourier transform, extra workspace for greater speed
C06FBF	8	Single one-dimensional Hermitian discrete Fourier transform, extra workspace for greater speed
C06FCF	8	Single one-dimensional complex discrete Fourier transform, extra workspace for greater speed
C06FFF	11	One-dimensional 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 one-dimensional real discrete Fourier transforms
C06FQF	12	Multiple one-dimensional Hermitian discrete Fourier transforms
C06FRF	12	Multiple one-dimensional complex discrete Fourier transforms
C06FUF	13	Two-dimensional complex discrete Fourier transform
C06FXF	17	Three-dimensional 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
C06PAF	19	Single one-dimensional real and Hermitian complex discrete Fourier transform, using complex data format for Hermitian sequences
C06PCF	19	Single one-dimensional complex discrete Fourier transform, complex data format
C06PFF	19	One-dimensional complex discrete Fourier transform of multi-dimensional data (using complex data type)
C06PJF	19	Multi-dimensional complex discrete Fourier transform of multi-dimensional data (using complex data type)
C06PKF	19	Circular convolution or correlation of two complex vectors
C06PPF	19	Multiple one-dimensional real and Hermitian complex discrete Fourier transforms, using complex data format for Hermitian sequences

C06PQF	19	Multiple one-dimensional real and Hermitian complex discrete Fourier transforms, using complex data format for Hermitian sequences and sequences stored as columns
C06PRF	19	Multiple one-dimensional complex discrete Fourier transforms using complex data format
C06PSF	19	Multiple one-dimensional complex discrete Fourier transforms using complex data format and sequences stored as columns
C06PUF	19	Two-dimensional complex discrete Fourier transform, complex data format
C06PXF	19	Three-dimensional complex discrete Fourier transform, complex data format
C06RAF	19	Discrete sine transform (easy-to-use)
C06RBF	19	Discrete cosine transform (easy-to-use)
C06RCF	19	Discrete quarter-wave sine transform (easy-to-use)
C06RDF	19	Discrete quarter-wave cosine transform (easy-to-use)

Chapter C06

Summation of Series

Contents

1 Scope of the Chapter

This chapter is concerned with the following tasks.

(a) Calculating the **discrete Fourier transform** of a sequence of real or complex data values.
(b) Calculating the **discrete convolution** or the **discrete correlation** of two sequences of real or complex data values using discrete Fourier transforms.
(c) Calculating the **inverse Laplace transform** of a user-supplied function.
(d) Direct summation of orthogonal series.
(e) 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 $\hat{z}_{-k} = \hat{z}_{n-k}$. Note also that 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.

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. (Note that the terms **forward** transform and **backward** transform are also used to mean the direct and inverse transforms respectively.)

The definition (1) of a one-dimensional transform can easily be extended to multi-dimensional transforms. For example, in two dimensions we have

$$\hat{z}_{k_1 k_2} = \frac{1}{\sqrt{n_1 n_2}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} z_{j_1 j_2} \exp\left(-i\frac{2\pi j_1 k_1}{n_1}\right) \exp\left(-i\frac{2\pi j_2 k_2}{n_2}\right).$$

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.

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, with two schemes being used in this chapter. In the first scheme, which will be referred to as the **real storage format** for Hermitian sequences, the real parts a_k for $0 \le k \le n/2$ 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. To clarify, if X is declared with bounds $(0{:}n-1)$ in your calling (sub)program, the following two tables illustrate the storage of the real and imaginary parts of $\hat{z}_k$ for the two cases: n even and n odd.

If n is even then the sequence has two purely real elements and is stored as follows:

Index of X	0	1	2	...	$n/2$	...	$n-2$	$n-1$
Sequence	a_0	$a_1 + ib_1$	$a_2 + ib_2$	...	$a_{n/2}$	...	$a_2 - ib_2$	$a_1 - ib_1$
Stored values	a_0	a_1	a_2	...	$a_{n/2}$	...	b_2	b_1

$X(k) = a_k$, for $k = 0, 1, \ldots, n/2$, and
$X(n-k) = b_k$, for $k = 1, 2, \ldots, n/2-1$.

If n is odd then the sequence has one purely real element and, letting $n = 2s+1$, is stored as follows:

Index of X	0	1	2	...	s	$s+1$	...	$n-2$	$n-1$
Sequence	a_0	$a_1 + ib_1$	$a_2 + ib_2$	...	$a_s + ib_s$	$a_s - ib_s$	...	$a_2 - ib_2$	$a_1 - ib_1$
Stored values	a_0	a_1	a_2	...	a_s	b_s	...	b_2	b_1

$X(k) = a_k$, for $k = 0, 1, \ldots, s$, and
$X(n-k) = b_k$, for $k = 1, 2, \ldots, s$.

The second storage scheme, referred to in this chapter as the **complex storage format** for Hermitian sequences, stores the real and imaginary parts a_k, b_k, for $0 \le k \le n/2$, in consecutive locations of an array X of length $n+2$. If X is declared with bounds $(0{:}n+1)$ in your calling (sub)program, the following two tables illustrate the storage of the real and imaginary parts of $\hat{z}_k$ for the two cases: n even and n odd.

If n is even then the sequence has two purely real elements and is stored as follows:

Index of X	0	1	2	3	...	$n-2$	$n-1$	n	$n+1$
Stored values	a_0	$b_0 = 0$	a_1	b_1	...	$a_{n/2-1}$	$b_{n/2-1}$	$a_{n/2}$	$b_{n/2} = 0$

$X(2*k) = a_k$, for $k = 0, 1, \ldots, n/2$, and
$X(2*k+1) = b_k$, for $k = 0, 1, \ldots, n/2$.

If n is odd then the sequence has one purely real element and, letting $n = 2s+1$, is stored as follows:

Index of X	0	1	2	3	...	$n-2$	$n-1$	n	$n+1$
Stored values	a_0	$b_0 = 0$	a_1	b_1	...	b_{s-1}	a_s	b_s	0

$X(2*k) = a_k$, for $k = 0,1,\ldots,s$, and
$X(2*k+1) = b_k$, for $k = 0,1,\ldots,s$.

Also, given a Hermitian sequence, the inverse (or backward) discrete transform produces a real sequence. That is,

$$x_j = \frac{1}{\sqrt{n}}\left(a_0 + 2\sum_{k=1}^{n/2-1}\left(a_k \cos\left(\frac{2\pi jk}{n}\right) - b_k \sin\left(\frac{2\pi jk}{n}\right)\right) + a_{n/2}\right)$$

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 (PDEs)

A further application of the fast Fourier transform, and in particular of the Fourier transforms of symmetric sequences, is in the solution of elliptic PDEs. 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 PDEs 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.,

$$|f(t)| \le Me^{\alpha t}$$

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 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. The choice is next determined by the users preferred storage format; where it is preferred for complex sequences to be stored in two separate real arrays or for Hermitian sequences to be stored in real storage format (see Section 2.1.2) then a real storage format routine should be used; where it is preferred for complex data to be stored in complex arrays or for Hermitian sequences to be stored in complex storage format then a complex storage format routine should be used.

Note also that the complex storage format routines have a reduced parameter list: there are no INIT or TRIG parameters.

Three groups, each of three routines, are provided in real storage format and three groups of two routines are provided in complex storage format.

	Group 1	Group 2	Group 3	Group 4
Real storage format				
Real sequences	C06EAF	C06FAF	C06FPF	
Hermitian sequences	C06EBF	C06FBF	C06FQF	
General complex sequences	C06ECF	C06FCF	C06FRF	
Complex storage format				
Real/Hermitian sequences		C06PAF	C06PPF	C06PQF
General complex sequences		C06PCF	C06PRF	C06PSF

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*** (***complex*** for C06PCF) work-array. 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 some Group 2 routines (C06FAF, C06FBF and C06FCF) 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 and Group 4 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 and Group 4 routines differ in the way sequences are stored: Group 3 routines store sequences as rows of a two-dimensional array while Group 4 routines store sequences as columns of a two-dimensional array.

Group 3 and Group 4 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 (backward) discrete Fourier transforms the real storage format 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. In the case of complex storage format routines, there is a **direction** parameter which determines the direction of the transform; a call to such a routine in the forward direction followed by a call in the backward direction reproduces the original data.

3.2 Half- and Quarter-wave Transforms

Eight routines are provided for computing fast Fourier transforms (FFTs) of real symmetric sequences. C06HAF and C06RAF compute multiple Fourier sine transforms, C06HBF and C06RBF compute multiple Fourier cosine transforms, C06HCF and C06RCF compute multiple quarter-wave Fourier sine transforms, and C06HDF and C06RDF compute multiple quarter-wave Fourier cosine transforms. There are two routines for each type of transform; the routines C06RAF, C06RBF, C06RCF and C06RDF have shorter parameter lists than their counterparts and are therefore simpler to use.

3.3 Application to Elliptic Partial Differential Equations

As described in Section 2.1, Fourier transforms may be used in the solution of elliptic PDEs.

C06HAF and C06RAF may be used to solve equations where the solution is specified along the boundary.

C06HBF and C06RBF may be used to solve equations where the derivative of the solution is specified along the boundary.

C06HCF and C06RCF 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 and C06RDF 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:

	Real storage	Complex storage
2 dimensions	C06FUF	C06PUF
3 dimensions	C06FXF	C06PXF
any number of dimensions	C06FJF	C06PJF

The real storage format routines store sequences of complex data in two ***real*** arrays containing the real and imaginary parts of the sequence respectively. The complex storage format routines store the sequences in ***complex*** arrays.

Note that complex storage format routines have a reduced parameter list, having no INIT or TRIG parameters.

C06FUF (C06PUF) and C06FXF (C06PXF) should be used in preference to C06FJF (C06PJF) for two- and three-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. C06PKF computes either the discrete convolution or the discrete correlation of two complex vectors.

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

The only routine available is C06DBF, which sums a finite Chebyshev series

$$\sum_{j=0}^{n} c_j T_j(x), \quad \sum_{j=0}^{n} c_j T_{2j}(x) \quad \text{or} \quad \sum_{j=0}^{n} c_j T_{2j+1}(x)$$

depending on the choice of a parameter.

3.8 Acceleration of Convergence

The only routine available is C06BAF.

4 Index

complex sequence, complex storage	C06PUF
three-dimensional	
complex sequence, real storage	C06FXF
complex sequence, complex storage	C06PXF
one-dimensional, multi-variable	
complex sequence, real storage	C06FFF
complex sequence, complex storage	C06PFF
one-dimensional, multiple transforms	
complex sequence, real storage by rows	C06FRF
complex sequence, complex storage by rows	C06PRF
complex sequence, complex storage by columns	C06PSF
Hermitian sequence, real storage by rows	C06FQF
real sequence, real storage by rows	C06FPF
Hermitian/real sequence, complex storage by rows	C06PPF
Hermitian/real sequence, complex storage by columns	C06PQF
one-dimensional, single transforms	
complex sequence, space saving, real storage	C06ECF
complex sequence, time-saving, real storage	C06FCF
complex sequence, time-saving, complex storage	C06PCF
Hermitian sequence, space-saving, real storage	C06EBF
Hermitian sequence, time-saving, real storage	C06FBF
real sequence, space-saving, real storage	C06EAF
real sequence, time-saving, real storage	C06FAF
Hermitian/real sequence, time-saving, complex storage	C06PAF
half- and quarter-wave transforms	
multiple Fourier sine transforms	C06HAF
multiple Fourier sine transforms, simple use	C06RAF
multiple Fourier cosine transforms	C06HBF
multiple Fourier cosine transforms, simple use	C06RBF
multiple quarter-wave sine transforms	C06HCF
multiple quarter-wave sine transforms, simple use	C06RCF
multiple quarter-wave cosine transforms	C06HDF
multiple quarter-wave cosine transforms, simple use	C06RDF
Inverse Laplace Transform	
Crump's method	C06LAF
Weeks' method	
compute coefficients of solution	C06LBF
evaluate solution	C06LCF
Summation of Chebyshev series	C06DBF

5 Routines Withdrawn or Scheduled for Withdrawal

None since Mark 13.

6 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

Chapter D01 – Quadrature

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D01AHF	8	One-dimensional quadrature, adaptive, finite interval, strategy due to Patterson, suitable for well-behaved integrands
D01AJF	8	One-dimensional quadrature, adaptive, finite interval, strategy due to Piessens and de Doncker, allowing for badly-behaved integrands
D01AKF	8	One-dimensional quadrature, adaptive, finite interval, method suitable for oscillating functions
D01ALF	8	One-dimensional quadrature, adaptive, finite interval, allowing for singularities at user-specified break-points
D01AMF	8	One-dimensional quadrature, adaptive, infinite or semi-infinite interval
D01ANF	8	One-dimensional quadrature, adaptive, finite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01APF	8	One-dimensional quadrature, adaptive, finite interval, weight function with end-point singularities of algebraico-logarithmic type
D01AQF	8	One-dimensional quadrature, adaptive, finite interval, weight function $1/(x-c)$, Cauchy principal value (Hilbert transform)
D01ARF	10	One-dimensional quadrature, non-adaptive, finite interval with provision for indefinite integrals
D01ASF	13	One-dimensional quadrature, adaptive, semi-infinite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01ATF	13	One-dimensional quadrature, adaptive, finite interval, variant of D01AJF efficient on vector machines
D01AUF	13	One-dimensional quadrature, adaptive, finite interval, variant of D01AKF efficient on vector machines
D01BAF	7	One-dimensional Gaussian quadrature
D01BBF	7	Pre-computed weights and abscissae for Gaussian quadrature rules, restricted choice of rule
D01BCF	8	Calculation of weights and abscissae for Gaussian quadrature rules, general choice of rule
D01BDF	8	One-dimensional quadrature, non-adaptive, finite interval
D01DAF	5	Two-dimensional quadrature, finite region
D01EAF	12	Multi-dimensional adaptive quadrature over hyper-rectangle, multiple integrands
D01FBF	8	Multi-dimensional Gaussian quadrature over hyper-rectangle
D01FCF	8	Multi-dimensional adaptive quadrature over hyper-rectangle
D01FDF	10	Multi-dimensional quadrature, Sag–Szekeres method, general product region or n-sphere
D01GAF	5	One-dimensional quadrature, integration of function defined by data values, Gill–Miller method
D01GBF	10	Multi-dimensional quadrature over hyper-rectangle, Monte Carlo method
D01GCF	10	Multi-dimensional quadrature, general product region, number-theoretic method
D01GDF	14	Multi-dimensional quadrature, general product region, number-theoretic method, variant of D01GCF efficient on vector machines
D01GYF	10	Korobov optimal coefficients for use in D01GCF or D01GDF, when number of points is prime

D01GZF	10	Korobov optimal coefficients for use in D01GCF or D01GDF, when number of points is product of two primes
D01JAF	10	Multi-dimensional quadrature over an n-sphere, allowing for badly-behaved integrands
D01PAF	10	Multi-dimensional quadrature over an n-simplex

Chapter D01

Quadrature

Contents

1 Scope of the Chapter

This chapter provides routines for the numerical evaluation of definite integrals in one or more dimensions and for evaluating weights and abscissae of integration rules.

2 Background to the Problems

The routines in this chapter are designed to estimate:

(a) the value of a one-dimensional definite integral of the form:

$$\int_a^b f(x)\,dx \tag{1}$$

where $f(x)$ is defined by the user, either at a set of points $(x_i, f(x_i))$, for $i = 1,2,\ldots,n$ where $a = x_1 < x_2 < \ldots < x_n = b$, or in the form of a function; and the limits of integration a, b may be finite or infinite.

Some methods are specially designed for integrands of the form

$$f(x) = w(x)g(x) \tag{2}$$

which contain a factor $w(x)$, called the weight-function, of a specific form. These methods take full account of any peculiar behaviour attributable to the $w(x)$ factor.

(b) the values of the one-dimensional indefinite integrals arising from (1) where the ranges of integration are interior to the interval $[a,b]$.

(c) the value of a multi-dimensional definite integral of the form:

$$\int_{R_n} f(x_1,x_2,\ldots,x_n)\,dx_n\ \ldots\ dx_2\,dx_1 \tag{3}$$

where $f(x_1,x_2,\ldots,x_n)$ is a function defined by the user and R_n is some region of n-dimensional space.

The simplest form of R_n is the n-rectangle defined by

$$a_i \leq x_i \leq b_i, \qquad i = 1,2,\ldots,n \tag{4}$$

where a_i and b_i are constants. When a_i and b_i are functions of x_j $(j < i)$, the region can easily be transformed to the rectangular form (see Davis and Rabinowitz [1], page 266). Some of the methods described incorporate the transformation procedure.

2.1 One-dimensional Integrals

To estimate the value of a one-dimensional integral, a quadrature rule uses an approximation in the form of a weighted sum of integrand values, i.e.,

$$\int_a^b f(x)\,dx \simeq \sum_{i=1}^{N} w_i f(x_i). \tag{5}$$

The points x_i within the interval $[a,b]$ are known as the abscissae, and the w_i are known as the weights.

More generally, if the integrand has the form (2), the corresponding formula is

$$\int_a^b w(x)g(x)\,dx \simeq \sum_{i=1}^{N} w_i g(x_i). \tag{6}$$

If the integrand is known only at a fixed set of points, these points must be used as the abscissae, and the weighted sum is calculated using finite-difference methods. However, if the functional form of the integrand is known, so that its value at any abscissa is easily obtained, then a wide variety of quadrature rules are available, each characterised by its choice of abscissae and the corresponding weights.

The appropriate rule to use will depend on the interval $[a, b]$ – whether finite or otherwise – and on the form of any $w(x)$ factor in the integrand. A suitable value of N depends on the general behaviour of $f(x)$; or of $g(x)$, if there is a $w(x)$ factor present.

Among possible rules, we mention particularly the Gaussian formulae, which employ a distribution of abscissae which is optimal for $f(x)$ or $g(x)$ of polynomial form.

The choice of basic rules constitutes one of the principles on which methods for one-dimensional integrals may be classified. The other major basis of classification is the implementation strategy, of which some types are now presented.

(a) Single rule evaluation procedures

A fixed number of abscissae, N, is used. This number and the particular rule chosen uniquely determine the weights and abscissae. No estimate is made of the accuracy of the result.

(b) Automatic procedures

The number of abscissae, N, within $[a, b]$ is gradually increased until consistency is achieved to within a level of accuracy (absolute or relative) requested by the user. There are essentially two ways of doing this; hybrid forms of these two methods are also possible:

(i) whole interval procedures (non-adaptive)

A series of rules using increasing values of N are successively applied over the whole interval $[a, b]$. It is clearly more economical if abscissae already used for a lower value of N can be used again as part of a higher-order formula. This principle is known as **optimal extension**. There is no overlap between the abscissae used in Gaussian formulae of different orders. However, the Kronrod formulae are designed to give an optimal $(2N + 1)$-point formula by adding $(N + 1)$ points to an N-point Gauss formula. Further extensions have been developed by Patterson.

(ii) adaptive procedures

The interval $[a, b]$ is repeatedly divided into a number of sub-intervals, and integration rules are applied separately to each sub-interval. Typically, the subdivision process will be carried further in the neighbourhood of a sharp peak in the integrand, than where the curve is smooth. Thus, the distribution of abscissae is adapted to the shape of the integrand.

Subdivision raises the problem of what constitutes an acceptable accuracy in each sub-interval. The usual **global acceptability criterion** demands that the sum of the absolute values of the error estimates in the sub-intervals should meet the conditions required of the error over the whole interval. Automatic extrapolation over several levels of subdivision may eliminate the effects of some types of singularities.

An ideal general-purpose method would be an automatic method which could be used for a wide variety of integrands, was efficient (i.e., required the use of as few abscissae as possible), and was reliable (i.e., always gave results to within the requested accuracy). Complete reliability is unobtainable, and generally higher reliability is obtained at the expense of efficiency, and vice versa. **It must therefore be emphasised that the automatic routines in this chapter cannot be assumed to be 100% reliable. In general, however, the reliability is very high.**

2.2 Multi-dimensional Integrals

A distinction must be made between cases of moderately low dimensionality (say, up to 4 or 5 dimensions), and those of higher dimensionality. Where the number of dimensions is limited, a one-dimensional method may be applied to each dimension, according to some suitable strategy, and high accuracy may be obtainable (using product rules). However, the number of integrand evaluations rises very rapidly with the number of dimensions, so that the accuracy obtainable with an acceptable amount of computational labour is limited; for example a product of 3-point rules in 20 dimensions would require more than 10^9 integrand evaluations. Special techniques such as the Monte Carlo methods can be used to deal with high dimensions.

(a) Products of one-dimensional rules

Using a two-dimensional integral as an example, we have

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2} f(x,y)\,dy\,dx \simeq \sum_{i=1}^{N} w_i \left[\int_{a_2}^{b_2} f(x_i,y)\,dy\right] \tag{7}$$

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2} f(x,y)\,dy\,dx \simeq \sum_{i=1}^{N}\sum_{j=1}^{N} w_i v_j f(x_i,y_j) \tag{8}$$

where (w_i,x_i) and (v_i,y_i) are the weights and abscissae of the rules used in the respective dimensions.

A different one-dimensional rule may be used for each dimension, as appropriate to the range and any weight function present, and a different strategy may be used, as appropriate to the integrand behaviour as a function of each independent variable.

For a rule-evaluation strategy in all dimensions, the formula (8) is applied in a straightforward manner. For automatic strategies (i.e., attempting to attain a requested accuracy), there is a problem in deciding what accuracy must be requested in the inner integral(s). Reference to formula (7) shows that the presence of a limited but random error in the y-integration for different values of x_i can produce a 'jagged' function of x, which may be difficult to integrate to the desired accuracy and for this reason products of automatic one-dimensional routines should be used with caution (see also Lyness [3]).

(b) Monte Carlo methods

These are based on estimating the mean value of the integrand sampled at points chosen from an appropriate statistical distribution function. Usually a variance reducing procedure is incorporated to combat the fundamentally slow rate of convergence of the rudimentary form of the technique. These methods can be effective by comparison with alternative methods when the integrand contains singularities or is erratic in some way, but they are of quite limited accuracy.

(c) Number theoretic methods

These are based on the work of Korobov and Conroy and operate by exploiting implicitly the properties of the Fourier expansion of the integrand. Special rules, constructed from so-called optimal coefficients, give a particularly uniform distribution of the points throughout n-dimensional space and from their number theoretic properties minimize the error on a prescribed class of integrals. The method can be combined with the Monte Carlo procedure.

(d) Sag–Szekeres method

By transformation this method seeks to induce properties into the integrand which make it accurately integrable by the trapezoidal rule. The transformation also allows effective control over the number of integrand evaluations.

(e) Automatic adaptive procedures

An automatic adaptive strategy in several dimensions normally involves division of the region into subregions, concentrating the divisions in those parts of the region where the integrand is worst behaved. It is difficult to arrange with any generality for variable limits in the inner integral(s). For this reason, some methods use a region where all the limits are constants; this is called a hyper-rectangle. Integrals over regions defined by variable or infinite limits may be handled by transformation to a hyper-rectangle. Integrals over regions so irregular that such a transformation is not feasible may be handled by surrounding the region by an appropriate hyper-rectangle and defining the integrand to be zero outside the desired region. Such a technique should always be followed by a Monte Carlo method for integration.

The method used locally in each subregion produced by the adaptive subdivision process is usually one of three types: Monte Carlo, number theoretic or deterministic. Deterministic methods are usually the most rapidly convergent but are often expensive to use for high dimensionality and not as robust as the other techniques.

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 following three sub-sections consider in turn routines for: one-dimensional integrals over a finite interval, and over a semi-infinite or an infinite interval; and multi-dimensional integrals. Within each sub-section, routines are classified by the type of method, which ranges from simple rule evaluation to automatic adaptive algorithms. The recommendations apply particularly when the primary objective is simply to compute the value of one or more integrals, and in these cases the automatic adaptive routines are generally the most convenient and reliable, although also the most expensive in computing time.

Note however that in some circumstances it may be counter-productive to use an automatic routine. If the results of the quadrature are to be used in turn as input to a further computation (e.g. an 'outer' quadrature or an optimization problem), then this further computation may be adversely affected by the 'jagged performance profile' of an automatic routine; a simple rule-evaluation routine may provide much better overall performance. For further guidance, the article by Lyness [3] is recommended.

3.1 One-dimensional Integrals over a Finite Interval

(a) Integrand defined at a set of points

If $f(x)$ is defined numerically at four or more points, then the Gill–Miller finite difference method (D01GAF) should be used. The interval of integration is taken to coincide with the range of x-values of the points supplied. It is in the nature of this problem that any routine may be unreliable. In order to check results independently and so as to provide an alternative technique the user may fit the integrand by Chebyshev series using E02ADF and then use routines E02AJF and E02AKF to evaluate its integral (which need not be restricted to the range of the integration points, as is the case for D01GAF). A further alternative is to fit a cubic spline to the data using E02BAF and then to evaluate its integral using E02BDF.

(b) Integrand defined as a function

If the functional form of $f(x)$ is known, then one of the following approaches should be taken. They are arranged in the order from most specific to most general, hence the first applicable procedure in the list will be the most efficient. **However, if the user does not wish to make any assumptions about the integrand, the most reliable routines to use will be D01AJF (or D01ATF) and D01AHF, although these will in general be less efficient for simple integrals.**

(i) Rule-evaluation routines

If $f(x)$ is known to be sufficiently well behaved (more precisely, can be closely approximated by a polynomial of moderate degree), a Gaussian routine with a suitable number of abscissae may be used.

D01BAF may be used if it is not required to examine the weights and abscissae.

D01BBF or D01BCF with D01FBF may be used if it is required to examine the weights and abscissae.

D01BBF is faster and more accurate, whereas D01BCF is more general.

If $f(x)$ is well behaved, apart from a weight-function of the form

$$\left|x-\frac{a+b}{2}\right|^c \quad \text{or} \quad (b-x)^c(x-a)^d,$$

D01BCF with D01FBF may be used.

(ii) Automatic whole-interval routines

If $f(x)$ is reasonably smooth, and the required accuracy is not too high, the automatic whole-interval routines, D01ARF or D01BDF may be used. D01ARF incorporates high-order extensions of the Kronrod rule and is the only routine which can also be used for indefinite integration.

(iii) Automatic adaptive routines

Firstly, several routines are available for integrands of the form $w(x)g(x)$ where $g(x)$ is a 'smooth' function (i.e., has no singularities, sharp peaks or violent oscillations in the interval of integration) and $w(x)$ is a weight function of one of the following forms:

if $w(x) = (b-x)^{\alpha}(x-a)^{\beta}(\log(b-x))^{k}(\log(x-a))^{l}$, where $k,l = 0$ or 1, $\alpha,\beta > -1$: use D01APF;

if $w(x) = \frac{1}{x-c}$: use D01AQF (this integral is called the Hilbert transform of g);

if $w(x) = \cos(\omega x)$ or $\sin(\omega x)$: use D01ANF (this routine can also handle certain types of singularities in $g(x)$).

Secondly, there are some routines for general $f(x)$. If $f(x)$ is known to be free of singularities, though it may be oscillatory, D01AKF or D01AUF may be used.

The most powerful of the finite interval integration routines are D01AJF and D01ATF, which can cope with singularities of several types, and D01AHF. They may be used if none of the more specific situations described above applies. D01AHF is likely to be more efficient, whereas D01AJF and D01ATF are somewhat more reliable, particularly where the integrand has singularities other than at an end-point, or has discontinuities or cusps, and is therefore recommended where the integrand is known to be badly behaved, or where its nature is completely unknown. It may sometimes be useful to use both routines as a check.

Most of the routines in this chapter require the user to supply a function or subroutine to evaluate the integrand at a single point. D01ATF and D01AUF use the same methods as D01AJF and D01AKF respectively, but have a different user-interface which can result in faster execution, especially on vector-processing machines (see Gladwell [2]). They require the user to provide a subroutine to return an array of values of the integrand at each of an array of points. This reduces the overhead of function calls, avoids repetition of computations common to each of the integrand evaluations, and offers greater scope for vectorisation of the user's code.

If $f(x)$ has singularities of certain types, discontinuities or sharp peaks **occurring at known points**, the integral should be evaluated separately over each of the subranges or D01ALF may be used.

3.2 One-dimensional Integrals over a Semi-infinite or Infinite Interval

(a) Integrand defined at a set of points

If $f(x)$ is defined numerically at four or more points, and the portion of the integral lying outside the range of the points supplied may be neglected, then the Gill–Miller finite difference method, D01GAF, should be used.

(b) Integrand defined as a function

(i) Rule evaluation routines

If $f(x)$ behaves approximately like a polynomial in x, apart from a weight function of the form

$e^{-\beta x}, \beta > 0$ (semi-infinite interval, lower limit finite); or

$e^{-\beta x}, \beta < 0$ (semi-infinite interval, upper limit finite); or

$e^{-\beta(x-\alpha)^2}, \beta > 0$ (infinite interval);

or if $f(x)$ behaves approximately like a polynomial in $(x+b)^{-1}$ (semi-infinite range), then the Gaussian routines may be used.

D01BAF may be used if it is not required to examine the weights and abscissae.

D01BBF or D01BCF with D01FBF may be used if it is required to examine the weights and abscissae.

D01BBF is faster and more accurate, whereas D01BCF is more general.

(ii) Automatic adaptive routines

D01AMF may be used, except for integrands which decay slowly towards an infinite end-point, and oscillate in sign over the entire range. For this class, it may be possible to calculate the integral by integrating between the zeros and invoking some extrapolation process (see C06BAF).

D01ASF may be used for integrals involving weight functions of the form $\cos(\omega x)$ and $\sin(\omega x)$ over a semi-infinite interval (lower limit finite).

The following alternative procedures are mentioned for completeness, though their use will rarely be necessary.

1. If the integrand decays rapidly towards an infinite end-point, a finite cut-off may be chosen, and the finite range methods applied.
2. If the only irregularities occur in the finite part (apart from a singularity at the finite limit, with which D01AMF can cope), the range may be divided, with D01AMF used on the infinite part.
3. A transformation to finite range may be employed, e.g.

$$x = \frac{1-t}{t} \quad \text{or} \quad x = -\log_e t$$

will transform $(0,\infty)$ to (1,0) while for infinite ranges we have

$$\int_{-\infty}^{\infty} f(x)\,dx = \int_0^{\infty} [f(x) + f(-x)]\,dx.$$

If the integrand behaves badly on $(-\infty,0)$ and well on $(0,\infty)$ or vice versa it is better to compute it as $\int_{-\infty}^{0} f(x)\,dx + \int_0^{\infty} f(x)\,dx$. This saves computing unnecessary function values in the semi-infinite range where the function is well behaved.

3.3 Multi-dimensional Integrals

A number of techniques are available in this area and the choice depends to a large extent on the dimension and the required accuracy. It can be advantageous to use more than one technique as a confirmation of accuracy particularly for high dimensional integrations. Many of the routines incorporate the transformation procedure REGION which allows general product regions to be easily dealt with in terms of conversion to the standard n-cube region.

(a) Products of one-dimensional rules (suitable for up to about 5 dimensions)

If $f(x_1,x_2,\ldots,x_n)$ is known to be a sufficiently well behaved function of each variable x_i, apart possibly from weight functions of the types provided, a product of Gaussian rules may be used. These are provided by D01BBF or D01BCF with D01FBF. Rules for finite, semi-infinite and infinite ranges are included.

For two-dimensional integrals only, unless the integrand is very badly-behaved, the automatic whole-interval product procedure of D01DAF may be used. The limits of the inner integral may be user-specified functions of the outer variable. Infinite limits may be handled by transformation (see Section 3.2); end-point singularities introduced by transformation should not be troublesome, as the integrand value will not be required on the boundary of the region.

If none of these routines proves suitable and convenient, the one-dimensional routines may be used recursively. For example, the two-dimensional integral

$$I = \int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x,y)\,dy\,dx$$

may be expressed as

$$I = \int_{a_1}^{b_1} F(x)\,dx, \quad \text{where } F(x) = \int_{a_2}^{b_2} f(x,y)\,dy.$$

The user segment to evaluate $F(x)$ will call the integration routine for the y-integration, which will call another user segment for $f(x,y)$ as a function of y (x being effectively a constant). Note that, as Fortran 77 is not a recursive language, **a different library integration routine must be used for each dimension**. Apart from this restriction, the following combinations are not permitted: D01AJF and D01ALF, D01ANF and D01APF, D01APF and D01AQF, D01AQF and D01ANF, D01ASF and D01ANF, D01ASF and D01AMF, D01AUF and D01ATF. Otherwise the full range of one-dimensional routines are available, for finite/infinite intervals, constant/variable limits, rule evaluation/automatic strategies etc.

(b) Sag–Szekeres method

Two routines are based on this method.

D01FDF is particularly suitable for integrals of very large dimension although the accuracy is generally not high. It allows integration over either the general product region (with built-in transformation to the n-cube) or the n-sphere. Although no error estimate is provided, two adjustable parameters may be varied for checking purposes or may be used to tune the algorithm to particular integrals.

D01JAF is also based on the Sag–Szekeres method and integrates over the n-sphere. It uses improved transformations which may be varied according to the behaviour of the integrand. Although it can yield very accurate results it can only practically be employed for dimensions not exceeding 4.

(c) Number Theoretic method

Two routines are based on this method.

D01GCF carries out multiple integration using the Korobov–Conroy method over a product region with built-in transformation to the n-cube. A stochastic modification of this method is incorporated hybridising the technique with the Monte Carlo procedure. An error estimate is provided in terms of the statistical standard error. The routine includes a number of optimal coefficient rules for up to 20 dimensions; others can be computed using D01GYF and D01GZF. Like the Sag–Szekeres method it is suitable for large dimensional integrals although the accuracy is not high.

D01GDF uses the same method as D01GCF, but has a different interface which can result in faster execution, especially on vector-processing machines. The user is required to provide two subroutines, the first to return an array of values of the integrand at each of an array of points, and the second to evaluate the limits of integration at each of an array of points. This reduces the overhead of function calls, avoids repetitions of computations common to each of the evaluations of the integral and limits of integration, and offers greater scope for vectorization of the user's code.

(d) A combinatorial extrapolation method

D01PAF computes a sequence of approximations and an error estimate to the integral of a function over a multi-dimensional simplex using a combinatorial method with extrapolation.

(e) Automatic routines (D01GBF and D01FCF)

Both routines are for integrals of the form

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2}\cdots\int_{a_n}^{b_n} f(x_1,x_2,\ldots,x_n)\,dx_n\,dx_{n-1}\,\ldots\,dx_1.$$

D01GBF is an adaptive Monte Carlo routine. This routine is usually slow and not recommended for high-accuracy work. It is a robust routine that can often be used for low-accuracy results with highly irregular integrands or when n is large.

D01FCF is an adaptive deterministic routine. Convergence is fast for well behaved integrands. Highly accurate results can often be obtained for n between 2 and 5, using significantly fewer integrand evaluations than would be required by D01GBF. The routine will usually work when the integrand is mildly singular and for $n \le 10$ should be used before D01GBF. If it is known in advance that the integrand is highly irregular, it is best to compare results from at least two different routines.

There are many problems for which one or both of the routines will require large amounts of computing time to obtain even moderately accurate results. The amount of computing time is controlled by the number of integrand evaluations allowed by the user, and users should set this parameter carefully, with reference to the time available and the accuracy desired.

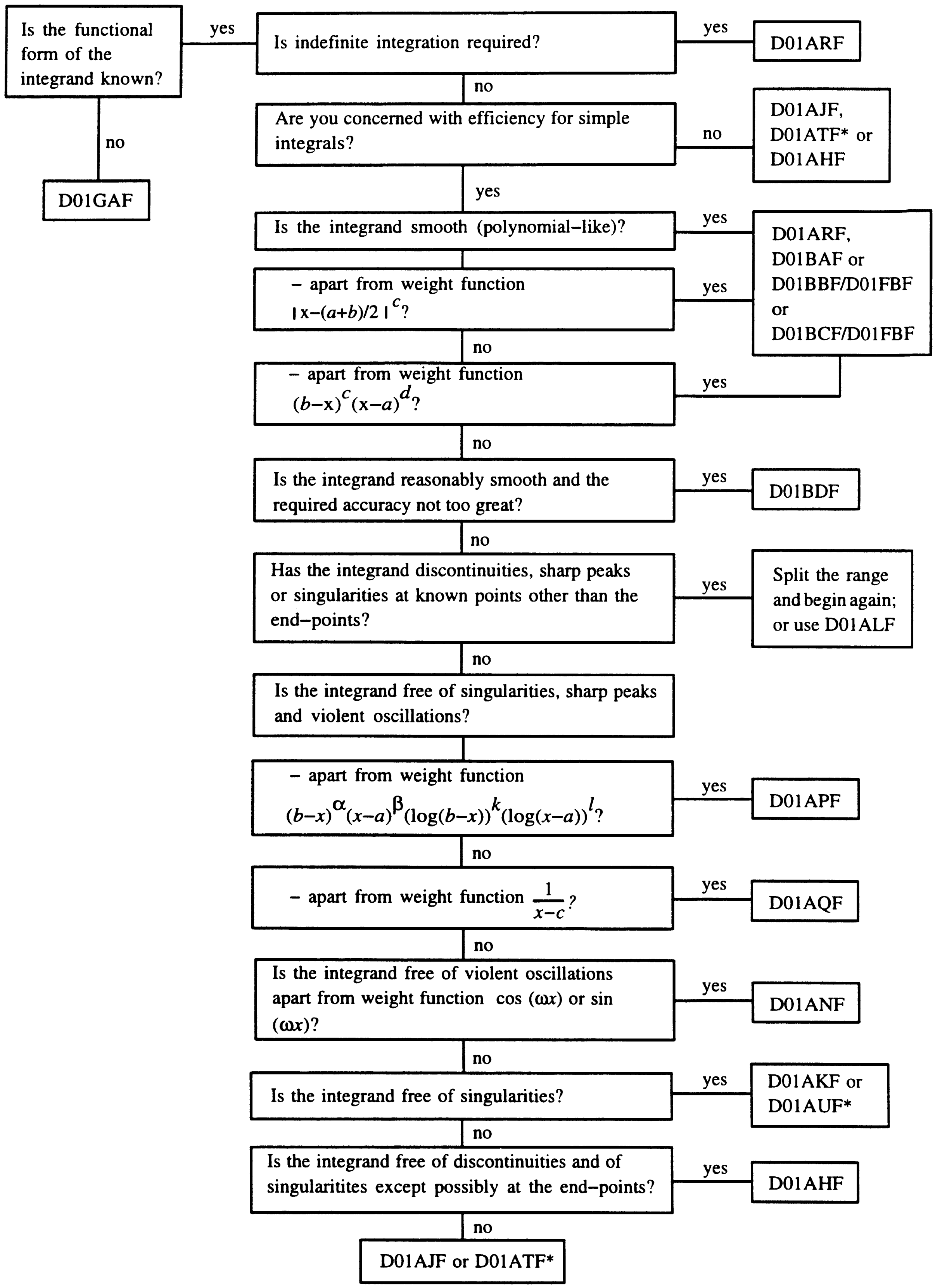
D01EAF extends the technique of D01FCF to integrate adaptively more than one integrand, that is to calculate the set of integrals

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_n}^{b_n} (f_1, f_2, \ldots, f_m)\, dx_n\, dx_{n-1} \ldots dx_1$$

for a set of similar integrands $f_1, f_2, \ldots, f_m$ where $f_i = f_i(x_1, x_2, \ldots, x_n)$.

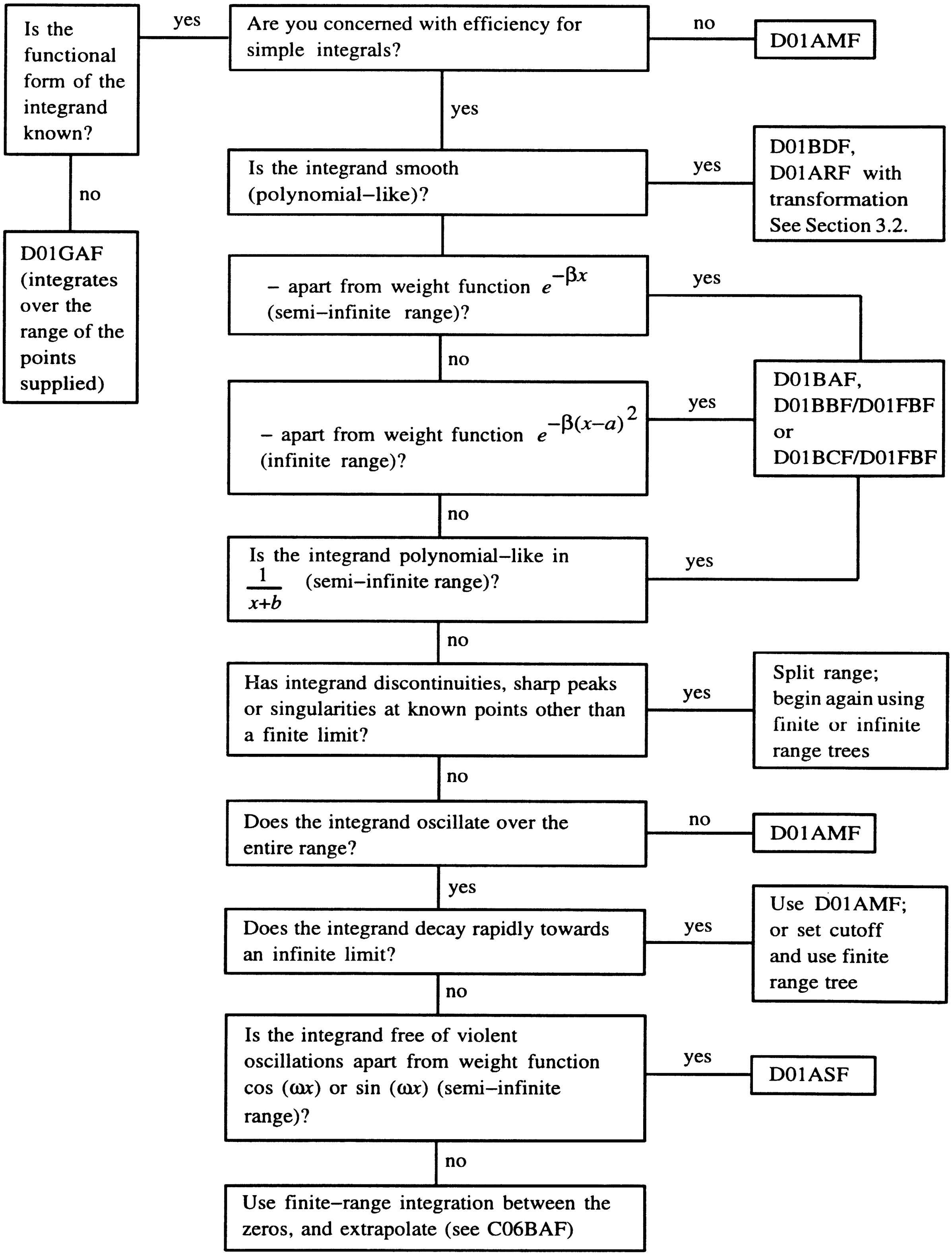
4 Decision Trees

(i) One-dimensional integrals over a finite interval
(If in doubt, follow the downward branch.)

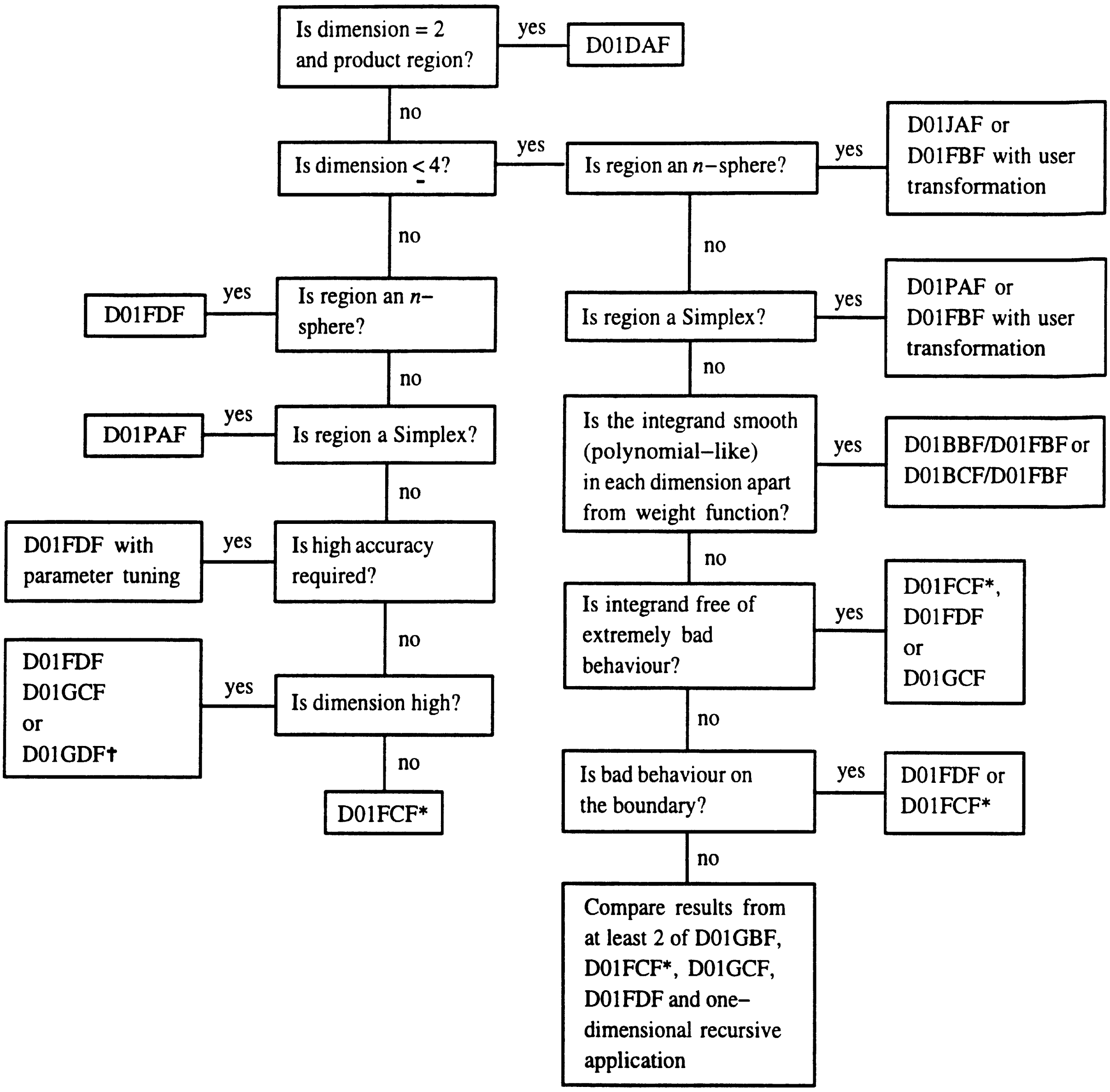


*D01ATF and D01AUF are likely to be more efficient than D01AJF and D01AKF, which use a more conventional user-interface, consistent with other routines in the chapter.

(ii) One-dimensional integrals over a semi-infinite or infinite interval
(If in doubt, follow the donward branch.)



(iii) Multi-dimensional integrals



* In the case where there are many integrals to be evaluated D01EAF should be preferred to D01FCF.

† D01GDF is likely to be more efficient than D01GCF, which uses a more conventional user-interface, consistent with other routines in the chapter.

5 References

[1] Davis P J and Rabinowitz P (1975) *Methods of Numerical Integration* Academic Press

[2] Gladwell I (1986) Vectorisation of one dimensional quadrature codes *Numerical Integration: Recent Developments, and Applications* (ed P Keast and G Fairweather) D Reidel Publishing Company, Holland 231–238

[3] Lyness J N (1983) When not to use an automatic quadrature routine *SIAM Rev.* **25** 63–87

[4] Piessens R, De Doncker-Kapenga E, Überhuber C and Kahaner D (1983) *QUADPACK, A Subroutine Package for Automatic Integration* Springer-Verlag

[5] Sobol I M (1974) *The Monte Carlo Method* The University of Chicago Press

[6] Stroud A H (1971) *Approximate Calculation of Multiple Integrals* Prentice-Hall

---

Chapter D02 – Ordinary Differential Equations

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D02AGF	2	ODEs, boundary value problem, shooting and matching technique, allowing interior matching point, general parameters to be determined
D02BGF	7	ODEs, IVP, Runge–Kutta–Merson method, until a component attains given value (simple driver)
D02BHF	7	ODEs, IVP, Runge–Kutta–Merson method, until function of solution is zero (simple driver)
D02BJF	18	ODEs, IVP, Runge–Kutta method, until function of solution is zero, integration over range with intermediate output (simple driver)
D02CJF	13	ODEs, IVP, Adams method, until function of solution is zero, intermediate output (simple driver)
D02EJF	12	ODEs, stiff IVP, BDF method, until function of solution is zero, intermediate output (simple driver)
D02GAF	8	ODEs, boundary value problem, finite difference technique with deferred correction, simple nonlinear problem
D02GBF	8	ODEs, boundary value problem, finite difference technique with deferred correction, general linear problem
D02HAF	8	ODEs, boundary value problem, shooting and matching, boundary values to be determined
D02HBF	8	ODEs, boundary value problem, shooting and matching, general parameters to be determined
D02JAF	8	ODEs, boundary value problem, collocation and least-squares, single nth-order linear equation
D02JBF	8	ODEs, boundary value problem, collocation and least-squares, system of first-order linear equations
D02KAF	7	Second-order Sturm–Liouville problem, regular system, finite range, eigenvalue only
D02KDF	7	Second-order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue only, user-specified break-points
D02KEF	8	Second-order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue and eigenfunction, user-specified break-points
D02LAF	13	Second-order ODEs, IVP, Runge–Kutta–Nystrom method
D02LXF	13	Second-order ODEs, IVP, set-up for D02LAF
D02LYF	13	Second-order ODEs, IVP, diagnostics for D02LAF
D02LZF	13	Second-order ODEs, IVP, interpolation for D02LAF
D02MVF	14	ODEs, IVP, DASSL method, set-up for D02M–N routines
D02MZF	14	ODEs, IVP, interpolation for D02M–N routines, natural interpolant
D02NBF	12	Explicit ODEs, stiff IVP, full Jacobian (comprehensive)
D02NCF	12	Explicit ODEs, stiff IVP, banded Jacobian (comprehensive)
D02NDF	12	Explicit ODEs, stiff IVP, sparse Jacobian (comprehensive)
D02NGF	12	Implicit/algebraic ODEs, stiff IVP, full Jacobian (comprehensive)
D02NHF	12	Implicit/algebraic ODEs, stiff IVP, banded Jacobian (comprehensive)
D02NJF	12	Implicit/algebraic ODEs, stiff IVP, sparse Jacobian (comprehensive)
D02NMF	12	Explicit ODEs, stiff IVP (reverse communication, comprehensive)
D02NNF	12	Implicit/algebraic ODEs, stiff IVP (reverse communication, comprehensive)
D02NRF	12	ODEs, IVP, for use with D02M–N routines, sparse Jacobian, enquiry routine
D02NSF	12	ODEs, IVP, for use with D02M–N routines, full Jacobian, linear algebra set-up

D02NTF	12	ODEs, IVP, for use with D02M–N routines, banded Jacobian, linear algebra set-up
D02NUF	12	ODEs, IVP, for use with D02M–N routines, sparse Jacobian, linear algebra set-up
D02NVF	12	ODEs, IVP, BDF method, set-up for D02M–N routines
D02NWF	12	ODEs, IVP, Blend method, set-up for D02M–N routines
D02NXF	12	ODEs, IVP, sparse Jacobian, linear algebra diagnostics, for use with D02M–N routines
D02NYF	12	ODEs, IVP, integrator diagnostics, for use with D02M–N routines
D02NZF	12	ODEs, IVP, set-up for continuation calls to integrator, for use with D02M–N routines
D02PCF	16	ODEs, IVP, Runge–Kutta method, integration over range with output
D02PDF	16	ODEs, IVP, Runge–Kutta method, integration over one step
D02PVF	16	ODEs, IVP, set-up for D02PCF and D02PDF
D02PWF	16	ODEs, IVP, resets end of range for D02PDF
D02PXF	16	ODEs, IVP, interpolation for D02PDF
D02PYF	16	ODEs, IVP, integration diagnostics for D02PCF and D02PDF
D02PZF	16	ODEs, IVP, error assessment diagnostics for D02PCF and D02PDF
D02QFF	13	ODEs, IVP, Adams method with root-finding (forward communication, comprehensive)
D02QGF	13	ODEs, IVP, Adams method with root-finding (reverse communication, comprehensive)
D02QWF	13	ODEs, IVP, set-up for D02QFF and D02QGF
D02QXF	13	ODEs, IVP, diagnostics for D02QFF and D02QGF
D02QYF	13	ODEs, IVP, root-finding diagnostics for D02QFF and D02QGF
D02QZF	13	ODEs, IVP, interpolation for D02QFF or D02QGF
D02RAF	8	ODEs, general nonlinear boundary value problem, finite difference technique with deferred correction, continuation facility
D02SAF	8	ODEs, boundary value problem, shooting and matching technique, subject to extra algebraic equations, general parameters to be determined
D02TGF	8	nth-order linear ODEs, boundary value problem, collocation and least-squares
D02TKF	17	ODEs, general nonlinear boundary value problem, collocation technique
D02TVF	17	ODEs, general nonlinear boundary value problem, set-up for D02TKF
D02TXF	17	ODEs, general nonlinear boundary value problem, continuation facility for D02TKF
D02TYF	17	ODEs, general nonlinear boundary value problem, interpolation for D02TKF
D02TZF	17	ODEs, general nonlinear boundary value problem, diagnostics for D02TKF
D02XJF	12	ODEs, IVP, interpolation for D02M–N routines, natural interpolant
D02XKF	12	ODEs, IVP, interpolation for D02M–N routines, C_1 interpolant
D02ZAF	12	ODEs, IVP, weighted norm of local error estimate for D02M–N routines

Chapter D02

Ordinary Differential Equations

Contents

1 Scope of the Chapter

This chapter is concerned with the numerical solution of ordinary differential equations. There are two main types of problem, those in which all boundary conditions are specified at one point (initial value problems), and those in which the boundary conditions are distributed between two or more points (boundary value problems and eigenvalue problems). Routines are available for initial value problems, two-point boundary value problems and Sturm–Liouville eigenvalue problems.

2 Background to the Problems

For most of the routines in this chapter a system of ordinary differential equations must be written in the form

$$y_1' = f_1(x, y_1, y_2, \ldots, y_n),$$

$$y_2' = f_2(x, y_1, y_2, \ldots, y_n),$$

$$\ldots..$$

$$y_n' = f_n(x, y_1, y_2, \ldots, y_n),$$

that is the system must be given in first-order form. The n dependent variables (also, the solution) $y_1, y_2, \ldots, y_n$ are functions of the independent variable x, and the differential equations give expressions for the first derivatives $y_i' = \frac{dy_i}{dx}$ in terms of x and $y_1, y_2, \ldots, y_n$. For a system of n first-order equations, n associated boundary conditions are usually required to define the solution.

A more general system may contain derivatives of higher order, but such systems can almost always be reduced to the first-order form by introducing new variables. For example, suppose we have the third-order equation

$$z''' + zz'' + k(l - z'^2) = 0.$$

We write $y_1 = z$, $y_2 = z'$, $y_3 = z''$, and the third-order equation may then be written as the system of first-order equations

$$y_1' = y_2$$

$$y_2' = y_3$$

$$y_3' = -y_1 y_3 - k(l - y_2^2).$$

For this system $n = 3$ and we require 3 boundary conditions in order to define the solution. These conditions must specify values of the dependent variables at certain points. For example, we have an **initial value problem** if the conditions are:

$$\begin{array}{lll} y_1 = 0 & \text{at} & x = 0 \\ y_2 = 0 & \text{at} & x = 0 \\ y_3 = 0.1 & \text{at} & x = 0. \end{array}$$

These conditions would enable us to integrate the equations numerically from the point $x = 0$ to some specified end-point. We have a **boundary value problem** if the conditions are:

$$\begin{array}{lll} y_1 = 0 & \text{at} & x = 0 \\ y_2 = 0 & \text{at} & x = 0 \\ y_2 = 1 & \text{at} & x = 10. \end{array}$$

These conditions would be sufficient to define a solution in the range $0 \leq x \leq 10$, but the problem could not be solved by direct integration (see Section 2.2). More general boundary conditions are permitted in the boundary value case.

It is sometimes advantageous to solve higher-order systems directly. In particular, there is an initial value routine to solve a system of second-order ordinary differential equations of the special form

$$y_1'' = f_1(x, y_1, y_2, \ldots, y_n),$$

$$y_2'' = f_2(x, y_1, y_2, \ldots, y_n),$$

$$\ldots\ldots$$

$$y_n'' = f_n(x, y_1, y_2, \ldots, y_n).$$

For this second-order system initial values of the derivatives of the dependent variables, y_i', for $i = 1, 2, \ldots, n$, are required.

There is also a boundary value routine that can treat directly a mixed order system of ordinary differential equations.

There is a broader class of initial value problems known as differential algebraic systems which can be treated. Such a system may be defined as

$$\begin{aligned} y' &= f(x, y, z) \\ 0 &= g(x, y, z) \end{aligned}$$

where y and f are vectors of length n and g and z are vectors of length m. The functions g represent the algebraic part of the system.

In addition implicit systems can also be solved, that is systems of the form

$$A(x, y)y' = f(x, y)$$

where A is a matrix of functions; such a definition can also incorporate algebraic equations. Note that general systems of this form may contain higher-order derivatives and that they can usually be transformed to first-order form, as above.

2.1 Initial Value Problems

To solve first-order systems, initial values of the dependent variables y_i, for $i = 1, 2, \ldots, n$, must be supplied at a given point, a. Also a point, b, at which the values of the dependent variables are required, must be specified. The numerical solution is then obtained by a step-by-step calculation which approximates values of the variables y_i, for $i = 1, 2, \ldots, n$, at finite intervals over the required range $[a, b]$. The routines in this chapter adjust the step length automatically to meet specified accuracy tolerances. Although the accuracy tests used are reliable over each step individually, in general an accuracy requirement cannot be guaranteed over a long range. For many problems there may be no serious accumulation of error, but for unstable systems small perturbations of the solution will often lead to rapid divergence of the calculated values from the true values. A simple check for stability is to carry out trial calculations with different tolerances; if the results differ appreciably the system is probably unstable. Over a short range, the difficulty may possibly be overcome by taking sufficiently small tolerances, but over a long range it may be better to try to reformulate the problem.

A special class of initial value problems are those for which the solutions contain rapidly decaying transient terms. Such problems are called **stiff**; an alternative way of describing them is to say that certain eigenvalues of the Jacobian matrix $\left(\frac{\partial f_i}{\partial y_j}\right)$ have large negative real parts when compared to others. These problems require special methods for efficient numerical solution; the methods designed for non-stiff problems when applied to stiff problems tend to be very slow, because they need small step lengths to avoid numerical instability. A full discussion is given in Hall and Watt [9] and a discussion of the methods for stiff problems is given in Berzins *et al.* [4].

2.2 Boundary Value Problems

In general, a system of nonlinear differential equations with boundary conditions at two or more points cannot be guaranteed to have a solution. The solution, if it exists, has to be determined iteratively. A comprehensive treatment of the numerical solution of boundary value problems can be found in [1] and [10]. The methods for this chapter are discussed in [3], [2] and [7].

2.2.1 Collocation methods

In the collocation method, the solution components are approximated by piecewise polynomials on a mesh. The coefficients of the polynomials form the unknowns to be computed. The approximation to the solution must satisfy the boundary conditions and the differential equations at collocation points in each mesh subinterval. A modified Newton method is used to solve the nonlinear equations. The mesh is refined by trying to equidistribute the estimated error over the whole interval. An initial estimate of the solution across the mesh is required.

2.2.2 Shooting methods

In the shooting method, the unknown boundary values at the initial point are estimated to form an initial value problem, and the equations are then integrated to the final point. At the final point the computed solution and the known boundary conditions should be equal. The condition for equality gives a set of nonlinear equations for the estimated values, which can be solved by Newton's method or one of its variants. The iteration cannot be guaranteed to converge, but it is usually successful if:

- the system has a solution,
- the system is not seriously unstable or very stiff for step-by-step solution, and
- good initial estimates can be found for the unknown boundary conditions.

It may be necessary to simplify the problem and carry out some preliminary calculations, in order to obtain suitable starting values. A fuller discussion is given in Chapters 16, 17 and 18 of Hall and Watt [9], Chapter 11 of Gladwell and Sayers [8] and Chapter 8 of Childs *et al.* [5].

2.2.3 Finite-difference methods

If a boundary value problem seems insoluble by the above methods and a good estimate for the solution of the problem is known at all points of the range then a finite-difference method may be used. Finite-difference equations are set up on a mesh of points and estimated values for the solution at the grid points are chosen. Using these estimated values as starting values a Newton iteration is used to solve the finite-difference equations. The accuracy of the solution is then improved by deferred corrections or the addition of points to the mesh or a combination of both. The method does not suffer from the difficulties associated with the shooting method but good initial estimates of the solution may be required in some cases and the method is unlikely to be successful when the solution varies very rapidly over short ranges. A discussion is given in Chapters 9 and 11 of Gladwell and Sayers [8] and Chapter 4 of Childs *et al.* [5].

2.3 Chebyshev Collocation for Linear Differential Equations

The collocation method gives a different approach to the solution of ordinary differential equations. It can be applied to problems of either initial value or boundary value type. Suppose the approximate solution is represented in polynomial form, say as a series of Chebyshev polynomials. The coefficients may be determined by matching the series to the boundary conditions, and making it satisfy the differential equation at a number of selected points in the range. The calculation is straightforward for linear differential equations (nonlinear equations may also be solved by an iterative technique based on linearisation). The result is a set of Chebyshev coefficients, from which the solution may be evaluated at any point using E02AKF. A fuller discussion is given in Chapter 24 of Childs *et al.* [5] and Chapter 11 of Gladwell and Sayers [8].

This method can be useful for obtaining approximations to standard mathematical functions. For example, suppose we require values of the Bessel function $J_{\frac{1}{3}}(x)$ over the range (0,5), for use in another calculation. We solve the Bessel differential equation by collocation and obtain the Chebyshev coefficients of the solution, which we can use to construct a function for $J_{\frac{1}{3}}(x)$. (Note that routines for many common standard functions are already available in the NAG Library, Chapter S).

2.4 Eigenvalue Problems

Sturm–Liouville problems of the form

$$(p(x)y')' + q(x,\lambda)y = 0$$

with appropriate boundary conditions given at two points, can be solved by a Scaled Prüfer method. In this method the differential equation is transformed to another which can be solved for a specified eigenvalue by a shooting method. A discussion is given in Chapter 11 of Gladwell and Sayers [8] and a complete description is given in Pryce [11]. Some more general eigenvalue problems can be solved by the methods described in Section 2.2.

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.

There are no routines which deal directly with COMPLEX equations. These may however be transformed to larger systems of real equations of the required form. Split each equation into its real and imaginary parts and solve for the real and imaginary parts of each component of the solution. Whilst this process doubles the size of the system and may not always be appropriate it does make available for use the full range of routines provided presently.

3.1 Initial Value Problems

In general, for non-stiff first-order systems, Runge–Kutta (RK) routines should be used. For the usual requirement of integrating across a range the appropriate routines are D02PVF and D02PCF; D02PVF is a setup routine for D02PCF. For more complex tasks there are a further five related routines, D02PDF, D02PWF, D02PXF, D02PYF and D02PZF. When a system is to be integrated over a long range or with relatively high accuracy requirements the variable-order, variable-step Adams codes may be more efficient. The appropriate routine in this case is D02CJF. For more complex tasks using an Adams code there are a further six related routines: D02QFF, D02QGF, D02QXF, D02QWF, D02QYF and D02QZF.

For stiff systems, that is those which usually contain rapidly decaying transient components, the Backward Differentiation Formula (BDF) variable-order, variable-step codes should be used. The appropriate routine in this case is D02EJF. For more complex tasks using a BDF code there are a collection of routines in the D02M–D02N Subchapter. These routines can treat implicit differential algebraic systems and contain methods alternative to BDF techniques which may be appropriate in some circumstances.

If users are not sure how to classify a problem, they are advised to perform some preliminary calculations with D02PCF, which can indicate whether the system is stiff. We also advise performing some trial calculations with D02PCF (RK), D02CJF (Adams) and D02EJF (BDF) so as to determine which type of routine is best applied to the problem. The conclusions should be based on the computer time used and the number of evaluations of the derivative function f_i. See Gladwell [6] for more details.

For second-order systems of the special form described in Section 2 the Runge–Kutta–Nystrom (RKN) routine D02LAF should be used.

3.1.1 Runge–Kutta routines

The basic RK routine is D02PDF which takes one integration step at a time. An alternative is D02PCF which provides output at user-specified points. The initialisation of either D02PCF or D02PDF and the setting of optional inputs, including choice of method, is made by a call to the setup routine D02PVF. Optional output information about the integration and about error assessment, if selected, can be obtained by calls to the diagnostic routines D02PYF and D02PZF respectively. D02PXF may be used to interpolate on information produced by D02PDF to give solution and derivative values between the integration points. D02PWF may be used to reset the end of the integration range whilst integrating using D02PDF.

There is a simple driving routine D02BJF which integrates a system over a range and, optionally, computes intermediate output and/or determines the position where a specified function of the solution is zero.

3.1.2 Adams routines

The general Adams variable-order variable-step routine is D02QFF which provides a choice of automatic error control and the option of a sophisticated root-finding technique. Reverse communication for both the differential equation and root definition function is provided in D02QGF, which otherwise has the same facilities as D02QFF. A reverse communication routine makes a return to the calling (sub)program for evaluations of equations rather than calling a user-supplied procedure. The initialisation of either of D02QFF and D02QGF and the setting of optional inputs is made by a call to the setup routine D02QWF. Optional output information about the integration and any roots detected can be obtained by calls to the diagnostic routines D02QXF and D02QYF respectively. D02QZF may be used to interpolate on information produced by D02QFF or D02QGF to give solution and derivative values between the integration points.

There is a simple driving routine D02CJF which integrates a system over a range and, optionally, computes intermediate output and/or determines the position where a specified function of the solution is zero.

3.1.3 BDF routines

General routines for explicit and implicit ordinary differential equations with a wide range of options for integrator choice and special forms of numerical linear algebra are provided in the D02M–D02N Subchapter. A separate document describing the use of this subchapter is given immediately before the routines of the subchapter.

There is a simple driving routine D02EJF which integrates a system over a range and, optionally, computes intermediate output and/or determines the position where a specified function of the solution is zero. It has a specification similar to the Adams routine D02CJF except that to solve the equations arising in the BDF method an approximation to the Jacobian $\left(\frac{\partial f_i}{\partial y_j}\right)$ is required. This approximation can be calculated internally but the user may supply an analytic expression. In most cases supplying a correct analytic expression will reduce the amount of computer time used.

3.1.4 Runge–Kutta–Nystrom routines

The Runge–Kutta–Nystrom routine D02LAF uses either a low- or high-order method (chosen by the user). The choice of method and error control and the setting of optional inputs is made by a call to the setup routine D02LXF. Optional output information about the integration can be obtained by a call to the diagnostic routine D02LYF. When the low-order method has been employed D02LZF may be used to interpolate on information produced by D02LAF to give solution and derivative values between the integration points.

3.2 Boundary Value Problems

In general, for a nonlinear system of mixed order with separated boundary conditions, the collocation method (D02TKF and its associated routines) can be used. Problems of a more general nature can often be transformed into a suitable form for treatment by D02TKF, for example nonseparated boundary conditions or problems with unknown parameters (see Section 8 of D02TVF for details).

For simple boundary value problems with assigned boundary values the user may prefer to use a code based on the shooting method or finite difference method for which there are routines with simple calling sequences (D02HAF and D02GAF).

For difficult boundary value problems, where the user needs to exercise some control over the calculation, and where the collocation method proves unsuccessful, the user may wish to try the alternative methods of shooting (D02SAF) or finite-differences (D02RAF).

Note that it is not possible to make a fully automatic boundary value routine, and the user should be prepared to experiment with different starting values or a different routine if the problem is at all difficult.

3.2.1 Collocation methods

The collocation routine D02TKF solves a nonlinear system of mixed order boundary value problems with separated boundary conditions. The initial mesh and accuracy requirements must be specified by a call to the setup routine D02TVF. Optional output information about the final mesh and estimated maximum error can be obtained by a call to the diagnostic routine D02TZF. The solution anywhere on the mesh can be computed by a call to the interpolation routine D02TYF. If D02TKF is being used to solve a sequence of related problems then the continuation routine D02TXF should also be used.

3.2.2 Shooting methods

D02HAF may be used for simple boundary value problems, where the unknown parameters are the missing boundary conditions. More general boundary value problems may be handled by using D02HBF.

This routine allows for a generalised parameter structure, and is fairly complicated. The older routine D02AGF has been retained for use when an interior matching-point is essential; otherwise the newer routine D02HBF should be preferred.

For particularly complicated problems where, for example, the parameters must be constrained or the range of integration must be split to enable the shooting method to succeed, the recommended routine is D02SAF which extends the facilities provided by D02HBF. D02SAF permits the sophisticated user much more control over the calculation than does D02HBF; in particular the user is permitted precise control of solution output and intermediate monitoring information.

3.2.3 Finite-difference methods

D02GAF may be used for simple boundary value problems with assigned boundary values. The calling sequence of D02GAF is very similar to that for D02HAF discussed above.

The user may find that convergence is difficult to achieve using D02GAF since only specifying the unknown boundary values and the position of the finite-difference mesh is permitted. In such cases the user may use D02RAF which permits specification of an initial estimate for the solution at all mesh points and allows the calculation to be influenced in other ways too. D02RAF is designed to solve a general nonlinear two-point boundary value problem with nonlinear boundary conditions.

A routine, D02GBF, is also supplied specifically for the general linear two-point boundary value problem written in a standard 'textbook' form.

The user is advised to use interpolation routines from the E01 Chapter to obtain solution values at points not on the final mesh.

3.3 Chebyshev Collocation Method

D02TGF may be used to obtain the approximate solution of a system of differential equations in the form of a Chebyshev series. The routine treats linear differential equations directly, and makes no distinction between initial value and boundary value problems. This routine is appropriate for problems where it is known that the solution is smooth and well-behaved over the range, so that each component can be represented by a single polynomial. Singular problems can be solved using D02TGF as long as their polynomial-like solutions are required.

D02TGF permits the differential equations to be specified in higher order form; that is without conversion to a first-order system. This type of specification leads to a complicated calling sequence. For the inexperienced user two simpler routines are supplied. D02JAF solves a single regular linear differential equation of any order whereas D02JBF solves a system of regular linear first-order differential equations.

3.4 Eigenvalue Problems

Two routines, D02KAF and D02KDF, may be used to find the eigenvalues of second-order Sturm–Liouville problems. D02KAF is designed to solve simple problems with regular boundary conditions. D02KAF calls D02KDF which is designed to solve more difficult problems, for example with singular boundary conditions or on infinite ranges or with discontinuous coefficients.

If the eigenfunctions of the Sturm–Liouville problem are also required, D02KEF should be used. (D02KEF solves the same types of problem as D02KDF.)

3.5 Summary of Recommended Routines

Problem	Routine		
	R K method	Adams method	BDF method
Initial–value Problems **Driver Routines**			
Integration over a range with optional intermediate output and optional determination of position where a function of the solution becomes zero	D02BJF	D02CJF	D02EJF
Integration over a range			
–with intermediate output	D02BJF	D02CJF	D02EJF
–until function of solution becomes zero	D02BJF	D02CJF	D02EJF
Comprehensive Integration routines	D02PCF, D02PDF D02PVF, D02PWF D02PXF, D02PYF	D02QFF, D02QGF D02QWF, D02QXF D02QYF, D02QZF	D02M routines D02N routines D02XKF, D02XJF and D02ZAF
Package for Solving Stiff Equations	D02M–D02N Subchapter		
Package for Solving Second–order Systems of Special Form	D02L routines		
Boundary–value Problems Collocation Method, Mixed Order	D02TKF, D02TVF, D02TXF, D02TYF, D02TZF		
Boundary–value Problems Shooting Method			
simple parameter	D02HAF		
generalised parameters	D02HBF, D02AGF		
additional facilities	D02SAF		
Boundary–value Problems Finite–difference Method			
simple parameter	D02GAF		
linear problem	D02GBF		
full nonlinear problem	D02RAF		
Chebyshev Collocation, Linear Problems			
single equation	D02JAF		
first–order system	D02JBF		
general system	D02TGF		
Sturm–Liouville Eigenvalue Problems			
regular problems	D02KAF		
general problems	D02KDF		
eigenfunction calculation	D02KEF		

4 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'.

D02BAF	D02BBF	D02BDF	D02CAF	D02CBF	D02CGF
D02CHF	D02EAF	D02EBF	D02EGF	D02EHF	D02PAF
D02QAF	D02QBF	D02QDF	D02QQF	D02XAF	D02XBF
D02XGF	D02XHF	D02YAF			

5 References

[1] Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice Hall, Englewood Cliffs, NJ

[2] Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

[3] Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

[4] Berzins M, Brankin R W and Gladwell I (1988) Design of the stiff integrators in the NAG Library *SIGNUM Newsl.* **23** 16–23

[5] Gladwell I (1979) The development of the boundary value codes in the ordinary differential equations chapter of the NAG Library *Codes for Boundary Value Problems in Ordinary Differential Equations. Lecture Notes in Computer Science* (ed B Childs, M Scott, J W Daniel, E Denman and P Nelson) **76** Springer-Verlag

[6] Gladwell I (1979) Initial value routines in the NAG Library *ACM Trans. Math. Software* **5** 386–400

[7] Gladwell I (1987) The NAG Library boundary value codes *Numerical Analysis Report* **134** Manchester University

[8] Gladwell I and Sayers D K (ed.) (1980) *Computational Techniques for Ordinary Differential Equations* Academic Press

[9] Hall G and Watt J M (ed.) (1976) *Modern Numerical Methods for Ordinary Differential Equations* Clarendon Press, Oxford

[10] Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

[11] Pryce J D (1986) Error estimation for phase-function shooting methods for Sturm–Liouville problems *IMA J. Numer. Anal.* **6** 103–123

Chapter D02M/N

Integrators for Stiff Ordinary Differential Systems

1 Introduction

This subchapter contains the specifications of the integrators, the setup routines and diagnostic routines which have been developed from the SPRINT package, Berzins and Furzeland [1].

The integrators D02NBF, D02NCF and D02NDF are designed for solving stiff systems of explicitly defined ordinary differential equations,

$$y' = g(t, y).$$

The integrators D02NGF, D02NHF and D02NJF are designed for solving stiff systems of implicitly defined ordinary differential equations,

$$A(t, y)y' = g(t, y).$$

This formulation permits solution of differential/algebraic systems (DAEs). The facilities provided are essentially those of the explicit solvers.

The integrator routines have almost identical calling sequences but each is designed to solve a problem where the Jacobian is of a particular structure: full matrix (D02NBF and D02NGF), banded matrix (D02NCF and D02NHF) or sparse matrix (D02NDF and D02NJF). Each of these structures has associated with it a linear algebra setup routine: D02NSF, D02NTF and D02NUF respectively. A linear algebra setup routine must be called before the first call to the appropriate integrator. These linear algebra setup routines check various parameters of the corresponding integrator routine and set certain parameters for the linear algebra computations. A routine, D02NXF, is supplied which permits extraction of diagnostic information after a call to either of the sparse linear algebra solvers D02NDF and D02NJF.

With the integrators are also associated three integrator setup routines D02NVF, D02NWF and D02MVF, one of which must be called before the first call to any integrator routine. They provide input to the Backward Differentiation Formulae (BDF), the Blend Formulae and the special fixed leading coefficient BDF codes respectively. On return from an integrator, if it is feasible to continue the integration, D02NZF may be called to reset various integration parameters. It is often of considerable interest to determine statistics concerning the integration process. D02NYF is provided with this aim in mind. It should prove especially useful to those who wish to integrate many similar problems as it provides suitable values for many of the input parameters and indications of the difficulties encountered when solving the problem.

Hence, the general form of a program calling one of the integrator routines D02NBF, D02NCF, D02NDF, D02NGF, D02NHF or D02NJF will be

```
declarations
     .
     .
call linear algebra setup routine
call integrator setup routine
call integrator
call integrator diagnostic routine (if required)
call linear algebra diagnostic routine (if appropriate and if required)
     .
     .
STOP
END
```

The required calling sequence for different Jacobian structures and system types is represented diagramatically in Figure 1.

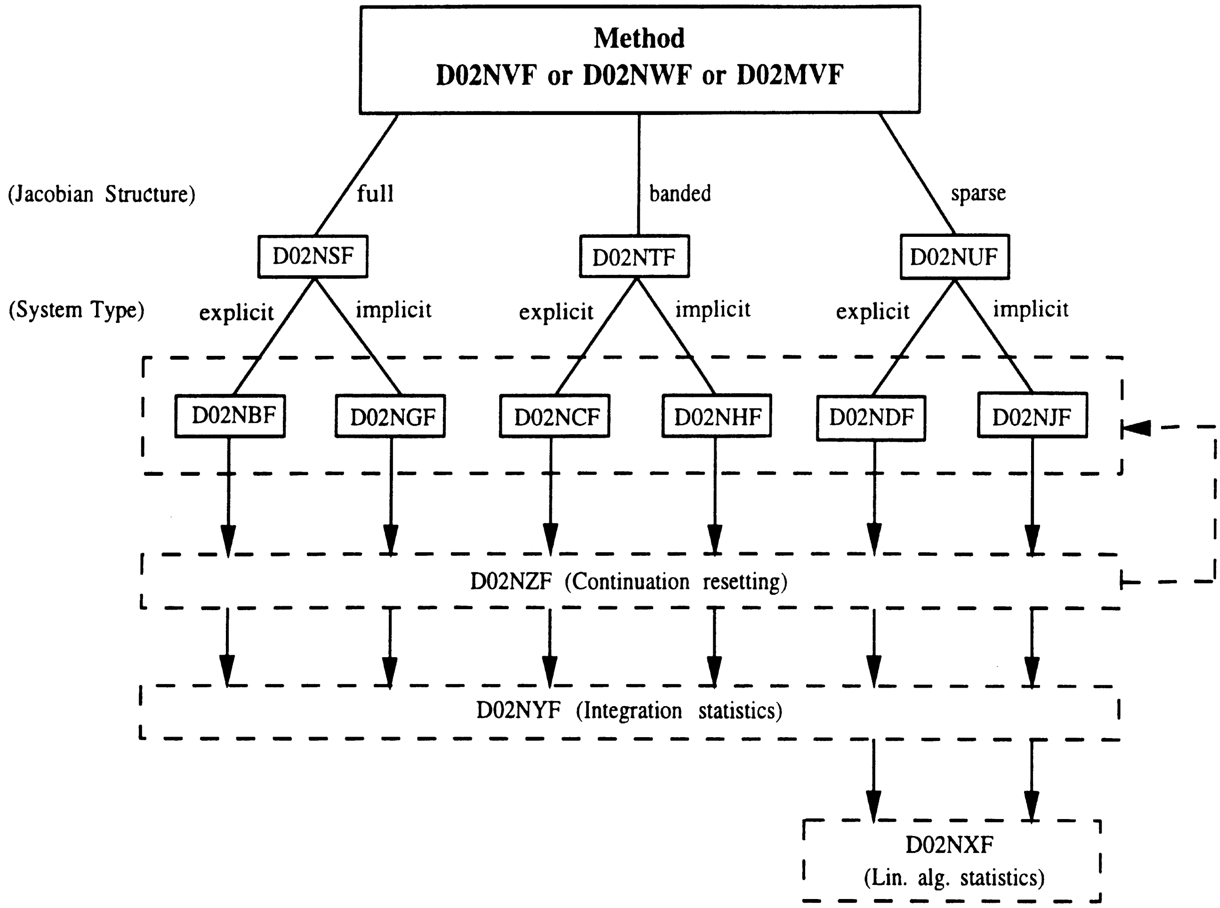


Figure 1
Schema for forward communication routine calling sequences

The integrators D02NMF and D02NNF are reverse communication routines designed for solving explicit and implicit stiff ordinary differential systems respectively. Users are warned that they should use these routines only when the integrators mentioned above are inadequate for their application. For example, if it is difficult to write one or more of the subroutines FCN (RESID) or JAC (or MONITR) or if the integrators are to be embedded in a package, it may be advisable to consider these routines.

Since these routines use reverse communication the user need define no EXTERNAL parameters. This makes them especially suitable for large scale computations where encapsulation of the definition of the differential system or its Jacobian matrix in subroutine form may be particularly difficult to achieve.

D02NMF is the reverse communication counterpart of the forward communication routines D02NBF, D02NCF and D02NDF whereas D02NNF is the reverse communication counterpart of the forward communication routines D02NGF, D02NHF and D02NJF. When using these reverse communication routines it is necessary to call the same linear algebra and integrator setup routines as for the forward communication counterpart. All the other continuation and interrogation routines available for use with the forward communication routines are also available to the user when calling the reverse communication routines.

There is also a routine, D02NRF, to inform the user how to supply the Jacobian when the sparse linear algebra option is being employed with either of D02NMF and D02NNF.

Hence, the general form of a program calling one of the integrator routines D02NMF or D02NNF will be

```
     declarations
     call linear algebra setup routine
     call integrator setup routine
     IREVCM = 0
1000 call integrator( ..., IREVCM, ...)
     IF (IREVCM.GT.0) THEN

     evaluate residual and Jacobian (including a call to D02NRF if
     sparse linear algebra is being used), call the MONITR routine etc.

     GO TO 1000
     ENDIF

     call integrator diagnostic routine (if required)
     call linear algebra diagnostic routine (if appropriate and if required)
     STOP
     END
```

The required calling sequence in the case of reverse communication, is represented diagramatically in Figure 2.

In the example programs for the eight integrators D02NBF, D02NCF, D02NDF, D02NGF, D02NHF, D02NJF, D02NMF and D02NNF we attempt to illustrate the various options available. Many of these options are available in all the routines and the user is invited to scan all the example programs for illustrations of their use. In each case we use as an example the stiff Robertson problem

$$\begin{aligned} a' &= -0.04a + 10^4bc \\ b' &= 0.04a - 10^4bc - 3\times10^7b^2 \\ c' &= 3\times10^7b^2 \end{aligned}$$

despite the fact that it is not a sensible choice to use either the banded or the sparse linear algebra for this problem. Their use here serves for illustration of the techniques involved. For the implicit integrators D02NGF, D02NHF and D02NJF we write the Robertson problem in residual form, as an implicit differential system and as a differential/algebraic system respectively. Here we are exploiting the fact that $a+b+c$ is constant and hence one of the equations may be replaced by $(a+b+c)'=0.0$ or $a+b+c=1.0$ (for our particular choice of initial conditions). For the reverse communication routines D02NMF and D02NNF our examples are intended only to illustrate the reverse communication technique.

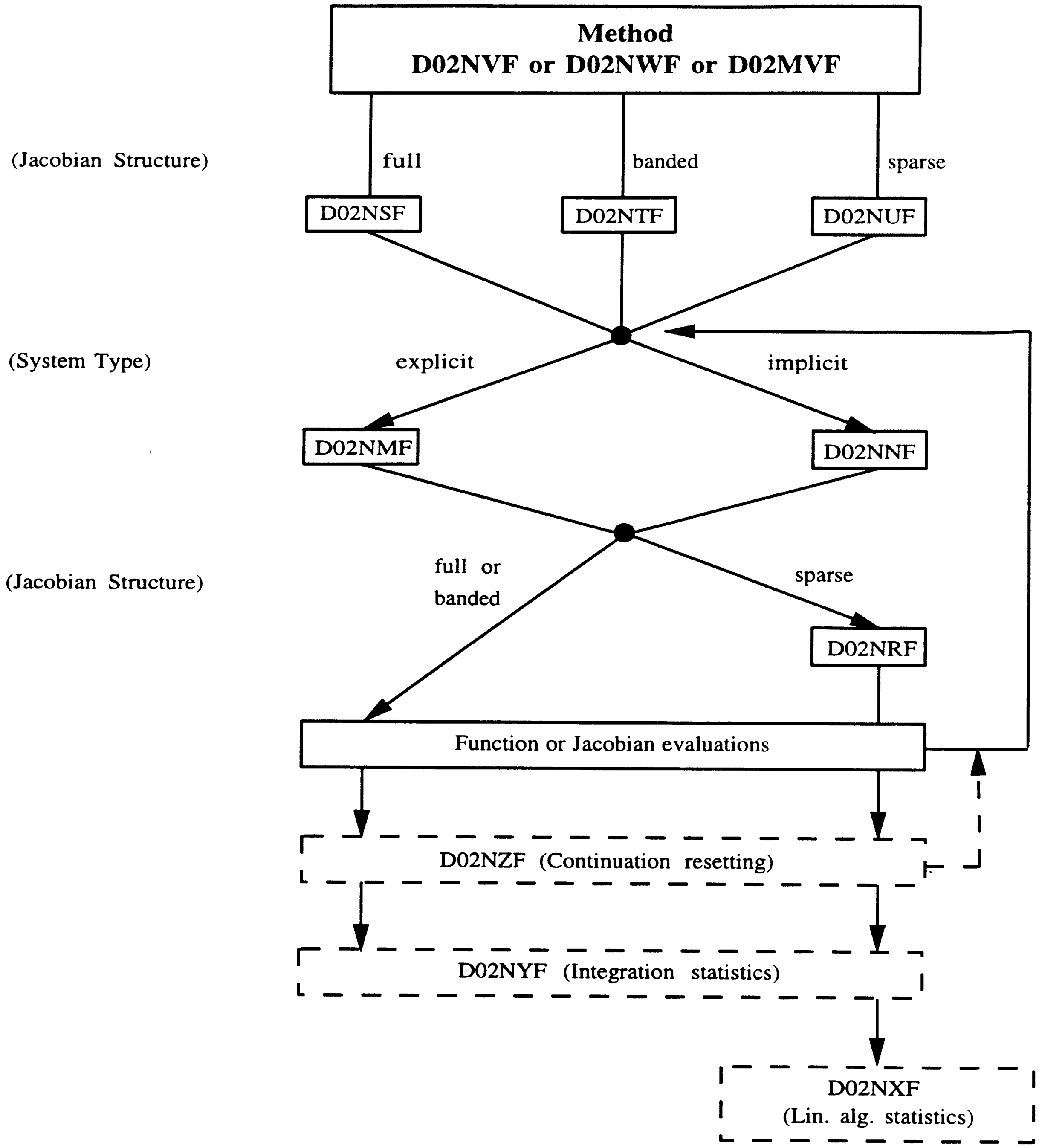


Figure 2
Schema for reverse communication routine calling sequences

2 References

[1] Berzins M and Furzeland R M (1985) A user's manual for SPRINT – A versatile software package for solving systems of algebraic, ordinary and partial differential equations: Part 1 – Algebraic and ordinary differential equations *Report TNER.85.085* Shell Research Limited

Chapter D03 – Partial Differential Equations

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D03EAF	7	Elliptic PDE, Laplace's equation, two-dimensional arbitrary domain
D03EBF	7	Elliptic PDE, solution of finite difference equations by SIP, five-point two-dimensional molecule, iterate to convergence
D03ECF	8	Elliptic PDE, solution of finite difference equations by SIP for seven-point three-dimensional molecule, iterate to convergence
D03EDF	12	Elliptic PDE, solution of finite difference equations by a multigrid technique
D03EEF	13	Discretize a second-order elliptic PDE on a rectangle
D03FAF	14	Elliptic PDE, Helmholtz equation, three-dimensional Cartesian co-ordinates
D03MAF	7	Triangulation of plane region
D03PCF	15	General system of parabolic PDEs, method of lines, finite differences, one space variable
D03PDF	15	General system of parabolic PDEs, method of lines, Chebyshev C^0 collocation, one space variable
D03PEF	16	General system of first-order PDEs, method of lines, Keller box discretisation, one space variable
D03PFF	17	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	15	General system of parabolic PDEs, coupled DAEs, method of lines, finite differences, one space variable
D03PJF	15	General system of parabolic PDEs, coupled DAEs, method of lines, Chebyshev C^0 collocation, one space variable
D03PKF	16	General system of first-order PDEs, coupled DAEs, method of lines, Keller box discretisation, one space variable
D03PLF	17	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	16	General system of parabolic PDEs, coupled DAEs, method of lines, finite differences, remeshing, one space variable
D03PRF	16	General system of first-order PDEs, coupled DAEs, method of lines, Keller box discretisation, remeshing, one space variable
D03PSF	17	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	17	Roe's approximate Riemann solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PVF	17	Osher's approximate Riemann solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PWF	18	Modified HLL Riemann solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PXF	18	Exact Riemann Solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PYF	15	PDEs, spatial interpolation with D03PDF or D03PJF
D03PZF	15	PDEs, spatial interpolation with D03PCF, D03PEF, D03PFF, D03PHF, D03PKF, D03PLF, D03PPF, D03PRF or D03PSF
D03RAF	18	General system of second-order PDEs, method of lines, finite differences, remeshing, two space variables, rectangular region

D03RBF	18	General system of second-order PDEs, method of lines, finite differences, remeshing, two space variables, rectilinear region
D03RYF	18	Check initial grid data in D03RBF
D03RZF	18	Extract grid data from D03RBF
D03UAF	7	Elliptic PDE, solution of finite difference equations by SIP, five-point two-dimensional molecule, one iteration
D03UBF	8	Elliptic PDE, solution of finite difference equations by SIP, seven-point three-dimensional molecule, one iteration

Chapter D03

Partial Differential Equations

Contents

1 Scope of the Chapter

This chapter is concerned with the numerical solution of partial differential equations.

2 Background to the Problems

The definition of a partial differential equation problem includes not only the equation itself but also the domain of interest and appropriate subsidiary conditions. Indeed, partial differential equations are usually classified as elliptic, hyperbolic or parabolic according to the form of the equation **and** the form of the subsidiary conditions which must be assigned to produce a well-posed problem. Ultimately it is hoped that this chapter will contain routines for the solution of equations of each of these types together with automatic mesh generation routines and other utility routines particular to the solution of partial differential equations. The routines in this chapter will often call upon routines from other chapters, such as Chapter F04 (Simultaneous Linear Equations) and Chapter D02 (Ordinary Differential Equations).

The classification of partial differential equations is easily described in the case of **linear** equations of the **second order** in two independent variables, i.e., equations of the form

$$au_{xx} + 2bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0, \tag{1}$$

where a, b, c, d, e, f and g are functions of x and y only. Equation (1) is called elliptic, hyperbolic or parabolic according to whether $ac - b^2$ is positive, negative or zero, respectively. Useful definitions of the concepts of elliptic, hyperbolic and parabolic character can also be given for differential equations in more than two independent variables, for systems and for nonlinear differential equations.

For **elliptic** equations, of which Laplace's equation

$$u_{xx} + u_{yy} = 0 \tag{2}$$

is the simplest example of second order, the subsidiary conditions take the form of **boundary** conditions, i.e., conditions which provide information about the solution at all points of a **closed** boundary. For example, if equation (2) holds in a plane domain D bounded by a contour C, a solution u may be sought subject to the condition

$$u = f \quad \text{on } C, \tag{3}$$

where f is a given function. The condition (3) is known as a Dirichlet boundary condition. Equally common is the Neumann boundary condition

$$u' = g \quad \text{on } C, \tag{4}$$

which is one form of a more general condition

$$u' + fu = g \quad \text{on } C, \tag{5}$$

where u' denotes the derivative of u normal to the contour C, and f and g are given functions. Provided that f and g satisfy certain restrictions, condition (5) yields a well-posed **boundary value problem** for Laplace's equation. In the case of the Neumann problem, one further piece of information, e.g. the value of u at a particular point, is necessary for uniqueness of the solution. Boundary conditions similar to the above are applicable to more general second-order elliptic equations, whilst two such conditions are required for equations of fourth order.

For **hyperbolic** equations, the wave equation

$$u_{tt} - u_{xx} = 0 \tag{6}$$

is the simplest example of second order. It is equivalent to a first-order system

$$u_t - v_x = 0, \quad v_t - u_x = 0. \tag{7}$$

The subsidiary conditions may take the form of **initial** conditions, i.e., conditions which provide information about the solution at points on a suitable **open** boundary. For example, if equation (6) is satisfied for $t > 0$, a solution u may be sought such that

$$u(x,0) = f(x), \quad u_t(x,0) = g(x), \tag{8}$$

where f and g are given functions. This is an example of an **initial value problem**, sometimes known as Cauchy's problem.

For **parabolic** equations, of which the heat conduction equation

$$u_t - u_{xx} = 0 \tag{9}$$

is the simplest example, the subsidiary conditions always include some of **initial** type and may also include some of **boundary** type. For example, if equation (9) is satisfied for $t > 0$ and $0 < x < 1$, a solution u may be sought such that

$$u(x,0) = f(x), \quad 0 < x < 1, \tag{10}$$

and

$$u(0,t) = 0, \quad u(1,t) = 1, \quad t > 0. \tag{11}$$

This is an example of a mixed **initial/boundary value problem**.

For all types of partial differential equations, finite difference methods (Mitchell and Griffiths [6]) and finite element methods (Wait and Mitchell [11]) are the most common means of solution and such methods obviously feature prominently either in this chapter or in the companion NAG Finite Element Library. Some of the utility routines in this chapter are concerned with the solution of the large sparse systems of equations which arise from finite difference and finite element methods.

Alternative methods of solution are often suitable for special classes of problems. For example, the method of characteristics is the most common for hyperbolic equations involving time and one space dimension (Smith [9]). The method of lines (see Mikhlin and Smolitsky [5]) may be used to reduce a parabolic equation to a (stiff) system of ordinary differential equations, which may be solved by means of routines from Chapter D02 – Ordinary Differential Equations. Similarly, integral equation or boundary element methods (Jaswon and Symm [3]) are frequently used for elliptic equations. Typically, in the latter case, the solution of a boundary value problem is represented in terms of certain boundary functions by an integral expression which satisfies the differential equation throughout the relevant domain. The boundary functions are obtained by applying the given boundary conditions to this representation. Implementation of this method necessitates discretization of only the boundary of the domain, the dimensionality of the problem thus being effectively reduced by one. The boundary conditions yield a full system of simultaneous equations, as opposed to the sparse systems yielded by finite difference and finite element methods, but the full system is usually of much lower order. Solution of this system yields the boundary functions, from which the solution of the problem may be obtained, by quadrature, as and where required.

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 choice of routine will depend first of all upon the type of partial differential equation to be solved. At present no special allowances are made for problems with boundary singularities such as may arise at corners of domains or at points where boundary conditions change. For such problems results should be treated with caution.

Users may wish to construct their own partial differential equation solution software for problems not solvable by the routines described in Section 3.1 to Section 3.6 below. In such cases users can employ appropriate routines from the Linear Algebra Chapters to solve the resulting linear systems; see Section 3.8 for further details.

3.1 Elliptic Equations

The routine D03EAF solves Laplace's equation in two dimensions, equation (2), by an integral equation method. This routine is applicable to an arbitrary domain bounded internally or externally by one or more closed contours, when the value of either the unknown function u or its normal derivative u' is given at each point of the boundary.

The routines D03EBF and D03ECF solve a system of simultaneous algebraic equations of five-point and seven-point molecule form (Mikhlin and Smolitsky [5]) on two-dimensional and three-dimensional topologically-rectangular meshes respectively, using Stone's Strongly Implicit Procedure (SIP). These routines, which make repeated calls of the utility routines D03UAF and D03UBF respectively, may be used to solve any boundary value problem whose finite difference representation takes the appropriate form.

The routine D03EDF solves a system of seven-point difference equations in a rectangular grid (in two dimensions), using the multigrid iterative method. The equations are supplied by the user, and the seven-point form allows cross-derivative terms to be represented (see Mitchell and Griffiths [6]). The method is particularly efficient for large systems of equations with diagonal dominance and should be preferred to D03EBF whenever it is appropriate for the solution of the problem.

The routine D03EEF discretizes a second-order equation on a two-dimensional rectangular region using finite differences and a seven-point molecule. The routine allows for cross-derivative terms, Dirichlet, Neumann or mixed boundary conditions, and either central or upwind differences. The resulting seven-diagonal difference equations are in a form suitable for passing directly to the multigrid routine D03EDF, although other solution methods could just as easily be used.

The routine D03FAF, based on the routine HW3CRT from FISHPACK (Swarztrauber and Sweet [10]), solves the Helmholtz equation in a three-dimensional cuboidal region, with any combination of Dirichlet, Neumann or periodic boundary conditions. The method used is based on the fast Fourier transform algorithm, and is likely to be particularly efficient on vector-processing machines.

3.2 Hyperbolic Equations

See Section 3.6.

3.3 Parabolic Equations

There are five routines available for solving parabolic equations in one space dimension: D03PCF, D03PDF, D03PHF, D02PJF and D03PPF. Equations may include nonlinear terms but the true derivative u_t should occur linearly and equations should usually contain a second-order space derivative u_{xx}. There are certain restrictions on the coefficients to try to ensure that the problems posed can be solved by the above routines.

The method of solution is to discretize the space derivatives using finite differences or collocation, and to solve the resulting system of ordinary differential equations using a 'stiff' solver.

D03PCF and D03PDF can solve a system of parabolic (and possibly elliptic) equations of the form

$$\sum_{j=1}^{n} P_{ij}(x,t,U,U_x)\frac{\partial U_j}{\partial t} + Q_i(x,t,U,U_x) = x^{-m}\frac{\partial}{\partial x}(x^m R_i(x,t,U,U_x)),$$

where $i = 1,2,\ldots,n$, $a \leq x \leq b$, $t \geq t_0$.

The parameter m allows the routine to handle different coordinate systems easily (Cartesian, cylindrical polars and spherical polars). D03PCF uses a finite differences spatial discretization and D03PDF uses a collocation spatial discretization.

D03PHF and D03PJF are similar to D03PCF and D03PDF respectively, except that they provide scope for coupled differential-algebraic systems. This extended functionality allows for the solution of more complex and more general problems, e.g. periodic boundary conditions and integro-differential equations.

D03PPF is similar to D03PHF but allows remeshing to take place in the spatial direction. This facility can be very useful when the nature of the solution in the spatial direction varies considerably over time.

For parabolic systems in two space dimensions see Section 3.5.

3.4 First Order Systems in One Space Dimension

There are three routines available for solving systems of first-order partial differential equations: D03PEF, D03PKF and D03PRF. Equations may include nonlinear terms but the time derivative should occur linearly. There are certain restrictions on the coefficients to ensure that the problems posed can be solved by the above routines.

The method of solution is to discretize the space derivatives using the Keller box scheme and to solve the resulting system of ordinary differential equations using a 'stiff' solver.

D03PEF is designed to solve a system of the form

$$\sum_{j=1}^{n} P_{ij}(x,t,U,U_x)\frac{\partial U_j}{\partial t} + Q_i(x,t,U,U_x) = 0,$$

where $i = 1,2,\ldots,n$, $a \leq x \leq b$, $t \geq t_0$.

D03PKF is similar to D03PEF except that it provides scope for coupled differential algebraic systems. This extended functionality allows for the solution of more complex problems.

D03PRF is similar to D03PKF but allows remeshing to take place in the spatial direction. This facility can be very useful when the nature of the solution in the spatial direction varies considerably over time.

D03PEF, D03PKF or D03PRF may also be used to solve systems of higher or mixed order partial differential equations which have been reduced to first order. Note that in general these routines are unsuitable for hyperbolic first-order equations, for which an appropriate upwind discretization scheme should be used (see Section 3.6 for example).

3.5 Second Order Systems in Two Space Dimensions

There are two routines available for solving nonlinear second order time-dependent systems in two space dimensions: D03RAF and D03RBF. These reoutines are formally applicable to the general nonlinear system:

$$F_j(t,x,y,u,u_t,u_x,u_y,u_{xx},u_{xy},u_{yy}) = 0$$

where $j = 1,2,\ldots,$NPDE, $(x,y) \in \Omega$, $t_0 \leq t \leq t_{out}$. However, they should not be used to solve purely hyperbolic systems, or time-independent problems.

D03RAF solves the nonlinear system in a rectangular domain, while D03RBF solves in a rectilinear region, i.e., a domain bounded by perpendicular straight lines.

Both routines use the method of lines and solve the resulting system of ordinary differential equations using a backward differentiation formula (BDF) method, modified Newton method, and BiCGSTAB iterative linear solver. Local uniform grid refinement is used to improve accuracy.

Utility routines D03RYF and D03RZF may be used in conjunction with D03RBF to check the user-supplied initial mesh, and extract mesh co-ordinate data.

3.6 Convection-diffusion Systems

There are three routines available for solving systems of convection-diffusion equations with optional source terms: D03PFF, D03PLF, D03PSF. Equations may include nonlinear terms but the time derivative should occur linearly. There are certain restrictions on the coefficients to ensure that the problems posed can be solved by the above routines, in particular the system must be posed in conservative form (see below). The routines may also be used to solve hyperbolic convection-only systems.

Convection terms are discretized using an upwind scheme involving a numerical flux function based on the solution of a Riemann problem at each mesh point [4]; and diffusion and source terms are discretized using central differences. The resulting system of ordinary differential equations is solved using a 'stiff' solver. In the case of Euler equations for a perfect gas various approximate and exact Riemann solvers are provided in D03PUF, D03PVF, D03PWF and D03PXF. These routines may be used in conjunction with D03PFF, D03PLF and D03PSF.

D03PFF is designed to solve systems of the form

$$\sum_{j=1}^{n} P_{ij}(x,t,U)\frac{\partial U_j}{\partial t} + \frac{\partial}{\partial x}F_i(x,t,U) = C_i(x,t,U)\frac{\partial}{\partial x}D_i(x,t,U,U_x) + S_i(x,t,U),$$

or hyperbolic convection-only systems of the form

$$\sum_{j=1}^{n} P_{ij}(x,t,U)\frac{\partial U_j}{\partial t} + \frac{\partial F_i(x,t,U)}{\partial x} = 0,$$

where $i = 1, 2, \ldots, n$, $a \leq x \leq b$, $t \geq t_0$.

D03PLF is similar to D03PFF except that it provides scope for coupled differential algebraic systems. This extended functionality allows for the solution of more complex problems.

D03PSF is similar to D03PLF but allows remeshing to take place in the spatial direction. This facility can be very useful when the nature of the solution in the spatial direction varies considerably over time.

3.7 Automatic Mesh Generation

The routine D03MAF places a triangular mesh over a given two-dimensional region. The region may have any shape and may include holes. It may also be used in conjunction with routines from the NAG Finite Element Library.

3.8 Utility Routines

D03UAF (D03UBF) calculates, by the Strongly Implicit Procedure, an approximate correction to a current estimate of the solution of a system of simultaneous algebraic equations for which the iterative update matrix is of five (seven) point molecule form on a two- (three-) dimensional topologically-rectangular mesh.

Routines are available in the Linear Algebra Chapters for the direct and iterative solution of linear equations. Here we point to some of the routines that may be of use in solving the linear systems that arise from finite difference or finite element approximations to partial differential equation solutions. Chapters F01, F04 and F11 should be consulted for further information and for the appropriate routine documents. Decision trees for the solution of linear systems are given in Section 3.6 of the the F04 Chapter Introduction.

The following routines allow the direct solution of symmetric positive-definite systems:

Band	F07HDF and F07HEF
Variable band (skyline)	F01MCF and F04MCF
Tridiagonal	F04FAF
Sparse	F11JAF* and F11JBF

(* the description of F11JBF explains how F11JAF should be called to obtain a direct method)

and the following routines allow the iterative solution of symmetric positive-definite and symmetric-indefinite systems:

Sparse	F11GAF, F11GBF, F11GCF, F11JAF, F11JCF and F11JEF

The latter two routines above are black box routines which include Incomplete Cholesky, SSOR or Jacobi preconditioning.

The following routines allow the direct solution of nonsymmetric systems:

Band	F07BDF and F07BEF
Almost block-diagonal	F01LHF and F04LHF
Tridiagonal	F01LEF and F04LEF, or F04EAF
Sparse	F01BRF (and F01BSF) and F04AXF

and the following routines allow the iterative solution of nonsymmetric systems:

Sparse	F11BAF, F11BBF, F11BCF, F11DAF, F11DCF and F11DEF

The latter two routines above are black box routines which include incomplete LU, SSOR and Jacobi preconditioning.

The routines D03PZF and D03PYF use linear interpolation to compute the solution to a parabolic problem and its first derivative at the user-specified points. D03PZF may be used in conjunction with D03PCF, D03PEF, D03PHF, D03PKF, D03PPF and D03PRF. D03PYF may be used in conjunction with D03PDF and D03PJF.

D03RYF and D03RZF are utility routines for use in conjunction with D03RBF. They can be called to check the user-specified initial mesh and to extract mesh co-ordinate data.

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'.

D03PAF D03PBF D03PGF

6 References

[1] Ames W F (1977) *Nonlinear Partial Differential Equations in Engineering* Academic Press (2nd Edition)

[2] Berzins M (1990) Developments in the NAG Library software for parabolic equations *Scientific Software Systems* (ed J C Mason and M G Cox) Chapman and Hall 59–72

[3] Jaswon M A and Symm G T (1977) *Integral Equation Methods in Potential Theory and Elastostatics* Academic Press

[4] LeVeque R J (1990) *Numerical Methods for Conservation Laws* Birkhäuser Verlag

[5] Mikhlin S G and Smolitsky K L (1967) *Approximate Methods for the Solution of Differential and Integral Equations* Elsevier

[6] Mitchell A R and Griffiths D F (1980) *The Finite Difference Method in Partial Differential Equations* Wiley

[7] Pennington S V and Berzins M (1994) New NAG Library software for first-order partial differential equations *ACM Trans. Math. Softw.* **20** 63–99

[8] Richtmyer R D and Morton K W (1967) *Difference Methods for Initial-value Problems* Interscience (2nd Edition)

[9] Smith G D (1985) *Numerical Solution of Partial Differential Equations: Finite Difference Methods* Oxford University Press (3rd Edition)

[10] Swarztrauber P N and Sweet R A (1979) Efficient Fortran subprograms for the solution of separable elliptic partial differential equations *ACM Trans. Math. Software* **5** 352–364

[11] Wait R and Mitchell A R (1985) *Finite Element Analysis and Application* Wiley

Chapter D04 – Numerical Differentiation

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D04AAF	5	Numerical differentiation, derivatives up to order 14, function of one real variable

Chapter D04

Numerical Differentiation

Contents

1 Scope of the Chapter

This chapter is concerned with calculating approximations to derivatives of a function f, where the user can supply a routine representing f.

2 Background to the Problems

2.1 Description of the Problem

The problem of numerical differentiation does not receive very much attention nowadays. Although the Taylor series plays a key role in much of classical analysis, the poor reputation enjoyed by numerical differentiation has led numerical analysts to construct techniques for most problems which avoid the explicit use of numerical differentiation.

One may briefly and roughly define the term numerical differentiation as any process in which numerical values of derivatives $f^{(s)}(x_0)$ are obtained from evaluations $f(x_i)$ of the function $f(x)$ at several abscissae x_i near x_0. This problem can be stable, well conditioned, and accurate when complex-valued abscissae are used. This was first pointed out by Lyness and Moler [1]. An item of numerical software for this appears in Lyness and Sande [2]. However, in many applications the use of complex-valued abscissae is either prohibitive or prohibited. The main difficulty in using real abscissae is that amplification of round-off error can completely obliterate meaningful results. In the days when one relied on hand calculating machines and tabular data, the frustration caused by this effect led to the abandonment of serious use of the process.

There are several reasons for believing that, in the present-day computing environment, numerical differentiation might find a useful role. The first is that, by present standards, it is rather a small-scale calculation, so its cost may well be negligible compared with any overall saving in cost that might result from its use. Secondly, the assignment of a step length h is now generally open. One does not have to rely on tabular data. Thirdly, although the amplification of round-off error is an integral part of the calculation, its effect can be measured reliably and automatically by the routine at the time of the calculation.

Thus the user does not have to gauge the round-off level (or noise level) of the function values or assess the effect of this on the accuracy of the results. A routine does this automatically, returning with each result an error estimate which has already taken round-off error amplification into account.

We now illustrate, by means of a very simple example, the importance of the round-off error effect. A very simple approximation of $f'(0)$, based on the identity

$$f'(0) = (f(h) - f(-h))/2h + (h^2/3!)f'''(\xi), \tag{1}$$

is

$$(f(h) - f(-h))/2h.$$

If there were no precision problem, this formula would be the only one needed in the theory of first-order numerical differentiation. We could simply take $h = 10^{-40}$ (or $h = 10^{-1000}$) to obtain an excellent approximation based on two function values. It is only when we consider in detail how a finite length machine comes to calculate $(f(h) - f(-h))/2h$ that the necessity for a sophisticated theory becomes apparent.

To simplify the subsequent description we shall assume that the quantities involved are neither so close to zero that the machine underflow characteristics need be considered nor so large that machine overflow occurs. The approximation mentioned above involves the function values $f(h)$ and $f(-h)$. In general no computer has available these numbers exactly. Instead it uses approximations $\hat{f}(h)$ and $\hat{f}(-h)$ whose relative accuracy is less than some tolerance ϵ_f. If the function $f(x)$ is a library function, for example $\sin x$, ϵ_f may coincide with the machine accuracy parameter ϵ_m. More generally the function $f(x)$ is calculated in a user-supplied routine and ϵ_f is larger than ϵ_m by a small factor 5 or 6 if the calculation is short or by some larger factor in an extended calculation. This factor is not usually known by the user.

$\hat{f}(h)$ and $\hat{f}(-h)$ are related to $f(h)$ and $f(-h)$ by:

$$\hat{f}(h) = f(h)(1 + \theta_1\epsilon_f), \qquad |\theta_1| \le 1$$

$$\hat{f}(-h) = f(-h)(1 + \theta_2\epsilon_f), \quad |\theta_2| \le 1.$$

Thus even if the rest of the calculation were carried out exactly, it is trivial to show that

$$\frac{\hat{f}(h)-\hat{f}(-h)}{2h}-\frac{f(h)-f(-h)}{2h}\simeq 2\phi\epsilon_f\frac{f(\xi)}{2h},\qquad |\phi|\le 1.$$

The difference between the quantity actually calculated and the quantity which one attempts to calculate may be as large as $\epsilon_f f(\xi)/h$; for example using $h=10^{-40}$ and $\epsilon_m=10^{-7}$ this gives a 'conditioning error' of $10^{33}f(\xi)$.

In practice much more sophisticated formulae than (1) above are used, and for these and for the corresponding higher-derivative formulae the error analysis is different and more complicated in detail. But invariably the theory contains the same overall feature. In a finite length calculation, the error is composed of two main parts: a discretisation error which increases as h becomes large and is zero for $h=0$; and a 'conditioning' error due to amplification of round-off error in function evaluation, which increases as h becomes small and is infinite for $h=0$.

The routine in this chapter has to take into account internally both these sources of error in the results. Thus it returns pairs of results, DER(j) and EREST(j) where DER(j) is an approximation to $f^{(j)}(x_0)$ and EREST(j) is an estimate of the error in the approximation DER(j). If the routine has not been misled, DER(j) and EREST(j) satisfy

$$|\mathrm{DER}(j)-f^{(j)}(x_0)|\le \mathrm{EREST}(j).$$

In this respect, numerical differentiation routines are fundamentally different from other routines. The user does not specify an error criterion. Instead the routine provides the error estimate and this may be very large.

We mention here a terminological distinction. A fully automatic routine is one in which the user does not need to provide any information other than that required to specify the problem. Such a routine (at a cost in computing time) decides an appropriate internal parameter such as the step length h by itself. On the other hand a routine which requires the user to provide a step length h, but automatically chooses from several different formulae each based on the specified step length, is termed a semi-automatic routine.

The situation described above must have seemed rather depressing when hand machines were in common use. For example in the simple illustration one does not know the values of the quantities $f'''(\xi)$ or ϵ_f involved in the error estimates, and the idea of altering the value of h and starting again must have seemed appalling. However by present-day standards, it is a relatively simple matter to write a program which carries out all the previously considered time-consuming calculations which may seem necessary. None of the routines envisaged for this chapter are particularly revolutionary in concept. They simply utilise the computer for the sort of task for which it was originally designed. It carries out simple tedious calculations which are necessary to estimate the accuracy of the results of other even simpler tedious calculations.

2.2 Examples of Applications for Numerical Differentiation Routines

(a) One immediate use to which a set of derivatives at a point is likely to be put is that of constructing a Taylor series representation:

$$f(x)=f(x_0)+\sum_{j=1}^{n}\frac{f^{(j)}(x_0)}{j!}(x-x_0)^j+\frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1},\ \ |\xi-x_0|\le x.$$

This infinite series converges so long as $|x-x_0|<R_c$ (the radius of convergence) and it is only for these values of x that such a series is likely to be used. In this case in forming the sum, the required accuracy in $f^{(j)}(x_0)$ diminishes with increasing j.

The series formed from the Taylor series using elementary operations such as integration or differentiation has the same overall characteristic. A technique based on a Taylor series expansion may be quite accurate, even if the individual derivatives are not, so long as the less accurate derivatives are associated with known small coefficients.

The error introduced by using n approximate derivatives DER(j) is bounded by:

$$\sum_{j=1}^{n}\mathrm{EREST}(j)\ |x-x_0|^j/j!$$

Thus, if the user is prepared to base the result on a truncated Taylor series, the additional error introduced by using approximate Taylor coefficients can be readily bounded from the values of EREST(j). However in an automatic code the user must be prepared to introduce some alternative approach in case this error bound turns out to be unduly high.

In this sort of application the accuracy of the result depends on the size of the errors in the numerical differentiation. There are other applications where the effect of large errors EREST(j) is merely to prolong a calculation, but not to impair the final accuracy.

(b) A simple Taylor series approach such as described in (a) is used to find a starting value for a rapidly converging but extremely local iterative process.

(c) The technique known as 'subtracting out the singularity' as a preliminary to numerical quadrature may be extended and may be carried out approximately. Thus suppose we are interested in evaluating

$$\int_0^1 x^{-(1/2)}\phi(x)dx,$$

we have an automatic quadrature routine available, and we have a routine available for $\phi(x)$ which we know to be an analytic function. An integrand function like $x^{-(1/2)}\phi(x)$ is generally accepted to be difficult for an automatic integrator because of the singularity. If $\phi(x)$ and some of its derivatives at the singularity $x = 0$ are known one may effectively 'subtract out' the singularity using the following identity:

$$\int_0^1 x^{-(1/2)}\phi(x)dx = \int_0^1 x^{-(1/2)}(\phi(x) - \phi(0) - Ax - Bx^2/2)dx + 2\phi(0) + 2A/3 + B/5 \qquad (2)$$

with $A = \phi'(0)$ and $B = \phi''(0)$.

The integrand function on the right of (2) has no singularity, but its third derivative does. Thus using numerical quadrature for this integral is much cheaper than using numerical quadrature for the original integral (in the left-hand side of (2)).

However (2) is an identity whatever values of A and B are assigned. Thus the same procedure can be used with A and B being approximations to $\phi'(0)$ and $\phi''(0)$ provided by a numerical differentiation routine. The integrand would now be more difficult to integrate than if A and B were correct but still much less difficult than the original integrand (on the left-hand side of (2)). But, assuming that the automatic quadrature routine is reliable, the overall result would still be correct. The effect of using approximate derivatives rather than exact derivatives does not impair the accuracy of the result. It simply makes the result more expensive to obtain, but not nearly as expensive as if no derivatives were used at all.

(d) The calculation of a definite integral may be based on the Euler–Maclaurin expansion

$$\int_0^1 f(x)dx = \frac{1}{m}\sum_{j=0}^{m}{}'' f(j/m) - \sum_{s=1}^{l}\frac{B_{2s}}{2s!}\frac{(f^{(2s-1)}(1) - f^{(2s-1)}(0))}{m^{2s}} + O(m^{(-2l-2)}).$$

Here B_{2s} is a Bernoulli number. If one fixes a value of l then as m is increased the right-hand side (without the remainder term) approaches the true value of the integral. This statement remains true whatever values are used to replace $f^{(2s-1)}(1)$ and $f^{(2s-1)}(0)$. If no derivatives are available, and this formula is used (effectively with the derivatives replaced by zero) the rate of convergence is slow (like m^{-2}) and a large number of function evaluations may be used in calculating the trapezoidal rule sum for large m before a sufficiently accurate result is attained. However if approximate derivatives are used, the initial rate of convergence is enhanced. In fact, in this example any derivative approximation which is closer than the approximation zero is helpful. Thus the use of inaccurate derivatives may have the effect of shortening the overall calculation, since a sufficiently accurate result may be obtained using a smaller value of m, without impairing the accuracy of the result. (The resemblance with Gregory's formula is superficial. Here l is kept fixed and m is increased, ensuring a convergent process.)

The examples given above are only intended to illustrate the sort of use to which approximate derivatives may be put. Very simple illustrations have been used for clarity, and in such simple cases there are usually more efficient approaches to the problem. The same ideas applied in a more complicated or restrictive setting may provide an efficient approach to a problem for which no simple standard approach exists.

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.

(a) At the present time there is only one numerical differentiation routine available in this chapter, D04AAF. This is a semi-automatic routine for obtaining approximations to the first fourteen derivatives of a real valued function $f(x)$ at a specified point x_0. The user provides a FUNCTION representing $f(x)$, the value of x_0, an upper limit $n \leq 14$ on the order of the derivatives required and a step length h. The routine returns a set of approximations DER(j) and corresponding error estimates EREST(j) which hopefully satisfy

$$|\mathrm{DER}(j) - f^{(j)}(x_0)| \leq \mathrm{EREST}(j), \quad j = 1, 2, \ldots, n \leq 14.$$

We term this routine a semi-automatic routine because the user provides a step length h and this is not needed to specify the problem.

It is important that the error estimate EREST(j) is taken into consideration by the user before any use of DER(j) is made. The actual size of EREST(j) depends on the analytic structure of the function, on the word length of the machine used and on the step size h, and is difficult to predict. Thus the user has to run the routine to find out how accurate the results are. Usually the user will find the higher-order derivatives are successively more inaccurate and that past a certain order, say 10 or 11, the size of EREST(j) actually exceeds DER(j). Clearly when this happens the approximation DER(j) is unusable.

(b) We have investigated a fully automatic routine, which has the same calling sequence with the exception that a step length is not required. This routine finds an appropriate step length h for itself. The cost seems to be greater by a factor of 3 to 5 but the returned values of EREST(j) are usually smaller. It is our intention to develop such a routine only if there is a demand for it in which case the experience of users with the presently available semi-automatic routine will be very helpful.

(c) There is available in the algorithm section of CACM [2] a semi-automatic Fortran routine for numerical differentiation of an analytical function $f(z)$ at a possibly complex abscissa z_0. This is a stable problem. It can be used for any problem for which D04AAF might be used and produces more accurate results, and derivatives of arbitrary high order. However even if z_0 is real and $f(z)$ is real for z, this routine requires a user-supplied FUNCTION which evaluates $f(z)$ for complex values of z and it makes use of complex arithmetic.

(d) Routines are available in Chapter E02 to differentiate functions which are polynomials (in Chebyshev series representation) or cubic splines (in B-spline representation).

4 References

[1] Lyness J N and Moler C B (1967) Numerical differentiation of analytic functions *SIAM J. Numer. Anal.* **4 (2)** 202–210

[2] Lyness J N and Ande G (1971) Algorithm 413, ENTCAF and ENTCRE: Evaluation of normalised Taylor coefficients of an analytic function *Comm. ACM* **14 (10)** 669–675

Chapter D05 – Integral Equations

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D05AAF	5	Linear non-singular Fredholm integral equation, second kind, split kernel
D05ABF	6	Linear non-singular Fredholm integral equation, second kind, smooth kernel
D05BAF	14	Nonlinear Volterra convolution equation, second kind
D05BDF	16	Nonlinear convolution Volterra–Abel equation, second kind, weakly singular
D05BEF	16	Nonlinear convolution Volterra–Abel equation, first kind, weakly singular
D05BWF	16	Generate weights for use in solving Volterra equations
D05BYF	16	Generate weights for use in solving weakly singular Abel-type equations

Chapter D05

Integral Equations

Contents

1 Scope of the Chapter

This chapter is concerned with the numerical solution of integral equations. Provision will be made for most of the standard types of equation (see below). The following are, however, specifically excluded:

(a) Equations arising in the solution of partial differential equations by integral equation methods. In cases where the prime purpose of an algorithm is the solution of a partial differential equation it will normally be included in Chapter D03.

(b) Calculation of inverse integral transforms. This problem falls within the ambit of Chapter C06.

2 Background to the Problems

2.1 Introduction

Any functional equation in which the unknown function appears under the sign of integration is called an integral equation. Integral equations arise in a great many branches of science; for example, in potential theory, acoustics, elasticity, fluid mechanics, radiative transfer, theory of population, etc. In many instances the integral equation originates from the conversion of a boundary-value problem or an initial-value problem associated with a partial or an ordinary differential equation, but many problems lead directly to integral equations and cannot be formulated in terms of differential equations.

Integral equations are of many types; here we attempt to indicate some of the main distinguishing features with particular regard to the use and construction of algorithms.

2.2 Classification of Integral Equations

In the classical theory of integral equations one distinguishes between *Volterra* equations and *Fredholm* equations. In a Fredholm equation the region of integration is fixed, whereas in a Volterra equation the region is variable. Thus, the equation

$$cy(t) = f(t) + \lambda \int_a^b K(t,s,y(s))\,ds, \quad a \le t \le b \tag{1}$$

is an example of Fredholm equation, and the equation

$$cy(t) = f(t) + \lambda \int_a^t K(t,s,y(s))\,ds, \quad a \le t \tag{2}$$

is an example of a Volterra equation.

Here the *forcing* function $f(t)$ and the *kernel* function $K(t,s,y(s))$ are prescribed, while $y(t)$ is the unknown function to be determined. (More generally the integration and the domain of definition of the functions may extend to more than one dimension.) The parameter λ is often omitted; it is, however, of importance in certain theoretical investigations (e.g. stability) and in the eigenvalue problem discussed below.

If in (1), or (2), $c = 0$, the integral equation is said to be of the *first kind.* If $c = 1$, the equation is said to be of the *second kind.*

Equations (1) and (2) are *linear* if the kernel $K(t,s,y(s)) = k(t,s)y(s)$, otherwise they are *nonlinear.*

Note. in a linear integral equation, $k(t,s)$ is usually referred to as the kernel. We adopt this convention throughout.

These two types of equations are broadly analogous to problems of initial- and boundary-value type for an ordinary differential equation (ODE); thus the Volterra equation, characterised by a variable upper limit of integration, is amenable to solution by methods of marching type whilst most methods for treating Fredholm equations lead ultimately to the solution of an approximating system of simultaneous algebraic equations. For comprehensive discussion of numerical methods see Atkinson [1], Baker [2], Brunner and van der Houwen [3] and Delves and Walsh [5]. In what follows, the term 'integral equation' is used in its general sense, and the type is distinguished when appropriate.

2.3 Structure of Kernel

When considering numerical methods for integral equations, particular attention should be paid to the character of the kernel, which is usually the main factor governing the choice of an appropriate quadrature formula or system of approximating functions. Various commonly occurring types of singularity call for individual treatment.

Likewise provision can be made for cases of symmetry, periodicity or other special structure, where the solution may have special properties and/or economies may be effected in the solution process. We note in particular the following cases to which we shall often have occasion to refer in the description of individual algorithms

(a) A linear integral equation with a kernel $k(t,s) = k(s,t)$ is said to be **symmetric**. This property plays a key role in the theory of Fredholm integral equations.

(b) If $k(t,s) = k(a+b-t, a+b-s)$ in a linear integral equation, the kernel is called **centro-symmetric**.

(c) If in Equations (1) or (2) the kernel has the form $K(t,s,y(s)) = k(t-s)g(s,y(s))$, the equation is called a **convolution** integral equation; in the linear case $g(s,y(s)) = y(s)$.

(d) If the kernel in (1) has the form

$$\begin{aligned} K(t,s,y(s)) &= K_1(t,s,y(s)), \quad a \leq s \leq t, \\ K(t,s,y(s)) &= K_2(t,s,y(s)), \quad t < s \leq b, \end{aligned}$$

where the functions K_1 and K_2 are well-behaved, whilst K or its s-derivative is possibly discontinuous, may be described as discontinuous or of 'split' type; in the linear case $K(t,s,y(s)) = k(t,s)y(s)$ and consequently $K_1 = k_1 y$ and $K_2 = k_2 y$. Examples are the commonly occurring kernels of the type $k(|t-s|)$ and the Green's functions (influence functions) which arise in the conversion of ODE boundary-value problems to integral equations. It is also of interest to note that the Volterra equation (2) may be conceived as a Fredholm equation with kernel of split type, with $K_2(t,s,y(s)) \equiv 0$; consequently methods designed for the solution of Fredholm equations with split kernels are also applicable to Volterra equations.

2.4 Singular and Weakly Singular Equations

An integral equation may be called singular if either

(a) its kernel contains a singularity, or
(b) the range of integration is infinite,

and it is said to be weakly-singular if the kernel becomes infinite at $s = t$.

Sometimes a solution can be effected by a simple adaptation of a method applicable to a non-singular equation: for example, an infinite range may be truncated at a suitably chosen point. In other cases, however, theoretical considerations will dictate the need for special methods and algorithms. Examples are:

(i) Integral equations with singular kernels of Cauchy type;

(ii) Equations of Wiener–Hopf type;

(iii) Various dual integral equations arising in the solution of boundary-value problems of mathematical physics;

(iv) The well-known Abel integral equation, an equation of Volterra type, whose kernel contains an inverse square-root singularity at $s = t$.

Problems of inversion of integral transforms also fall under this heading but, as already remarked, they lie outside the scope of this chapter.

2.5 Fredholm Integral Equations

2.5.1 Eigenvalue problem

Closely connected with the linear Fredholm integral equation of the second kind is the eigenvalue problem represented by the homogeneous equation

$$y(t) - \lambda \int_a^b k(t,s)y(s)\,ds = 0, \quad a \leq t \leq b. \tag{3}$$

If λ is chosen arbitrarily this equation in general possesses only the trivial solution $y(t) = 0$. However, for a certain critical set of values of λ, the **characteristic values** or eigenvalues (the latter term is sometimes reserved for the reciprocals $\mu = 1/\lambda$), there exist non-trivial solutions $y(t)$, termed **characteristic functions** or **eigenfunctions**, which are of fundamental importance in many investigations. The analogy with the eigenproblem of linear algebra is readily apparent, and indeed most methods of solution of equation (3) entail reduction to an approximately equivalent algebraic problem

$$(K - \mu I)y = 0. \tag{4}$$

2.5.2 Equations of the first kind

The Fredholm integral equation of the first kind

$$\int_a^b k(t,s)y(s)\ ds = f(t), \quad a \le t \le b, \tag{5}$$

belong to the class of 'ill-posed' problems; even supposing that a solution corresponding to the prescribed $f(t)$ exists, a slight perturbation of $f(t)$ may give rise to an arbitrarily large variation in the solution $y(t)$. Hence the equation may be closely satisfied by a function bearing little resemblance to the 'true' solution. The difficulty associated with this instability is aggravated by the fact that in practice the specification of $f(t)$ is usually inexact.

Nevertheless a great many physical problems (e.g. in radiography, spectroscopy, stereology, chemical analysis) are appropriately formulated in terms of integral equations of the first kind, and useful and meaningful 'solutions' can be obtained with the aid of suitable stabilising procedures. See Chapters 12 and 13 of Delves and Walsh [5] for further discussion and references.

2.5.3 Equations of the second kind

Consider the nonlinear Fredholm equation of the second kind

$$y(t) = f(t) + \int_a^b K(t,s,y(s))\ ds, \quad a \le t \le b. \tag{6}$$

The numerical solution of equation (6) is usually accomplished either by simple iteration or by a more sophisticated iterative scheme based on Newton's method; in the latter case it is necessary to solve a sequence of linear integral equations. Convergence may be demonstrated subject to suitable conditions of Lipschitz continuity of the functions K with respect to the argument y.

Examples of Fredholm type (for which the provision of algorithms is contemplated) are:

(a) the Uryson equation

$$u(t) - \int_0^1 F(t,s,u(s))\ ds = 0, \quad 0 \le t \le 1, \tag{7}$$

(b) the Hammerstein equation

$$u(t) - \int_0^1 k(t,s)g(s,u(s))\ ds = 0, \quad 0 \le t \le 1, \tag{8}$$

where F and g are arbitrary functions.

2.6 Volterra Integral Equations

2.6.1 Equations of the first kind

Consider the Volterra integral equation of the first kind

$$\int_a^t k(t,s)y(s)\ ds = f(t), \quad a \le t. \tag{9}$$

Clearly it is necessary that $f(a) = 0$; otherwise no solution to (9) can exist.

The following types of Volterra integral equations of the first kind occur in real life problems:

equations with unbounded kernel at $s = t$,

equations with sufficiently smooth kernel.

These types belong also to the class of 'ill-posed' problems. However, the instability is appreciably less severe in the equations with unbounded kernel. In general, a non-singular Volterra equation of the first kind presents less computational difficulty than the Fredholm equation (5) with a smooth kernel.

A Volterra equation of the first kind may, under suitable conditions, be converted by differentiation to one of the second kind or by integration by parts to an equation of the second kind for the integral of the wanted function.

2.6.2 Equations of the second kind

A very general Volterra equation of the second kind is given by

$$y(t) = f(t) + \int_a^t K(t,s,y(s))\,ds, \quad a \leq t. \tag{10}$$

The resemblance of Volterra equations to ODEs suggests that the underlying methods for ODE problems can be applied to Volterra equations. Indeed this turns out to be the case. The main advantages of implementing these methods are their well-developed theoretical background, i.e., convergence and stability, see Brunner and van der Houwen [3], Wolkenfelt [6].

Many Volterra integral equations arising in real life problems have a convolution kernel (cf. Section 2.3(c)), see [3] for references. However, a subclass of these equations which have kernels of the form

$$k(t-s) = \sum_{j=0}^{M} \lambda_j (t-s)^j, \tag{11}$$

where $\{\lambda_j\}$ are real, can be converted into a system of linear or nonlinear ODEs, see [3].

For more information on the theoretical and the numerical treatment of integral equations we refer the user to Atkinson [1], Baker [2], Brunner and van der Houwen [3], Cochran [4] and Delves and Walsh [5].

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 choice of routine will depend first of all upon the type of integral equation to be solved.

3.1 Fredholm Equations of Second Kind

(a) Linear equations

D05AAF is applicable to an equation with a discontinuous or 'split' kernel as defined in 2.3.(d). Here, however, both the functions k_1 and k_2 are required to be defined (and well-behaved) throughout the square $a \leq s, t \leq b$.

D05ABF is applicable to an equation with a smooth kernel. Note that D05AAF may also be applied to this case, by setting $k_1 = k_2 = k$, but D05ABF is more efficient.

3.2 Volterra Equations of Second Kind

(a) Linear equations

D05AAF may be used to solve a Volterra equation by defining k_2 (or k_1) to be identically zero. (See also (b).)

(b) Nonlinear equations

D05BAF is applicable to a nonlinear convolution Volterra integral equation of the second kind. The kernel function has the form

$$K(t,s,y(s)) = k(t-s)g(s,y(s)).$$

The underlying methods used in the routine are the reducible linear multi-step methods. The user has a choice of variety of these methods. This routine can also be used for linear g.

D05BDF is applicable to a nonlinear convolution equation having a weakly-singular kernel (Abel). The kernel function has the form

$$K(t,s,y(s)) = \frac{k(t-s)}{\sqrt{t-s}}g(s,y(s)).$$

The underlying methods used in the routine are the fractional linear multistep methods based on BDF methods. This routine can also be used for linear g.

3.3 Volterra Equations of First Kind

(a) Linear equations

See (b).

(b) Nonlinear qquations

D05BEF is applicable to a nonlinear equation having a weakly-singular kernel (Abel). The kernel function has the form

$$K(t,s,y(s)) = \frac{k(t-s)}{\sqrt{t-s}}g(s,y(s)).$$

The underlying methods used in the routine are the fractional linear multistep methods based on BDF methods. This routine can also be used for linear g.

3.4 Utility Routines

D05BWF generates the weights associated with Adams and BDF linear multistep methods. These weights can be used for the solution of non-singular Volterra integral and integro-differential equations of general type.

D05BYF generates the weights associated with BDF linear multistep methods. These weights can be used for the solution of weakly-singular Volterra (Abel) integral equations of general type.

3.5 User-supplied Routines

All the routines in this chapter require the user to supply functions or real procedures defining the kernels and other given functions in the equations. It is important to test these independently before using them in conjunction with NAG Library routines.

4 Index

5 References

[1] Atkinson K E (1976) *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind* SIAM, Philadelphia

[2] Baker C T H (1977) *The Numerical Treatment of Integral Equations* Oxford University Press

[3] Brunner H and Van Der Houwen P J (1986) *The Numerical Solution of Volterra Equations* CWI Monographs, North-Holland, Amsterdam

[4] Cochran J A (1972) *The Analysis of Linear Integral Equations* McGraw-Hill

[5] Delves L M and Walsh J (1974) *Numerical Solution of Integral Equations* Clarendon Press, Oxford

[6] Wolkenfelt P H M (1982) The construction of reducible quadrature rules for Volterra integral and integro-differential equations *IMA J. Numer. Anal.* **2** 131–152

Chapter E01 – Interpolation

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
E01AAF	1	Interpolated values, Aitken's technique, unequally spaced data, one variable
E01ABF	1	Interpolated values, Everett's formula, equally spaced data, one variable
E01AEF	8	Interpolating functions, polynomial interpolant, data may include derivative values, one variable
E01BAF	8	Interpolating functions, cubic spline interpolant, one variable
E01BEF	13	Interpolating functions, monotonicity-preserving, piecewise cubic Hermite, one variable
E01BFF	13	Interpolated values, interpolant computed by E01BEF, function only, one variable
E01BGF	13	Interpolated values, interpolant computed by E01BEF, function and first derivative, one variable
E01BHF	13	Interpolated values, interpolant computed by E01BEF, definite integral, one variable
E01DAF	14	Interpolating functions, fitting bicubic spline, data on rectangular grid
E01RAF	9	Interpolating functions, rational interpolant, one variable
E01RBF	9	Interpolated values, evaluate rational interpolant computed by E01RAF, one variable
E01SAF	13	Interpolating functions, method of Renka and Cline, two variables
E01SBF	13	Interpolated values, evaluate interpolant computed by E01SAF, two variables
E01SEF*	13	Interpolating functions, modified Shepard's method, two variables
E01SFF*	13	Interpolated values, evaluate interpolant computed by E01SEF, two variables
E01SGF	18	Interpolating functions, modified Shepard's method, two variables
E01SHF	18	Interpolated values, evaluate interpolant computed by E01SGF, function and first derivatives, two variables
E01TGF	18	Interpolating functions, modified Shepard's method, three variables
E01THF	18	Interpolated values, evaluate interpolant computed by E01TGF, function and first derivatives, three variables

* This routine is scheduled for withdrawal at Mark 20. See the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines' for details of the recommended replacement routine.

Chapter E01

Interpolation

Contents

1 Scope of the Chapter

This chapter is concerned with the interpolation of a function of one, two or three variables. When provided with the value of the function (and possibly one or more of its lowest-order derivatives) at each of a number of values of the variable(s), the routines provide either an interpolating function or an interpolated value. For some of the interpolating functions, there are supporting routines to evaluate, differentiate or integrate them.

2 Background to the Problems

In motivation and in some of its numerical processes, this chapter has much in common with the E02 Chapter Introduction (Curve and Surface Fitting). For this reason, we shall adopt the same terminology and refer to dependent variable and independent variable(s) instead of function and variable(s). Where there is only one independent variable, we shall denote it by x and the dependent variable by y. Thus, in the basic problem considered in this chapter, we are given a set of distinct values $x_1, x_2, \ldots, x_m$ of x and a corresponding set of values $y_1, y_2, \ldots, y_m$ of y, and we shall describe the problem as being one of interpolating the data points (x_r, y_r), rather than interpolating a function. In modern usage, however, **interpolation** can have either of two rather different meanings, both relevant to routines in this chapter. They are

(a) the determination of a function of x which takes the value y_r at $x = x_r$, for $r = 1, 2, \ldots, m$ (an **interpolating function** or **interpolant**),

(b) the determination of the value (**interpolated value** or **interpolate**) of an interpolating function at any given value, say $\hat{x}$, of x within the range of the x_r (so as to estimate the value at $\hat{x}$ of the function underlying the data).

The latter is the older meaning, associated particularly with the use of mathematical tables. The term 'function underlying the data', like the other terminology described above, is used so as to cover situations additional to those in which the data points have been computed from a known function, as with a mathematical table. In some contexts, the function may be unknown, perhaps representing the dependency of one physical variable on another, say temperature upon time.

Whether the underlying function is known or unknown, the object of interpolation will usually be to approximate it to acceptable accuracy by a function which is easy to evaluate anywhere in some range of interest. Polynomials, rational functions (ratios of two polynomials) and piecewise polynomials, such as cubic splines (see Section 2.2 of the E02 Chapter Introduction for definitions of terms in the latter case), being easy to evaluate and also capable of approximating a wide variety of functions, are the types of function mostly used in this chapter as interpolating functions. An interpolating polynomial is taken to have degree $m-1$ when there are m data points, and so it is unique. It is called the **Lagrange interpolating polynomial**. The rational function, in the special form used, is also unique. An interpolating spline, on the other hand, depends on the choice made for the knots.

One way of achieving the objective in (b) above is, of course, through (a), but there are also methods which do not involve the explicit computation of the interpolating function. Everett's formula and Aitken's successive linear interpolation (see Froberg [2]) provide two such methods. Both are used in this chapter and determine a value of the Lagrange interpolating polynomial.

It is important to appreciate, however, that the Lagrange interpolating polynomial often exhibits unwanted fluctuations between the data points. These tend to occur particularly towards the ends of the data range, and to get larger with increasing number of data points. In severe cases, such as with 30 or 40 equally spaced values of x, the polynomial can take on values several orders of magnitude larger than the data values. (Closer spacing near the ends of the range tends to improve the situation, and wider spacing tends to make it worse.) Clearly, therefore, the Lagrange polynomial often gives a very poor approximation to the function underlying the data. On the other hand, it can be perfectly satisfactory when its use is restricted to providing interpolated values away from the ends of the data range from a reasonably small number of data values.

In contrast, a cubic spline which interpolates a large number of data points can often be used satisfactorily over the whole of the data range. Unwanted fluctuations can still arise but much less frequently and much less severely than with polynomials. Rational functions, when appropriate, would also be used over the whole data range. The main danger with these functions is that their polynomial denominators may take zero values within that range. Unwanted fluctuations are avoided altogether by a routine using piecewise cubic polynomials having only first derivative continuity. It is designed especially for monotonic data, but for other data still provides an interpolant which increases, or decreases, over the same intervals as the data.

The concept of interpolation can be generalised in a number of ways. Firstly, at each x, the interpolating function may be required to take on not only a given value but also given values for all its derivatives up to some specified order (which can vary with r). This is the Hermite–Birkoff interpolation problem. Secondly, we may be required to estimate the value of the underlying function at a value $\hat{x}$ outside the range of the data. This is the process of **extrapolation**. In general, it is a good deal less accurate than interpolation and is to be avoided whenever possible.

Interpolation can also be extended to the case of two or more independent variables. If the data values are given at the intersections of a regular two-dimensional mesh bicubic splines (see Section 2.3.2 of the E02 Chapter Introduction) are very suitable and usually very effective for the problem. For other cases, perhaps where the data values are quite arbitrarily scattered, polynomials and splines are not at all appropriate and special forms of interpolating function have to be employed. Many such forms have been devised and two of the most successful are in routines in this chapter. They both have continuity in first, but not higher, derivatives.

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 General

Before undertaking interpolation, in other than the simplest cases, the user should seriously consider the alternative of using a routine from the E02 Chapter Introduction to approximate the data by a polynomial or spline containing significantly fewer coefficients than the corresponding interpolating function. This approach is much less liable to produce unwanted fluctuations and so can often provide a better approximation to the function underlying the data.

When interpolation is employed to approximate either an underlying function or its values, the user will need to be satisfied that the accuracy of approximation achieved is adequate. There may be a means for doing this which is particular to the application, or the routine used may itself provide a means. In other cases, one possibility is to repeat the interpolation using one or more extra data points, if they are available, or otherwise one or more fewer, and to compare the results. Other possibilities, if it is an interpolating function which is determined, are to examine the function graphically, if that gives sufficient accuracy, or to observe the behaviour of the differences in a finite-difference table, formed from evaluations of the interpolating function at equally-spaced values of x over the range of interest. The spacing should be small enough to cause the typical size of the differences to decrease as the order of difference increases.

3.2 One Independent Variable

3.2.1 Interpolated values: data without derivatives

When the underlying function is well represented by data points on both sides of the value, $\hat{x}$, at which an interpolated value is required, E01ABF should be tried first if the data points are equally spaced, E01AAF if they are not. Both compute a value of the Lagrange interpolating polynomial, the first using Everett's formula, the second Aitken's successive linear interpolation. The first routine requires an equal (or nearly equal) number of data points on each side of $\hat{x}$; such a distribution of points is preferable also for the second routine. If there are many data points, this will be achieved simply by using only an appropriate subset for each value of $\hat{x}$. Ten to twelve data points are the most that would be required for many problems. Both routines provide a means of assessing the accuracy of an interpolated value, with E01ABF by examination of the size of the finite differences supplied, with E01AAF by intercomparison of the set of interpolated values obtained from polynomials of increasing degree.

In other cases, or when the above routines fail to produce a satisfactory result, one of the routines discussed in the next section should be used. The spline and other piecewise polynomial routines are the most generally applicable. They are particularly appropriate when interpolated values towards the ends of the range are required. They are also likely to be preferable, for reasons of economy, when many interpolated values are required.

E01AAF above, and three of the routines discussed in the next section, can be used to compute extrapolated values. These three are E01AEF, E01BEF and E01RAF based on polynomials, piecewise polynomials and rational functions respectively. Extrapolation is not recommended in general, but can sometimes give acceptable results if it is to a point not far outside the data range, and only the few nearest data points are used in the process. E01RAF is most likely to be successful.

3.2.2 Interpolating function: data without derivatives

E01AEF computes the Lagrange interpolating polynomial by a method (based on **Newton's formula with divided differences** [1]) which has proved numerically very stable. Thus, it can sometimes be used to provide interpolated values in more difficult cases than can E01AAF (see previous section). However, the likelihood of the polynomial having unwanted fluctuations, particularly near the ends of the data range when a moderate or large number of data points are used, should be remembered.

Such fluctuations of the polynomial can be avoided if the user is at liberty to choose the x-values at which to provide data points. In this case, a routine from Chapter E02, namely E02AFF, should be used in the manner and with the x-values discussed in Section 3.2.2 of the E02 Chapter Introduction.

Usually however, when the whole of the data range is of interest, it is preferable to use a cubic spline as the interpolating function. E01BAF computes an interpolating cubic spline, using a particular choice for the set of knots which has proved generally satisfactory in practice. If the user wishes to choose a different set, a cubic spline routine from Chapter E02, namely E02BAF, may be used in its interpolating mode, setting NCAP7 = M + 4 and all elements of the parameter W to unity.

The cubic spline does not always avoid unwanted fluctuations, especially when the data show a steep slope close to a region of small slope, or when the data inadequately represent the underlying curve. In such cases, E01BEF can be very useful. It derives a piecewise cubic polynomial (with first derivative continuity) which, between any adjacent pair of data points, either increases all the way, or decreases all the way (or stays constant). It is especially suited to data which are monotonic over their whole range.

In this routine, the interpolating function is represented simply by its value and first derivative at the data points. Supporting routines compute its value and first derivative elsewhere, as well as its definite integral over an arbitary interval. The other routines above provide the interpolating function either in Chebyshev-series form or in B-spline form (see Section 2.2.1 of the E02 Chapter Introduction and Section 2.2.2 of the E02 Chapter Introduction). Routines for evaluating, differentiating and integrating these forms are discussed in Section 3.7 of the E02 Chapter Introduction. The splines and other piecewise cubics will normally provide better estimates of the derivatives of the underlying function than will interpolating polynomials, at any rate away from the central part of the data range.

E01RAF computes an interpolating rational function. It is intended mainly for those cases where the user knows that this form of function is appropriate. However, it is also worth trying in cases where the other routines have proved unsatisfactory. E01RBF is available to compute values of the function provided by E01RAF.

3.2.3 Data containing derivatives

E01AEF (see previous section) can also compute the polynomial which, at each x_r, has not only a specified value y_r but also a specified value of each derivative up to order p_r.

3.3 Two Independent Variables

3.3.1 Data on a rectangular mesh

Given the value f_{qr} of the dependent variable f at the point (x_q, y_r) in the plane of the independent variables x and y, for each $q = 1, 2, \ldots, m$ and $r = 1, 2, \ldots, n$ (so that the points (x_q, y_r) lie at the $m \times n$ intersections of a rectangular mesh), E01DAF computes an interpolating bicubic spline, using a particular choice for each of the spline's knot-set. This choice, the same as in E01BAF, has proved generally satisfactory in practice. If, instead, the user wishes to specify his own knots, a routine from Chapter E02, namely E02DAF, may be adapted (it is more cumbersome for the purpose, however, and much slower for larger problems). Using m and n in the above sense, the parameter M must be set to

$m \times n$, PX and PY must be set to $m + 4$ and $n + 4$ respectively and all elements of W should be set to unity. The recommended value for EPS is zero.

3.3.2 Arbitrary data

As remarked at the end of Section 2, special types of interpolating are required for this problem, which can often be difficult to solve satisfactorily. Two of the most successful are employed in E01SAF and E01SGF, the two routines which (with their respective evaluation routines E01SBF and E01SHF) are provided for the problem. Definitions can be found in the routine documents. Both interpolants have first derivative continuity and are 'local', in that their value at any point depends only on data in the immediate neighbourhood of the point. This latter feature is necessary for large sets of data to avoid prohibitive computing time. E01SHF allows evaluation of the interpolant and its first partial derivatives.

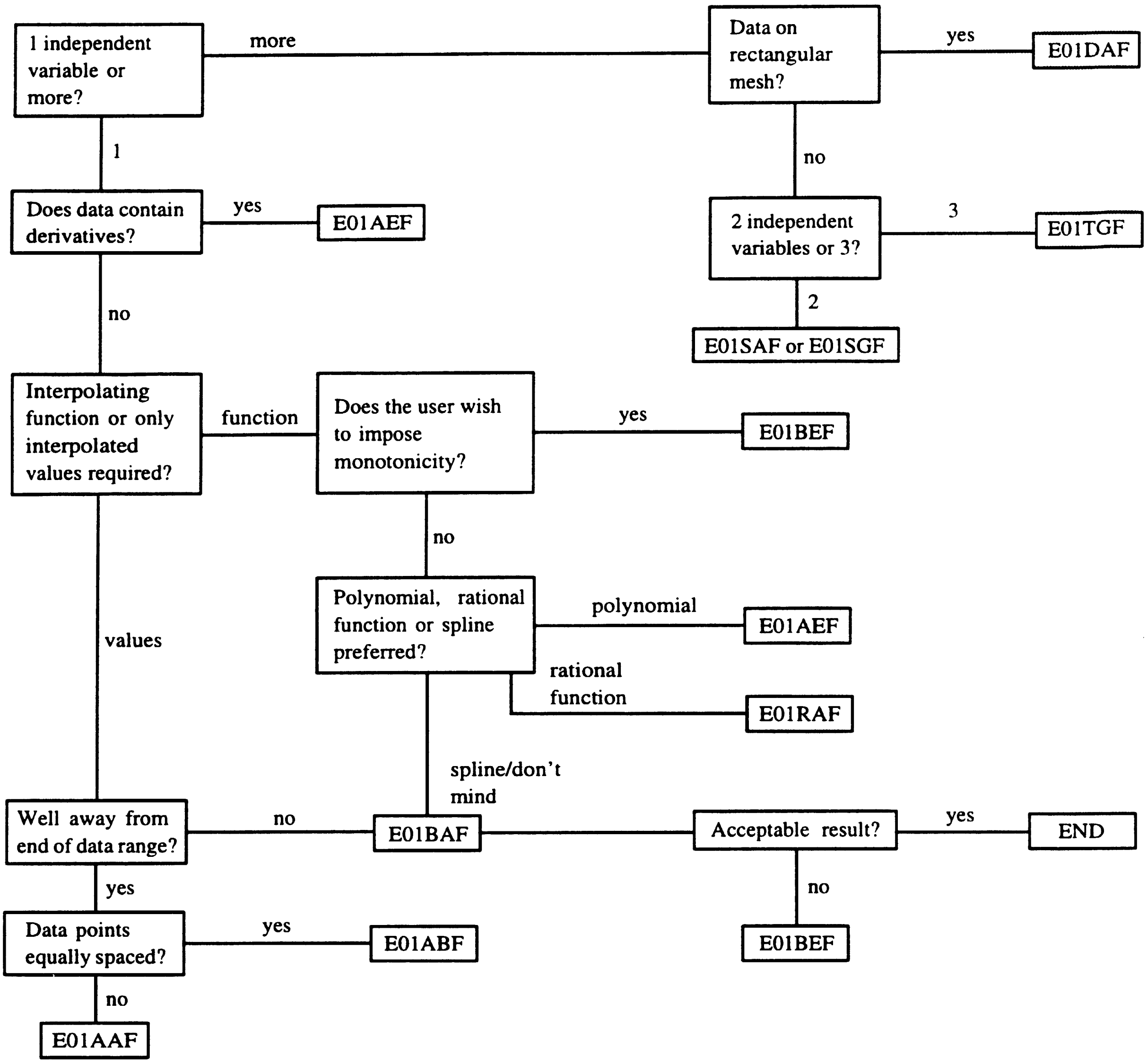
The relative merits of the two methods vary with the data and it is not possible to predict which will be the better in any particular case.

3.4 Three Independent Variables

3.4.1 Arbitrary data

The routine E01TGF and its evaluation routine E01THF are provided for interpolation of three dimensional scattered data. As in the case of two independent variables, the method is local, and produces an interpolant with first derivative continuity. E01THF allows evaluation of the interpolant and its first partial derivatives.

4 Decision Trees



5 Index

from polynomial,	
equally spaced data	E01ABF
general data	E01AAF
from rational function	E01RBF
Interpolated values, two variables,	
from interpolant from E01SAF	E01SBF
from interpolant from E01SGF	E01SHF
Interpolating function, one variable,	
cubic spline	E01BAF
other piecewise polynomial	E01BEF
polynomial, data with or without derivatives	E01AEF
rational function	E01RAF
Interpolating function, two variables	
bicubic spline	E01DAF
other piecewise polynomial	E01SAF
modified Shepard method	E01SGF
Interpolating function, three variables	
modified Shepard method	E01TGF

6 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Those routines indicated by a dagger are still present at Mark 19, but will be omitted at a future date. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

E01ACF   E01SEF†   E01SFF†

7 References

[1] Froberg C E (1970) *Introduction to Numerical Analysis* Addison–Wesley

[2] Dahlquist G and Björck Å(1974) *Numerical Methods* Prentice–Hall

Chapter E02 – Curve and Surface Fitting

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
E02ACF	1	Minimax curve fit by polynomials
E02ADF	5	Least-squares curve fit, by polynomials, arbitrary data points
E02AEF	5	Evaluation of fitted polynomial in one variable from Chebyshev series form (simplified parameter list)
E02AFF	5	Least-squares polynomial fit, special data points (including interpolation)
E02AGF	8	Least-squares polynomial fit, values and derivatives may be constrained, arbitrary data points
E02AHF	8	Derivative of fitted polynomial in Chebyshev series form
E02AJF	8	Integral of fitted polynomial in Chebyshev series form
E02AKF	8	Evaluation of fitted polynomial in one variable from Chebyshev series form
E02BAF	5	Least-squares curve cubic spline fit (including interpolation)
E02BBF	5	Evaluation of fitted cubic spline, function only
E02BCF	7	Evaluation of fitted cubic spline, function and derivatives
E02BDF	7	Evaluation of fitted cubic spline, definite integral
E02BEF	13	Least-squares cubic spline curve fit, automatic knot placement
E02CAF	7	Least-squares surface fit by polynomials, data on lines
E02CBF	7	Evaluation of fitted polynomial in two variables
E02DAF	6	Least-squares surface fit, bicubic splines
E02DCF	13	Least-squares surface fit by bicubic splines with automatic knot placement, data on rectangular grid
E02DDF	13	Least-squares surface fit by bicubic splines with automatic knot placement, scattered data
E02DEF	14	Evaluation of fitted bicubic spline at a vector of points
E02DFF	14	Evaluation of fitted bicubic spline at a mesh of points
E02GAF	7	L_1-approximation by general linear function
E02GBF	7	L_1-approximation by general linear function subject to linear inequality constraints
E02GCF	8	L_∞-approximation by general linear function
E02RAF	7	Padé-approximants
E02RBF	7	Evaluation of fitted rational function as computed by E02RAF
E02ZAF	6	Sort two-dimensional data into panels for fitting bicubic splines

Chapter E02

Curve and Surface Fitting

Contents

1 Scope of the Chapter

The main aim of this chapter is to assist the user in finding a function which approximates a set of data points. Typically the data contain random errors, as of experimental measurement, which need to be smoothed out. To seek an approximation to the data, it is first necessary to specify for the approximating function a mathematical form (a polynomial, for example) which contains a number of unspecified coefficients: the appropriate fitting routine then derives for the coefficients the values which provide the best fit of that particular form. The chapter deals mainly with curve and surface fitting (i.e., fitting with functions of one and of two variables) when a polynomial or a cubic spline is used as the fitting function, since these cover the most common needs. However, fitting with other functions and/or more variables can be undertaken by means of general linear or nonlinear routines (some of which are contained in other chapters) depending on whether the coefficients in the function occur linearly or nonlinearly. Cases where a graph rather than a set of data points is given can be treated simply by first reading a suitable set of points from the graph.

The chapter also contains routines for evaluating, differentiating and integrating polynomial and spline curves and surfaces, once the numerical values of their coefficients have been determined.

There is, too, a routine for computing a Padé approximant of a mathematical function (see Section 2.6 and Section 3.8).

2 Background to the Problems

2.1 Preliminary Considerations

In the curve-fitting problems considered in this chapter, we have a dependent variable y and an independent variable x, and we are given a set of data points (x_r, y_r), for $r = 1, 2, \ldots, m$. The preliminary matters to be considered in this section will, for simplicity, be discussed in this context of curve-fitting problems. In fact, however, these considerations apply equally well to surface and higher-dimensional problems. Indeed, the discussion presented carries over essentially as it stands if, for these cases, we interpret x as a vector of several independent variables and correspondingly each x_r as a vector containing the rth data value of each independent variable.

We wish, then, to approximate the set of data points as closely as possible with a specified function, $f(x)$ say, which is as smooth as possible: $f(x)$ may, for example, be a polynomial. The requirements of smoothness and closeness conflict, however, and a balance has to be struck between them. Most often, the smoothness requirement is met simply by limiting the number of coefficients allowed in the fitting function – for example, by restricting the degree in the case of a polynomial. Given a particular number of coefficients in the function in question, the fitting routines of this chapter determine the values of the coefficients such that the 'distance' of the function from the data points is as small as possible. The necessary balance is struck by the user comparing a selection of such fits having different numbers of coefficients. If the number of coefficients is too low, the approximation to the data will be poor. If the number is too high, the fit will be too close to the data, essentially following the random errors and tending to have unwanted fluctuations between the data points. Between these extremes, there is often a group of fits all similarly close to the data points and then, particularly when least-squares polynomials are used, the choice is clear: it is the fit from this group having the smallest number of coefficients.

The above process can be seen as the user minimizing the smoothness measure (i.e., the number of coefficients) subject to the distance from the data points being acceptably small. Some of the routines, however, do this task themselves. They use a different measure of smoothness (in each case one that is continuous) and minimize it subject to the distance being less than a threshold specified by the user. This is a much more automatic process, requiring only some experimentation with the threshold.

2.1.1 Fitting criteria: norms

A measure of the above 'distance' between the set of data points and the function $f(x)$ is needed. The distance from a single data point (x_r, y_r) to the function can simply be taken as

$$\epsilon_r = y_r - f(x_r), \qquad (1)$$

and is called the **residual** of the point. (With this definition, the residual is regarded as a function of the coefficients contained in $f(x)$; however, the term is also used to mean the particular value of ϵ_r which corresponds to the fitted values of the coefficients.) However, we need a measure of distance for the set of data points as a whole. Three different measures are used in the different routines (which measure to select, according to circumstances, is discussed later in this sub-section). With ϵ_r defined in (1), these measures, or **norms**, are

$$\sum_{r=1}^{m} |\epsilon_r|, \tag{2}$$

$$\sqrt{\sum_{r=1}^{m} \epsilon_r^2}, \tag{3}$$

and

$$\max_r |\epsilon_r|, \tag{4}$$

respectively the l_1 norm, the l_2 norm and the l_∞ norm.

Minimization of one or other of these norms usually provides the fitting criterion, the minimization being carried out with respect to the coefficients in the mathematical form used for $f(x)$: with respect to the b_i for example if the mathematical form is the power series in (8) below. The fit which results from minimizing (2) is known as the l_1 fit, or the fit in the l_1 norm: that which results from minimizing (3) is the l_2 fit, the well-known least-squares fit (minimizing (3) is equivalent to minimizing the square of (3), i.e., the sum of squares of residuals, and it is the latter which is used in practice), and that from minimizing (4) is the l_∞, or minimax, fit.

Strictly speaking, implicit in the use of the above norms are the statistical assumptions that the random errors in the y_r are independent of one another and that any errors in the x_r are negligible by comparison. From this point of view, the use of the l_2 norm is appropriate when the random errors in the y_r have a normal distribution, and the l_∞ norm is appropriate when they have a rectangular distribution, as when fitting a table of values rounded to a fixed number of decimal places. The l_1 norm is appropriate when the error distribution has its frequency function proportional to the negative exponential of the modulus of the normalised error – not a common situation.

However, the user is often indifferent to these statistical considerations, and simply seeks a fit which can be assessed by inspection, perhaps visually from a graph of the results. In this event, the l_1 norm is particularly appropriate when the data are thought to contain some 'wild' points (since fitting in this norm tends to be unaffected by the presence of a small number of such points), though of course in simple situations the user may prefer to identify and reject these points. The l_∞ norm should be used only when the maximum residual is of particular concern, as may be the case for example when the data values have been obtained by accurate computation, as of a mathematical function. Generally, however, a routine based on least-squares should be preferred, as being computationally faster and usually providing more information on which to assess the results. In many problems the three fits will not differ significantly for practical purposes.

Some of the routines based on the l_2 norm do not minimize the norm itself but instead minimize some (intuitively acceptable) measure of smoothness subject to the norm being less than a user-specified threshold. These routines fit with cubic or bicubic splines (see (10) and (14) below) and the smoothing measures relate to the size of the discontinuities in their third derivatives. A much more automatic fitting procedure follows from this approach.

2.1.2 Weighting of data points

The use of the above norms also assumes that the data values y_r are of equal (absolute) accuracy. Some of the routines enable an allowance to be made to take account of differing accuracies. The allowance takes the form of 'weights' applied to the y-values so that those values known to be more accurate have a greater influence on the fit than others. These weights, to be supplied by the user, should be calculated from estimates of the absolute accuracies of the y-values, these estimates being expressed as standard deviations, probable errors or some other measure which has the same dimensions as y. Specifically, for each y_r the corresponding weight w_r should be inversely proportional to the accuracy estimate of y_r. For example, if the percentage accuracy is the same for all y_r, then the absolute accuracy of y_r is proportional to y_r (assuming y_r to be positive, as it usually is in such cases) and so $w_r = K/y_r$, for $r = 1, 2, \ldots, m$, for an arbitrary positive constant K. (This definition of weight is stressed because often weight is defined as the square of that used here.) The norms (2), (3) and (4) above are then replaced respectively by

$$\sum_{r=1}^{m} |w_r \epsilon_r|, \tag{5}$$

$$\sqrt{\sum_{r=1}^{m} w_r^2 \epsilon_r^2}, \tag{6}$$

and

$$\max_r |w_r \epsilon_r|. \tag{7}$$

Again it is the square of (6) which is used in practice rather than (6) itself.

2.2 Curve Fitting

When, as is commonly the case, the mathematical form of the fitting function is immaterial to the problem, polynomials and cubic splines are to be preferred because their simplicity and ease of handling confer substantial benefits. The **cubic spline** is the more versatile of the two. It consists of a number of cubic polynomial segments joined end to end with continuity in first and second derivatives at the joins. The third derivative at the joins is in general discontinuous. The x-values of the joins are called **knots**, or, more precisely, interior knots. Their number determines the number of coefficients in the spline, just as the degree determines the number of coefficients in a polynomial.

2.2.1 Representation of polynomials

Two different forms for representing a polynomial are used in different routines. One is the usual power-series form

$$f(x) \equiv b_0 + b_1 x + b_2 x^2 + \ldots + b_k x^k. \tag{8}$$

The other is the Chebyshev series form

$$f(x) \equiv \tfrac{1}{2} a_0 T_0(x) + a_1 T_1(x) + a_2 T_2(x) + \ldots + a_k T_k(x), \tag{9}$$

where $T_i(x)$ is the Chebyshev polynomial of the first kind of degree i (see Cox and Hayes [1], page 9), and where the range of x has been normalised to run from -1 to $+1$. The use of either form leads theoretically to the same fitted polynomial, but in practice results may differ substantially because of the effects of rounding error. The Chebyshev form is to be preferred, since it leads to much better accuracy in general, both in the computation of the coefficients and in the subsequent evaluation of the fitted polynomial at specified points. This form also has other advantages: for example, since the later terms in (9) generally decrease much more rapidly from left to right than do those in (8), the situation is more often encountered where the last terms are negligible and it is obvious that the degree of the polynomial can be reduced (note that on the interval $-1 \le x \le 1$ for all i, $T_i(x)$ attains the value unity but never exceeds it, so that the coefficient a_i gives directly the maximum value of the term containing it). If the power-series form is used it is most advisable to work with the variable x normalised to the range -1 to $+1$, carrying out the normalisation before entering the relevant routine. This will often substantially improve computational accuracy.

2.2.2 Representation of cubic splines

A cubic spline is represented in the form

$$f(x) \equiv c_1 N_1(x) + c_2 N_2(x) + \ldots + c_p N_p(x), \tag{10}$$

where $N_i(x)$, for $i = 1, 2, \ldots, p$, is a normalised cubic B-spline (see Hayes [2]). This form, also, has advantages of computational speed and accuracy over alternative representations.

2.3 Surface Fitting

There are now two independent variables, and we shall denote these by x and y. The dependent variable, which was denoted by y in the curve-fitting case, will now be denoted by f. (This is a rather different notation from that indicated for the general-dimensional problem in the first paragraph of Section 2.1, but it has some advantages in presentation.)

Again, in the absence of contrary indications in the particular application being considered, polynomials and splines are the approximating functions most commonly used.

2.3.1 Representation of bivariate polynomials

The type of bivariate polynomial currently considered in the chapter can be represented in either of the two forms

$$f(x,y) \equiv \sum_{i=0}^{k}\sum_{j=0}^{l} b_{ij}x^i y^j, \tag{11}$$

and

$$f(x,y) \equiv {\sum_{i=0}^{k}}'{\sum_{j=0}^{l}}' a_{ij}T_i(x)T_j(y), \tag{12}$$

where $T_i(x)$ is the Chebyshev polynomial of the first kind of degree i in the argument x (see Cox and Hayes [1] page 9), and correspondingly for $T_j(y)$. The prime on the two summation signs, following standard convention, indicates that the first term in each sum is halved, as shown for one variable in equation (9). The two forms (11) and (12) are mathematically equivalent, but again the Chebyshev form is to be preferred on numerical grounds, as discussed in Section 2.2.1.

2.3.2 Bicubic splines: definition and representation

The bicubic spline is defined over a rectangle R in the (x,y) plane, the sides of R being parallel to the x- and y-axes. R is divided into rectangular panels, again by lines parallel to the axes. Over each panel the bicubic spline is a bicubic polynomial, that is it takes the form

$$\sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j. \tag{13}$$

Each of these polynomials joins the polynomials in adjacent panels with continuity up to the second derivative. The constant x-values of the dividing lines parallel to the y-axis form the set of interior knots for the variable x, corresponding precisely to the set of interior knots of a cubic spline. Similarly, the constant y-values of dividing lines parallel to the x-axis form the set of interior knots for the variable y. Instead of representing the bicubic spline in terms of the above set of bicubic polynomials, however, it is represented, for the sake of computational speed and accuracy, in the form

$$f(x,y) = \sum_{i=1}^{p}\sum_{j=1}^{q} c_{ij}M_i(x)N_j(y), \tag{14}$$

where $M_i(x)$, for $i = 1,2,\ldots,p$, and $N_j(y)$, for $j = 1,2,\ldots,q$, are normalised B-splines (see Hayes and Halliday [4] for further details of bicubic splines and Hayes [2] for normalised B-splines).

2.4 General Linear and Nonlinear Fitting Functions

We have indicated earlier that, unless the data-fitting application under consideration specifically requires some other type of fitting function, a polynomial or a spline is usually to be preferred. Special routines for these functions, in one and in two variables, are provided in this chapter. When the application does specify some other fitting function, however, it may be treated by a routine which deals with a general linear function, or by one for a general nonlinear function, depending on whether the coefficients in the given function occur linearly or nonlinearly.

The general linear fitting function can be written in the form

$$f(x) \equiv c_1\phi_1(x) + c_2\phi_2(x) + \ldots + c_p\phi_p(x), \tag{15}$$

where x is a vector of one or more independent variables, and the ϕ_i are any given functions of these variables (though they must be linearly independent of one another if there is to be the possibility of a unique solution to the fitting problem). This is not intended to imply that each ϕ_i is necessarily a function of all the variables: we may have, for example, that each ϕ_i is a function of a different single variable, and even that one of the ϕ_i is a constant. All that is required is that a value of each $\phi_i(x)$ can be computed when a value of each independent variable is given.

When the fitting function $f(x)$ is not linear in its coefficients, no more specific representation is available in general than $f(x)$ itself. However, we shall find it helpful later on to indicate the fact that $f(x)$ contains a number of coefficients (to be determined by the fitting process) by using instead the notation $f(x;c)$, where c denotes the vector of coefficients. An example of a nonlinear fitting function is

$$f(x;c) \equiv c_1 + c_2 \exp(-c_4 x) + c_3 \exp(-c_5 x), \tag{16}$$

which is in one variable and contains five coefficients. Note that here, as elsewhere in this Chapter Introduction, we use the term 'coefficients' to include all the quantities whose values are to be determined by the fitting process, not just those which occur linearly. We may observe that it is only the presence of the coefficients c_4 and c_5 which makes the form (16) nonlinear. If the values of these two coefficients were known beforehand, (16) would instead be a linear function which, in terms of the general linear form (15), has $p = 3$ and

$$\phi_1(x) \equiv 1,\ \phi_2(x) \equiv \exp(-c_4 x),\ \text{and}\ \phi_3(x) \equiv \exp(-c_5 x).$$

We may note also that polynomials and splines, such as (9) and (14), are themselves linear in their coefficients. Thus if, when fitting with these functions, a suitable special routine is not available (as when more than two independent variables are involved or when fitting in the l_1 norm), it is appropriate to use a routine designed for a general linear function.

2.5 Constrained Problems

So far, we have considered only fitting processes in which the values of the coefficients in the fitting function are determined by an unconstrained minimization of a particular norm. Some fitting problems, however, require that further restrictions be placed on the determination of the coefficient values. Sometimes these restrictions are contained explicitly in the formulation of the problem in the form of equalities or inequalities which the coefficients, or some function of them, must satisfy. For example, if the fitting function contains a term $A\exp(-kx)$, it may be required that $k \geq 0$. Often, however, the equality or inequality constraints relate to the value of the fitting function or its derivatives at specified values of the independent variable(s), but these too can be expressed in terms of the coefficients of the fitting function, and it is appropriate to do this if a general linear or nonlinear routine is being used. For example, if the fitting function is that given in (10), the requirement that the first derivative of the function at $x = x_0$ be non-negative can be expressed as

$$c_1 N_1'(x_0) + c_2 N_2'(x_0) + \ldots + c_p N_p'(x_0) \geq 0, \tag{17}$$

where the prime denotes differentiation with respect to x and each derivative is evaluated at $x = x_0$. On the other hand, if the requirement had been that the derivative at $x = x_0$ be exactly zero, the inequality sign in (17) would be replaced by an equality.

Routines which provide a facility for minimizing the appropriate norm subject to such constraints are discussed in Section 3.6.

2.6 Padé Approximants

A Padé approximant to a function $f(x)$ is a rational function (ratio of two polynomials) whose Maclaurin-series expansion is the same as that of $f(x)$ up to and including the term in x^k, where k is the sum of the degrees of the numerator and denominator of the approximant. Padé approximation can be a useful technique when values of a function are to be obtained from its Maclaurin series but convergence of the series is unacceptably slow or even non-existent.

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 General

The choice of a routine to treat a particular fitting problem will depend first of all on the fitting function and the norm to be used. Unless there is good reason to the contrary, the fitting function should be a polynomial or a cubic spline (in the appropriate number of variables) and the norm should be the l_2 norm (leading to the least-squares fit). If some other function is to be used, the choice of routine will depend on whether the function is nonlinear (in which case see Section 3.5.2) or linear in its coefficients (see Section 3.5.1), and, in the latter case, on whether the l_1, l_2 or l_∞ norm is to be used. The latter section is appropriate for polynomials and splines, too, if the l_1 or l_∞ norm is preferred, with one exception: there is a special routine for fitting polynomial curves in the unweighted l_∞ norm (see Section 3.2.3).

In the case of a polynomial or cubic spline, if there is only one independent variable, the user should choose a spline (Section 3.3) when the curve represented by the data is of complicated form, perhaps with several peaks and troughs. When the curve is of simple form, first try a polynomial (see Section 3.2) of low degree, say up to degree 5 or 6, and then a spline if the polynomial fails to provide a satisfactory fit. (Of course, if third-derivative discontinuities are unacceptable to the user, a polynomial is the only choice.) If the problem is one of surface fitting, the polynomial routine (Section 3.4.1) should be tried first if the data arrangement happens to be appropriate, otherwise one of the spline routines (Section 3.4.2). If the problem has more than two independent variables, it may be treated by the general linear routine in Section 3.5.1, again using a polynomial in the first instance.

Another factor which affects the choice of routine is the presence of constraints, as previously discussed in Section 2.5. Indeed this factor is likely to be overriding at present, because of the limited number of routines which have the necessary facility. Consequently those routines have been grouped together for discussion in Section 3.6.

3.1.1 Data considerations

A satisfactory fit cannot be expected by any means if the number and arrangement of the data points do not adequately represent the character of the underlying relationship: sharp changes in behaviour, in particular, such as sharp peaks, should be well covered. Data points should extend over the whole range of interest of the independent variable(s): extrapolation outside the data ranges is most unwise. Then, with polynomials, it is advantageous to have additional points near the ends of the ranges, to counteract the tendency of polynomials to develop fluctuations in these regions. When, with polynomial curves, the user can precisely choose the x-values of the data, the special points defined in Section 3.2.2 should be selected. With polynomial surfaces, each of these same x-values should, where possible, be combined with each of a corresponding set of y-values (not necessarily with the same value of n), thus forming a rectangular grid of (x, y)-values. With splines the choice is less critical as long as the character of the relationship is adequately represented. All fits should be tested graphically before accepting them as satisfactory.

For this purpose it should be noted that it is not sufficient to plot the values of the fitted function only at the data values of the independent variable(s); at the least, its values at a similar number of intermediate points should also be plotted, as unwanted fluctuations may otherwise go undetected. Such fluctuations are the less likely to occur the lower the number of coefficients chosen in the fitting function. No firm guide can be given, but as a rough rule, at least initially, the number of coefficients should not exceed half the number of data points (points with equal or nearly equal values of the independent variable, or both independent variables in surface fitting, counting as a single point for this purpose). However, the situation may be such, particularly with a small number of data points, that a satisfactorily close fit to the data cannot be achieved without unwanted fluctuations occurring. In such cases, it is often possible to improve the situation by a transformation of one or more of the variables, as discussed in the next section: otherwise it will be necessary to provide extra data points. Further advice on curve fitting is given in Cox and Hayes [1] and, for polynomials only, in Hayes [3]. Much of the advice applies also to surface fitting; see also the routine documents.

3.1.2 Transformation of variables

Before starting the fitting, consideration should be given to the choice of a good form in which to deal with each of the variables: often it will be satisfactory to use the variables as they stand, but sometimes the use of the logarithm, square root, or some other function of a variable will lead to a better-behaved relationship. This question is customarily taken into account in preparing graphs and tables of a relationship and the same considerations apply when curve or surface fitting. The practical context will often give a guide. In general, it is best to avoid having to deal with a relationship whose behaviour in one region is radically different from that in another. A steep rise at the left-hand end of a curve, for example, can often best be treated by curve fitting in terms of $\log(x + c)$ with some suitable value of the constant c. A case when such a transformation gave substantial benefit is discussed in page 60 of Hayes [3]. According to the features exhibited in any particular case, transformation of either dependent variable or independent variable(s) or both may be beneficial. When there is a choice it is usually better to transform the independent variable(s): if the dependent variable is transformed, the weights attached to the data points must be adjusted. Thus (denoting the dependent variable by y, as in the notation for curves) if the y_r to be fitted have been obtained by a transformation $y = g(Y)$ from original data values Y_r, with weights W_r, for $r = 1, 2, \ldots, m$, we must take

$$w_r = W_r/(dy/dY), \tag{18}$$

where the derivative is evaluated at Y_r. Strictly, the transformation of Y and the adjustment of weights are valid only when the data errors in the Y_r are small compared with the range spanned by the Y_r, but this is usually the case.

3.2 Polynomial Curves

3.2.1 Least-squares polynomials: arbitrary data points

E02ADF fits to arbitrary data points, with arbitrary weights, polynomials of all degrees up to a maximum degree k, which is a choice. If the user is seeking only a low-degree polynomial, up to degree 5 or 6 say, $k = 10$ is an appropriate value, providing there are about 20 data points or more. To assist in deciding the degree of polynomial which satisfactorily fits the data, the routine provides the root-mean-square residual s_i for all degrees $i = 1, 2, \ldots, k$. In a satisfactory case, these s_i will decrease steadily as i increases and then settle down to a fairly constant value, as shown in the example

i	s_i
0	3.5215
1	0.7708
2	0.1861
3	0.0820
4	0.0554
5	0.0251
6	0.0264
7	0.0280
8	0.0277
9	0.0297
10	0.0271

If the s_i values settle down in this way, it indicates that the closest polynomial approximation justified by the data has been achieved. The degree which first gives the approximately constant value of s_i (degree 5 in the example) is the appropriate degree to select. (Users who are prepared to accept a fit higher than sixth degree should simply find a high enough value of k to enable the type of behaviour indicated by the example to be detected: thus they should seek values of k for which at least 4 or 5 consecutive values of s_i are approximately the same.) If the degree were allowed to go high enough, s_i would, in most cases, eventually start to decrease again, indicating that the data points are being fitted too closely and that undesirable fluctuations are developing between the points. In some cases, particularly with a small number of data points, this final decrease is not distinguishable from the initial decrease in s_i. In such cases, users may seek an acceptable fit by examining the graphs of several of the polynomials obtained. Failing this, they may (a) seek a transformation of variables which improves the behaviour, (b) try fitting a spline, or (c) provide more data points. If data can be provided simply by drawing an approximating curve by hand and reading points from it, use the points discussed in Section 3.2.2.

3.2.2 Least-squares polynomials: selected data points

When users are at liberty to choose the x-values of data points, such as when the points are taken from a graph, it is most advantageous when fitting with polynomials to use the values $x_r = \cos(\pi r/n)$, for $r = 0, 1, \ldots, n$ for some value of n, a suitable value for which is discussed at the end of this section. Note that these x_r relate to the variable x after it has been normalised so that its range of interest is -1 to $+1$. E02ADF may then be used as in Section 3.2.1 to seek a satisfactory fit. However, if the ordinate values are of equal weight, as would often be the case when they are read from a graph, E02AFF is to be preferred, as being simpler to use and faster. This latter algorithm provides the coefficients a_j, for $j = 0, 1, \ldots, n$, in the Chebyshev series form of the polynomial of degree n which interpolates the data. In a satisfactory case, the later coefficients in this series, after some initial significant ones, will exhibit a random behaviour, some positive and some negative, with a size about that of the errors in the data or less. All these 'random' coefficients should be discarded, and the remaining (initial) terms of the series be taken as the approximating polynomial. This truncated polynomial is a least-squares fit to the data, though with the point at each end of the range given half the weight of each of the other points. The following example illustrates a case in which degree 5 or perhaps 6 would be chosen for the approximating polynomial.

j	a_j
0	9.315
1	−8.030
2	0.303
3	−1.483
4	0.256
5	−0.386
6	0.076
7	0.022
8	0.014
9	0.005
10	0.011
11	−0.040
12	0.017
13	−0.054
14	0.010
15	−0.034
16	−0.001

Basically, the value of n used needs to be large enough to exhibit the type of behaviour illustrated in the above example. A value of 16 is suggested as being satisfactory for very many practical problems, the required cosine values for this value of n being given in Cox and Hayes [1], page 11. If a satisfactory fit is not obtained, a spline fit should be tried, or, if the user is prepared to accept a higher degree of polynomial, n should be increased: doubling n is an advantageous strategy, since the set of values $\cos(\pi r/n)$, for $r = 0, 1, \ldots, n$, contains all the values of $\cos(\pi r/2n)$, for $r = 0, 1, \ldots, 2n$, so that the old data set will then be re-used in the new one. Thus, for example, increasing n from 16 to 32 will require only 16 new data points, a smaller number than for any other increase of n. If data points are particularly expensive to obtain, a smaller initial value than 16 may be tried, provided the user is satisfied that the number is adequate to reflect the character of the underlying relationship. Again, the number should be doubled if a satisfactory fit is not obtained.

3.2.3 Minimax space polynomials

E02ACF determines the polynomial of given degree which is a minimax space fit to arbitrary data points with equal weights. (If unequal weights are required, the polynomial must be treated as a general linear function and fitted using E02GCF.) To arrive at a satisfactory degree it will be necessary to try several different degrees and examine the results graphically. Initial guidance can be obtained from the value of the maximum residual: this will vary with the degree of the polynomial in very much the same way as does s_i in least-squares fitting, but it is much more expensive to investigate this behaviour in the same detail.

The algorithm uses the power-series form of the polynomial so for numerical accuracy it is advisable to normalise the data range of x to $[-1, 1]$.

3.3 Cubic Spline Curves

3.3.1 Least-squares cubic splines

E02BAF fits to arbitrary data points, with arbitrary weights, a cubic spline with interior knots specified by the user. The choice of these knots so as to give an acceptable fit must largely be a matter of trial and error, though with a little experience a satisfactory choice can often be made after one or two trials. It is usually best to start with a small number of knots (too many will result in unwanted fluctuations in the fit, or even in there being no unique solution) and, examining the fit graphically at each stage, to add a few knots at a time at places where the fit is particularly poor. Moving the existing knots towards these places will also often improve the fit. In regions where the behaviour of the curve underlying the data is changing rapidly, closer knots will be needed than elsewhere. Otherwise, positioning is not usually very critical and equally-spaced knots are often satisfactory. See also the next section, however.

A useful feature of the routine is that it can be used in applications which require the continuity to be less than the normal continuity of the cubic spline. For example, the fit may be required to have a discontinuous slope at some point in the range. This can be achieved by placing three coincident knots at the given point. Similarly a discontinuity in the second derivative at a point can be achieved by placing two knots there. Analogy with these discontinuous cases can provide guidance in more usual cases: for example, just as three coincident knots can produce a discontinuity in slope, so three close knots can produce a rapid change in slope. The closer the knots are, the more rapid can the change be.

An example set of data is given in Figure 1. It is a rather tricky set, because of the scarcity of data on the right, but it will serve to illustrate some of the above points and to show some of the dangers to be avoided. Three interior knots (indicated by the vertical lines at the top of the diagram) are chosen as a start. We see that the resulting curve is not steep enough in the middle and fluctuates at both ends, severely on the right. The spline is unable to cope with the shape and more knots are needed.

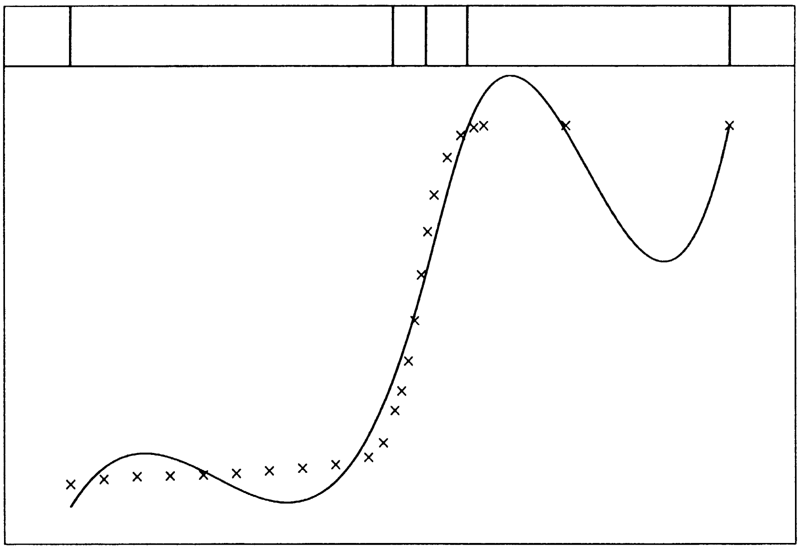


Figure 1

In Figure 2, three knots have been added in the centre, where the data shows a rapid change in behaviour, and one further out at each end, where the fit is poor. The fit is still poor, so a further knot is added in this region and, in Figure 3, disaster ensues in rather spectacular fashion.

The reason is that, at the right-hand end, the fits in Figure 1 and 2 have been interpreted as poor simply because of the fluctuations about the curve underlying the data (or what it is naturally assumed to be). But the fitting process knows only about the data and nothing else about the underlying curve, so it is important to consider only closeness to the data when deciding goodness of fit.

Thus, in Figure 1, the curve fits the last two data points quite well compared with the fit elsewhere, so no knot should have been added in this region. In Figure 2, the curve goes exactly through the last two points, so a further knot is certainly not needed here.

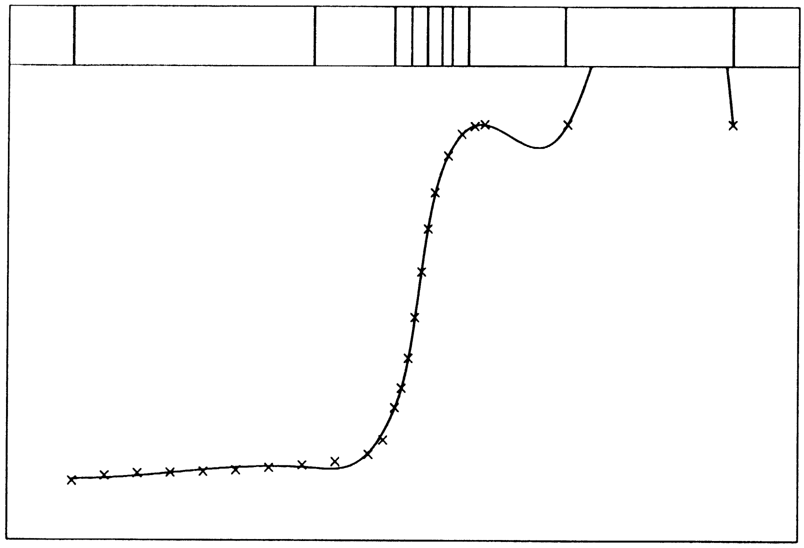


Figure 2

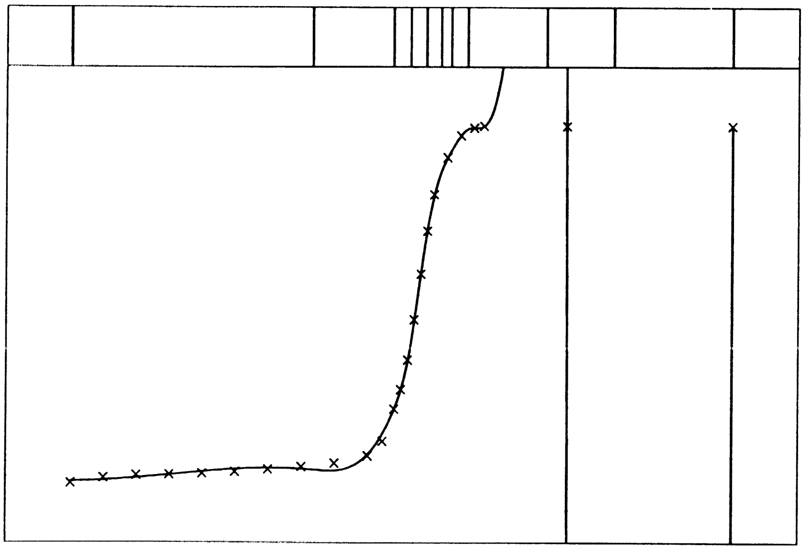


Figure 3

Figure 4 shows what can be achieved without the extra knot on each of the flat regions. Remembering that within each knot interval the spline is a cubic polynomial, there is really no need to have more than one knot interval covering each flat region.

What we have, in fact, in Figures 2 and 3 is a case of too many knots (so too many coefficients in the spline equation) for the number of data points. The warning in the second paragraph of Section 2.1 was that the fit will then be too close to the data, tending to have unwanted fluctuations between the data points. The warning applies locally for splines, in the sense that, in localities where there are plenty of data points, there can be a lot of knots, as long as there are few knots where there are few points, especially near the ends of the interval. In the present example, with so few data points on the right, just the one extra knot in Figure 2 is too many! The signs are clearly present, with the last two points fitted exactly (at least to the graphical accuracy and actually much closer than that) and fluctuations **within** the last two knot-intervals (cf. Figure 1, where only the final point is fitted exactly and one of the wobbles spans several data points).

The situation in Figure 3 is different. The fit, if computed exactly, **would** still pass through the last two data points, with even more violent fluctuations. However, the problem has become so ill-conditioned that all accuracy has been lost. Indeed, if the last interior knot were moved a tiny amount to the right,

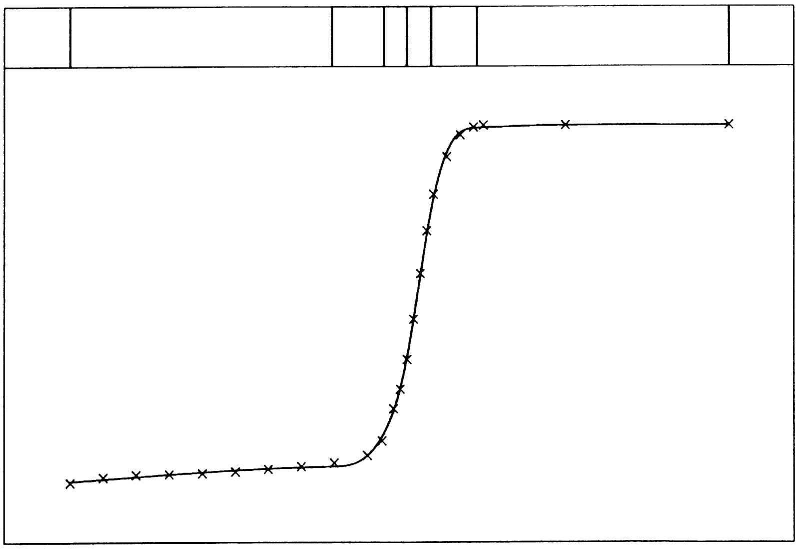


Figure 4

there would be no unique solution and an error message would have been caused. **Near-singularity** is, sadly, not picked up by the routine, but can be spotted readily in a graph, as Figure 3. B-spline coefficients becoming large, with alternating signs, is another indication. However, it is better to avoid such situations, firstly by providing, whenever possible, data adequately covering the range of interest, and secondly by placing knots only where there is a reasonable amount of data.

The example here could, in fact, have utilised from the start the observation made in the second paragraph of this section, that three close knots can produce a rapid change in slope. The example has two such rapid changes and so requires two sets of three close knots (in fact, the two sets can be so close that one knot can serve in both sets, so only five knots prove sufficient in Figure 4). It should be noted, however, that the rapid turn occurs within the range spanned by the three knots. This is the reason that the six knots in Figure 2 are not satisfactory as they do not quite span the two turns.

Some more examples to illustrate the choice of knots are given in Cox and Hayes [1].

3.3.2 Automatic fitting with cubic splines

E02BEF also fits cubic splines to arbitrary data points with arbitrary weights but itself chooses the number and positions of the knots. The user has to supply only a threshold for the sum of squares of residuals. The routine first builds up a knot set by a series of trial fits in the l_2 norm. Then, with the knot set decided, the final spline is computed to minimize a certain smoothing measure subject to satisfaction of the chosen threshold. Thus it is easier to use than E02BAF (see previous section), requiring only some experimentation with this threshold. It should therefore be first choice unless the user has a preference for the ordinary least-squares fit or, for example, wishes to experiment with knot positions, trying to keep their number down (E02BEF aims only to be reasonably frugal with knots).

3.4 Polynomial and Spline Surfaces

3.4.1 Least-squares polynomials

E02CAF fits bivariate polynomials of the form (12), with k and l specified by the user, to data points in a particular, but commonly occurring, arrangement. This is such that, when the data points are plotted in the plane of the independent variables x and y, they lie on lines parallel to the x-axis. Arbitrary weights are allowed. The matter of choosing satisfactory values for k and l is discussed in Section 8 of the routine document.

3.4.2 Least-squares bicubic splines

E02DAF fits to arbitrary data points, with arbitrary weights, a bicubic spline with its two sets of interior knots specified by the user. For choosing these knots, the advice given for cubic splines, in Section 3.3.1

above, applies here too. (See also the next section, however.) If changes in the behaviour of the surface underlying the data are more marked in the direction of one variable than of the other, more knots will be needed for the former variable than the latter. Note also that, in the surface case, the reduction in continuity caused by coincident knots will extend across the whole spline surface: for example, if three knots associated with the variable x are chosen to coincide at a value L, the spline surface will have a discontinuous slope across the whole extent of the line $x = L$.

With some sets of data and some choices of knots, the least-squares bicubic spline will not be unique. This will not occur, with a reasonable choice of knots, if the rectangle R is well covered with data points: here R is defined as the smallest rectangle in the (x, y) plane, with sides parallel to the axes, which contains all the data points. Where the least-squares solution is not unique, the minimal least-squares solution is computed, namely that least-squares solution which has the smallest value of the sum of squares of the B-spline coefficients c_{ij} (see the end of Section 2.3.2 above). This choice of least-squares solution tends to minimize the risk of unwanted fluctuations in the fit. The fit will not be reliable, however, in regions where there are few or no data points.

3.4.3 Automatic fitting with bicubic splines

E02DDF also fits bicubic splines to arbitrary data points with arbitrary weights but chooses the knot sets itself. The user has to supply only a threshold for the sum of squares of residuals. Just like the automatic curve E02BEF (Section 3.3.2), E02DDF then builds up the knot sets and finally fits a spline minimizing a smoothing measure subject to satisfaction of the threshold. Again, this easier to use routine is normally to be preferred, at least in the first instance.

E02DCF is a very similar routine to E02DDF but deals with data points of equal weight which lie on a rectangular mesh in the (x, y) plane. This kind of data allows a very much faster computation and so is to be preferred when applicable. Substantial departures from equal weighting can be ignored if the user is not concerned with statistical questions, though the quality of the fit will suffer if this is taken too far. In such cases, the user should revert to E02DDF.

3.5 General Linear and Nonlinear Fitting Functions

3.5.1 General linear functions

For the general linear function (15), routines are available for fitting in all three norms. The least-squares routines (which are to be preferred unless there is good reason to use another norm – see Section 2.1.1) are in Chapter F04. The l_∞ routine is E02GCF. Two routines for the l_1 norm are provided, E02GAF and E02GBF. Of these two, the former should be tried in the first instance, since it will be satisfactory in most cases, has a much shorter code and is faster. E02GBF, however, uses a more stable computational algorithm and therefore may provide a solution when E02GAF fails to do so. It also provides a facility for imposing linear inequality constraints on the solution (see Section 3.6).

All the above routines are essentially linear algebra routines, and in considering their use we need to view the fitting process in a slightly different way from hitherto. Taking y to be the dependent variable and x the vector of independent variables, we have, as for equation (1) but with each x_r now a vector,

$$\epsilon_r = y_r - f(x_r), \quad r = 1, 2, \ldots, m.$$

Substituting for $f(x)$ the general linear form (15), we can write this as

$$c_1\phi_1(x_r) + c_2\phi_2(x_r) + \ldots + c_p\phi_p(x_r) = y_r - \epsilon_r, \quad r = 1, 2, \ldots, m. \tag{19}$$

Thus we have a system of linear equations in the coefficients c_j. Usually, in writing these equations, the ϵ_r are omitted and simply taken as implied. The system of equations is then described as an overdetermined system (since we must have $m \geq p$ if there is to be the possibility of a unique solution to our fitting problem), and the fitting process of computing the c_j to minimize one or other of the norms (2), (3) and (4) can be described, in relation to the system of equations, as solving the overdetermined system in that particular norm. In matrix notation, the system can be written as

$$\Phi c = y, \tag{20}$$

where Φ is the m by p matrix whose element in row r and column j is $\phi_j(x_r)$, for $r = 1, 2, \ldots, m$; $j = 1, 2, \ldots, p$. The vectors c and y respectively contain the coefficients c_j and the data values y_r.

All four routines, however, use the standard notation of linear algebra, the overdetermined system of equations being denoted by

$$Ax = b. \tag{21}$$

(In fact, F04AMF can deal with several right-hand sides simultaneously, and thus is concerned with a matrix of right-hand sides, denoted by B, instead of the single vector b, and correspondingly with a matrix X of solutions instead of the single vector x.) The correspondence between this notation and that which we have used for the data-fitting problem (20) is therefore given by

$$\begin{aligned} A &\equiv \Phi, \\ x &\equiv c, \\ b &\equiv y. \end{aligned} \tag{22}$$

Note that the norms used by these routines are the unweighted norms (2), (3) and (4). If the user wishes to apply weights to the data points, that is to use the norms (5), (6) or (7), the equivalences (22) should be replaced by

$$\begin{aligned} A &\equiv D\Phi, \\ x &\equiv c, \\ b &\equiv Dy, \end{aligned}$$

where D is a diagonal matrix with w_r as the rth diagonal element. Here w_r, for $r = 1, 2, \ldots, m$, is the weight of the rth data point as defined in Section 2.1.2.

3.5.2 Nonlinear functions

Routines for fitting with a nonlinear function in the l_2 norm are provided in Chapter E04. The the E04 Chapter Introduction should be consulted for the appropriate choice of routine. Again, however, the notation adopted is different from that we have used for data fitting. In the latter, we denote the fitting function by $f(x;c)$, where x is the vector of independent variables and c is the vector of coefficients, whose values are to be determined. The squared l_2 norm, to be minimized with respect to the elements of c, is then

$$\sum_{r=1}^{m} w_r^2 [y_r - f(x_r;c)]^2 \tag{23}$$

where y_r is the rth data value of the dependent variable, x_r is the vector containing the rth values of the independent variables, and w_r is the corresponding weight as defined in Section 2.1.2.

On the other hand, in the nonlinear least-squares routines of Chapter E04, the function to be minimized is denoted by

$$\sum_{i=1}^{m} f_i^2(x), \tag{24}$$

the minimization being carried out with respect to the elements of the vector x. The correspondence between the two notations is given by

$$x \equiv c \text{ and } f_i(x) \equiv w_r[y_r - f(x_r;c)], \quad i = r = 1, 2, \ldots, m.$$

Note especially that the vector x of variables of the nonlinear least-squares routines is the vector c of coefficients of the data-fitting problem, and in particular that, if the selected routine requires derivatives of the $f_i(x)$ to be provided, these are derivatives of $w_r[y_r - f(x_r;c)]$ with respect to the coefficients of the data-fitting problem.

3.6 Constraints

At present, there are only a limited number of routines which fit subject to constraints. E02GBF allows the imposition of linear inequality constraints (the inequality (17) for example) when fitting with the general linear function in the l_1 norm. In addition, Chapter E04 contains a routine, E04UNF, which can be used for fitting with a nonlinear function in the l_2 norm subject to general equality or inequality constraints.

The remaining two constraint routines relate to fitting with polynomials in the l_2 norm. E02AGF deals with polynomial curves and allows precise values of the fitting function and (if required) all its derivatives up to a given order to be prescribed at one or more values of the independent variable. The related surface-fitting E02CAF, designed for data on lines as discussed in Section 3.4.1, has a feature which permits precise values of the function and its derivatives to be imposed all along one or more lines parallel to the x- or y-axes (see the routine document for the relationship between these normalised variables and the user's original variables). In this case, however, the prescribed values cannot be supplied directly to the routine: instead, the user must provide modified data ordinates $F_{r,s}$ and polynomial factors $\gamma_1(x)$ and $\gamma_2(x)$, as defined on page 95 of Hayes [5].

3.7 Evaluation, Differentiation and Integration

Routines are available to evaluate, differentiate and integrate polynomials in Chebyshev-series form and cubic or bicubic splines in B-spline form. These polynomials and splines may have been produced by the various fitting routines or, in the case of polynomials, from prior calls of the differentiation and integration routines themselves.

E02AEF and E02AKF evaluate polynomial curves: the latter has a longer parameter list but does not require the user to normalise the values of the independent variable and can accept coefficients which are not stored in contiguous locations. E02CBF evaluates polynomial surfaces, E02BBF cubic spline curves, and E02DEF and E02DFF bicubic spline surfaces.

Differentiation and integration of polynomial curves are carried out by E02AHF and E02AJF respectively. The results are provided in Chebyshev-series form and so repeated differentiation and integration are catered for. Values of the derivative or integral can then be computed using the appropriate evaluation routine. Polynomial surfaces can be treated by a sequence of calls of one or other of the same two routines, differentiating or integrating the form (12) piece by piece. For example, if, for some given value of j, the coefficients a_{ij}, for $i = 0, 1, \ldots, k$, are supplied to E02AHF, we obtain coefficients $\bar{a}_{ij}$ say, for $i = 0, 1, \ldots, k-1$, which are the coefficients in the derivative with respect to x of the polynomial

$$\sum_{i=0}^{k}{}' a_{ij} T_i(x).$$

If this is repeated for all values of j, we obtain all the coefficients in the derivative of the surface with respect to x, namely

$$\sum_{i=0}^{k-1}{}' \sum_{j=0}^{l}{}' \bar{a}_{ij} T_j(y). \tag{25}$$

The derivative of (12), or of (25), with respect to y can be obtained in a corresponding manner. In the latter case, for example, for each value of i in turn we supply the coefficients $\bar{a}_{i0}, \bar{a}_{i1}, \bar{a}_{i2}, \ldots$, to the routine. Values of the resulting polynomials, such as (25), can subsequently be computed using E02CBF. It is important, however, to note one exception: the process described will not give valid results for differentiating or integrating a surface with respect to y if the normalisation of x was made dependent upon y, an option which is available in the fitting routine E02CAF.

For splines the differentiation and integration routines provided are of a different nature from those for polynomials. E02BCF provides values of a cubic spline curve and its first three derivatives (the rest, of course, are zero) at a given value of x. E02BDF computes the value of the definite integral of a cubic spline over its whole range. Again the routines can be applied to surfaces, this time of the form (14). For example, if, for each value of j in turn, the coefficients c_{ij}, for $i = 1, 2, \ldots, p$, are supplied to E02BCF with $x = x_0$ and on each occasion we select from the output the value of the second derivative, d_j say, and if the whole set of d_j are then supplied to the same routine with $x = y_0$, the output will contain all the values at (x_0, y_0) of

$$\frac{\partial^2 f}{\partial x^2} \text{ and } \frac{\partial^{r+2} f}{\partial x^2 \partial y^r}, \quad r = 1, 2, 3.$$

Equally, if after each of the first p calls of E02BCF we had selected the function value (E02BBF would also provide this) instead of the second derivative and we had supplied these values to E02BDF, the result obtained would have been the value of

$$\int_A^B f(x_0, y)\, dy,$$

where A and B are the end-points of the y interval over which the spline was defined.

3.8 Padé Approximants

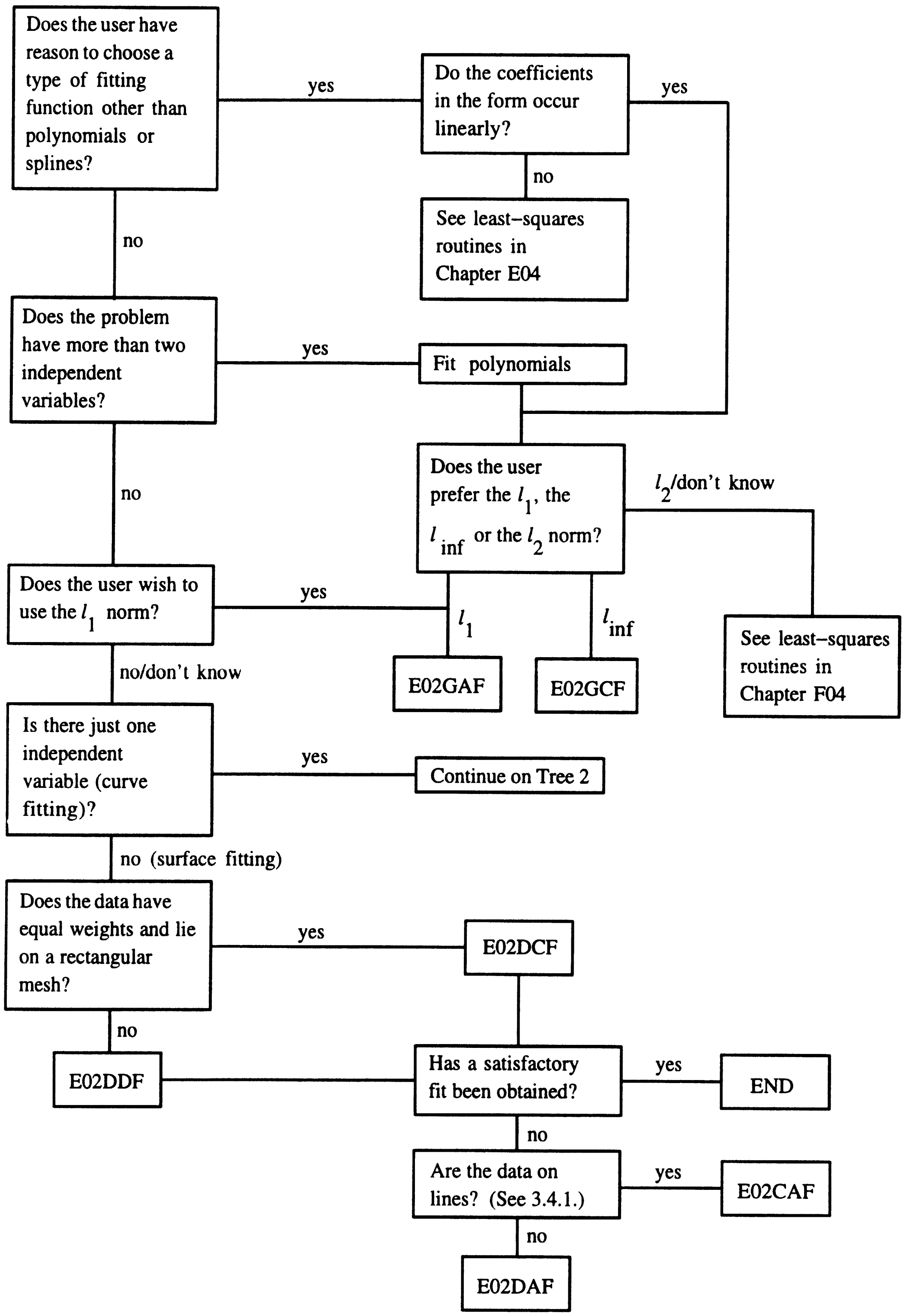
Given two non-negative integers l and m, and the coefficients in the Maclaurin expansion of a function up to degree $l+m$, E02RAF calculates the Padé approximant of degree l in the numerator and degree m in the denominator. For advice on the use of this routine, see the routine document, Sections 3 and 10.

E02RBF is provided to compute values of the Padé approximant, once it has been obtained.

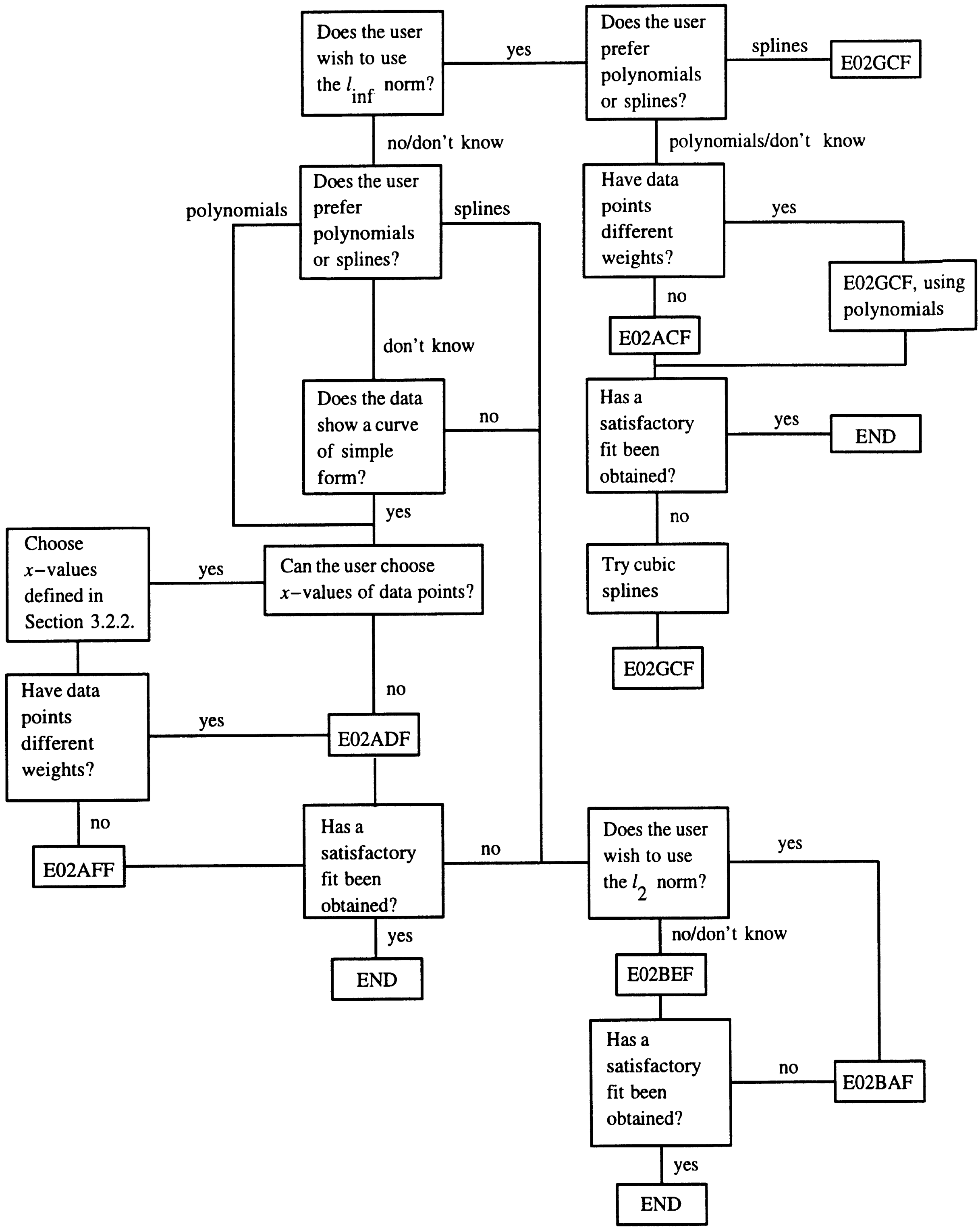
4 Decision Trees

Note. These Decision Trees are concerned with unconstrained fitting: for constrained fitting, consult Section 3.6.

Tree 1



Tree 2


Does the user wish to use the l_{inf} norm?
yes
Does the user prefer polynomials or splines?
splines
E02GCF
polynomials/don't know
no/don't know
Have data points different weights?
yes
E02GCF, using polynomials
no
E02ACF
Does the user prefer polynomials or splines?
polynomials
splines
don't know
Does the data show a curve of simple form?
no
yes
Has a satisfactory fit been obtained?
yes
END
no
Try cubic splines
E02GCF
Choose x–values defined in Section 3.2.2.
yes
Can the user choose x–values of data points?
no
Have data points different weights?
yes
E02ADF
no
E02AFF
Has a satisfactory fit been obtained?
no
yes
END
Does the user wish to use the l_2 norm?
yes
no/don't know
E02BEF
Has a satisfactory fit been obtained?
no
E02BAF
yes
END

5 Index

6 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Those routines indicated by a dagger are still present at Mark 18, but will be omitted at a future date. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

E02DBF

7 References

[1] Cox M G and Hayes J G (1973) Curve fitting: A guide and suite of algorithms for the non-specialist user *NPL Report NAC 26* National Physical Laboratory

[2] Hayes J G (1974) Numerical methods for curve and surface fitting *Bull. Inst. Math. Appl.* **10** 144–152

[3] Hayes J G (ed.) (1970) Curve fitting by polynomials in one variable *Numerical Approximation to Functions and Data* Athlone Press, London

[4] Hayes J G and Halliday J (1974) The least-squares fitting of cubic spline surfaces to general data sets *J. Inst. Math. Appl.* **14** 89–103

[5] Hayes J G (ed.) (1970) Fitting data in more than one variable *Numerical Approximation to Functions and Data* Athlone Press, London

[6] Baker G A (1975) *Essentials of Padé Approximants* Academic Press, New York

---

Chapter E04 – Minimizing or Maximizing a Function

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
E04ABF	6	Minimum, function of one variable using function values only
E04BBF	6	Minimum, function of one variable, using first derivative
E04CCF	1	Unconstrained minimum, simplex algorithm, function of several variables using function values only (comprehensive)
E04DGF	12	Unconstrained minimum, preconditioned conjugate gradient algorithm, function of several variables using first derivatives (comprehensive)
E04DJF	12	Read optional parameter values for E04DGF from external file
E04DKF	12	Supply optional parameter values to E04DGF
E04FCF	7	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using function values only (comprehensive)
E04FYF	18	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using function values only (easy-to-use)
E04GBF	7	Unconstrained minimum of a sum of squares, combined Gauss–Newton and quasi-Newton algorithm using first derivatives (comprehensive)
E04GDF	7	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using first derivatives (comprehensive)
E04GYF	18	Unconstrained minimum of a sum of squares, combined Gauss–Newton and quasi-Newton algorithm, using first derivatives (easy-to-use)
E04GZF	18	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm using first derivatives (easy-to-use)
E04HCF	6	Check user's routine for calculating first derivatives of function
E04HDF	6	Check user's routine for calculating second derivatives of function
E04HEF	7	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm, using second derivatives (comprehensive)
E04HYF	18	Unconstrained minimum of a sum of squares, combined Gauss–Newton and modified Newton algorithm, using second derivatives (easy-to-use)
E04JYF	18	Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using function values only (easy-to-use)
E04KDF	6	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (comprehensive)
E04KYF	18	Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using first derivatives (easy-to-use)
E04KZF	18	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (easy-to-use)
E04LBF	6	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first and second derivatives (comprehensive)
E04LYF	18	Minimum, function of several variables, modified Newton algorithm, simple bounds, using first and second derivatives (easy-to-use)
E04MFF	16	LP problem (dense)
E04MGF	16	Read optional parameter values for E04MFF from external file
E04MHF	16	Supply optional parameter values to E04MFF
E04MZF	18	Converts MPSX data file defining LP or QP problem to format required by E04NKF
E04NCF	12	Convex QP problem or linearly-constrained linear least-squares problem (dense)
E04NDF	12	Read optional parameter values for E04NCF from external file
E04NEF	12	Supply optional parameter values to E04NCF
E04NFF	16	QP problem (dense)

E04NGF	16	Read optional parameter values for E04NFF from external file
E04NHF	16	Supply optional parameter values to E04NFF
E04NKF	18	LP or QP problem (sparse)
E04NLF	18	Read optional parameter values for E04NKF from external file
E04NMF	18	Supply optional parameter values to E04NKF
E04UCF	12	Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (forward communication, comprehensive)
E04UDF	12	Read optional parameter values for E04UCF or E04UFF from external file
E04UEF	12	Supply optional parameter values to E04UCF or E04UFF
E04UFF	18	Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (reverse communication, comprehensive)
E04UGF	19	NLP problem (sparse)
E04UHF	19	Read optional parameter values for E04UGF from external file
E04UJF	19	Supply optional parameter values to E04UGF
E04UNF	17	Minimum of a sum of squares, nonlinear constraints, sequential QP method, using function values and optionally first derivatives (comprehensive)
E04UQF	14	Read optional parameter values for E04UNF from external file
E04URF	14	Supply optional parameter values to E04UNF
E04XAF	12	Estimate (using numerical differentiation) gradient and/or Hessian of a function
E04YAF	7	Check user's routine for calculating Jacobian of first derivatives
E04YBF	7	Check user's routine for calculating Hessian of a sum of squares
E04YCF	11	Covariance matrix for nonlinear least-squares problem (unconstrained)
E04ZCF	11	Check user's routines for calculating first derivatives of function and constraints

Chapter E04

Minimizing or Maximizing a Function

Contents

1 Scope of the Chapter

An optimization problem involves minimizing a function (called the **objective function**) of several variables, possibly subject to restrictions on the values of the variables defined by a set of **constraint functions**. The routines in the Library are concerned with function **minimization** only, since the problem of maximizing a given objective function $F(x)$ is equivalent to minimizing $-F(x)$.

This introduction is only a brief guide to the subject of optimization designed for the casual user. Anyone with a difficult or protracted problem to solve will find it beneficial to consult a more detailed text, such as Gill *et al.* [5] or Fletcher [3].

Users who are unfamiliar with the mathematics of the subject may find some sections difficult at first reading; if so, they should **concentrate** on Sections 2.1, 2.2, 2.5, 2.6 and 3.

2 Background to the Problems

2.1 Types of Optimization Problems

The solution of optimization problems by a single, all-purpose, method is cumbersome and inefficient. Optimization problems are therefore classified into particular categories, where each category is defined by the properties of the objective and constraint functions, as illustrated by some examples below.

Properties of Objective Function	**Properties of Constraints**
Nonlinear	Nonlinear
Sums of squares of nonlinear functions	Sparse linear
Quadratic	Linear
Sums of squares of linear functions	Bounds
Linear	None

For instance, a specific problem category involves the minimization of a nonlinear objective function subject to bounds on the variables. In the following sections we define the particular categories of problems that can be solved by routines contained in this chapter. Not every category is given special treatment in the current version of the Library; however, the long-term objective is to provide a comprehensive set of routines to solve problems in all such categories.

2.1.1 Unconstrained minimization

In unconstrained minimization problems there are no constraints on the variables. The problem can be stated mathematically as follows:

$$\underset{x}{\text{minimize}}\, F(x)$$

where $x \in R^n$, that is, $x = (x_1, x_2, \ldots, x_n)^T$.

2.1.2 Nonlinear least-squares problems

Special consideration is given to the problem for which the function to be minimized can be expressed as a sum of squared functions. The least-squares problem can be stated mathematically as follows:

$$\underset{x}{\text{minimize}} \left\{ f^T f = \sum_{i=1}^{m} f_i^2(x) \right\}, x \in R^n$$

where the ith element of the m-vector f is the function $f_i(x)$.

2.1.3 Minimization subject to bounds on the variables

These problems differ from the unconstrained problem in that at least one of the variables is subject to a simple bound (or restriction) on its value, e.g., $x_5 \leq 10$, but no constraints of a more general form are present.

The problem can be stated mathematically as follows:

$$\underset{x}{\text{minimize}}\, F(x), \qquad x \in R^n$$

subject to $l_i \leq x_i \leq u_i$, $i = 1, 2, \ldots, n$.

This format assumes that upper and lower bounds exist on all the variables. By conceptually allowing $u_i = +\infty$ and $l_i = -\infty$ all the variables need not be restricted.

2.1.4 Minimization subject to linear constraints

A general linear constraint is defined as a constraint function that is linear in more than one of the variables, e.g. $3x_1 + 2x_2 \geq 4$. The various types of linear constraint are reflected in the following mathematical statement of the problem:

$$\underset{x}{\text{minimize}}\, F(x), \quad x \in R^n$$

subject to the

equality constraints:	$a_i^T x = b_i$	$i = 1, 2, \ldots, m_1;$
inequality constraints:	$a_i^T x \geq b_i$	$i = m_1 + 1, m_1 + 2, \ldots, m_2;$
	$a_i^T x \leq b_i$	$i = m_2 + 1, m_2 + 2, \ldots, m_3;$
range constraints:	$s_j \leq a_i^T x \leq t_j$	$i = m_3 + 1, m_3 + 2, \ldots, m_4;$
		$j = 1, 2, \ldots, m_4 - m_3;$
bounds constraints:	$l_i \leq x_i \leq u_i$	$i = 1, 2, \ldots, n$

where each a_i is a vector of length n; b_i, s_j and t_j are constant scalars; and any of the categories may be empty.

Although the bounds on x_i could be included in the definition of general linear constraints, we prefer to distinguish between them for reasons of computational efficiency.

If $F(x)$ is a linear function, the linearly-constrained problem is termed a **linear programming** problem (LP problem); if $F(x)$ is a quadratic function, the problem is termed a **quadratic programming** problem (QP problem). For further discussion of LP and QP problems, including the dual formulation of such problems, see Dantzig [2].

2.1.5 Minimization subject to nonlinear constraints

A problem is included in this category if at least one constraint function is nonlinear, e.g. $x_1^2+x_3+x_4-2 \geq 0$. The mathematical statement of the problem is identical to that for the linearly-constrained case, except for the addition of the following constraints:

equality constraints:	$c_i(x) = 0$	$i = 1, 2, \ldots, m_5;$
inequality constraints:	$c_i(x) \geq 0$	$i = m_5 + 1, m_5 + 2, \ldots, m_6;$
range constraints:	$v_j \leq c_i(x) \leq w_j$	$i = m_6 + 1, m_6 + 2, \ldots, m_7,$
		$j = 1, 2, \ldots, m_7 - m_6$

where each c_i is a nonlinear function; v_j and w_j are constant scalars; and any category may be empty. Note that we do not include a separate category for constraints of the form $c_i(x) \leq 0$, since this is equivalent to $-c_i(x) \geq 0$.

Although the general linear constraints could be included in the definition of nonlinear constraints, again we prefer to distinguish between them for reasons of computational efficiency.

If $F(x)$ is a nonlinear function, the nonlinearly-constrained problem is termed a **nonlinear programming** problem (NLP problem). For further discussion of NLP problems, see Gill *et al.* [5] or Fletcher [3].

2.1.6 Minimization subject to bounds on the objective function

In all of the above problem categories it is assumed that

$$a \leq F(x) \leq b$$

where $a = -\infty$ and $b = +\infty$. Problems in which a and/or b are finite can be solved by adding an extra constraint of the appropriate type (i.e., linear or nonlinear) depending on the form of $F(x)$. Further advice is given in Section 3.4.

2.2 Geometric Representation and Terminology

To illustrate the nature of optimization problems it is useful to consider the following example in two dimensions:

$$F(x) = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1).$$

(This function is used as the example function in the documentation for the unconstrained routines.)

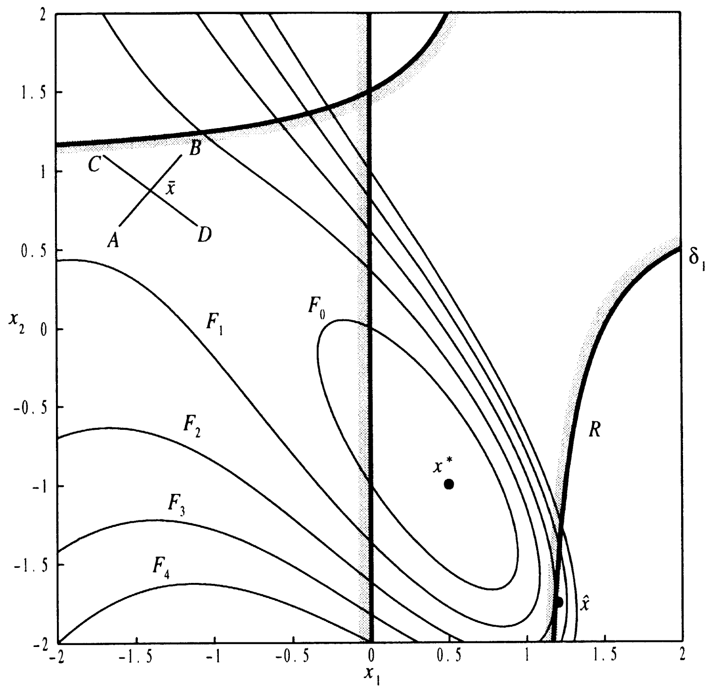


Figure 1

Figure 1 is a contour diagram of $F(x)$. The contours labelled $F_0, F_1, \ldots, F_4$ are isovalue contours, or lines along which the function $F(x)$ takes specific constant values. The point $x^* = (\frac{1}{2}, -1)^T$ is a **local unconstrained minimum**, that is, the value of $F(x^*)$ $(= 0)$ is less than at all the neighbouring points. A function may have several such minima. The lowest of the local minima is termed a **global minimum**. In the problem illustrated in Figure 1, x^* is the only local minimum. The point $\bar{x}$ is said to be a **saddle point** because it is a minimum along the line AB, but a maximum along CD.

If we add the constraint $x_1 \geq 0$ (a simple bound) to the problem of minimizing $F(x)$, the solution remains unaltered. In Figure 1 this constraint is represented by the straight line passing through $x_1 = 0$, and the shading on the line indicates the unacceptable region (i.e., $x_1 < 0$). The region in R^n satisfying the constraints of an optimization problem is termed the **feasible region**. A point satisfying the constraints is defined as a **feasible point**.

If we add the nonlinear constraint $c_1(x) : x_1 + x_2 - x_1x_2 - \frac{3}{2} \geq 0$, represented by the curved shaded line in Figure 1, then x^* is not a feasible point because $c_1(x^*) < 0$. The solution of the new constrained problem is $\hat{x} \simeq (1.1825, -1.7397)^T$, the feasible point with the smallest function value (where $F(\hat{x}) \simeq 3.0607$).

2.2.1 Gradient vector

The vector of first partial derivatives of $F(x)$ is called the **gradient vector**, and is denoted by $g(x)$, i.e.,

$$g(x) = \left[\frac{\partial F(x)}{\partial x_1}, \frac{\partial F(x)}{\partial x_2}, \ldots, \frac{\partial F(x)}{\partial x_n}\right]^T.$$

For the function illustrated in Figure 1,

$$g(x) = \begin{bmatrix} F(x) + e^{x_1}(8x_1 + 4x_2) \\ e^{x_1}(4x_2 + 4x_1 + 2) \end{bmatrix}.$$

The gradient vector is of importance in optimization because it must be zero at an unconstrained minimum of any function with continuous first derivatives.

2.2.2 Hessian matrix

The matrix of second partial derivatives of a function is termed its **Hessian matrix**. The Hessian matrix of $F(x)$ is denoted by $G(x)$, and its (i,j)th element is given by $\partial^2 F(x)/\partial x_i \partial x_j$. If $F(x)$ has continuous second derivatives, then $G(x)$ must be positive semi-definite at any unconstrained minimum of F.

2.2.3 Jacobian matrix; matrix of constraint normals

In nonlinear least-squares problems, the matrix of first partial derivatives of the vector-valued function $f(x)$ is termed the **Jacobian matrix** of $f(x)$ and its (i,j)th component is $\partial f_i / \partial x_j$.

The vector of first partial derivatives of the constraint $c_i(x)$ is denoted by

$$a_i(x) = \left[\frac{\partial c_i(x)}{\partial x_1}, \frac{\partial c_i(x)}{\partial x_2}, \ldots, \frac{\partial c_i(x)}{\partial x_n}\right]^T .$$

The matrix whose columns are the vectors $\{a_i\}$ is termed the **matrix of constraint normals**. At a point $\hat{x}$, the vector $a_i(\hat{x})$ is orthogonal (normal) to the isovalue contour of $c_i(x)$ passing through $\hat{x}$; this relationship is illustrated for a two-dimensional function in Figure 2.

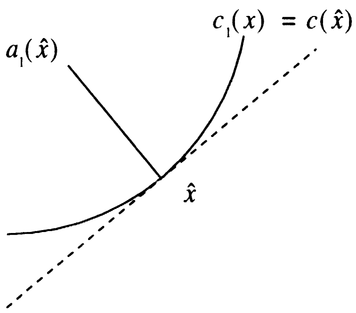


Figure 2

Note that if $c_i(x)$ is a linear constraint involving $a_i^T x$, then its vector of first partial derivatives is simply the vector a_i.

2.3 Sufficient Conditions for a Solution

All nonlinear functions will be assumed to have continuous second derivatives in the neighbourhood of the solution.

2.3.1 Unconstrained minimization

The following conditions are sufficient for the point x^* to be an unconstrained local minimum of $F(x)$:

(i) $\|g(x^*)\| = 0$; and
(ii) $G(x^*)$ is positive-definite,

where $\|g\|$ denotes the Euclidean length of g.

2.3.2 Minimization subject to bounds on the variables

At the solution of a bounds-constrained problem, variables which are not on their bounds are termed **free variables**. If it is known in advance which variables are on their bounds at the solution, the problem can be solved as an unconstrained problem in just the free variables; thus, the sufficient conditions for a solution are similar to those for the unconstrained case, applied only to the free variables.

Sufficient conditions for a feasible point x^* to be the solution of a bounds-constrained problem are as follows:

(i) $\|\bar{g}(x^*)\| = 0$; and
(ii) $\bar{G}(x^*)$ is positive-definite; and
(iii) $g_j(x^*) < 0, x_j = u_j$; $g_j(x^*) > 0, x_j = l_j$,

where $\bar{g}(x)$ is the gradient of $F(x)$ with respect to the free variables, and $\bar{G}(x)$ is the Hessian matrix of $F(x)$ with respect to the free variables. The extra condition (iii) ensures that $F(x)$ cannot be reduced by moving off one or more of the bounds.

2.3.3 Linearly-constrained minimization

For the sake of simplicity, the following description does not include a specific treatment of bounds or range constraints, since the results for general linear inequality constraints can be applied directly to these cases.

At a solution x^*, of a linearly-constrained problem, the constraints which hold as equalities are called the **active** or **binding** constraints. Assume that there are t active constraints at the solution x^*, and let $\hat{A}$ denote the matrix whose columns are the columns of A corresponding to the active constraints, with $\hat{b}$ the vector similarly obtained from b; then

$$\hat{A}^T x^* = \hat{b}.$$

The matrix Z is defined as an $n \times (n-t)$ matrix satisfying:

$$\hat{A}^T Z = 0;$$
$$Z^T Z = I.$$

The columns of Z form an orthogonal basis for the set of vectors orthogonal to the columns of $\hat{A}$.

Define

$g_Z(x) = Z^T g(x)$, the **projected gradient vector** of $F(x)$;
$G_Z(x) = Z^T G(x) Z$, the **projected Hessian matrix** of $F(x)$.

At the solution of a linearly-constrained problem, the projected gradient vector must be zero, which implies that the gradient vector $g(x^*)$ can be written as a linear combination of the columns of $\hat{A}$, i.e., $g(x^*) = \sum_{i=1}^{t} \lambda_i^* \hat{a}_i = \hat{A}\lambda^*$. The scalar λ_i^* is defined as the **Lagrange-multiplier** corresponding to the ith active constraint. A simple interpretation of the ith Lagrange-multiplier is that it gives the gradient of $F(x)$ along the ith active constraint normal; a convenient definition of the Lagrange-multiplier vector (although not a recommended method for computation) is:

$$\lambda^* = (\hat{A}^T \hat{A})^{-1} \hat{A}^T g(x^*).$$

Sufficient conditions for x^* to be the solution of a linearly-constrained problem are:

(i) x^* is feasible, and $\hat{A}^T x^* = \hat{b}$; and
(ii) $\|g_Z(x^*)\| = 0$, or equivalently, $g(x^*) = \hat{A}\lambda^*$; and
(iii) $G_Z(x^*)$ is positive-definite; and
(iv) $\lambda_i^* > 0$ if λ_i^* corresponds to a constraint $\hat{a}_i^T x^* \geq \hat{b}_i$;

$\lambda_i^* < 0$ if λ_i^* corresponds to a constraint $\hat{a}_i^T x^* \leq \hat{b}_i$.

The sign of λ_i^* is immaterial for equality constraints, which by definition are always active.

2.3.4 Nonlinearly-constrained minimization

For nonlinearly-constrained problems, much of the terminology is defined exactly as in the linearly-constrained case. The set of active constraints at x again means the set of constraints that hold as equalities at x, with corresponding definitions of $\hat{c}$ and $\hat{A}$: the vector $\hat{c}(x)$ contains the active constraint functions, and the columns of $\hat{A}(x)$ are the gradient vectors of the active constraints. As before, Z is defined in terms of $\hat{A}(x)$ as a matrix such that:

$$\hat{A}^T Z = 0;$$
$$Z^T Z = I$$

where the dependence on x has been suppressed for compactness.

The projected gradient vector $g_Z(x)$ is the vector $Z^T g(x)$. At the solution x^* of a nonlinearly-constrained problem, the projected gradient must be zero, which implies the existence of Lagrange-multipliers corresponding to the active constraints, i.e., $g(x^*) = \hat{A}(x^*)\lambda^*$.

The **Lagrangian function** is given by:

$$L(x, \lambda) = F(x) - \lambda^T \hat{c}(x).$$

We define $g_L(x)$ as the gradient of the Lagrangian function; $G_L(x)$ as its Hessian matrix, and $\hat{G}_L(x)$ as its projected Hessian matrix, i.e., $\hat{G}_L = Z^T G_L Z$.

Sufficient conditions for x^* to be the solution of a nonlinearly-constrained problem are:

(i) x^* is feasible, and $\hat{c}(x^*) = 0$; and
(ii) $\|g_Z(x^*)\| = 0$, or, equivalently, $g(x^*) = \hat{A}(x^*)\lambda^*$; and
(iii) $\hat{G}_L(x^*)$ is positive-definite; and
(iv) $\lambda_i^* > 0$ if λ_i^* corresponds to a constraint of the form $\hat{c}_i \geq 0$.

The sign of λ_i^* is immaterial for equality constraints, which by definition are always active.

Note that condition (ii) implies that the projected gradient of the Lagrangian function must also be zero at x^*, since the application of Z^T annihilates the matrix $\hat{A}(x^*)$.

2.4 Background to Optimization Methods

All the algorithms contained in this chapter generate an iterative sequence $\{x^{(k)}\}$ that converges to the solution x^* in the limit, except for some special problem categories (i.e., linear and quadratic programming). To terminate computation of the sequence, a convergence test is performed to determine whether the current estimate of the solution is an adequate approximation. The convergence tests are discussed in Section 2.6.

Most of the methods construct a sequence $\{x^{(k)}\}$ satisfying:

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)},$$

where the vector $p^{(k)}$ is termed the **direction of search**, and $\alpha^{(k)}$ is the **steplength**. The steplength $\alpha^{(k)}$ is chosen so that $F(x^{(k+1)}) < F(x^{(k)})$ and is computed using one of the techniques for one-dimensional optimization referred to in Section 2.4.1.

2.4.1 One-dimensional optimization

The Library contains two special routines for minimizing a function of a single variable. Both routines are based on safeguarded polynomial approximation. One routine requires function evaluations only and fits a quadratic polynomial whilst the other requires function and gradient evaluations and fits a cubic polynomial. See Section 4.1. of Gill *et al.* [5].

2.4.2 Methods for unconstrained optimization

The distinctions among methods arise primarily from the need to use varying levels of information about derivatives of $F(x)$ in defining the search direction. We describe three basic approaches to unconstrained problems, which may be extended to other problem categories. Since a full description of the methods would fill several volumes, the discussion here can do little more than allude to the processes involved, and direct the user to other sources for a full explanation.

(a) **Newton-type** Methods (Modified Newton Methods)

Newton-type methods use the Hessian matrix $G(x^{(k)})$, or a finite-difference approximation to $G(x^{(k)})$, to define the search direction. The routines in the Library either require a subroutine that computes the elements of $G(x^{(k)})$ directly, or they approximate $G(x^{(k)})$ by finite-differences.

Newton-type methods are the most powerful methods available for general problems and will find the minimum of a quadratic function in one iteration. See Sections 4.4. and 4.5.1. of Gill *et al.* [5].

(b) **Quasi-Newton** Methods

Quasi-Newton methods approximate the Hessian $G(x^{(k)})$ by a matrix $B^{(k)}$ which is modified at each iteration to include information obtained about the curvature of F along the current search direction $p^{(k)}$. Although not as robust as Newton-type methods, quasi-Newton methods can be more efficient because $G(x^{(k)})$ is not computed directly, or approximated by finite-differences. Quasi-Newton methods minimize a quadratic function in n iterations. See Section 4.5.2 of Gill *et al.* [5].

(c) **Conjugate-Gradient** Methods

Unlike Newton-type and quasi-Newton methods, conjugate-gradient methods do not require the storage of an n by n matrix and so are ideally suited to solve large problems. Conjugate-gradient type methods are not usually as reliable or efficient as Newton-type, or quasi-Newton methods. See Section 4.8.3 of Gill *et al.* [5].

2.4.3 Methods for nonlinear least-squares problems

These methods are similar to those for unconstrained optimization, but exploit the special structure of the Hessian matrix to give improved computational efficiency.

Since

$$F(x) = \sum_{i=1}^{m} f_i^2(x)$$

the Hessian matrix $G(x)$ is of the form

$$G(x) = 2\left(J(x)^T J(x) + \sum_{i=1}^{m} f_i(x) G_i(x)\right),$$

where $J(x)$ is the Jacobian matrix of $f(x)$, and $G_i(x)$ is the Hessian matrix of $f_i(x)$.

In the neighbourhood of the solution, $\|f(x)\|$ is often small compared to $\|J(x)^T J(x)\|$ (for example, when $f(x)$ represents the goodness of fit of a nonlinear model to observed data). In such cases, $2J(x)^T J(x)$ may be an adequate approximation to $G(x)$, thereby avoiding the need to compute or approximate second derivatives of $\{f_i(x)\}$. See Section 4.7 of Gill *et al.* [5].

2.4.4 Methods for handling constraints

Bounds on the variables are dealt with by fixing some of the variables on their bounds and adjusting the remaining free variables to minimize the function. By examining estimates of the Lagrange-multipliers it is possible to adjust the set of variables fixed on their bounds so that eventually the bounds active at the solution should be correctly identified. This type of method is called an **active set method**. One feature of such methods is that, given an initial feasible point, all approximations $x^{(k)}$ are feasible. This approach can be extended to general linear constraints. At a point, x, the set of constraints which hold as equalities being used to predict, or approximate, the set of active constraints is called the **working set**.

Nonlinear constraints are more difficult to handle. If at all possible, it is usually beneficial to avoid including nonlinear constraints during the formulation of the problem. The methods currently implemented in the Library handle nonlinearly constrained problems by transforming them into a sequence of quadratic programming problems. A feature of such methods is that $x^{(k)}$ is not guaranteed to be feasible except in the limit, and this is certainly true of the routines currently in the Library. See Chapter 6, particularly Sections 6.4 and 6.5, of Gill *et al.* [5].

Anyone interested in a detailed description of methods for optimization should consult the references.

2.5 Scaling

Scaling (in a broadly defined sense) often has a significant influence on the performance of optimization methods. Since convergence tolerances and other criteria are necessarily based on an implicit definition of 'small' and 'large', problems with unusual or unbalanced scaling may cause difficulties for some algorithms. Although there are currently no user-callable scaling routines in the Library, scaling is automatically performed by default in the routines which solve sparse LP, QP or NLP problems. The following sections present some general comments on problem scaling.

2.5.1 Transformation of variables

One method of scaling is to transform the variables from their original representation, which may reflect the physical nature of the problem, to variables that have certain desirable properties in terms of optimization. It is generally helpful for the following conditions to be satisfied:

(i) the variables are all of similar magnitude in the region of interest;

(ii) a fixed change in any of the variables results in similar changes in $F(x)$. Ideally, a unit change in any variable produces a unit change in $F(x)$;

(iii) the variables are transformed so as to avoid cancellation error in the evaluation of $F(x)$.

Normally, users should restrict themselves to linear transformations of variables, although occasionally nonlinear transformations are possible. The most common such transformation (and often the most appropriate) is of the form

$$x_{\mathrm{new}} = Dx_{\mathrm{old}},$$

where D is a diagonal matrix with constant coefficients. Our experience suggests that more use should be made of the transformation

$$x_{\mathrm{new}} = Dx_{\mathrm{old}} + v,$$

where v is a constant vector.

Consider, for example, a problem in which the variable x_3 represents the position of the peak of a Gaussian curve to be fitted to data for which the extreme values are 150 and 170; therefore x_3 is known to lie in the range 150–170. One possible scaling would be to define a new variable $\bar{x}_3$, given by

$$\bar{x}_3 = \frac{x_3}{170}.$$

A better transformation, however, is given by defining $\bar{x}_3$ as

$$\bar{x}_3 = \frac{x_3 - 160}{10}.$$

Frequently, an improvement in the accuracy of evaluation of $F(x)$ can result if the variables are scaled before the routines to evaluate $F(x)$ are coded. For instance, in the above problem just mentioned of Gaussian curve fitting, x_3 may always occur in terms of the form $(x_3 - x_m)$, where x_m is a constant representing the mean peak position.

2.5.2 Scaling the objective function

The objective function has already been mentioned in the discussion of scaling the variables. The solution of a given problem is unaltered if $F(x)$ is multiplied by a positive constant, or if a constant value is added to $F(x)$. It is generally preferable for the objective function to be of the order of unity in the region of interest; thus, if in the original formulation $F(x)$ is always of the order of 10^{+5} (say), then the value of $F(x)$ should be multiplied by 10^{-5} when evaluating the function within an optimization routine. If a constant is added or subtracted in the computation of $F(x)$, usually it should be omitted – i.e., it is better to formulate $F(x)$ as $x_1^2 + x_2^2$ rather than as $x_1^2 + x_2^2 + 1000$ or even $x_1^2 + x_2^2 + 1$. The inclusion of such a constant in the calculation of $F(x)$ can result in a loss of significant figures.

2.5.3 Scaling the constraints

A 'well scaled' set of constraints has two main properties. Firstly, each constraint should be well conditioned with respect to perturbations of the variables. Secondly, the constraints should be balanced with respect to each other, i.e., all the constraints should have 'equal weight' in the solution process.

The solution of a linearly- or nonlinearly-constrained problem is unaltered if the ith constraint is multiplied by a positive weight w_i. At the approximation of the solution determined by a Library routine, any active linear constraints will (in general) be satisfied 'exactly' (i.e., to within the tolerance defined by ***machine precision***) if they have been properly scaled. This is in contrast to any active nonlinear constraints, which will not (in general) be satisfied 'exactly' but will have 'small' values (for example, $\hat{c}_1(x^*) = 10^{-8}$, $\hat{c}_2(x^*) = -10^{-6}$, and so on). In general, this discrepancy will be minimized if the constraints are weighted so that a unit change in x produces a similar change in each constraint.

A second reason for introducing weights is related to the effect of the size of the constraints on the Lagrange-multiplier estimates and, consequently, on the active set strategy. This means that different sets of weights may cause an algorithm to produce different sequences of iterates. Additional discussion is given in Gill *et al.* [5].

2.6 Analysis of Computed Results

2.6.1 Convergence criteria

The convergence criteria inevitably vary from routine to routine, since in some cases more information is available to be checked (for example, is the Hessian matrix positive-definite?), and different checks need to be made for different problem categories (for example, in constrained minimization it is necessary to verify whether a trial solution is feasible). Nonetheless, the underlying principles of the various criteria are the same; in non-mathematical terms, they are:

(i) is the sequence $\{x^{(k)}\}$ converging?
(ii) is the sequence $\{F^{(k)}\}$ converging?
(iii) are the necessary and sufficient conditions for the solution satisfied?

The decision as to whether a sequence is converging is necessarily speculative. The criterion used in the present routines is to assume convergence if the relative change occurring between two successive iterations is less than some prescribed quantity. Criterion (iii) is the most reliable but often the conditions cannot be checked fully because not all the required information may be available.

2.6.2 Checking results

Little *a priori* guidance can be given as to the quality of the solution found by a nonlinear optimization algorithm, since no guarantees can be given that the methods will not fail. Therefore, the user should always check the computed solution even if the routine reports success. Frequently a 'solution' may have been found even when the routine does not report a success. The reason for this apparent contradiction is that the routine needs to assess the accuracy of the solution. This assessment is not an exact process and consequently may be unduly pessimistic. Any 'solution' is in general only an approximation to the exact solution, and it is possible that the accuracy specified by the user is too stringent.

Further confirmation can be sought by trying to check whether or not convergence tests are almost satisfied, or whether or not some of the sufficient conditions are nearly satisfied. When it is thought that a routine has returned a non-zero value of IFAIL only because the requirements for 'success' were too stringent it may be worth restarting with increased convergence tolerances.

For nonlinearly-constrained problems, check whether the solution returned is feasible, or nearly feasible; if not, the solution returned is not an adequate solution.

Confidence in a solution may be increased by re-solving the problem with a different initial approximation to the solution. See Section 8.3 of Gill *et al.* [5] for further information.

2.6.3 Monitoring progress

Many of the routines in the chapter have facilities to allow the user to monitor the progress of the minimization process, and users are encouraged to make use of these facilities. Monitoring information can be a great aid in assessing whether or not a satisfactory solution has been obtained, and in indicating difficulties in the minimization problem or in the ability of the routine to cope with the problem.

The behaviour of the function, the estimated solution and first derivatives can help in deciding whether a solution is acceptable and what to do in the event of a return with a non-zero value of IFAIL.

2.6.4 Confidence intervals for least-squares solutions

When estimates of the parameters in a nonlinear least-squares problem have been found, it may be necessary to estimate the variances of the parameters and the fitted function. These can be calculated from the Hessian of $F(x)$ at the solution.

In many least-squares problems, the Hessian is adequately approximated at the solution by $G = 2J^TJ$ (see Section 2.4.3). The Jacobian, J, or a factorization of J is returned by all the comprehensive least-squares routines and, in addition, a routine is available in the Library to estimate variances of the parameters following the use of most of the nonlinear least-squares routines, in the case that $G = 2J^TJ$ is an adequate approximation.

Let H be the inverse of G, and S be the sum of squares, both calculated at the solution $\bar{x}$; an unbiased estimate of the **variance** of the ith parameter x_i is

$$\operatorname{var}\bar{x}_i = \frac{2S}{m-n}H_{ii}$$

and an unbiased estimate of the covariance of $\bar{x}_i$ and $\bar{x}_j$ is

$$covar(\bar{x}_i,\bar{x}_j) = \frac{2S}{m-n}H_{ij}.$$

If x^* is the true solution, then the $100(1-\beta)\%$ **confidence interval** on $\bar{x}$ is

$$\bar{x}_i - \sqrt{\operatorname{var}\bar{x}_i}.t_{(1-\beta/2,m-n)} < x_i^* < \bar{x}_i + \sqrt{\operatorname{var}\bar{x}_i}.t_{(1-\beta/2,m-n)}, \quad i = 1,2,\ldots,n$$

where $t_{(1-\beta/2,m-n)}$ is the $100(1-\beta)/2$ percentage point of the t–distribution with $m-n$ degrees of freedom.

In the majority of problems, the residuals f_i, for $i = 1,2,\ldots,m$, contain the difference between the values of a model function $\phi(z,x)$ calculated for m different values of the independent variable z, and the corresponding observed values at these points. The minimization process determines the parameters, or constants x, of the fitted function $\phi(z,x)$. For any value, $\bar{z}$, of the independent variable z, an unbiased estimate of the **variance** of ϕ is

$$\operatorname{var}\phi = \frac{2S}{m-n}\sum_{i=1}^{n}\sum_{j=1}^{n}\left[\frac{\partial\phi}{\partial x_i}\right]_{\bar{z}}\left[\frac{\partial\phi}{\partial x_j}\right]_{\bar{z}}H_{ij}.$$

The $100(1-\beta)\%$ **confidence interval** on F at the point $\bar{z}$ is

$$\phi(\bar{z},\bar{x}) - \sqrt{\operatorname{var}\phi}.t_{(\beta/2,m-n)} < \phi(\bar{z},x^*) < \phi(\bar{z},\bar{x}) + \sqrt{\operatorname{var}\phi}.t_{(\beta/2,m-n)}.$$

For further details on the analysis of least-squares solutions see Bard [1] and Wolberg [7].

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 choice of routine depends on several factors: the type of problem (unconstrained, etc.); the level of derivative information available (function values only, etc.); the experience of the user (there are easy-to-use versions of some routines); whether or not storage is a problem; and whether computational time has a high priority. Not all choices are catered for in the current version of the Library.

3.1 Easy-to-use and Comprehensive Routines

Many routines appear in the Library in two forms: a comprehensive form and an easy-to-use form. The objective in the easy-to-use forms is to make the routine simple to use by including in the calling sequence only those parameters absolutely essential to the definition of the problem, as opposed to parameters relevant to the solution method. The comprehensive routines have additional parameters which allow the experienced user to improve their efficiency by 'tuning' the method to a particular problem. For the casual or inexperienced user, this feature is of little value and may in some cases cause a failure because of a poor choice of some parameters.

In the easy-to-use routines, these extra parameters are determined either by fixing them at a known safe and reasonably efficient value, or by an auxiliary routine which generates a 'good' value automatically.

For routines introduced since Mark 12 of the Library a different approach has been adopted towards the choice of easy-to-use and comprehensive routines. The optimization routine has an easy-to-use parameter list, but additional parameters may be changed from their default values by calling an 'option' setting routine prior to the call to the main optimization routine. This approach has the advantages of allowing the options to be given in the form of keywords and requiring only those options that are to be different from their default values to be set.

3.2 Reverse Communication Routines

Most of the routines in this chapter are called just once in order to compute the minimum of a given objective function subject to a set of constraints on the variables. The objective function and nonlinear constraints (if any) are specified by the user and written as subroutines to a very rigid format described in the relevant routine document. Such subroutines usually appear in the argument list of the minimization routine.

For the majority of applications this is the simplest and most convenient usage. Sometimes however this approach can be restrictive:

(i) when the required format of the user's subroutine does not allow useful information to be passed conveniently to and from the user's calling program;

(ii) when the minimization routine is being called from another computer language, such as Visual Basic, which does not fully support procedure arguments in a way that is compatible with the Library.

A way around these problems is to supply **reverse communication** routines. Instead of performing complete optimizations, these routines perform one step in the solution process before returning to the calling program with an appropriate flag (IREVCM) set. The value of IREVCM determines whether the minimization process has finished or whether fresh information is required. In the latter case the user calculates this information (in the form of a vector or as a scalar, as appropriate) and re-enters the reverse communication routine with the information contained in appropriate arguments. Thus the user has the responsibility for providing the iterative loop in the minimization process, but as compensation, has an extremely flexible and basic user-interface to the reverse communication routine.

The only reverse communication routine in this chapter is E04UFF, which solves dense NLP problems and uses exactly the same method as E04UCF.

3.3 Service Routines

One of the most common errors in the use of optimization routines is that user-supplied subroutines do not evaluate the relevant partial derivatives correctly. Because exact gradient information normally enhances efficiency in all areas of optimization, the user should be encouraged to provide analytical derivatives whenever possible. However, mistakes in the computation of derivatives can result in serious and obscure run-time errors. Consequently, **service routines** are provided to perform an elementary check on the user-supplied gradients. These routines are inexpensive to use in terms of the number of calls they require to user-supplied routines.

The appropriate checking routines are as follows:

Minimization routine	**Checking routine(s)**
E04KDF	E04HCF
E04LBF	E04HCF and E04HDF
E04GBF	E04YAF
E04GDF	E04YAF
E04HEF	E04YAF and E04YBF

It should be noted that routines E04UCF, E04UFF, E04UGF and E04UNF each incorporate a check on the gradients being supplied. This involves verifying the gradients at the first point that satisfies the linear constraints and bounds. There is also an option to perform a more reliable (but more expensive) check on the individual gradient elements being supplied. Note that the checks are not infallible.

A second type of service routine computes a set of finite-differences to be used when approximating first derivatives. Such differences are required as input parameters by some routines that use only function evaluations.

E04YCF estimates selected elements of the variance-covariance matrix for the computed regression parameters following the use of a nonlinear least-squares routine.

E04XAF estimates the gradient and Hessian of a function at a point, given a routine to calculate function values only, or estimates the Hessian of a function at a point, given a routine to calculate function and gradient values.

3.4 Function Evaluations at Infeasible Points

All the routines for constrained problems will ensure that any evaluations of the objective function occur at points which **approximately** satisfy any **simple bounds** or **linear constraints**. Satisfaction of such constraints is only approximate because routines which estimate derivatives by finite-differences may require function evaluations at points which just violate such constraints even though the current iteration just satisfies them.

There is no attempt to ensure that the current iteration satisfies any nonlinear constraints. Users who wish to prevent their objective function being evaluated outside some known region (where it may be undefined or not practically computable), may try to confine the iteration within this region by imposing suitable simple bounds or linear constraints (but beware as this may create new local minima where these constraints are active).

Note also that some routines allow the user-supplied routine to return a parameter (IFLAG or MODE) with a negative value to force an immediate clean exit from the minimization routine when the objective function (or nonlinear constraints where appropriate) cannot be evaluated.

3.5 Related Problems

Apart from the standard types of optimization problem, there are other related problems which can be solved by routines in this or other chapters of the Library.

H02BBF solves **dense integer LP** problems, H02CBF solves **dense integer QP** problems, H02CEF solves **sparse integer QP** problems and H03ABF solves a special type of such problem known as a **'transportation' problem**.

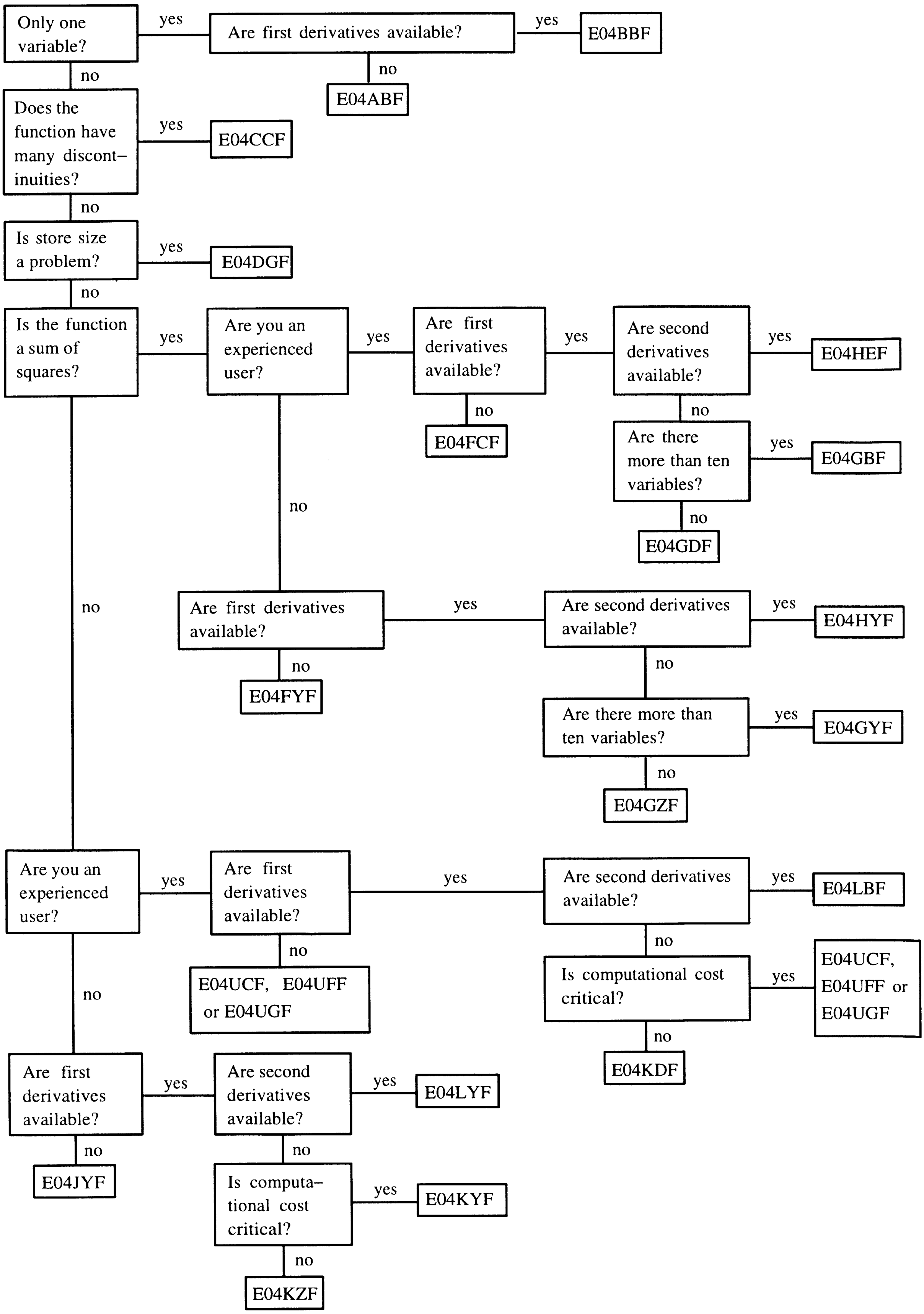
Several routines in Chapter F04 solve **linear least-squares problems**, i.e., minimize $\sum_{i=1}^{m} r_i(x)^2$ where $r_i(x) = b_i - \sum_{j=1}^{n} a_{ij} x_j$.

E02GAF solves an overdetermined system of linear equations in the l_1 norm, i.e., minimizes $\sum_{i=1}^{m} |r_i(x)|$, with r_i as above, and E02GBF solves the same problem subject to linear inequality constraints.

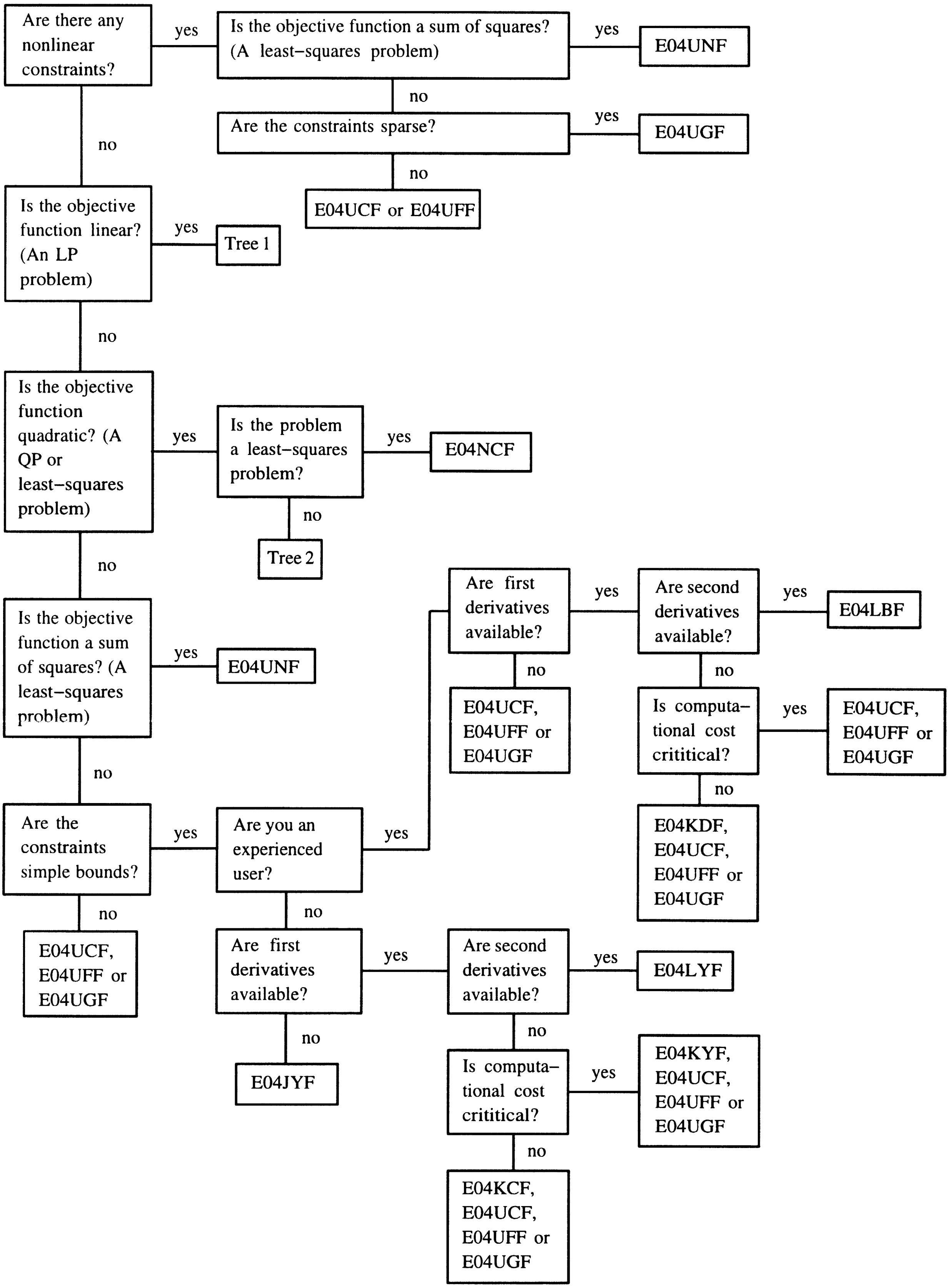
E02GCF solves an overdetermined system of linear equations in the l_∞ norm, i.e., minimizes $\max_i |r_i(x)|$, with r_i as above.

4 Decision Trees

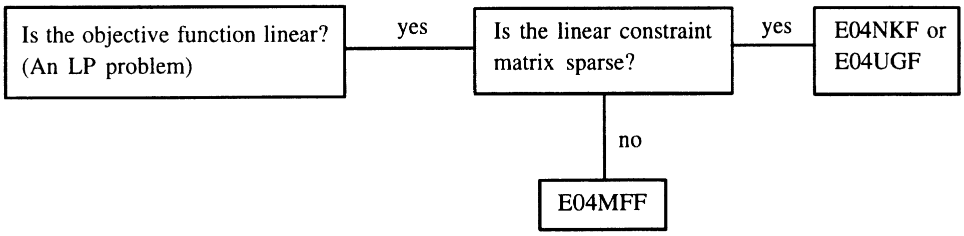
Selection chart for unconstrained problems



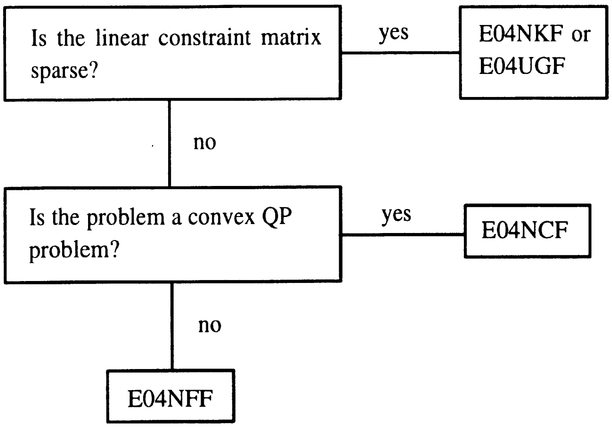
Selection chart for bound-constrained, linearly-constrained and nonlinearly-constrained problems



Tree 1



Tree 2



5 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

E04CGF	E04DBF	E04DEF	E04DFF	E04EBF	E04FDF
E04GCF	E04GEF	E04HFF	E04HBF	E04JAF	E04JBF
E04KAF	E04KBF	E04KCF	E04LAF	E04MBF	E04NAF
E04UAF	E04UPF	E04VCF	E04VDF		

6 References

[1] Bard Y (1974) *Nonlinear Parameter Estimation* Academic Press

[2] Dantzig G B (1963) *Linear Programming and Extensions* Princeton University Press

[3] Fletcher R (1987) *Practical Methods of Optimization* Wiley (2nd Edition)

[4] Gill P E and Murray W (ed.) (1974) *Numerical Methods for Constrained Optimization* Academic Press

[5] Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

[6] Murray W (ed.) (1972) *Numerical Methods for Unconstrained Optimization* Academic Press

[7] Wolberg J R (1967) *Prediction Analysis* Van Nostrand

Chapter F – Linear Algebra

Chapter F

Linear Algebra

Contents

1 Introduction

The F Chapters of the Library are concerned with linear algebra and cover a large area. This general introduction is intended to help users decide which particular F Chapter is relevant to their problem. There are currently nine F Chapters with the following titles:

Chapter F01 – Matrix Operations, including Inversion

Chapter F02 – Eigenvalues and Eigenvectors

Chapter F03 – Determinants

Chapter F04 – Simultaneous Linear Equations

Chapter F05 – Orthogonalisation

Chapter F06 – Linear Algebra Support Routines

Chapter F07 – Linear Equations (LAPACK)

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

Chapter F11 – Sparse Linear Algebra

The principal problem areas addressed by the above Chapters are:

Systems of linear equations

Linear least-squares problems

Eigenvalue and singular value problems

The solution of these problems usually involves several matrix operations, such as a matrix factorization followed by the solution of the factorized form, and the routines for these operations themselves utilize lower level support routines, typically from Chapter F06. Most users will not normally need to be concerned with these support routines.

NAG has been involved in a project, called LAPACK [1], to develop a linear algebra package for modern high-performance computers, and the routines developed within that project are being incorporated into the Library as Chapter F07 and Chapter F08. It should be emphasised that, while the LAPACK project has been concerned with high-performance computers, the routines do not compromise efficiency on conventional machines.

Chapter F11 contains a suite of routines for solving sparse systems of linear equations. Earlier routines for sparse linear algebra are still located in Chapter F01, Chapter F02 and Chapter F04. For an alternative selection of routines for sparse linear algebra, users are referred to the Harwell Sparse Matrix Library (available from NAG).

For background information on numerical algorithms for the solution of linear algebra problems see Golub and Van Loan [5]. For the three main problem areas listed above the user generally has the choice of selecting a single routine to solve the problem, a so-called *Black Box* routine, or selecting more than one routine to solve the problem, such as a factorization routine followed by a solve routine, so-called *General Purpose* routines. The following sections indicate which chapters are relevant to particular problem areas.

2 Linear Equations

The Black Box routines for solving linear equations of the form

$$Ax = b \text{ and } AX = B,$$

where A is an n by n real or complex non-singular matrix, are to be found in Chapter F04. Such equations can also be solved by selecting a General Purpose factorization routine from Chapter F01 or Chapter F03 and combining them with a solve routine in Chapter F04, or by selecting a factorization and a solve routine from Chapter F07. For sparse symmetric problems, routines from Chapter F11 should be used. In addition there are routines to estimate condition numbers in Chapter F04 and Chapter F07, and routines to give error estimates in Chapter F07.

There are routines to cater for a variety of types of matrix, including general, symmetric or Hermitian, symmetric or Hermitian positive definite, banded, skyline and sparse matrices.

In order to select the appropriate routine, users are recommended to consult the F04 Chapter Introduction in the first instance, although the decision trees for the General Purpose routines will usually in fact point to an F07 or F11 routine.

3 Linear Least-squares

The Black Box routines for solving linear least-squares problems of the form

$$\underset{x}{\text{minimize}}\ r^T r, \text{ where } r = b - Ax,$$

where A is an m by n, possibly rank deficient, matrix are to be found in Chapter F04. Such problems can also be solved by selecting a General Purpose factorization routine from Chapter F02 or Chapter F08 and combining them with a solve routine in Chapter F04, which also contains a routine to compute covariance matrices. Linear least-squares problems can also be solved by routines in the statistical Chapter G02.

In order to select the appropriate routine, users are recommended to consult the F04 Chapter Introduction in the first instance, but users with additional statistical requirements may prefer to consult Section 2.2 of the G02 Chapter Introduction.

4 Eigenvalue Problems and Singular Value Problems

The Black Box routines for solving standard matrix eigenvalue problems of the form

$$Ax = \lambda x,$$

where A is an n by n real or complex matrix, and generalized matrix eigenvalue problems of the form

$$Ax = \lambda Bx \text{ and } ABx = \lambda x,$$

where B is also an n by n matrix, are to be found in Chapter F02 and Chapter F08. These eigenvalue problems can also be solved by a combination of General Purpose routines (which are mostly in Chapter F08, but a few in Chapter F02).

There are routines to cater for various types of matrices, including general, symmetric or Hermitian, banded and sparse matrices.

Similarly, the Black Box routines for finding singular values and/or singular vectors of an m by n real or complex matrix A are to be found in Chapter F02, and such problems may also be solved by combining routines from Chapter F08.

In order to select the appropriate routine, users are recommended to consult the F02 Chapter and F08 Chapter Introductions in the first instance.

5 Inversion and Determinants

Routines for matrix inversion are to be found in Chapter F01 and Chapter F07. Users are recommended to consult the F01 Chapter Introduction in the first instance, although the decision tree will often in fact point to an F07 routine. It should be noted that users are strongly encouraged not to use matrix inversion routines for the solution of linear equations, since these can be solved more efficiently and accurately using routines directed specifically at such problems. Indeed many problems, which superficially appear to be matrix inversion, can be posed as the solution of a system of linear equations and this is almost invariably preferable.

Routines to compute determinants of matrices are to be found in Chapter F03. Users are recommended to consult the F03 Chapter Introduction in the first instance.

6 Matrix Operations

Routines for various sorts of matrix operation are to be found in Chapter F01, including matrix transposition, addition and multiplication, and conversion between different matrix representation storage formats. Facilities for matrix manipulation can also be found in Chapter F06 (see next section).

7 Support Routines

Chapter F06 contains a variety of routines to perform elementary algebraic operations involving scalars, vectors and matrices, such as setting up a plane rotation, performing a dot product and computing a matrix norm. Chapter F06 contains routines that meet the specification of the BLAS (Basic Linear Algebra Subprograms) [6], [2], [4] and [3]. The routines in this chapter will not normally be required by the general user, but are intended for use by those who require to build specialist linear algebra modules. These routines, especially the BLAS, are extensively used by other NAG Fortran Library routines.

8 References

[1] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A, Ostrouchov S and Sorensen D (1995) *LAPACK Users' Guide* (2nd Edition) SIAM, Philadelphia

[2] Dodson D S, Grimes R G and Lewis J G (1991) Sparse extensions to the Fortran basic linear algebra subprograms *ACM Trans. Math. Software* **17** 253–263

[3] Dongarra J J, Du Croz J J, Duff I S and Hammarling S (1990) A set of Level 3 basic linear algebra subprograms *ACM Trans. Math. Software* **16** 1–28

[4] Dongarra J J, Du Croz J J, Hammarling S and Hanson R J (1988) An extended set of FORTRAN basic linear algebra subprograms *ACM Trans. Math. Software* **14** 1–32

[5] Golub G H and Van Loan C F (1996) *Matrix Computations* Johns Hopkins University Press (3rd Edition), Baltimore

[6] Lawson C L, Hanson R J, Kincaid D R and Krogh F T (1979) Basic linear algebra subprograms for Fortran usage *ACM Trans. Math. Software* **5** 308–325

Chapter F01 – Matrix Factorizations

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
F01ABF	1	Inverse of real symmetric positive-definite matrix using iterative refinement
F01ADF	2	Inverse of real symmetric positive-definite matrix
F01BLF	5	Pseudo-inverse and rank of real m by n matrix ($m \geq n$)
F01BRF	7	LU factorization of real sparse matrix
F01BSF	7	LU factorization of real sparse matrix with known sparsity pattern
F01BUF	7	$ULDL^TU^T$ factorization of real symmetric positive-definite band matrix
F01BVF	7	Reduction to standard form, generalized real symmetric-definite banded eigenproblem
F01CKF	2	Matrix multiplication
F01CRF	7	Matrix transposition
F01CTF	14	Sum or difference of two real matrices, optional scaling and transposition
F01CWF	14	Sum or difference of two complex matrices, optional scaling and transposition
F01LEF	11	LU factorization of real tridiagonal matrix
F01LHF	13	LU factorization of real almost block diagonal matrix
F01MCF	8	LDL^T factorization of real symmetric positive-definite variable-bandwidth matrix
F01QGF	14	RQ factorization of real m by n upper trapezoidal matrix ($m \leq n$)
F01QJF	14	RQ factorization of real m by n matrix ($m \leq n$)
F01QKF	14	Operations with orthogonal matrices, form rows of Q, after RQ factorization by F01QJF
F01RGF	14	RQ factorization of complex m by n upper trapezoidal matrix ($m \leq n$)
F01RJF	14	RQ factorization of complex m by n matrix ($m \leq n$)
F01RKF	14	Operations with unitary matrices, form rows of Q, after RQ factorization by F01RJF
F01ZAF	14	Convert real matrix between packed triangular and square storage schemes
F01ZBF	14	Convert complex matrix between packed triangular and square storage schemes
F01ZCF	14	Convert real matrix between packed banded and rectangular storage schemes
F01ZDF	14	Convert complex matrix between packed banded and rectangular storage schemes

Chapter F01

Matrix Factorizations

Contents

1 Scope of the Chapter

This chapter provides facilities for three types of problem:

(i) Matrix Inversion
(ii) Matrix Factorizations
(iii) Matrix Arithmetic and Manipulation

These problems are discussed separately in Section 2.1, Section 2.2 and Section 2.3.

2 Background to the Problems

2.1 Matrix Inversion

(i) Non-singular square matrices of order n.

If A, a square matrix of order n, is non-singular (has rank n), then its inverse X exists and satisfies the equations $AX = XA = I$ (the identity or unit matrix).

It is worth noting that if $AX - I = R$, so that R is the 'residual' matrix, then a bound on the relative error is given by $\|R\|$, i.e.,

$$\frac{\|X - A^{-1}\|}{\|A^{-1}\|} \leq \|R\|.$$

(ii) General real rectangular matrices.

A real matrix A has no inverse if it is square (n by n) and singular (has rank $< n$), or if it is of shape (m by n) with $m \neq n$, but there is a **Generalized** or **Pseudo Inverse** Z which satisfies the equations

$$AZA = A, \quad ZAZ = Z, \quad (AZ)^T = AZ, \quad (ZA)^T = ZA$$

(which of course are also satisfied by the inverse X of A if A is square and non-singular).

(a) if $m \geq n$ and rank$(A) = n$ then A can be factorized using a $\boldsymbol{QR}$ **factorization**, given by

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix},$$

where Q is an m by m orthogonal matrix and R is an n by n, non-singular, upper triangular matrix. The pseudo-inverse of A is then given by

$$Z = R^{-1}\tilde{Q}^T,$$

where $\tilde{Q}$ consists of the first n columns of Q.

(b) if $m \leq n$ and rank$(A) = m$ then A can be factorized using an $\boldsymbol{RQ}$ **factorization**, given by

$$A = (R \;\; 0)P^T$$

where P is an n by n orthogonal matrix and R is an m by m, non-singular, upper triangular matrix. The pseudo-inverse of A is then given by

$$Z = \tilde{P}R^{-1},$$

where $\tilde{P}$ consists of the first m columns of P.

(c) if $m \geq n$ and rank$(A) = r \leq n$ then A can be factorized using a QR factorization, with column interchanges, as

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}P^T,$$

where Q is an m by m orthogonal matrix, R is an r by n upper trapezoidal matrix and P is an n by n permutation matrix. The pseudo inverse of A is then given by

$$Z = PR^T(RR^T)^{-1}\tilde{Q}^T,$$

where $\tilde{Q}$ consists of the first r columns of Q.

(d) if $\text{rank}(A) = r \leq k = \min(m,n)$, then A can be factorized as the **singular value decomposition**

$$A = QDP^T,$$

where Q is an m by m orthogonal matrix, P is an n by n orthogonal matrix and D is an m by n diagonal matrix with non-negative diagonal elements. The first k columns of Q and P are the **left-** and **right-hand singular vectors** of A respectively and the k diagonal elements of D are the **singular values** of A. D may be chosen so that

$$d_1 \geq d_2 \geq \ldots \geq d_k \geq 0$$

and in this case if $\text{rank}(A) = r$ then

$$d_1 \geq d_2 \geq \ldots \geq d_r > 0, \quad d_{r+1} = \ldots = d_k = 0.$$

If $\tilde{Q}$ and $\tilde{P}$ consist of the first r columns of Q and P respectively and Σ is an r by r diagonal matrix with diagonal elements $d_1, d_2, \ldots, d_r$ then A is given by

$$A = \tilde{Q}\Sigma\tilde{P}^T$$

and the pseudo inverse of A is given by

$$Z = \tilde{P}\Sigma^{-1}\tilde{Q}^T.$$

Notice that

$$A^TA = P(D^TD)P^T$$

which is the classical eigenvalue (spectral) factorization of A^TA.

(e) if A is complex then the above relationships are still true if we use 'unitary' in place of 'orthogonal' and conjugate transpose in place of transpose. For example, the singular value decomposition of A is

$$A = QDP^H,$$

where Q and P are unitary, P^H the conjugate transpose of P and D is as in (d) above.

2.2 Matrix Factorizations

The routines in this section perform matrix factorizations which are required for the solution of systems of linear equations with various special structures. A few routines which perform associated computations are also included.

Other routines for matrix factorizations are to be found in Chapter F03, Chapter F07, Chapter F08 and Chapter F11.

This section also contains a few routines associated with eigenvalue problems (see Chapter F02). (Historical note: this section used to contain many more such routines, but they have now been superseded by routines in Chapter F08.)

2.3 Matrix Arithmetic and Manipulation

The intention of routines in this section (sub-chapters F01C and F01Z) is to cater for some of the commonly occurring operations in matrix manipulation, e.g. transposing a matrix or adding part of one matrix to another, and for conversion between different storage formats, e.g. conversion between rectangular band matrix storage and packed band matrix storage. For vector or matrix-vector or matrix-matrix operations refer to Chapter F06.

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 Matrix Inversion

Note. Before using any routine for matrix inversion, consider carefully whether it is really needed.

Although the solution of a set of linear equations $Ax = b$ can be written as $x = A^{-1}b$, the solution should **never** be computed by first inverting A and then computing $A^{-1}b$; the routines in Chapter F04 or Chapter F07 should **always** be used to solve such sets of equations directly; they are faster in execution, and numerically more stable and accurate. Similar remarks apply to the solution of least-squares problems which again should be solved by using the routines in Chapter F04 rather than by computing a pseudo inverse.

(a) Non-singular square matrices of order n

This chapter describes techniques for inverting a general real matrix A and matrices which are positive-definite (have all eigenvalues positive) and are either real and symmetric or complex and Hermitian. It is wasteful and uneconomical not to use the appropriate routine when a matrix is known to have one of these special forms. A general routine must be used when the matrix is not known to be positive-definite. In all routines the inverse is computed by solving the linear equations $Ax_i = e_i$, for $i = 1, 2, \ldots, n$, where e_i is the ith column of the identity matrix.

Routines are given for calculating the approximate inverse, that is solving the linear equations just once, and also for obtaining the accurate inverse by successive iterative corrections of this first approximation. The latter, of course, are more costly in terms of time and storage, since each correction involves the solution of n sets of linear equations and since the original A and its LU decomposition must be stored together with the first and successively corrected approximations to the inverse. In practice the storage requirements for the 'corrected' inverse routines are about double those of the 'approximate' inverse routines, though the extra computer time is not prohibitive since the same matrix and the same LU decomposition is used in every linear equation solution.

Despite the extra work of the 'corrected' inverse routines they are superior to the 'approximate' inverse routines. A correction provides a means of estimating the number of accurate figures in the inverse or the number of 'meaningful' figures relating to the degree of uncertainty in the coefficients of the matrix.

The residual matrix $R = AX - I$, where X is a computed inverse of A, conveys useful information. Firstly $\|R\|$ is a bound on the relative error in X and secondly $\|R\| < \frac{1}{2}$ guarantees the convergence of the iterative process in the 'corrected' inverse routines.

The decision trees for inversion show which routines in chapters F04 and F07 should be used for the inversion of other special types of matrices not treated in the chapter.

(b) General real rectangular matrices

For real matrices F08AEF and F01QJF return QR and RQ factorizations of A respectively and F08BEF returns the QR factorization with column interchanges. The corresponding complex routines are F08ASF, F01RJF and F08BSF respectively. Routines are also provided to form the orthogonal matrices and transform by the orthogonal matrices following the use of the above routines. F01QGF and F01RGF form the RQ factorization of an upper trapezoidal matrix for the real and complex cases respectively.

F01BLF uses the QR factorization as described in Section 2.1(ii)(a) and is the only routine that explicitly returns a pseudo inverse. If $m \geq n$, then the routine will calculate the pseudo inverse Z of the matrix A. If $m < n$, then the n by m matrix A^T should be used. The routine will calculate the pseudo inverse Z of A^T and the required pseudo inverse will be Z^T. The routine also attempts to calculate the rank, r, of the matrix given a tolerance to decide when elements can be regarded as zero. However, should this routine fail due to an incorrect determination of the rank, the singular value decomposition method (described below) should be used.

F02WEF and F02XEF compute the singular value decomposition as described in Section 2 for real and complex matrices respectively. If A has rank $r \leq k = \min(m, n)$ then the $k - r$ smallest singular values will be negligible and the pseudo inverse of A can be obtained as $Z = P\Sigma^{-1}Q^T$ as described in Section 2. If the rank of A is not known in advance it can be estimated from the singular values (see Section 2.2 of the F04 Chapter Introduction). In the real case with $m \geq n$, F02WDF provides details of the QR factorization or the singular value decomposition depending on whether or not A is of full rank and for some problems provides an attractive alternative to F02WEF.

3.2 Matrix Factorizations

Each of these routines serves a special purpose required for the solution of sets of simultaneous linear equations or the eigenvalue problem. For further details users should consult sections on 'Recommendations on Choice and Use of Routines' and 'Decision Trees' of the F02 Chapter Introduction and the F04 Chapter Introduction, and individual routine documents.

F01BRF and F01BSF are provided for factorizing general real sparse matrices. For factorizing real symmetric positive-definite sparse matrices, see F11JAF. These routines should be used only when A is **not** banded and when the total number of non-zero elements is less than 10% of the total number of elements. In all other cases either the band routines or the general routines should be used.

3.3 Matrix Arithmetic and Manipulation

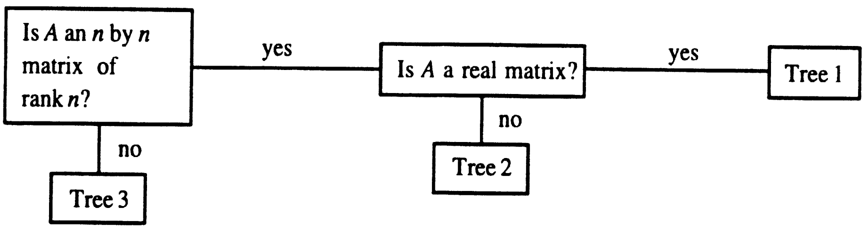
The routines in the F01C section are designed for the general handling of m by n matrices. Emphasis has been placed on flexibility in the parameter specifications and on avoiding, where possible, the use of internally declared arrays. They are therefore suited for use with large matrices of variable row and column dimensions. Routines are included for the addition and subtraction of sub-matrices of larger matrices, as well as the standard manipulations of full matrices. Those routines involving matrix multiplication may use additional-precision arithmetic for the accumulation of inner products. See also Chapter F06.

The routines in the F01Z section are designed to allow conversion between square storage and the packed storage schemes required by some of the routines in Chapter F02, Chapter F04, Chapter F06, Chapter F07 and Chapter F08.

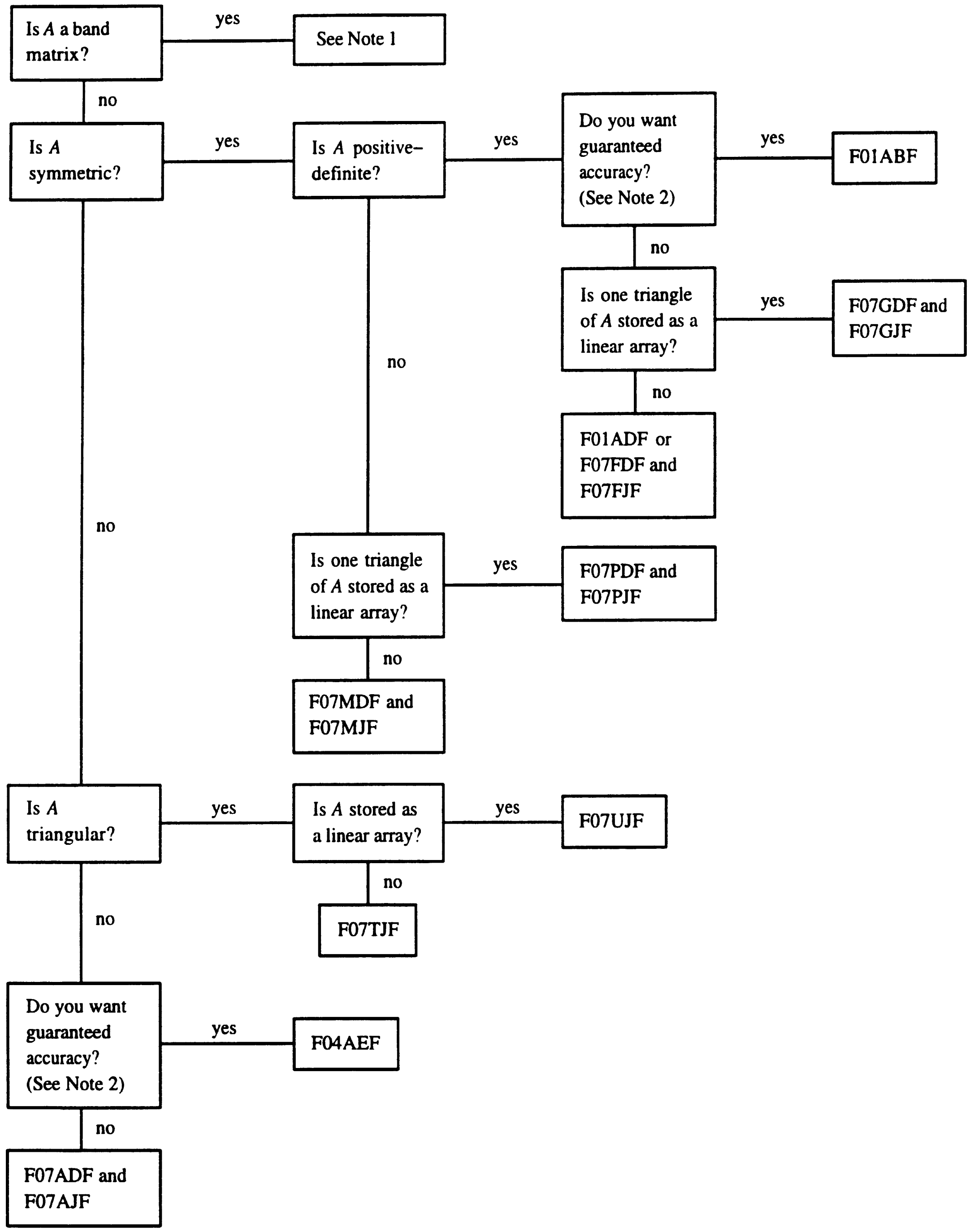
4 Decision Trees

The decision trees show the routines in this chapter and in chapter F04 that should be used for inverting matrices of various types. Routines marked with an asterisk (*) only perform part of the computation – see Section 3.1 for further advice.

(i) **Matrix Inversion:**



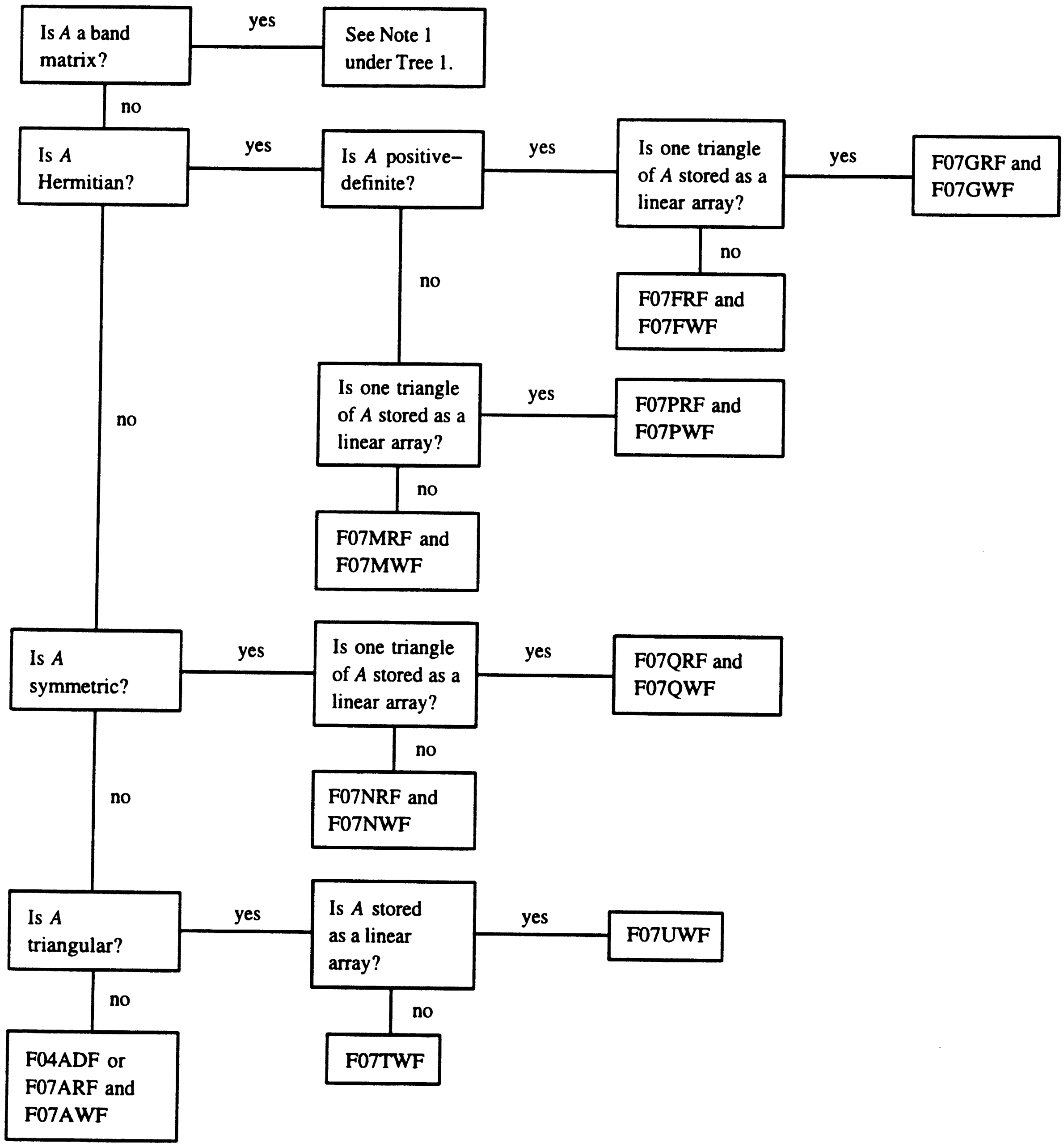
Tree 1: Inverse of a real n by n matrix of full rank



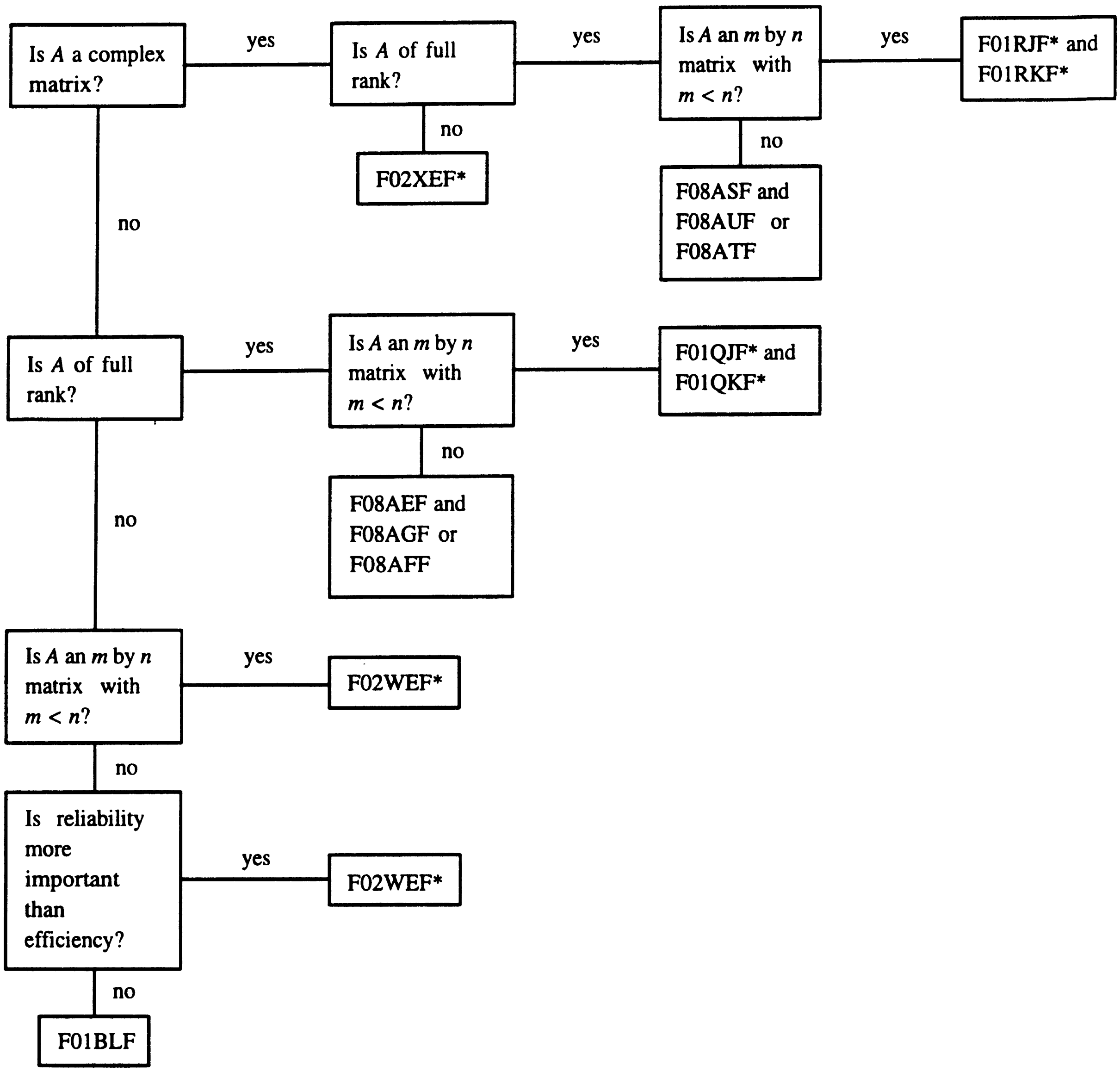
Note 1: the inverse of a band matrix A does not in general have the same shape as A, and no routines are provided specifically for finding such an inverse. The matrix must either be treated as a full matrix, or the equations $AX = B$ must be solved, where B has been initialised to the identity matrix I. In the latter case, see the decision trees in the F04 Chapter Introduction.

Note2: by 'guaranteed accuracy' we mean that the accuracy of the inverse is improved by use of the iterative refinement technique using additional precision.

Tree 2: Inverse of a complex n by n matrix of full rank



Tree 3: Pseudo-inverses



(ii) **Matrix Factorizations:** See the Decision Trees in Section 4 of the F02 Chapter Introduction and Section 4 of the F04 Chapter Introduction.

(iii) **Matrix Arithmetic and Manipulation:** Not appropriate.

5 Index

(i) **Inversion**

Real m by n Matrix, Pseudo Inverse,	F01BLF
Real Symmetric Positive-definite Matrix, Accurate Inverse,	F01ABF
Real Symmetric Positive-definite Matrix, Approximate Inverse,	F01ADF

(ii) **Matrix Transformations**

Complex m by n ($m \leq n$) Matrix, RQ Factorization,	F01RJF
Complex Matrix, Form Unitary Matrix,	F01RKF
Complex Upper Trapezoidal Matrix, RQ Factorization,	F01RGF
Eigenproblem $Ax = \lambda Bx, A, B$ Banded, Reduction to Standard Symmetric Problem,	F01BVF
Tridiagonal matrix, LU Factorization,	F01LEF
Real Almost Block-diagonal Matrix, LU Factorization,	F01LHF
Real Band Symmetric Positive-definite Matrix, $ULDL^TU^T$ Factorization,	F01BUF

Real Band Symmetric Positive-definite Matrix, Variable Bandwidth, Cholesky Factorization,	F01MCF
Real m by n ($m \le n$) Matrix, RQ Factorization,	F01QJF
Real Matrix, Form Orthogonal Matrix,	F01QKF
Real Upper Trapezoidal Matrix, RQ Factorization,	F01QGF
Real Sparse Matrix, Factorization,	F01BRF
Real Sparse Matrix, Factorization, Known Sparsity Pattern,	F01BSF

(iii) **Matrix Arithmetic and Manipulation**

Matrix Addition,	
Real Matrices	F01CTF
Complex Matrices	F01CWF
Matrix Multiplication,	F01CKF
Matrix Storage Conversion	
Packed Triangular ↔ Square Storage	
Real Matrices	F01ZAF
Complex Matrices	F01ZBF
Packed Band ↔ Rectangular Storage	
Real Matrices	F01ZCF
Complex Matrices	F01ZDF
Matrix Subtraction,	
Real Matrices	F01CTF
Complex Matrices	F01CWF
Matrix Transpose,	F01CRF

6 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

F01AAF	F01ACF	F01AEF	F01AFF	F01AGF	F01AHF
F01AJF	F01AKF	F01ALF	F01AMF	F01ANF	F01APF
F01ATF	F01AUF	F01AVF	F01AWF	F01AXF	F01AYF
F01AZF	F01BCF	F01BDF	F01BEF	F01BNF	F01BPF
F01BQF	F01BTF	F01BWF	F01BXF	F01CAF	F01CBF
F01CDF	F01CEF	F01CFF	F01CGF	F01CHF	F01CLF
F01CMF	F01CNF	F01CPF	F01CQF	F01CSF	F01DAF
F01DBF	F01DCF	F01DDF	F01DEF	F01LBF	F01LZF
F01MAF	F01NAF	F01QAF	F01QBF	F01QCF	F01QDF
F01QEF	F01QFF	F01RCF	F01RDF	F01REF	F01RFF

7 References

[1] Golub G H and Van Loan C F (1996) *Matrix Computations* Johns Hopkins University Press (3rd Edition), Baltimore

[2] Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation II, Linear Algebra* Springer–Verlag

[3] Wilkinson J H (1965) *The Algebraic Eigenvalue Problem* Oxford University Press, London

[4] Wilkinson J H (1977) Some recent advances in numerical linear algebra *The State of the Art in Numerical Analysis* (ed D A H Jacobs) Academic Press

Chapter F02 – Eigenvalues and Eigenvectors

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
F02BJF	6	All eigenvalues and optionally eigenvectors of generalized eigenproblem by QZ algorithm, real matrices (Black Box)
F02EAF	16	All eigenvalues and Schur factorization of real general matrix (Black Box)
F02EBF	16	All eigenvalues and eigenvectors of real general matrix (Black Box)
F02ECF	17	Selected eigenvalues and eigenvectors of real nonsymmetric matrix (Black Box)
F02FAF	16	All eigenvalues and eigenvectors of real symmetric matrix (Black Box)
F02FCF	17	Selected eigenvalues and eigenvectors of real symmetric matrix (Black Box)
F02FDF	16	All eigenvalues and eigenvectors of real symmetric-definite generalized problem (Black Box)
F02FHF	11	All eigenvalues of generalized banded real symmetric-definite eigenproblem (Black Box)
F02FJF	11	Selected eigenvalues and eigenvectors of sparse symmetric eigenproblem (Black Box)
F02GAF	16	All eigenvalues and Schur factorization of complex general matrix (Black Box)
F02GBF	16	All eigenvalues and eigenvectors of complex general matrix (Black Box)
F02GCF	17	Selected eigenvalues and eigenvectors of complex nonsymmetric matrix (Black Box)
F02GJF	8	All eigenvalues and optionally eigenvectors of generalized complex eigenproblem by QZ algorithm (Black Box)
F02HAF	16	All eigenvalues and eigenvectors of complex Hermitian matrix (Black Box)
F02HCF	17	Selected eigenvalues and eigenvectors of complex Hermitian matrix (Black Box)
F02HDF	16	All eigenvalues and eigenvectors of complex Hermitian-definite generalized problem (Black Box)
F02SDF	8	Eigenvector of generalized real banded eigenproblem by inverse iteration
F02WDF	8	QR factorization, possibly followed by SVD
F02WEF	13	SVD of real matrix (Black Box)
F02WUF	14	SVD of real upper triangular matrix (Black Box)
F02XEF	13	SVD of complex matrix (Black Box)
F02XUF	13	SVD of complex upper triangular matrix (Black Box)

Chapter F02

Eigenvalues and Eigenvectors

Contents

1 Scope of the Chapter

This chapter provides routines for various types of matrix eigenvalue problem:

- standard eigenvalue problems (finding eigenvalues and eigenvectors of a square matrix A)
- singular value problems (finding singular values and singular vectors of a rectangular matrix A)
- generalized eigenvalue problems (finding eigenvalues and eigenvectors of a matrix pencil $A - \lambda B$)

Routines are provided for both real and complex data.

Additional routines for these problems can be found in Chapter F08 which contains software derived from LAPACK (see Anderson *et al.* [1]). However, you should read the introduction to this chapter, F02, before turning to F08, especially if you are a new user.

Chapter F02 contains **Black Box** routines that enable many problems to be solved by a call to a single routine, and the decision trees in Section 4 direct you to the most appropriate routines in Chapter F02. These Black Box routines call routines in Chapter F07 and Chapter F08 wherever possible to perform the computations, and there are pointers in Section 4 to the relevant decision trees in Chapter F08.

2 Background to the Problems

Here we describe the different types of problem which can be tackled by the routines in this chapter, and give a brief outline of the methods used to solve them. If you have one specific type of problem to solve, you need only read the relevant subsection and then turn to Section 3. Consult a standard textbook for a more thorough discussion, for example Golub and Van Loan [2] or Parlett [3].

In each subsection, we first describe the problem in terms of real matrices. The changes needed to adapt the discussion to complex matrices are usually simple and obvious: a matrix transpose such as Q^T must be replaced by its conjugate transpose Q^H; symmetric matrices must be replaced by Hermitian matrices, and orthogonal matrices by unitary matrices. Any additional changes are noted at the end of the subsection.

2.1 Standard Eigenvalue Problems

Let A be a square matrix of order n. The *standard eigenvalue problem* is to find eigenvalues, λ, and corresponding eigenvectors, $x \neq 0$, such that

$$Ax = \lambda x. \tag{1}$$

(The phrase 'eigenvalue problem' is sometimes abbreviated to *eigenproblem.*)

2.1.1 Standard symmetric eigenvalue problems

If A is real symmetric, the eigenvalue problem has many desirable features, and it is advisable to take advantage of symmetry whenever possible.

The eigenvalues λ are all real, and the eigenvectors can be chosen to be mutually orthogonal. That is, we can write

$$Az_i = \lambda_i z_i \quad \text{for} \quad i = 1, 2, \ldots, n$$

or equivalently:

$$AZ = Z\Lambda \tag{2}$$

where Λ is a real diagonal matrix whose diagonal elements λ_i are the eigenvalues, and Z is a real orthogonal matrix whose columns z_i are the eigenvectors. This implies that $z_i^T z_j = 0$ if $i \neq j$, and $\|z_i\|_2 = 1$.

Equation (2) can be rewritten

$$A = Z\Lambda Z^T. \tag{3}$$

This is known as the *eigen-decomposition* or *spectral factorization* of A.

Eigenvalues of a real symmetric matrix are well-conditioned, that is, they are not unduly sensitive to perturbations in the original matrix A. The sensitivity of an eigenvector depends on how small the gap is between its eigenvalue and any other eigenvalue: the smaller the gap, the more sensitive the eigenvector. More details on the accuracy of computed eigenvalues and eigenvectors are given in the routine documents, and in the F08 Chapter Introduction.

For dense or band matrices, the computation of eigenvalues and eigenvectors proceeds in the following stages:

(1) A is reduced to a symmetric tridiagonal matrix T by an orthogonal similarity transformation: $A = QTQ^T$, where Q is orthogonal. (A *tridiagonal* matrix is zero except for the main diagonal and the first subdiagonal and superdiagonal on either side.) T has the same eigenvalues as A and is easier to handle.

(2) Eigenvalues and eigenvectors of T are computed as required. If all eigenvalues (and optionally eigenvectors) are required, they are computed by the QR algorithm, which effectively factorizes T as $T = S\Lambda S^T$, where S is orthogonal. If only selected eigenvalues are required, they are computed by bisection, and if selected eigenvectors are required, they are computed by inverse iteration. If s is an eigenvector of T, then Qs is an eigenvector of A.

All the above remarks also apply – with the obvious changes – to the case when A is a complex Hermitian matrix. The eigenvectors are complex, but the eigenvalues are all real, and so is the tridiagonal matrix T.

If A is large and sparse, the methods just described would be very wasteful in both storage and computing time, and therefore an alternative algorithm, known as *subspace iteration*, is provided (for real problems only) to find a (usually small) subset of the eigenvalues and their corresponding eigenvectors.

2.1.2 Standard nonsymmetric eigenvalue problems

A real nonsymmetric matrix A may have complex eigenvalues, occurring as complex conjugate pairs. If x is an eigenvector corresponding to a complex eigenvalue λ, then the complex conjugate vector $\bar{x}$ is the eigenvector corresponding to the complex conjugate eigenvalue $\bar{\lambda}$. Note that the vector x defined in equation (1) is sometimes called a *right eigenvector*; a *left eigenvector* y is defined by:

$$y^H A = \lambda y^H \quad \text{or} \quad A^T y = \bar{\lambda} y.$$

Routines in this chapter only compute right eigenvectors (the usual requirement), but routines in Chapter F08 can compute left or right eigenvectors or both.

The eigenvalue problem can be solved via the *Schur factorization* of A, defined as

$$A = ZTZ^T,$$

where Z is an orthogonal matrix and T is a real upper *quasi-triangular* matrix, with the same eigenvalues as A. T is called the *Schur form* of A. If all the eigenvalues of A are real, then T is upper triangular, and its diagonal elements are the eigenvalues of A. If A has complex conjugate pairs of eigenvalues, then T has 2 by 2 diagonal blocks, whose eigenvalues are the complex conjugate pairs of eigenvalues of A. (The structure of T is simpler if the matrices are complex – see below.)

For example, the following matrix is in quasi-triangular form

$$\begin{pmatrix} 1 & * & * & * \\ 0 & 2 & -1 & * \\ 0 & 1 & 2 & * \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

and has eigenvalues 1, $2 \pm i$, and 3. (The elements indicated by '$*$' may take any values.)

The columns of Z are called the *Schur vectors*. For each k ($1 \le k \le n$), the first k columns of Z form an orthonormal basis for the invariant subspace corresponding to the first k eigenvalues on the diagonal of T. (An *invariant subspace* (for A) is a subspace S such that for any vector v in S, Av is also in S.) Because this basis is orthonormal, it is preferable in many applications to compute Schur vectors rather than eigenvectors. It is possible to order the Schur factorization so that any desired set of k eigenvalues occupy the k leading positions on the diagonal of T, and routines for this purpose are provided in Chapter F08.

Note that if A is symmetric, the Schur vectors are the same as the eigenvectors, but if A is nonsymmetric, they are distinct, and the Schur vectors, being orthonormal, are often more satisfactory to work with in numerical computation.

Eigenvalues and eigenvectors of an nonsymmetric matrix may be ill-conditioned, that is, sensitive to perturbations in A. Chapter F08 contains routines which compute or estimate the condition numbers of eigenvalues and eigenvectors, and the Introduction to that Chapter gives more details about the error analysis of nonsymmetric eigenproblems. The accuracy with which eigenvalues and eigenvectors can be obtained is often improved by *balancing* a matrix. This is discussed further in Section 3.4 below.

Computation of eigenvalues, eigenvectors or the Schur factorization proceeds in the following stages:

(1) A is reduced to an upper Hessenberg matrix H by an orthogonal similarity transformation: $A = QHQ^T$, where Q is orthogonal. (An *upper Hessenberg* matrix is zero below the first subdiagonal.) H has the same eigenvalues as A, and is easier to handle.

(2) The upper Hessenberg matrix H is reduced to Schur form T by the QR algorithm, giving the Schur factorization $H = STS^T$. The eigenvalues of A are obtained from the diagonal blocks of T. The matrix Z of Schur vectors (if required) is computed as $Z = QS$.

(3) After the eigenvalues have been found, eigenvectors may be computed, if required, in two different ways. Eigenvectors of H can be computed by inverse iteration, and then pre-multiplied by Q to give eigenvectors of A; this approach is usually preferred if only a few eigenvectors are required. Alternatively, eigenvectors of T can be computed by back-substitution, and pre-multiplied by Z to give eigenvectors of A.

All the above remarks also apply – with the obvious changes – to the case when A is a complex matrix. The eigenvalues are in general complex, so there is no need for special treatment of complex conjugate pairs, and the Schur form T is simply a complex upper triangular matrix.

2.2 The Singular Value Decomposition

The *singular value decomposition* (SVD) of a real m by n matrix A is given by

$$A = U\Sigma V^T,$$

where U and V are orthogonal and Σ is an m by n diagonal matrix with real diagonal elements, σ_i, such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0.$$

The σ_i are the *singular values* of A and the first $\min(m,n)$ columns of U and V are, respectively, the *left* and *right singular vectors* of A. The singular values and singular vectors satisfy

$$Av_i = \sigma_i u_i \text{ and } A^T u_i = \sigma_i v_i$$

where u_i and v_i are the ith columns of U and V respectively.

The singular value decomposition of A is closely related to the eigen-decompositions of the symmetric matrices A^TA or AA^T, because:

$$A^TAv_i = \sigma_i^2 v_i \text{ and } AA^Tu_i = \sigma_i^2 u_i.$$

However, these relationships are not recommended as a means of computing singular values or vectors.

Singular values are well-conditioned, that is, they are not unduly sensitive to perturbations in A. The sensitivity of a singular vector depends on how small the gap is between its singular value and any other singular value: the smaller the gap, the more sensitive the singular vector. More details on the accuracy of computed singular values and vectors are given in the routine documents and in the the F08 Chapter Introduction.

The singular value decomposition is useful for the numerical determination of the rank of a matrix, and for solving linear least-squares problems, especially when they are rank-deficient (or nearly so). See Chapter F04.

Computation of singular values and vectors proceeds in the following stages:

(1) A is reduced to an upper bidiagonal matrix B by an orthogonal transformation $A = U_1 B V_1^T$, where U_1 and V_1 are orthogonal. (An *upper bidiagonal* matrix is zero except for the main diagonal and the first superdiagonal.) B has the same singular values as A, and is easier to handle.

(2) The SVD of the bidiagonal matrix B is computed as $B = U_2 \Sigma V_2^T$, where U_2 and V_2 are orthogonal and Σ is diagonal as described above. Then in the SVD of A, $U = U_1 U_2$ and $V = V_1 V_2$.

All the above remarks also apply – with the obvious changes – to the case when A is a complex matrix. The singular vectors are complex, but the singular values are real and non-negative, and the bidiagonal matrix B is also real.

2.3 Generalized Eigenvalue Problems

Let A and B be square matrices of order n. The *generalized eigenvalue problem* is to find eigenvalues, λ, and corresponding eigenvectors, $x \neq 0$, such that

$$Ax = \lambda Bx. \tag{4}$$

For given A and B, the set of all matrices of the form $A - \lambda B$ is called a *pencil*, and λ and x are said to be an eigenvalue and eigenvector of the pencil $A - \lambda B$.

When B is non-singular, equation (4) is mathematically equivalent to $(B^{-1}A)x = \lambda x$, and when A is non-singular, it is equivalent to $(A^{-1}B)x = (1/\lambda)x$. Thus, in theory, if one of the matrices A or B is known to be nonsingular, the problem could be reduced to a standard eigenvalue problem.

However, for this reduction to be satisfactory from the point of view of numerical stability, it is necessary not only that B (or A) should be nonsingular, but that it should be well-conditioned with respect to inversion. The nearer B is to singularity, the more unsatisfactory $B^{-1}A$ will be as a vehicle for determining the required eigenvalues. Well-determined eigenvalues of the original problem (4) may be poorly determined even by the correctly rounded version of $B^{-1}A$.

We consider first a special class of problems in which B is known to be non-singular, and then return to the general case in the following subsection.

2.3.1 Generalized symmetric-definite eigenvalue problems

If A and B are *symmetric* and B is *positive-definite*, then the generalized eigenvalue problem has desirable properties similar to those of the standard symmetric eigenvalue problem. The eigenvalues are all real, and the eigenvectors, while not orthogonal in the usual sense, satisfy the relations $z_i^T B z_j = 0$ for $i \neq j$ and can be normalized so that $z_i^T B z_i = 1$.

Note that it is not enough for A and B to be symmetric; B must also be positive-definite, which implies non-singularity. Eigenproblems with these properties are referred to as *symmetric-definite* problems.

If Λ is the diagonal matrix whose diagonal elements are the eigenvalues, and Z is the matrix whose columns are the eigenvectors, then:

$$Z^T A Z = \Lambda \text{ and } Z^T B Z = I.$$

To compute eigenvalues and eigenvectors, the problem can be reduced to a standard symmetric eigenvalue problem, using the Cholesky factorization of B as LL^T or $U^T U$ (see Chapter F07). Note, however, that this reduction does implicitly involve the inversion of B, and hence this approach should *not* be used if B is ill-conditioned with respect to inversion.

For example, with $B = LL^T$, we have

$$Az = \lambda Bz \Leftrightarrow (L^{-1}A(L^{-1})^T)(L^T z) = \lambda(L^T z).$$

Hence the eigenvalues of $Az = \lambda Bz$ are those of $Cy = \lambda y$, where C is the symmetric matrix $C = L^{-1}A(L^{-1})^T$ and $y = L^T z$. The standard symmetric eigenproblem $Cy = \lambda y$ may be solved by the methods described in Section 2.1.1. The eigenvectors z of the original problem may be recovered by computing $z = (L^T)^{-1} y$.

Most of the routines which solve this class of problems can also solve the closely related problems:

$$ABx = \lambda x \text{ or } BAx = \lambda x$$

where again A and B are symmetric and B is positive-definite. See the routine documents for details.

All the above remarks also apply – with the obvious changes – to the case when A and B are complex Hermitian matrices. Such problems are called *Hermitian-definite*. The eigenvectors are complex, but the eigenvalues are all real.

If A and B are large and sparse, reduction to an equivalent standard eigenproblem as described above would almost certainly result in a large dense matrix C, and hence would be very wasteful in both storage and computing time. The method of subspace iteration, mentioned in Section 2.1.1, can also be used for large sparse generalized symmetric-definite problems.

2.3.2 Generalized nonsymmetric eigenvalue problems

Any generalized eigenproblem which is not symmetric-definite with well-conditioned B must be handled as if it were a general nonsymmetric problem.

If B is singular, the problem has infinite eigenvalues. These are not a problem; they are equivalent to zero eigenvalues of the problem $Bx = \mu Ax$. Computationally they appear as very large values.

If A and B are both singular and have a common null-space, then $A - \lambda B$ is singular for all λ; in other words, any value λ can be regarded as an eigenvalue. Pencils with this property are called *singular*.

As with standard nonsymmetric problems, a real problem may have complex eigenvalues, occurring as complex conjugate pairs.

The generalized eigenvalue problem can be solved via the *generalized Schur factorization* of A and B:

$$A = QUZ^T, \quad B = QVZ^T$$

where Q and Z are orthogonal, V is upper triangular, and U is upper quasi-triangular (defined just as in Section 2.1.2).

If all the eigenvalues are real, then U is upper triangular; the eigenvalues are given by $\lambda_i = u_{ii}/v_{ii}$. If there are complex conjugate pairs of eigenvalues, then U has 2 by 2 diagonal blocks.

Eigenvalues and eigenvectors of a generalized nonsymmetric problem may be ill-conditioned, that is, sensitive to perturbations in A or B.

Particular care must be taken if, for some i, $u_{ii} = v_{ii} = 0$, or in practical terms if u_{ii} and v_{ii} are both small; this means that the pencil is singular, or approximately so. Not only is the particular value λ_i undetermined, but also **no reliance** can be placed on **any** of the computed eigenvalues. See also the routine documents.

Computation of eigenvalues and eigenvectors proceeds in the following stages:

(1) The pencil $A - \lambda B$ is reduced by an orthogonal transformation to a pencil $H - \lambda K$ in which H is upper Hessenberg and K is upper triangular: $A = Q_1 H Z_1^T$ and $B = Q_1 K Z_1^T$. The pencil $H - \lambda K$ has the same eigenvalues as $A - \lambda B$, and is easier to handle.

(2) The upper Hessenberg matrix H is reduced to upper quasi-triangular form, while K is maintained in upper triangular form, using the QZ algorithm. This gives the generalized Schur factorization: $H = Q_2 U Z_2$ and $K = Q_2 V Z_2$.

(3) Eigenvectors of the pencil $U - \lambda V$ are computed (if required) by backsubstitution, and pre-multiplied by $Z_1 Z_2$ to give eigenvectors of A.

All the above remarks also apply – with the obvious changes – to the case when A and B are complex matrices. The eigenvalues are in general complex, so there is no need for special treatment of complex conjugate pairs, and the matrix U in the generalized Schur factorization is simply a complex upper triangular matrix.

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 Black Box Routines and General Purpose Routines

Routines in the NAG Library for solving eigenvalue problems fall into two categories:

Black Box Routines: These are designed to solve a standard type of problem in a single call – for example, to compute all the eigenvalues and eigenvectors of a real symmetric matrix. You are recommended to use a black box routine if there is one to meet your needs; refer to the decision tree in Section 4.1 or the index in Section 5.

General Purpose Routines: These perform the computational subtasks which make up the separate stages of the overall task, as described in Section 2 – for example, reducing a real symmetric matrix to tridiagonal form. General purpose routines are to be found, for historical reasons, some in this Chapter, a few in Chapter F01, but most in Chapter F08. If there is no black box routine that meets your needs, you will need to use one or more general purpose routines.

Here are some of the more likely reasons why you may need to do this:

> Your problem is already in one of the reduced forms – for example, your symmetric matrix is already tridiagonal.
>
> You wish to economize on storage for symmetric matrices (see Section 3.3).
>
> You wish to find selected eigenvalues or eigenvectors of a generalized symmetric-definite eigenproblem (see also Section 3.2).

The decision trees in Section 4.2 list the combinations of general purpose routines which are needed to solve many common types of problem.

Sometimes a combination of a black box routine and one or more general purpose routines will be the most convenient way to solve your problem: the black box routine can be used to compute most of the results, and a general purpose routine can be used to perform a subsidiary computation, such as computing condition numbers of eigenvalues and eigenvectors.

3.2 Computing Selected Eigenvalues and Eigenvectors

The decision trees and the routine documents make a distinction between routines which compute *all* eigenvalues or eigenvectors, and routines which compute *selected* eigenvalues or eigenvectors; the two classes of routine use different algorithms.

It is difficult to give clear guidance on which of these two classes of routine to use in a particular case, especially with regard to computing eigenvectors. If you only wish to compute a very few eigenvectors, then a routine for selected eigenvectors will be more economical, but if you want to compute a substantial subset (an old rule of thumb suggested more than 25%), then it may be more economical to compute all of them. Conversely, if you wish to compute all the eigenvectors of a sufficiently large symmetric tridiagonal matrix, the routine for selected eigenvectors may be faster.

The choice depends on the properties of the matrix and on the computing environment; if it is critical, you should perform your own timing tests.

For nonsymmetric eigenproblems, there are no algorithms provided for computing selected eigenvalues; it is always necessary to compute all the eigenvalues, but you can then select specific eigenvectors for computation by inverse iteration.

3.3 Storage Schemes for Symmetric Matrices

Routines which handle symmetric matrices are usually designed to use either the upper or lower triangle of the matrix; it is not necessary to store the whole matrix. If either the upper or lower triangle is stored conventionally in the upper or lower triangle of a 2-dimensional array, the remaining elements of the array can be used to store other useful data. However, that is not always convenient, and if it is important to economize on storage, the upper or lower triangle can be stored in a 1-dimensional array of length

$n(n+1)/2$ – in other words, the storage is almost halved. This storage format is referred to as *packed storage*.

Routines designed for packed storage are usually less efficient, especially on high-performance computers, so there is a trade-off between storage and efficiency.

A ***band*** matrix is one whose non-zero elements are confined to a relatively small number of sub-diagonals or super-diagonals on either side of the main diagonal. Algorithms can take advantage of bandedness to reduce the amount of work and storage required.

Routines which take advantage of packed storage or bandedness are provided for both standard symmetric eigenproblems and generalized symmetric-definite eigenproblems.

3.4 Balancing for Nonsymmmetric Eigenproblems

There are two preprocessing steps which one may perform on an nonsymmetric matrix A in order to make its eigenproblem easier. Together they are referred to as *balancing*.

Permutation: This involves reordering the rows and columns to make A more nearly upper triangular (and thus closer to Schur form): $A' = PAP^T$, where P is a permutation matrix. If A has a significant number of zero elements, this preliminary permutation can reduce the amount of work required, and also improve the accuracy of the computed eigenvalues. In the extreme case, if A is permutable to upper triangular form, then no floating-point operations are needed to reduce it to Schur form.

Scaling: A diagonal matrix D is used to make the rows and columns of A' more nearly equal in norm: $A'' = DA'D^{-1}$. Scaling can make the matrix norm smaller with respect to the eigenvalues, and so possibly reduce the inaccuracy contributed by roundoff (see Chap. II/11, of [4]).

Routines are provided in Chapter F08 for performing either or both of these pre-processing steps, and also for transforming computed eigenvectors or Schur vectors back to those of the original matrix.

Black box routines in this chapter which compute the Schur factorization perform only the permutation step, since diagonal scaling is not in general an orthogonal transformation. The black box routines which compute eigenvectors perform both forms of balancing.

3.5 Non-uniqueness of Eigenvectors and Singular Vectors

Eigenvectors, as defined by equations (1) or (4), are ***not uniquely defined.*** If x is an eigenvector, then so is kx where k is any non-zero scalar. Eigenvectors computed by different algorithms, or on different computers, may appear to disagree completely, though in fact they differ only by a scalar factor (which may be complex). These differences should not be significant in any application in which the eigenvectors will be used, but they can arouse uncertainty about the correctness of computed results.

Even if eigenvectors x are normalized so that $\|x\|_2 = 1$, this is not sufficient to fix them uniquely, since they can still be multiplied by a scalar factor k such that $|k| = 1$. To counteract this inconvenience, most of the routines in this chapter, and in Chapter F08, normalize eigenvectors (and Schur vectors) so that $\|x\|_2 = 1$ ***and*** the component of x with largest absolute value is real and positive. (There is still a possible indeterminacy if there are two components of equal largest absolute value – or in practice if they are very close – but this is rare.)

In symmetric problems the computed eigenvalues are sorted into ascending order, but in nonsymmetric problems the order in which the computed eigenvalues are returned is dependent on the detailed working of the algorithm and may be sensitive to rounding errors. The Schur form and Schur vectors depend on the ordering of the eigenvalues and this is another possible cause of non-uniqueness when they are computed. However, it must be stressed again that variations in the results from this cause should not be significant. (Routines in Chapter F08 can be used to transform the Schur form and Schur vectors so that the eigenvalues appear in any given order if this is important.)

In singular value problems, the left and right singular vectors u and v which correspond to a singular value σ cannot be normalized independently: if u is multiplied by a factor k such that $|k| = 1$, then v must also be multiplied by k.

Non-uniqueness also occurs among eigenvectors which correspond to a multiple eigenvalue, or among singular vectors which correspond to a multiple singular value. In practice, this is more likely to be apparent as the extreme sensitivity of eigenvectors which correspond to a cluster of close eigenvalues (or of singular vectors which correspond to a cluster of close singular values).

4 Decision Trees

4.1 Black Box routines

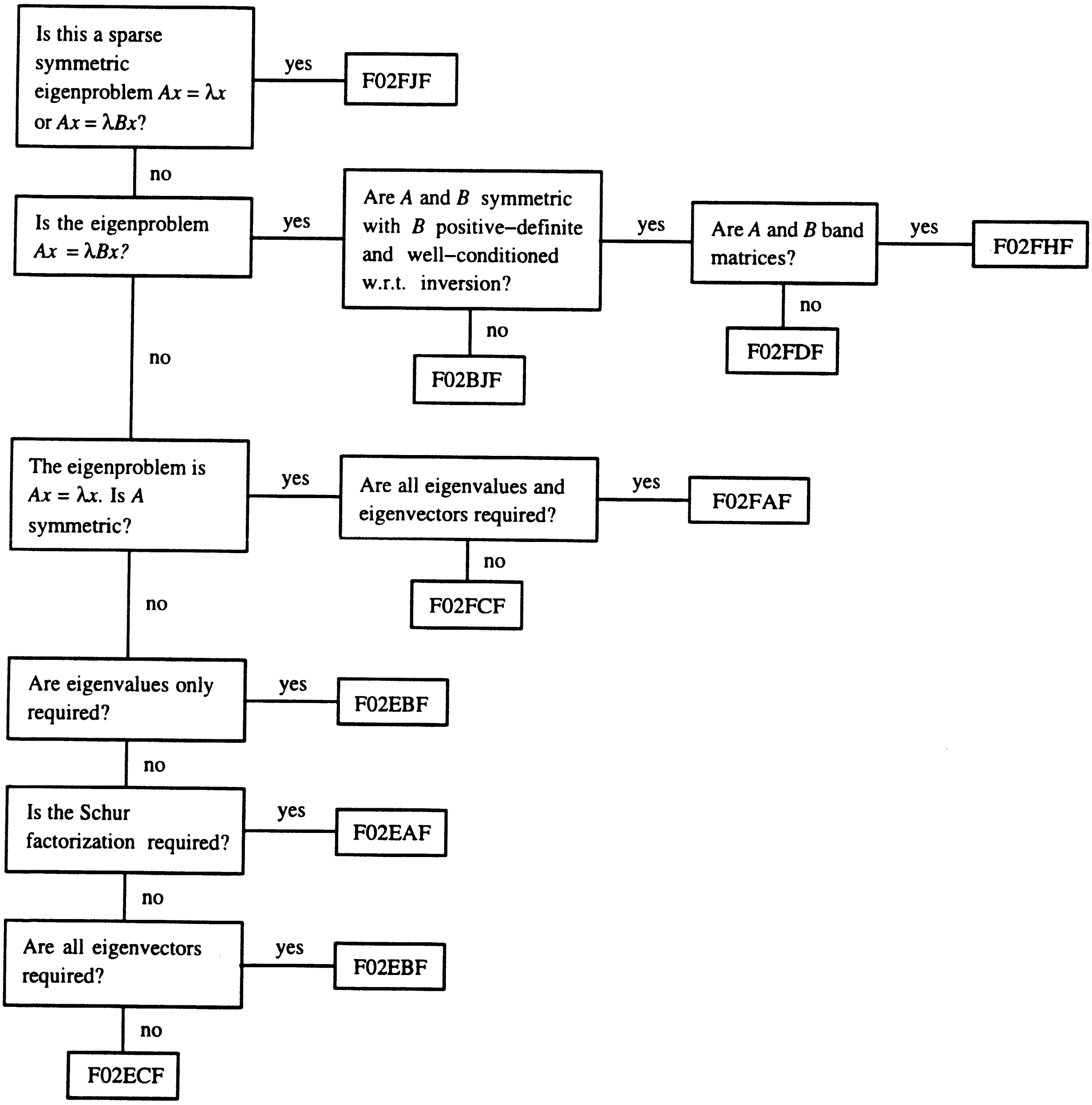
The decision tree for this section is divided into three sub-trees:

Tree 1: Eigenvalues and eigenvectors of real matrices

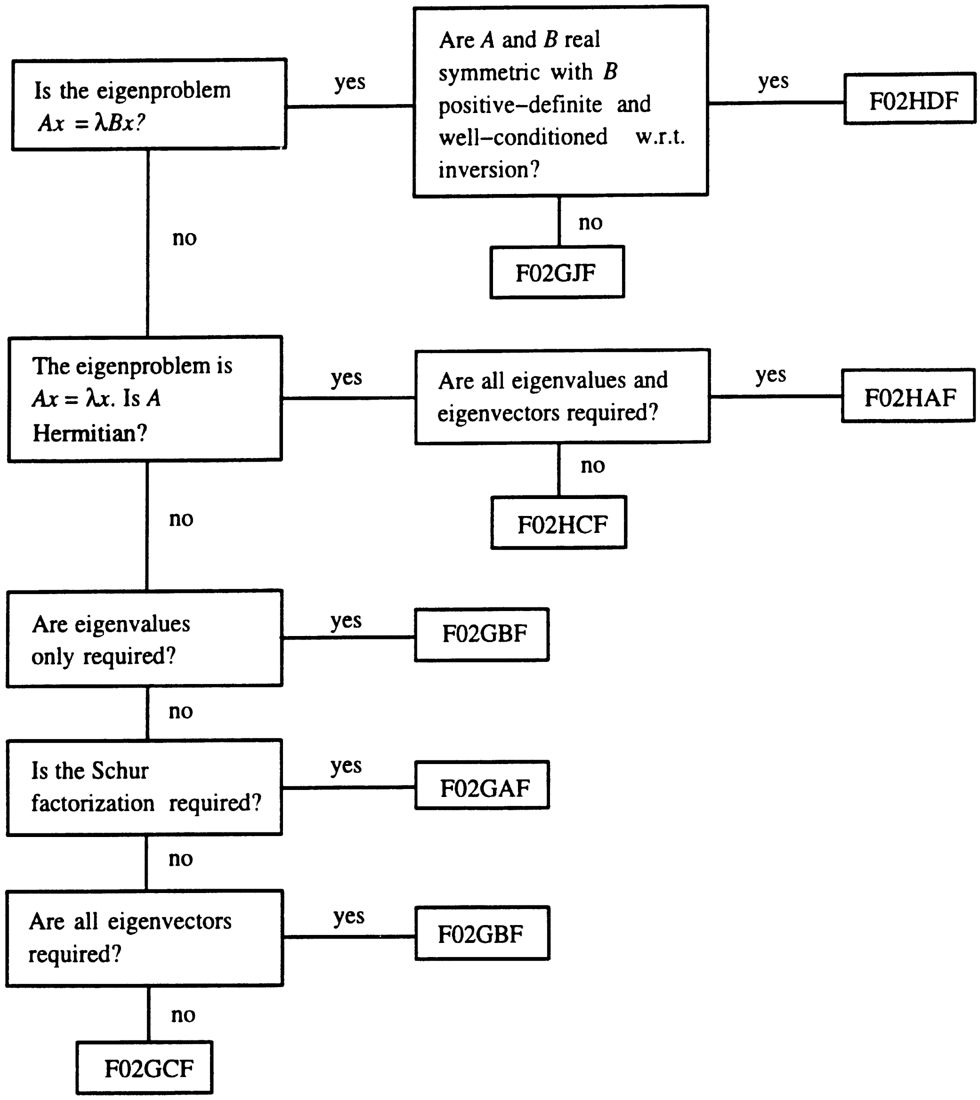
Tree 2: Eigenvalues and eigenvectors of complex matrices

Tree 3: Singular values and singular vectors

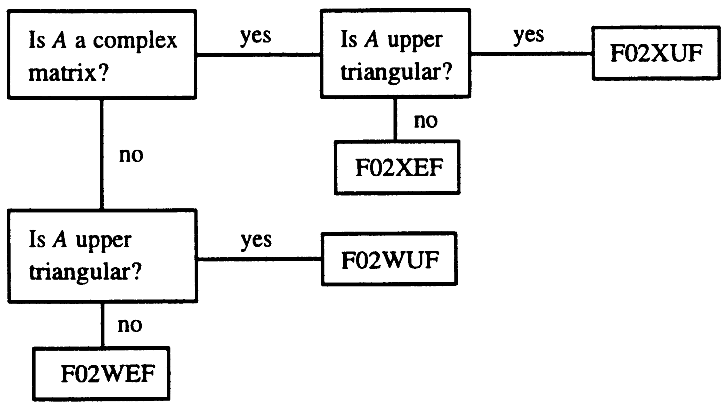
Tree 1: Eigenvalues and Eigenvectors of Real Matrices



Tree 2: Eigenvalues and Eigenvectors of Complex Matrices



Tree 3: Singular Values and Singular Vectors



4.2 General purpose routines (eigenvalues and eigenvectors)

The decision tree for this section is divided into six sub-trees. These are given in Section 4 of the F08 Chapter Introduction:

Tree 1: Real symmetric eigenproblems, $Ax = \lambda x$.

Tree 2: Real generalized symmetric-definite eigenproblems, $Ax = \lambda Bx, ABx =$ or $BAx = \lambda x$.

Tree 3: Real nonsymmetric eigenproblems, $Ax = \lambda x$.

Tree 4: Complex Hermitian eigenproblems, $Ax = \lambda x$.

Tree 5: Complex generalized Hermitian-definite eigenproblems, $Ax = \lambda Bx, ABx = \lambda x$ or $BAx = \lambda x$.

Tree 6: Complex non-Hermitian eigenproblems, $Ax = \lambda x$.

Note. There are no general purpose routines for the problem $Ax = \lambda Bx$, where A and B are real or complex but not otherwise specialised. Use the Black Box routines F02BJF or F02GJF.

As it is very unlikely that one of the routines in this section will be called on its own, the other routines required to solve a given problem are listed in the order in which they should be called.

4.3 General purpose routines (singular value decomposition)

See Section 4.2 of the F08 Chapter Introduction.

5 Index

Black Box Routines

Complex Hermitian Matrix, All Eigenvalues and Eigenvectors	F02HAF
Complex Hermitian Matrix, Selected Eigenvalues and Eigenvectors	F02HCF
Complex Matrix, All Eigenvalues and Eigenvectors	F02GBF
Complex Matrix, Schur Factorization	F02GAF
Complex Matrix, Selected Eigenvalues and Eigenvectors	F02GCF
Complex Upper Triangular Matrix, Singular Values and, optionally, Left and/or Right Singular Vectors	F02XUF
Complex m by n Matrix, Singular Values and, optionally, Left and/or Right Singular Vectors	F02XEF
Generalized Complex Eigenproblem	F02GJF
Generalized Complex Hermitian-Definite Eigenproblem, All Eigenvalues and Eigenvectors	F02HDF
Generalized Real Eigenproblem	F02BJF
Generalized Real Band Symmetric-Definite Eigenproblem, Eigenvalues	F02FHF
Generalized Real Sparse Symmetric-Definite Eigenproblem, Selected Eigenvalues and Eigenvectors	F02FJF
Generalized Real Symmetric-Definite Eigenproblem, All Eigenvalues and Eigenvectors	F02FDF
Real Sparse Symmetric Matrix, Selected Eigenvalues and Eigenvectors	F02FJF
Real Symmetric Matrix, All Eigenvalues and Eigenvectors	F02FAF
Real Symmetric Matrix, Selected Eigenvalues and Eigenvectors	F02FCF
Real Matrix, All Eigenvalues and Eigenvectors	F02EBF
Real Matrix, Schur Factorization	F02EAF
Real Matrix, Selected Eigenvalues and Eigenvectors	F02ECF
Real Upper Triangular Matrix, Singular Values and, optionally, Left and/or Right Singular Vectors	F02WUF
Real m by n Matrix, Singular Values and, optionally, Left and/or Right Singular Vectors	F02WEF

General purpose routines (see also Chapter F08)

Real m by n Matrix ($m \geq n$), QR factorization and SVD	F02WDF
Real Band Matrix, Selected Eigenvector, $A - \lambda B$	F02SDF

6 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

F02AAF	F02ABF	F02ADF	F02AEF	F02AFF	F02AGF
F02AJF	F02AKF	F02AMF	F02ANF	F02APF	F02AQF
F02ARF	F02AVF	F02AWF	F02AXF	F02AYF	F02BBF
F02BCF	F02BDF	F02BEF	F02BFF	F02BKF	F02BLF
F02SWF	F02SXF	F02SYF	F02SZF	F02UWF	F02UXF
F02UYF	F02WAF	F02WBF	F02WCF		

7 References

[1] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A, Ostrouchov S and Sorensen D (1995) *LAPACK Users' Guide* (2nd Edition) SIAM, Philadelphia

[2] Golub G H and Van Loan C F (1996) *Matrix Computations* Johns Hopkins University Press (3rd Edition), Baltimore

[3] Parlett B N (1980) *The Symmetric Eigenvalue Problem* Prentice–Hall

[4] Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation II, Linear Algebra* Springer–Verlag

Chapter F03 – Determinants

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
F03AAF	1	Determinant of real matrix (Black Box)
F03ABF	1	Determinant of real symmetric positive-definite matrix (Black Box)
F03ACF	1	Determinant of real symmetric positive-definite band matrix (Black Box)
F03ADF	1	Determinant of complex matrix (Black Box)
F03AEF	2	LL^T factorization and determinant of real symmetric positive-definite matrix
F03AFF	2	LU factorization and determinant of real matrix

Chapter F03

Determinants

Contents

1 Scope of the Chapter

This chapter is concerned with the calculation of determinants of square matrices.

2 Background to the Problems

The routines in this chapter compute the determinant of a square matrix A. The matrix is first decomposed into triangular factors

$$A = LU.$$

If A is positive-definite, then $U = L^T$, and the determinant is the product of the squares of the diagonal elements of L. Otherwise, the routines in this chapter use the Crout form of the LU decomposition, where U has unit elements on its diagonal. The determinant is then the product of the diagonal elements of L, taking account of possible sign changes due to row interchanges.

To avoid overflow or underflow in the computation of the determinant, some scaling is associated with each multiplication in the product of the relevant diagonal elements. The final value is represented by:

$$\det A = d1 \times 2^{d2}$$

where $d2$ is an integer and

$$\tfrac{1}{16} \leq |d1| < 1.$$

Most of the routines of the chapter are based on those published in the book edited by Wilkinson and Reinsch [2]. We are very grateful to the late Dr J H Wilkinson, F R S, for his help and interest during the implementation of this chapter of the Library.

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 General Discussion

It is extremely wasteful of computer time and storage to use an inappropriate routine, for example one for a complex matrix when A is real. Most programmers will know whether their matrix is real or complex, but may be less certain whether or not a real symmetric matrix A is positive-definite, i.e., all eigenvalues of $A > 0$. A real symmetric matrix A not known to be positive-definite must be treated as a general real matrix. In all other cases either the band routine or the general routines must be used.

The routines in this chapter fall into easily defined categories.

(i) **Black Box Routines**

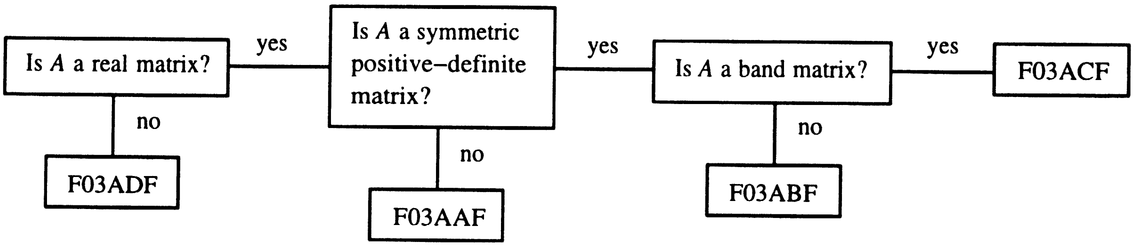
These should be used if only the determinant is required. The scaled representation $d1 \times 2^{d2}$ is evaluated as a floating-point number and a failure is indicated if the floating-point number is outside the range of the machine.

(ii) **General Purpose Routines**

These give the value of the determinant in its scaled form, $d1$ and $d2$, and also give the triangular decomposition of the matrix A in a form suitable for input to either the inversion routines of Chapter F01 or the solution of linear equation routines in Chapter F04.

4 Decision Tree

If at any stage the answer to a question is 'Don't know' this should be read as 'No'.



5 Index

Black Box Routines	
Complex Matrix	**F03ADF**
Real Matrix	**F03AAF**
Real Symmetric Positive-Definite Matrix	**F03ABF**
Real Symmetric Positive-Definite Band Matrix	**F03ACF**
General Purpose Routines	
Including the decomposition into triangular factors:	
Real Matrix	**F03AFF**
Real Symmetric Positive-Definite Matrix	**F03AEF**

6 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'.

F03AGF F03AHF F03AMF

7 References

[1] Fox L (1964) *An Introduction to Numerical Linear Algebra* Oxford University Press

[2] Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation II, Linear Algebra* Springer-Verlag

Chapter F04 – Simultaneous Linear Equations

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
F04AAF	2	Solution of real simultaneous linear equations with multiple right-hand sides (Black Box)
F04ABF	2	Solution of real symmetric positive-definite simultaneous linear equations with multiple right-hand sides using iterative refinement (Black Box)
F04ACF	2	Solution of real symmetric positive-definite banded simultaneous linear equations with multiple right-hand sides (Black Box)
F04ADF	2	Solution of complex simultaneous linear equations with multiple right-hand sides (Black Box)
F04AEF	2	Solution of real simultaneous linear equations with multiple right-hand sides using iterative refinement (Black Box)
F04AFF	2	Solution of real symmetric positive-definite simultaneous linear equations using iterative refinement (coefficient matrix already factorized by F03AEF)
F04AGF	2	Solution of real symmetric positive-definite simultaneous linear equations (coefficient matrix already factorized by F03AEF)
F04AHF	2	Solution of real simultaneous linear equations using iterative refinement (coefficient matrix already factorized by F03AFF)
F04AJF	2	Solution of real simultaneous linear equations (coefficient matrix already factorized by F03AFF)
F04AMF	2	Least-squares solution of m real equations in n unknowns, rank $= n$, $m \geq n$ using iterative refinement (Black Box)
F04ARF	4	Solution of real simultaneous linear equations, one right-hand side (Black Box)
F04ASF	4	Solution of real symmetric positive-definite simultaneous linear equations, one right-hand side using iterative refinement (Black Box)
F04ATF	4	Solution of real simultaneous linear equations, one right-hand side using iterative refinement (Black Box)
F04AXF	7	Solution of real sparse simultaneous linear equations (coefficient matrix already factorized)
F04EAF	11	Solution of real tridiagonal simultaneous linear equations, one right-hand side (Black Box)
F04FAF	11	Solution of real symmetric positive-definite tridiagonal simultaneous linear equations, one right-hand side (Black Box)
F04FEF	15	Solution of the Yule–Walker equations for real symmetric positive-definite Toeplitz matrix, one right-hand side
F04FFF	15	Solution of real symmetric positive-definite Toeplitz system, one right-hand side
F04JAF	8	Minimal least-squares solution of m real equations in n unknowns, rank $\leq n$, $m \geq n$
F04JDF	8	Minimal least-squares solution of m real equations in n unknowns, rank $\leq n$, $m \geq n$
F04JGF	8	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	17	Real general Gauss–Markov linear model (including weighted least-squares)
F04JMF	17	Equality-constrained real linear least-squares problem
F04KLF	17	Complex general Gauss–Markov linear model (including weighted least-squares)
F04KMF	17	Equality-constrained complex linear least-squares problem

F04LEF	11	Solution of real tridiagonal simultaneous linear equations (coefficient matrix already factorized by F01LEF)
F04LHF	13	Solution of real almost block diagonal simultaneous linear equations (coefficient matrix already factorized by F01LHF)
F04MCF	8	Solution of real symmetric positive-definite variable-bandwidth simultaneous linear equations (coefficient matrix already factorized by F01MCF)
F04MEF	15	Update solution of the Yule–Walker equations for real symmetric positive-definite Toeplitz matrix
F04MFF	15	Update solution of real symmetric positive-definite Toeplitz system
F04QAF	11	Sparse linear least-squares problem, m real equations in n unknowns
F04YAF	11	Covariance matrix for linear least-squares problems, m real equations in n unknowns
F04YCF	13	Norm estimation (for use in condition estimation), real matrix
F04ZCF	13	Norm estimation (for use in condition estimation), complex matrix

Chapter F04

Simultaneous Linear Equations

Contents

1 Scope of the Chapter

This chapter is concerned with the solution of the matrix equation $AX = B$, where B may be a single vector or a matrix of multiple right-hand sides. The matrix A may be real, complex, symmetric, Hermitian, positive-definite, positive-definite Toeplitz or banded. It may also be rectangular, in which case a least-squares solution is obtained.

For a general introduction to sparse systems of equations, see the F11 Chapter Introduction, which currently provides routines for sparse symmetric systems. Some routines for sparse problems are also included in this chapter; they are described in Section 3.4.

2 Background to the Problems

A set of linear equations may be written in the form

$$Ax = b$$

where the known matrix A, with real or complex coefficients, is of size m by n, (m rows and n columns), the known right-hand vector b has m components (m rows and one column), and the required solution vector x has n components (n rows and one column). There may also be p vectors b_i, $i = 1,2,\ldots,p$ on the right-hand side and the equations may then be written as

$$AX = B,$$

the required matrix X having as its p columns the solutions of $Ax_i = b_i$, $i = 1,2,\ldots,p$. Some routines deal with the latter case, but for clarity only the case $p = 1$ is discussed here.

The most common problem, the determination of the unique solution of $Ax = b$, occurs when $m = n$ and A is not singular, that is $\mathrm{rank}(A) = n$. This is discussed in Section 2.1 below. The next most common problem, discussed in Section 2.2 below, is the determination of the least-squares solution of $Ax \simeq b$ required when $m > n$ and $\mathrm{rank}(A) = n$, i.e., the determination of the vector x which minimizes the norm of the residual vector $r = b - Ax$. All other cases are rank deficient, and they are treated in Section 2.3.

Most of the routines of the chapter are based on those published in the book edited by Wilkinson and Reinsch [3]. We are very grateful to the late Dr J H Wilkinson, F R S, for his help and interest during the implementation of this chapter of the Library.

2.1 Unique Solution of $Ax = b$

Most routines in this chapter solve this particular problem. The computation starts with the triangular decomposition $A = PLU$, where L and U are respectively lower and upper triangular matrices and P is a permutation matrix, chosen so as to ensure that the decomposition is numerically stable. The solution is then obtained by solving in succession the simpler equations

$$\begin{aligned} Ly &= P^T b \\ Ux &= y \end{aligned}$$

the first by forward-substitution and the second by back-substitution.

If A is real symmetric and positive-definite, $U = L^T$, while if A is complex Hermitian and positive-definite, $U = L^H$; in both these cases P is the identity matrix (i.e., no permutations are necessary). In all other cases either U or L has unit diagonal elements.

Due to rounding errors the computed 'solution'x_0, say, is only an approximation to the true solution x. This approximation will sometimes be satisfactory, agreeing with x to several figures, but if the problem is ill-conditioned then x and x_0 may have few or even no figures in common, and at this stage there is no means of estimating the 'accuracy' of x_0.

There are three possible approaches to estimating the accuracy of a computed solution.

One way to do so, and to 'correct' x_0 when this is meaningful (see next paragraph), involves computing the residual vector $r = b - Ax_0$ in extended precision arithmetic, and obtaining a correction vector d by solving $PLUd = r$. The new approximate solution $x_0 + d$ is usually more accurate and the correcting process is repeated until (a) further corrections are negligible or (b) they show no further decrease.

It must be emphasised that the 'true' solution x may not be meaningful, that is correct to all figures quoted, if the elements of A and b are known with certainty only to say p figures, where p is smaller than the word-length of the computer. The first correction vector d will then give some useful information about the number of figures in the 'solution' which probably remain unchanged with respect to maximum possible uncertainties in the coefficients.

An alternative approach to assessing the accuracy of the solution is to compute or estimate the **condition number** of A, defined as

$$\kappa(A) = \|A\|\|A^{-1}\|.$$

Roughly speaking, errors or uncertainties in A or b may be amplified in the solution by a factor $\kappa(A)$. Thus, for example, if the data in A and b are only accurate to 5 digits and $\kappa(A) \approx 10^3$, then the solution cannot be guaranteed to have more than 2 correct digits. If $\kappa(A) \geq 10^5$, the solution may have no meaningful digits.

To be more precise, suppose that

$$Ax = b \quad \text{and} \quad (A + \delta A)(x + \delta x) = b + \delta b.$$

Here δA and δb represent perturbations to the matrices A and b which cause a perturbation δx in the solution. We can define measures of the relative sizes of the perturbations in A, b and x as

$$\rho_A = \frac{\|\delta A\|}{\|A\|}, \quad \rho_b = \frac{\|\delta b\|}{\|b\|} \quad \text{and} \quad \rho_x = \frac{\|\delta x\|}{\|x\|} \quad \text{respectively.}$$

Then

$$\rho_x \leq \frac{\kappa(A)}{1 - \kappa(A)\rho_A}(\rho_A + \rho_b)$$

provided that $\kappa(A)\rho_A < 1$. Often $\kappa(A)\rho_A \ll 1$ and then the bound effectively simplifies to

$$\rho_x \leq \kappa(A)(\rho_A + \rho_b).$$

Hence, if we know $\kappa(A)$, ρ_A and ρ_b, we can compute a bound on the relative errors in the solution. Note that ρ_A, ρ_b and ρ_x are defined in terms of the norms of A, b and x. If A, b or x contains elements of widely differing magnitude, then ρ_A, ρ_b and ρ_x will be dominated by the errors in the larger elements, and ρ_x will give no information about the relative accuracy of smaller elements of x.

A third way to obtain useful information about the accuracy of a solution is to solve two sets of equations, one with the given coefficients, which are assumed to be known with certainty to p figures, and one with the coefficients rounded to $(p-1)$ figures, and to count the number of figures to which the two solutions agree. In ill-conditioned problems this can be surprisingly small and even zero.

2.2 The Least-squares Solution of $Ax \simeq b$, $m > n$, rank$(A) = n$

The least-squares solution is the vector $\hat{x}$ which minimizes the sum of the squares of the residuals,

$$S = (b - A\hat{x})^T(b - A\hat{x}) = \|b - A\hat{x}\|_2^2.$$

The solution is obtained in two steps:

(i) Householder Transformations are used to reduce A to 'simpler form' via the equation $QA = R$, where R has the appearance

$$\begin{pmatrix} \hat{R} \\ \hline 0 \end{pmatrix}$$

with $\hat{R}$ a non-singular upper triangular n by n matrix and 0 a zero matrix of shape $(m-n)$ by n. Similar operations convert b to $Qb = c$, where

$$c = \begin{pmatrix} c_1 \\ \hline c_2 \end{pmatrix}$$

with c_1 having n rows and c_2 having $(m-n)$ rows.

(ii) The required least-squares solution is obtained by back-substitution in the equation

$$\hat{R}\hat{x} = c_1.$$

Again due to rounding errors the computed $\hat{x}_0$ is only an approximation to the required $\hat{x}$, but as in Section 2.1, this can be improved by 'iterative refinement'. The first correction d is the solution of the least-squares problem

$$Ad = b - A\hat{x}_0 = r$$

and since the matrix A is unchanged, this computation takes less time than that of the original $\hat{x}_0$. The process can be repeated until further corrections are (a) negligible or (b) show no further decrease.

2.3 Rank-deficient Cases

If, in the least-squares problem just discussed, $\text{rank}(A) < n$, then a least-squares solution exists but it is not unique. In this situation it is usual to ask for the least-squares solution 'of minimal length', i.e., the vector x which minimizes $\|x\|_2$, among all those x for which $\|b - Ax\|_2$ is a minimum.

This can be computed from the Singular Value Decomposition (SVD) of A, in which A is factorized as

$$A = QDP^T$$

where Q is an m by n matrix with orthonormal columns, P is an n by n orthogonal matrix and D is an n by n diagonal matrix. The diagonal elements of D are called the 'singular values' of A; they are non-negative and can be arranged in decreasing order of magnitude:

$$d_1 \geq d_2 \geq \ldots \geq d_n \geq 0.$$

The columns of Q and P are called respectively the left and right singular vectors of A. If the singular values $d_{r+1}, \ldots, d_n$ are zero or negligible, but d_r is not negligible, then the rank of A is taken to be r (see also Section 2.4) and the minimal length least-squares solution of $Ax \simeq b$ is given by

$$\hat{x} = D^\dagger Q^T b$$

where $D^\dagger$ is the diagonal matrix with diagonal elements $d_1^{-1}, d_2^{-1}, \ldots, d_r^{-1}, 0, \ldots, 0$.

The SVD may also be used to find solutions to the homogeneous system of equations $Ax = 0$, where A is m by n. Such solutions exist if and only if $\text{rank}(A) < n$, and are given by

$$x = \sum_{i=r+1}^{n} \alpha_i p_i$$

where the α_i are arbitrary numbers and the p_i are the columns of P which correspond to negligible elements of D.

The general solution to the rank-deficient least-squares problem is given by $\hat{x} + x$, where $\hat{x}$ is the minimal length least-squares solution and x is any solution of the homogeneous system of equations $Ax = 0$.

2.4 The Rank of a Matrix

In theory the rank is r if $n - r$ elements of the diagonal matrix D of the singular value decomposition are exactly zero. In practice, due to rounding and/or experimental errors, some of these elements have very small values which usually can and should be treated as zero.

For example, the following 5 by 8 matrix has rank 3 in exact arithmetic

$$\begin{pmatrix} 22 & 14 & -1 & -3 & 9 & 9 & 2 & 4 \\ 10 & 7 & 13 & -2 & 8 & 1 & -6 & 5 \\ 2 & 10 & -1 & 13 & 1 & -7 & 6 & 0 \\ 3 & 0 & -11 & -2 & -2 & 5 & 5 & -2 \\ 7 & 8 & 3 & 4 & 4 & -1 & 1 & 2 \end{pmatrix}.$$

On a computer with 7 decimal digits of precision the computed singular values were:

$$3.5 \times 10^1, \quad 2.0 \times 10^1, \quad 2.0 \times 10^1, \quad 1.3 \times 10^{-6}, \quad 5.5 \times 10^{-7}$$

and the rank would be correctly taken to be 3.

It is not, however, always certain that small computed singular values are really zero. With the 7 by 7 Hilbert matrix, for example, where $a_{ij} = 1/(i+j-1)$, the singular values are

$$1.7, \quad 2.7\times10^{-1}, \quad 2.1\times10^{-2}, \quad 1.0\times10^{-3}, \quad 2.9\times10^{-5}, \quad 4.9\times10^{-7}, \quad 3.5\times10^{-9}.$$

Here there is no clear cut-off between small (i.e., negligible) singular values and larger ones. In fact, in exact arithmetic, the matrix is known to have full rank and none of its singular values is zero. On a computer with 7 decimal digits of precision, the matrix is effectively singular, but should its rank be taken to be 6, or 5, or 4?

It is therefore impossible to give an infallible rule, but generally the rank can be taken to be the number of singular values which are neither zero nor very small compared with other singular values. For example, if there is a sharp decrease in singular values from numbers of order unity to numbers of order 10^{-7}, then the latter will almost certainly be zero in a machine in which 7 significant decimal figures is the maximum accuracy. Similarly for a least-squares problem in which the data is known to about four significant figures and the largest singular value is of order unity then a singular value of order 10^{-4} or less should almost certainly be regarded as zero.

It should be emphasised that rank determination and least-squares solutions can be sensitive to the scaling of the matrix. If at all possible the units of measurement should be chosen so that the elements of the matrix have data errors of approximately equal magnitude.

2.5 Generalized Linear Least Squares Problems

The simple type of linear least-squares problem described in Section 2.2 can be generalized in various ways.

(1) linear least-squares problems with **equality constraints**:

$$\text{find } x \text{ to minimize } \; S = \|c - Ax\|_2^2 \; \text{ subject to } \; Bx = d,$$

where A is m by n and B is p by n, with $p \le n \le m+p$. The equations $Bx = d$ may be regarded as a set of equality constraints on the problem of minimizing S. Alternatively the problem may be regarded as solving an overdetermined system of equations

$$\begin{pmatrix} A \\ B \end{pmatrix} x = \begin{pmatrix} c \\ d \end{pmatrix},$$

where some of the equations (those involving B) are to be solved exactly, and the others (those involving A) are to be solved in a least-squares sense. The problem has a unique solution on the assumptions that B has full row rank p and the matrix $\begin{pmatrix} A \\ B \end{pmatrix}$ has full column rank n. (For linear least-squares problems with **inequality constraints**, refer to Chapter E04.)

(2) **general Gauss–Markov linear model problems**:

$$\text{minimize } \; \|y\|_2 \; \text{ subject to } \; d = Ax + By,$$

where A is m by n and B is m by p, with $n \le m \le n+p$. When $B = I$, the problem reduces to an ordinary linear least-squares problem. When B is square and nonsingular, it is equivalent to a **weighted linear least-squares problem**:

$$\text{find } \; x \; \text{ to minimize } \; \|B^{-1}(d - Ax)\|_2.$$

The problem has a unique solution on the assumptions that A has full column rank n, and the matrix (A, B) has full row rank m.

2.6 Calculating the Inverse of a Matrix

The routines in this chapter can also be used to calculate the inverse of a square matrix A by solving the equation

$$AX = I$$

where I is the identity matrix. However, solving the equations $AX = B$ by calculation of the inverse of the coefficient matrix A, i.e., by $X = A^{-1}B$, is **definitely not recommended**.

Similar remarks apply to the calculation of the pseudo inverse of a singular or rectangular matrix.

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 Black Box and General Purpose Routines

Most of the routines in this chapter are categorised as Black Box routines or General Purpose routines.

Black Box routines solve the equations $Ax_i = b_i$, $i = 1, 2, \ldots, p$ in a single call with the matrix A and the right-hand sides b_i being supplied as data. These are the simplest routines to use and are suitable when all the right-hand sides are known in advance and do not occupy too much storage.

General Purpose routines, in general, require a previous call to a routine in Chapter F01 or Chapter F03 to factorize the matrix A. This factorization can then be used repeatedly to solve the equations for one or more right-hand sides which may be generated in the course of the computation. The Black Box routines simply call a factorization routine and then a general purpose routine to solve the equations.

The two routines F04MBF and F04QAF which use an iterative method for sparse systems of equations do not fit easily into this categorisation, but are classified as general purpose routines in the decision trees and indexes.

3.2 Systems of Linear Equations

Most of the routines in this chapter solve linear equations $Ax = b$ when A is n by n and a unique solution is expected (case 2.1). If this turns out to be untrue the routines go to a failure exit. The matrix A may be 'general' real or complex, or may have special structure or properties, e.g. it may be banded, tridiagonal, almost block-diagonal, sparse, symmetric, Hermitian, positive-definite (or various combinations of these).

It must be emphasised that it is a waste of computer time and space to use an inappropriate routine, for example one for the complex case when the equations are real. It is also unsatisfactory to use the special routines for a positive-definite matrix if this property is not known in advance.

Routines are given for calculating the **approximate solution**, that is solving the linear equations just once, and also for obtaining the **accurate solution** by successive iterative corrections of this first approximation, as described in Section 2.1. The latter, of course, are more costly in terms of time and storage, since each correction involves the solution of n sets of linear equations and since the original A and its LU decomposition must be stored together with the first and successively corrected approximations to the solution. In practice the storage requirements for the 'corrected' routines are about double those of the 'approximate' routines, though the extra computer time is not prohibitive since the same matrix and the same LU decomposition is used in every linear equation solution.

Two routines are provided – F04YCF for real matrices, F04ZCF for complex matrices – which can return a cheap but reliable estimate of $\|A^{-1}\|$, and hence an estimate of the **condition number** $\kappa(A)$ (see Section 2.1). These routines can be used in conjunction with most of the linear equation solving routines in this chapter: further advice is given in the routine documents.

Other routines for solving linear equation systems, computing inverse matrices, and estimating condition numbers can be found in Chapter F07, which contains LAPACK software.

3.3 Linear Least-squares Problems

For case 2.2, when $m \geq n$ and a unique least-squares solution is expected, there are two routines for a general real A, one of which (F04JGF) computes a first approximation and the other (F04AMF) computes iterative corrections. If it transpires that $\text{rank}(A) < n$, so that the least-squares solution is not unique, then F04AMF takes a failure exit, but F04JGF proceeds to compute the **minimal length** solution by using the SVD (see below).

If A is expected to be of less than full rank then one of the routines for calculating the minimal length solution may be used. Currently these routines are only for the 'approximate' solution. These routines determine the rank based upon a user-supplied tolerance to decide which elements are negligible, routines based upon the SVD providing the most reliable guide.

For $m \gg n$ the use of the SVD is not significantly more expensive than the use of routines based upon the QR factorization.

If A is complex and rank(A) = n, the problem can be solved by calling F04KMF with $p = 0$ (dummy arrays of dimension 1 must be supplied for the parameters B and D). If A is expected to be of less than full rank, the problem can be solved by calls to F02XEF (to compute the SVD of A) and F06SAF (CGEMV/ZGEMV).

Problems with linear **equality constraints** can be solved by F04JLF (for real data) or by F04KLF (for complex data), provided that the problems are of full rank. Problems with linear **inequality constraints** can be solved by E04NCF in Chapter E04.

General Gauss–Markov linear model problems, as formulated in Section 2.5, can be solved by F04JMF (for real data) or by F04KMF (for complex data).

3.4 Sparse Matrix Routines

Routines specifically for real sparse matrices should be used only when the number of non-zero elements is very small, less than, say, 10% of the n^2 elements of A, and the matrix does not have a relatively small band width.

Chapter F11 contains routines for the iterative solution of real sparse symmetric linear systems, as well as a routine F11JBF which may be used for direct solution of real sparse symmetric positive-definite problems. There are two routines in Chapter F04 for solving sparse linear equations (F04AXF and F04QAF). F04AXF utilizes a factorization of the matrix A obtained from F01BRF or F01BSF, while F04QAF uses an iterative technique and requires a user-supplied function to compute matrix-vector products Ac and A^Tc for any given vector c. F04AXF can be utilised to solve for several right-hand sides, but the original matrix has to be explicitly supplied and is overwritten by the factorization, and the storage requirements will usually be substantially more than those of the iterative routines.

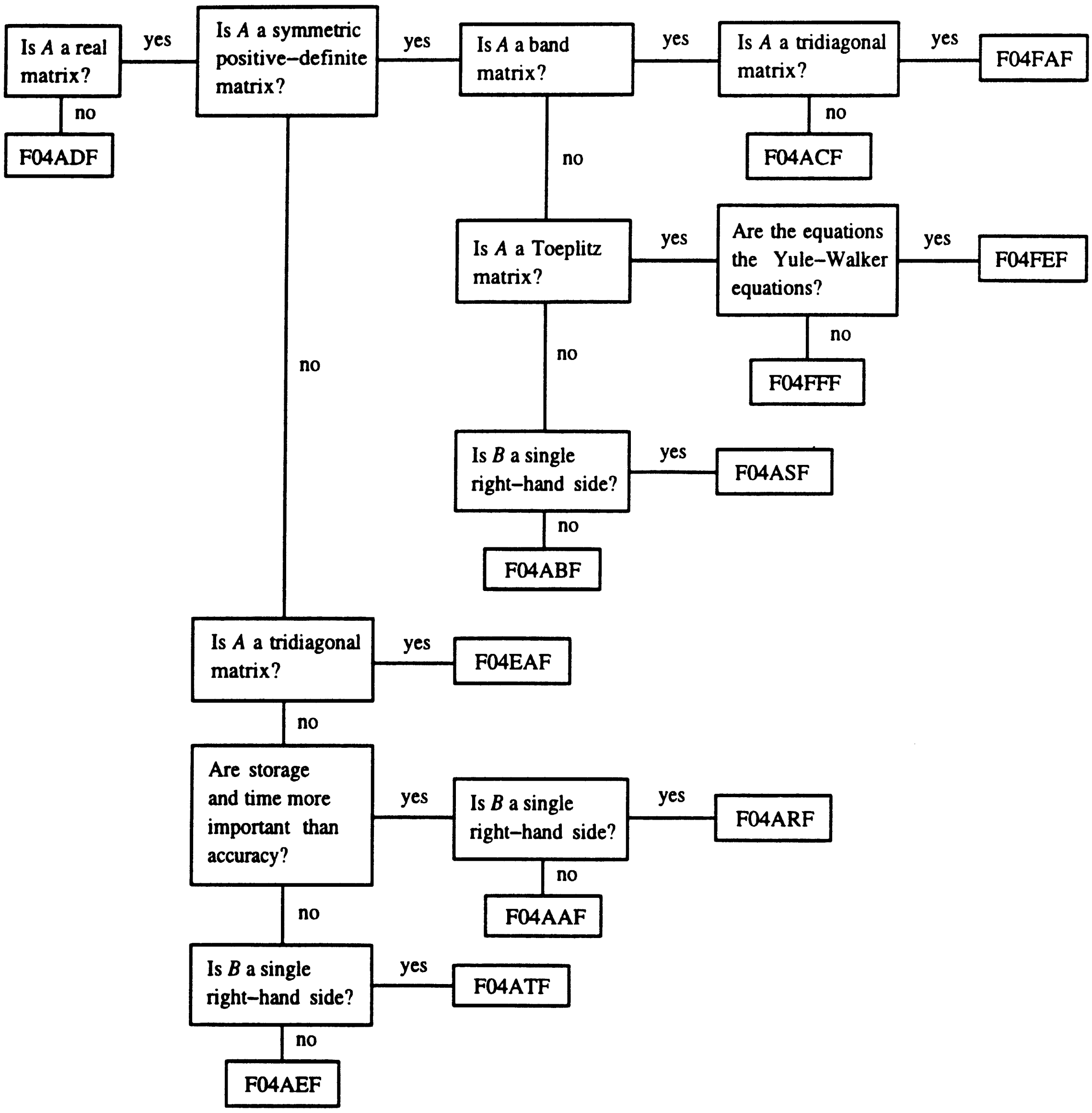
F04QAF solves sparse least-squares problems by an iterative technique, and also allows the solution of damped (regularised) least-squares problems (see the routine document for details).

4 Decision Trees

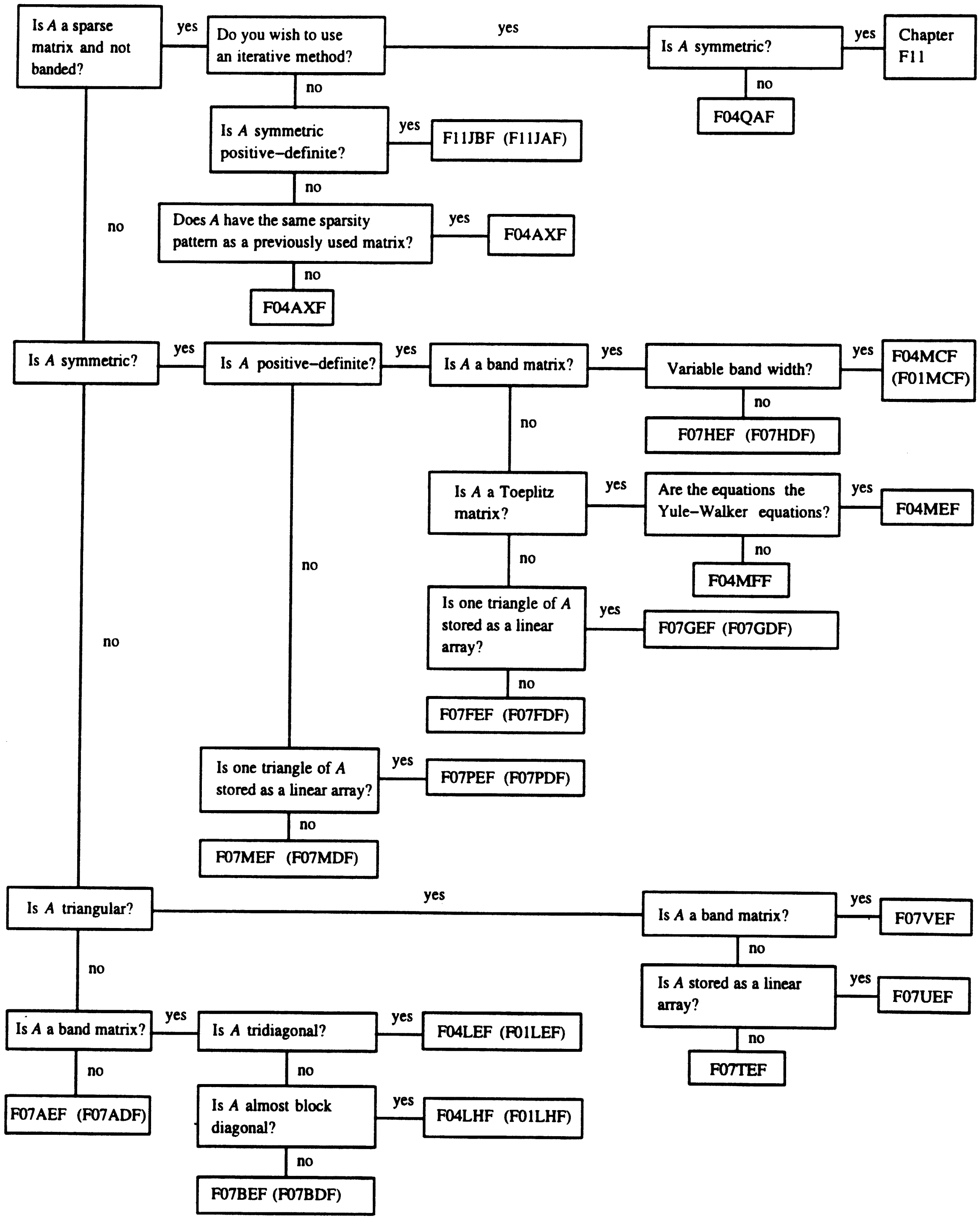
If at any stage the answer to a question is 'Don't know' this should be read as 'No'.

The name of the routine (if any) that should be used to factorize the matrix A is given in brackets after the name of the routine for solving the equations.

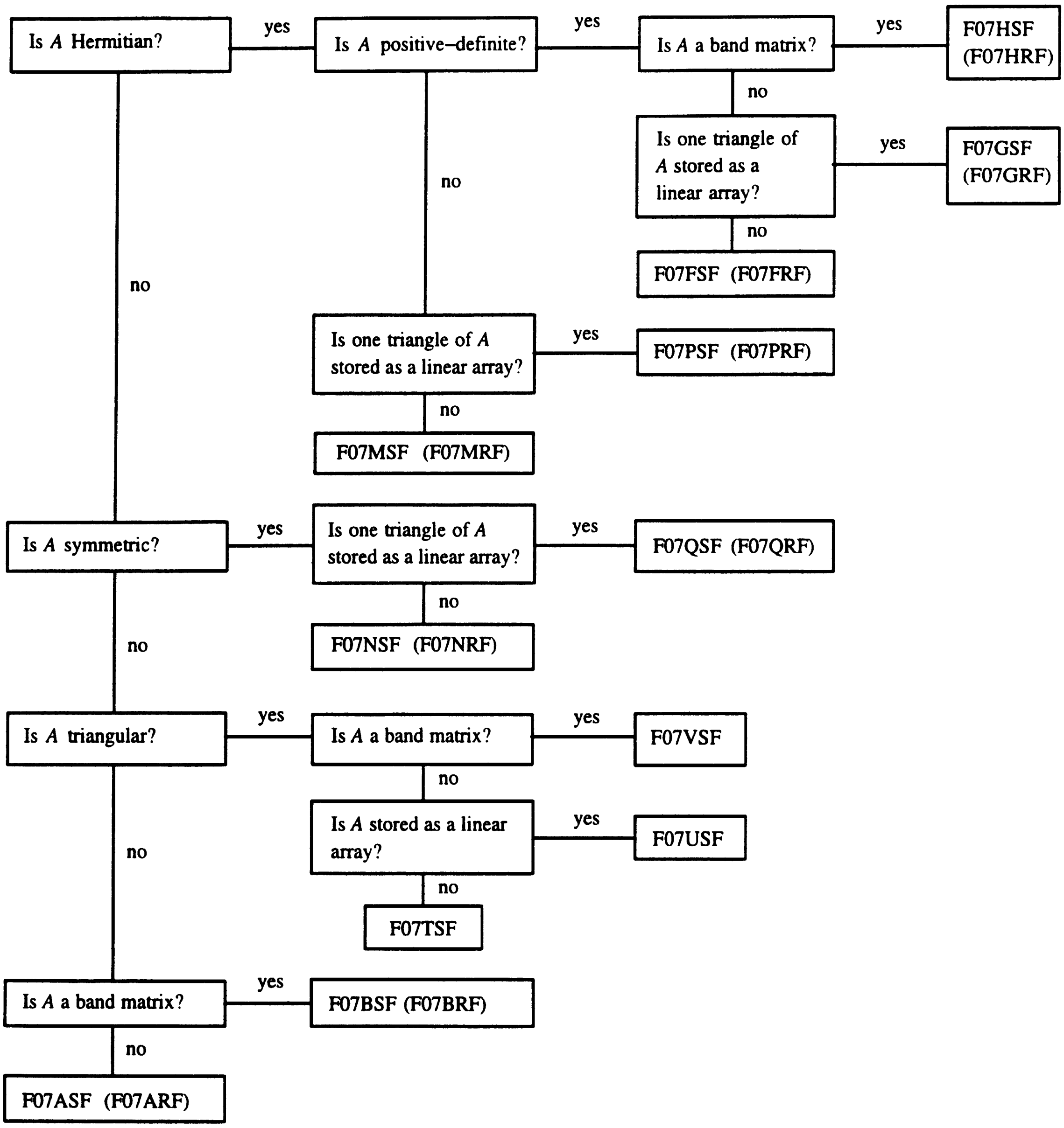
Tree 1: Black Box routines for unique solution of $Ax = b$



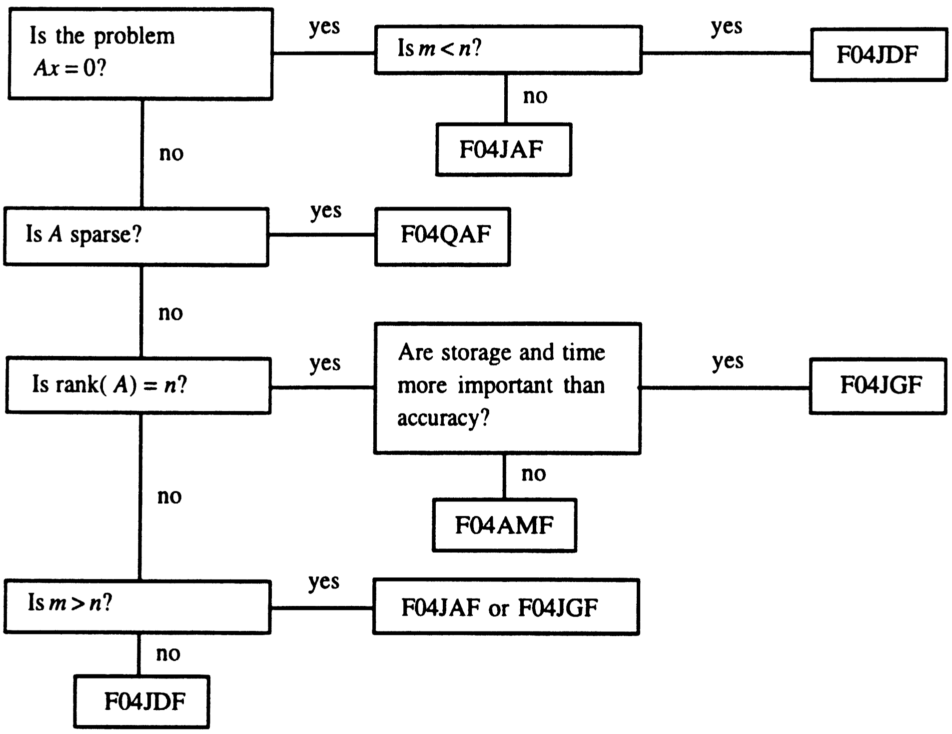
Tree 2: General Purpose routines for unique solution of $Ax = b$
(a) Real matrix



(b) Complex matrix



Tree 3: Least-squares and homogeneous equations (without constraints)



5 Index

(i) **Black Box Routines, $Ax = b$**

(ii) **General Purpose Routines, $Ax = b$**

Real Symmetric Positive-definite Toeplitz Matrix, General Right-hand Side, Update Solution,	F04MFF
Real Symmetric Positive-definite Toeplitz Matrix, Yule–Walker Equations, Update Solution,	F04MEF
Real Tridiagonal Matrix,	F04LEF

(iii) **Least-squares and Homogeneous Equations**

Complex general Gauss–Markov linear model problem	F04KLF
Complex problem with linear equality constraints	F04KMF
Real m by n Matrix, $m \geq n$, Minimal Solution,	F04JAF
Real m by n Matrix, $m \geq n$, Rank $= n$ or Minimal Solution,	F04JGF
Real m by n Matrix, $m \leq n$, Minimal Solution,	F04JDF
Real m by n Matrix, Rank $= n$, Iterative Refinement,	F04AMF
Real general Gauss–Markov linear model problem	F04JLF
Real problem with linear equality constraints	F04JMF
Real Sparse Matrix,	F04QAF

(iv) **Service Routines**

Complex Matrix, Norm and Condition Number Estimation	F04ZCF
Real Matrix, Norm and Condition Number Estimation	F04YCF

6 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

F04ALF	F04ANF	F04AQF	F04AWF	F04AYF	F04AZF
F04LDF	F04MAF	F04MBF	F04NAF		

7 References

[1] Golub G H and Van Loan C F (1996) *Matrix Computations* Johns Hopkins University Press (3rd Edition), Baltimore

[2] Lawson C L and Hanson R J (1974) *Solving Least-squares Problems* Prentice–Hall

[3] Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation II, Linear Algebra* Springer–Verlag

Chapter F05 – Orthogonalisation

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
F05AAF	5	Gram–Schmidt orthogonalisation of n vectors of order m

Chapter F05

Orthogonalisation

Contents

1 Scope of the Chapter

This chapter is concerned with the orthogonalisation of vectors in a finite dimensional space.

2 Background to the Problems

Let $a_1, a_2, \ldots, a_n$ be a set of n linearly independent vectors in m-dimensional space; $m \geq n$.

We wish to construct a set of n vectors $q_1, q_2, \ldots, q_n$ such that:

- the vectors $\{q_i\}$ form an orthonormal set, that is: $q_i^T q_j = 0$ for $i \neq j$, and $\|q_i\|_2 = 1$.
- each a_i is linearly dependent on the set $\{q_i\}$.

2.1 Gram–Schmidt Orthogonalisation

The classical Gram–Schmidt orthogonalisation process is described in many textbooks; see for example Golub and Van Loan [2], Chapter 5.

It constructs the orthonormal set progressively. Suppose it has computed orthonormal vectors $q_1, q_2, \ldots, q_k$ which orthogonalise the first k vectors $a_1, a_2, \ldots, a_k$. It then uses a_{k+1} to compute q_{k+1} as follows:

$$\begin{aligned} z_{k+1} &= a_{k+1} - \sum_{i=1}^{k} (q_i^T a_{k+1}) q_i \\ q_{k+1} &= z_{k+1} / \|z_{k+1}\|_2. \end{aligned}$$

In finite precision computation, this process can result in a set of vectors $\{q_i\}$ which are far from being orthogonal. This is caused by $\|z_{k+1}\|$ being small compared with $\|a_{k+1}\|$. If this situation is detected, it can be remedied by reorthogonalising the computed q_{k+1} against $q_1, q_2, \ldots, q_k$, that is, repeating the process with the computed q_{k+1} instead of a_{k+1}. See Daniel *et al.* [1].

2.2 Householder Orthogonalisation

An alternative approach to orthogonalizing a set of vectors is based on the QR factorization (see the F08 Chapter Introduction), which is usually performed by Householder's method. See Golub and Van Loan [2], Chapter 5.

Let A be the m by n matrix whose columns are the n vectors to be orthogonalised. The QR factorization gives:

$$A = QR$$

where R is an n by n upper triangular matrix and Q is an m by n matrix, whose columns are the required orthonormal set.

Moreover, for any k such that $1 \leq k \leq n$, the first k columns of Q are an orthonormal basis for the first k columns of A.

Householder's method requires twice as much work as the Gram–Schmidt method, provided that no reorthogonalization is required in the latter. However, it has satisfactory numerical properties and yields vectors which are close to orthogonality even when the original vectors a_i are close to being linearly dependent.

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 single routine in this chapter, F05AAF, uses the Gram–Schmidt method, with reorthogonalisation to ensure that the computed vectors are close to being exactly orthogonal. This method is only available for real vectors.

To apply Householder's method, you must use routines in Chapter F08:

for real vectors:	F08AEF, followed by F08AFF
for complex vectors:	F08ASF, followed by F08ATF

The example programs for F08AEF or F08ASF illustrate the necessary calls to these routines.

4 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'.

F05ABF

5 References

[1] Danial J W, Gragg W B, Kaufman L and Stewart G W (1976) Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization *Math. Comput.* **30** 772–795

[2] Golub G H and Van Loan C F (1989) *Matrix Computations* Johns Hopkins University Press (2nd Edition), Baltimore

Chapter F06 – Linear Algebra Support Routines

Chapter F06

Linear Algebra Support Routines

Contents

1 Scope of the Chapter

This chapter is concerned with basic linear algebra routines which perform elementary algebraic operations involving scalars, vectors and matrices.

2 Background to the Problems

A number of the routines in this chapter meet the specification of the Basic Linear Algebra Subprograms (BLAS) as described in Lawson *et al.* [7], Dodson *et al.* [2], Dongarra *et al.* [4] and [5]. The first reference describes a set of routines concerned with operations on scalars and vectors: these will be referred to here as the Level-0 and the Level-1 BLAS; the second reference describes a set of routines concerned with operations on sparse vectors: these will be referred to here as the Level-1 Sparse BLAS; the third reference describes a set of routines concerned with matrix-vector operations: these will be referred to here as the Level-2 BLAS; and the fourth reference describes a set of routines concerned with matrix-matrix operations: these will be referred to here as the Level-3 BLAS.

More generally we refer to the scalar routines in the chapter as Level-0 routines, to the vector routines as Level-1 routines, to the matrix-vector and matrix routines as Level-2 routines, and to the matrix-matrix routines as Level-3 routines. The terminology reflects the number of operations involved. For example, a Level-2 routine involves $O(n^2)$ operations for an $n \times n$ matrix.

Table 1 indicates the naming scheme for the routines in this chapter. The heading BLAS in the table indicates that routines in that category meet the specification of the BLAS, the heading 'mixed type' is for routines where a mixture of data types is involved, such as a routine that returns the real Euclidean length of a complex vector. In future marks of the Library, routines may be included in categories that are currently empty and further categories may be introduced.

		Level-0	Level-1	Level-2	Level-3
integer	F06 routine	–	F06D_F	–	–
'real'	BLAS routine	F06A_F	F06E_F	F06P_F	F06Y_F
'real'	F06 routine	F06B_F	F06F_F	F06Q_F F06R_F	–
'complex'	BLAS routine	–	F06G_F	F06S_F	F06Z_F
'complex'	F06 routine	F06C_F	F06H_F	F06T_F F06U_F	
'mixed type'	BLAS routine	–	F06J_F	–	–
'mixed type'	F06 routine	–	F06K_F	F06V_F	–

Table 1

The routines in this chapter do not have full routine documents, but instead are covered by some relevant background material, in Section 2.2, together with general descriptions, in Section 4, sufficient to enable their use. Descriptions of the individual routines are included in the NAG online documentation. As this chapter is concerned only with basic linear algebra operations, the routines will not normally be required by the general user. The functionality of each routine is indicated in Section 4 so that those users requiring these routines to build specialist linear algebra modules can determine which routines are of interest.

2.1 The Use of BLAS Names

Many of the routines in other chapters of the Library call the routines in this chapter, and in particular a number of the BLAS are called. These routines are usually called by the BLAS name and so, for correct operation of the Library, it is essential that you do not attempt to link your own versions of these routines. If you are in any doubt about how to avoid this, please consult your computer centre or the NAG Response Centre.

The BLAS names are used in order to make use of efficient implementations of the routines when these exist. Such implementations are stringently tested before being used, to ensure that they correctly meet the specification of the BLAS, and that they return the desired accuracy (see, for example, Dodson *et al.* [2], Dongarra *et al.* [4] and [5]).

2.2 Background Information

Most of the routines in this chapter implement straightforward scalar, vector and matrix operations that need no further explanation beyond a statement of the purpose of the routine. In this section we give some additional background information to those few cases where additional explanation may be necessary. A sub-section is devoted to each topic.

2.2.1 Real plane rotations

There are a number of routines in the chapter concerned with setting up and applying plane rotations. This section discusses the real case and the next section looks at the complex case. For further background information see Golub and Van Loan [6].

A plane rotation matrix for the (i,j) plane, R_{ij}, is an orthogonal matrix that is different from the unit matrix only in the elements r_{ii}, r_{jj}, r_{ij} and r_{ji}. If we put

$$R = \begin{pmatrix} r_{ii} & r_{ij} \\ r_{ji} & r_{jj} \end{pmatrix},$$

then, in the real case, it is usual to choose R_{ij} so that

$$R = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \quad c = \cos\theta, \;\; s = \sin\theta. \tag{1}$$

An exception is routine F06FPF which applies the so-called symmetric rotation for which

$$R = \begin{pmatrix} c & s \\ s & -c \end{pmatrix}. \tag{2}$$

The application of plane rotations is straightforward and needs no further elaboration, so further comment is made only on the construction of plane rotations.

The most common use of plane rotations is to choose c and s so that for given a and b,

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}. \tag{3}$$

In such an application the matrix R is often termed a **Givens rotation** matrix. There are two approaches to the construction of real Givens rotations in Chapter F06.

The BLAS routine F06AAF (SROTG/DROTG), see Lawson *et al.* [7] and Dodson and Grimes [1], computes c, s and d as

$$d = \sigma(a^2+b^2)^{1/2},$$

$$c = \begin{cases} a/d, & d \neq 0, \\ 1, & d = 0, \end{cases} \quad s = \begin{cases} b/d, & d \neq 0, \\ 0, & d = 0, \end{cases} \tag{4}$$

where $\sigma = \begin{cases} \text{sign } a, & |a| > |b| \\ \text{sign } b, & |a| \leq |b| \end{cases}$.

The value z defined as

$$z = \begin{cases} s, & |s| < c \text{ or } c = 0 \\ 1/c, & 0 < |c| \leq s \end{cases} \tag{5}$$

is also computed and this enables c and s to be reconstructed from the single value z as

$$c = \begin{cases} 0, & z = 1 \\ (1-z^2)^{1/2}, & |z| < 1 \\ 1/z, & |z| > 1 \end{cases} \quad s = \begin{cases} 1, & z = 1 \\ z, & |z| < 1 \\ (1-c^2)^{1/2}, & |z| > 1 \end{cases}$$

The other F06 routines for contructing Givens rotations are based on the computation of the tangent, $t = \tan\theta$. t is computed as

$$t = \begin{cases} 0, & b = 0 \\ b/a, & |b| \leq |a|.flmax, b \neq 0 \\ \text{sign}(b/a).flmax, & |b| > |a|.flmax \\ \text{sign}(b).flmax, & b \neq 0, a = 0 \end{cases} \tag{6}$$

where $flmax = 1/flmin$ and $flmin$ is the small positive value returned by X02AMF. The values of c and s are then computed or reconstructed via t as

$$c = \begin{cases} 1/(1+t^2)^{1/2}, & \sqrt{eps} \le |t| \le 1/\sqrt{eps} \\ 1, & |t| < \sqrt{eps} \\ 1/|t|, & |t| > 1/\sqrt{eps} \end{cases} \qquad s = \begin{cases} c.t, & \sqrt{eps} \le |t| \le 1/\sqrt{eps} \\ t, & |t| < \sqrt{eps} \\ \text{sign } t, & |t| > 1/\sqrt{eps} \end{cases} \tag{7}$$

where eps is the ***machine precision***. Note that c is always non-negative in this scheme and that the same expressions are used in the initial computation of c and s from a and b as in any subsequent recovery of c and s via t. This is the approach used by many of the NAG Fortran Library routines that require plane rotations. d is computed simply as

$$d = c.a + s.b.$$

You need not be too concerned with the above detail, since routines are provided for setting up, recovering and applying such rotations.

Another use of plane rotations is to choose c and s so that for given x, y and z

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x & y \\ y & z \end{pmatrix} \begin{pmatrix} c & -s \\ s & c \end{pmatrix} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}. \tag{8}$$

In such an application the matrix R is often termed a **Jacobi rotation** matrix. The routine that generates a Jacobi rotation (F06BEF) first computes the tangent t and then computes c and s via t as described above for the Givens rotation.

2.2.2 Complex plane rotations

In the complex case a plane rotation matrix for the (i,j) plane, R_{ij} is a unitary matrix and, analagously to the real case, it is usual to choose R_{ij} so that

$$R = \begin{pmatrix} \bar{c} & \bar{s} \\ -s & c \end{pmatrix}, \quad |c|^2 + |s|^2 = 1, \tag{9}$$

where $\bar{a}$ denotes the complex conjugate of a. The BLAS (Lawson *et al.* [7]) do not contain a routine for the generation of complex rotations, and so the routines in Chapter F06 are all based upon computing c and s via $t = b/a$ in an analagous manner to the real case. R can be chosen to have either c real, or s real and there are routines for both cases.

When c is real then it is non-negative and the transformation

$$\begin{pmatrix} c & \bar{s} \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix} \tag{10}$$

is such that if a is real then d is also real.

When s is real then the transformation

$$\begin{pmatrix} \bar{c} & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix} \tag{11}$$

is such that if b is real then d is also real.

2.2.3 Elementary real (Householder) reflections

There are a number of routines in the chapter concerned with setting up and applying Householder transformations. This section discusses the real case and the next section looks at the complex case. For further background information see Golub and Van Loan [6].

A real elementary reflector, P, is a matrix of the form

$$P = I - \mu u u^T, \qquad \mu u^T u = 2, \tag{12}$$

where μ is a scalar and u is a vector, and P is both symmetric and orthogonal. In the routines in Chapter F06, u is expressed in the form

$$u = \begin{pmatrix} \zeta \\ z \end{pmatrix}, \quad \zeta \text{ a scalar} \tag{13}$$

because in many applications ζ and z are not contiguous elements. The usual use of elementary reflectors is to choose μ and u so that for given α and x

$$P\begin{pmatrix}\alpha\\x\end{pmatrix}=\begin{pmatrix}\beta\\0\end{pmatrix},\qquad \alpha \text{ and } \beta \text{ scalars.} \tag{14}$$

Such a transformation is often termed a **Householder transformation**. There are two choices of μ and u available in Chapter F06.

The first form of the Householder transformation is compatible with that used by LINPACK (see Dongarra *et al.* [3]) and has

$$\mu = 1/\zeta. \tag{15}$$

This choice makes ζ satisfy

$$1 \le \zeta \le 2.$$

The second form, and the form used by many of the NAG Fortran Library routines, has

$$\mu = 1 \tag{16}$$

which makes

$$1 \le \zeta \le \sqrt{2}.$$

In both cases the special setting

$$\zeta = 0 \tag{17}$$

is used by the routines to flag the case where $P = I$.

Note that while there are routines to apply an elementary reflector to a vector, there are no routines available in Chapter F06 to apply an elementary reflector to a matrix. This is because such transformations can readily and efficiently be achieved by calls to the matrix-vector Level 2 BLAS routines. For example, to form PA for a given matrix

$$\begin{aligned} PA &= (I-\mu uu^T)A = A - \mu uu^T A \\ &= A - \mu ub^T, \quad b = A^T u, \end{aligned} \tag{18}$$

and so we can call a matrix-vector product routine to form $b = A^T u$ and then call a rank-one update routine to form $(A - \mu ub^T)$. Of course, we must skip the transformation when ζ has been set to zero.

2.2.4 Elementary complex (Householder) reflections

A complex elementary reflector, P, is a matrix of the form

$$P = I - \mu uu^{\mathrm{H}}, \qquad \mu u^{\mathrm{H}} u = 2, \qquad \mu \text{ real},$$

where u^{H} denotes the complex conjugate of u^T, and P is both Hermitian and unitary. For convenience in a number of applications this definition can be generalized slightly by allowing μ to be complex and so defining the generalized elementary reflector as

$$P = I - \mu uu^{\mathrm{H}}, \qquad |\mu|^2 u^H u = \mu + \bar{\mu} \tag{18}$$

for which P is still unitary, but is no longer Hermitian.

The F06 routines choose μ and ζ so that

$$\mathrm{Re}(\mu) = 1, \qquad \mathrm{Im}(\zeta) = 0 \tag{20}$$

and this reduces to (12) with the choice (16) when μ and u are real. This choice is used because μ and u can now be chosen so that in the Householder transformation (14) we can make

$$\mathrm{Im}(\beta) = 0$$

and, as in the real case,

$$1 \le \zeta \le \sqrt{2}.$$

Rather than returning μ and ζ as separate parameters the F06 routines return the single complex value θ defined as

$$\theta = \zeta + i.\,\mathrm{Im}(\mu), \quad i = \sqrt{-1}.$$

Obviously ζ and μ can be recovered as

$$\zeta = \mathrm{Re}(\theta), \; \mu = 1 + i.\,\mathrm{Im}(\theta).$$

The special setting

$$\theta = 0$$

is used to flag the case where $P = I$, and

$$\mathrm{Re}(\theta) \leq 0, \qquad \mathrm{Im}(\theta) \neq 0$$

is used to flag the case where

$$P = \begin{pmatrix} \gamma & 0 \\ 0 & I \end{pmatrix}, \qquad \gamma \text{ a scalar} \tag{21}$$

and in this case θ actually contains the value of γ. Notice that with both (18) and (21) we merely have to supply $\bar{\theta}$ rather than θ in order to represent P^H.

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.

This section lists the routines in each of the categories Level-0 (scalar), Level-1 (vector), Level-2 (matrix-vector and matrix) and Level-3 (matrix-matrix). In each case a separate sub-section is given for the routines that meet the specification of the BLAS and for the other F06 routines. For routines that meet the specification of the BLAS, the corresponding BLAS name is indicated in brackets; in single precision implementations the first of the names in the brackets is the appropriate name and in double precision implementations it is the second of the names that is appropriate.

Within each section routines are listed in alphabetic order of the fifth character in the routine name, so that corresponding real and complex routines may have adjacent entries.

3.1 The Level-0 Scalar Routines

The Level-0 routines just perform scalar operations such as generating a plane rotation.

3.1.1 The BLAS Level-0 scalar routine

F06AAF (SROTG/DROTG) generates a real plane rotation

3.1.2 The F06 Level-0 scalar routines

F06BAF generates a real plane rotation, storing the tangent
F06CAF generates a complex plane rotation, storing the tangent (real cosine)
F06CBF generates a complex plane rotation, storing the tangent (real sine)
F06BCF recovers the cosine and sine from a given real tangent
F06CCF recovers the cosine and sine from a given complex tangent (real cosine)
F06CDF recovers the cosine and sine from a given complex tangent (real sine)
F06BEF generates a real Jacobi plane rotation
F06BHF applies a real similarity rotation to a 2×2 symmetric matrix
F06CHF applies a complex similarity rotation to a 2×2 Hermitian matrix
F06BLF divides two real scalars, with an overflow flag
F06CLF divides two complex scalars, with an overflow flag
F06BMF calculates the Euclidean length of a vector following the use of routines F06FJF or F06KJF
F06BNF computes the value $(a^2 + b^2)^{1/2}$; a, b real
F06BPF computes an eigenvalue of a 2×2 real symmetric matrix

3.2 The Level-1 Vector Routines

The Level-1 routines perform operations on or between vectors, such as computing dot products and Euclidean lengths.

3.2.1 The BLAS Level-1 vector and sparse vector routines

F06EAF (SDOT/DDOT)	computes the dot product of two real vectors
F06GAF (CDOTU/ZDOTU)	computes the dot product of two complex vectors (unconjugated)
F06GBF (CDOTC/ZDOTC)	computes the dot product of two complex vectors (conjugated)
F06ECF (SAXPY/DAXPY)	adds a scalar times a vector to another real vector
F06GCF (CAXPY/ZAXPY)	adds a scalar times a vector to another complex vector
F06EDF (SSCAL/DSCAL)	multiplies a real vector by a scalar
F06GDF (CSCAL/ZSCAL)	multiplies a complex vector by a scalar
F06JDF (CSSCAL/ZDSCAL)	multiplies a complex vector by a real scalar
F06EFF (SCOPY/DCOPY)	copies a real vector
F06GFF (CCOPY/ZCOPY)	copies a complex vector
F06EGF (SSWAP/DSWAP)	swaps two real vectors
F06GGF (CSWAP/ZSWAP)	swaps two complex vectors
F06EJF (SNRM2/DNRM2)	computes the Euclidean length of a real vector
F06JJF (SCNRM2/DZNRM2)	computes the Euclidean length of a complex vector
F06EKF (SASUM/DASUM)	sums the absolute values of the elements of a real vector
F06JKF (SCASUM/DZASUM)	sums the absolute values of the elements of a complex vector
F06JLF (ISAMAX/IDAMAX)	finds the index of the element of largest absolute value of a real vector
F06JMF (ICAMAX/IZAMAX)	finds the index of the element of largest absolute value of a complex vector
F06EPF (SROT/DROT)	applies a real plane rotation
F06ERF (SDOTI/DDOTI)	computes the dot product of two real sparse vectors
F06GRF (CDOTUI/ZDOTUI)	computes the dot product of two complex sparse vectors (unconjugated)
F06GSF (CDOTCI/ZDOTCI)	computes the dot product of two complex sparse vectors (conjugated)
F06ETF (SAXPYI/DAXPYI)	adds a scalar times a sparse vector to another real sparse vector
F06GTF (CAXPYI/ZAXPYI)	adds a scalar times a sparse vector to another complex sparse vector
F06EUF (SGTHR/DGTHR)	gathers a real sparse vector
F06GUF (CGTHR/ZGTHR)	gathers a complex sparse vector
F06EVF (SGTHRZ/DGTHRZ)	gathers and sets to zero a real sparse vector
F06GVF (CGTHRZ/ZGTHRZ)	gathers and sets to zero a complex sparse vector
F06EWF (SSCTR/DSCTR)	scatters a real sparse vector
F06GWF (CSCTR/ZSCTR)	scatters a complex sparse vector
F06EXF (SROTI/DROTI)	applies a plane rotation to two real sparse vectors

3.2.2 The F06 Level-1 vector routines

F06FAF	computes the cosine of the angle between two real vectors
F06DBF	loads a scalar into each element of an integer vector
F06FBF	loads a scalar into each element of a real vector
F06HBF	loads a scalar into each element of a complex vector
F06FCF	multiplies a real vector by a diagonal matrix
F06HCF	multiplies a complex vector by a diagonal matrix

F06KCF	multiplies a complex vector by a real diagonal matrix
F06FDF	multiplies a real vector by a scalar, preserving the input vector
F06HDF	multiplies a complex vector by a scalar, preserving the input vector
F06KDF	multiplies a complex vector by a real scalar, preserving the input vector
F06DFF	copies an integer vector
F06KFF	copies a real vector to a complex vector
F06FGF	negates a real vector
F06HGF	negates a complex vector
F06FJF	updates the Euclidean length of a real vector in scaled form
F06KJF	updates the Euclidean length of a complex vector in scaled form
F06FKF	finds the weighted Euclidean length of a real vector
F06FLF	finds the elements of largest and smallest absolute value of a real vector
F06KLF	finds the last non-negligible element of a real vector
F06FPF	applies a real symmetric plane rotation
F06HPF	applies a complex plane rotation
F06KPF	applies a real plane rotation to two complex vectors
F06FQF	generates a sequence of real plane rotations
F06HQF	generates a sequence of complex plane rotations
F06FRF	generates a real elementary reflection (NAG style)
F06HRF	generates a complex elementary reflection
F06FSF	generates a real elementary reflection (LINPACK style)
F06FTF	applies a real elementary reflection (NAG style)
F06HTF	applies a complex elementary reflection
F06FUF	applies a real elementary reflection (LINPACK style)

3.3 The Level-2 Matrix-vector and Matrix Routines

The Level-2 routines perform matrix-vector and matrix operations, such as forming the product between a matrix and a vector, computing Frobenius norms and applying a sequence of plane rotations.

3.3.1 The BLAS Level-2 matrix-vector routines

F06PAF (SGEMV/DGEMV)	computes a matrix-vector product; real general matrix
F06SAF (CGEMV/ZGEMV)	computes a matrix-vector product; complex general matrix
F06PBF (SGBMV/DGBMV)	computes a matrix-vector product; real general band matrix
F06SBF (CGBMV/ZGBMV)	computes a matrix-vector product; complex general band matrix
F06PCF (SSYMV/DSYMV)	computes a matrix-vector product; real symmetric matrix
F06SCF (CHEMV/ZHEMV)	computes a matrix-vector product; complex Hermitian matrix
F06PDF (SSBMV/DSBMV)	computes a matrix-vector product; real symmetric band matrix
F06SDF (CHBMV/ZHBMV)	computes a matrix-vector product; complex Hermitian band matrix
F06PEF (SSPMV/DSPMV)	computes a matrix-vector product; real symmetric packed matrix
F06SEF (CHPMV/ZHPMV)	computes a matrix-vector product; complex Hermitian packed matrix
F06PFF (STRMV/DTRMV)	computes a matrix-vector product; real triangular matrix
F06SFF (CTRMV/ZTRMV)	computes a matrix-vector product; complex triangular matrix
F06PGF (STBMV/DTBMV)	computes a matrix-vector product; real triangular band matrix
F06SGF (CTBMV/ZTBMV)	computes a matrix-vector product; complex triangular band matrix
F06PHF (STPMV/DTPMV)	computes a matrix-vector product; real triangular packed matrix

F06SHF (CTPMV/ZTPMV)	computes a matrix-vector product; complex triangular packed matrix
F06PJF (STRSV/DTRSV)	solves a system of equations; real triangular coefficient matrix
F06SJF (CTRSV/ZTRSV)	solves a system of equations; complex triangular coefficient matrix
F06PKF (STBSV/DTBSV)	solves a system of equations; real triangular band coefficient matrix
F06SKF (CTBSV/ZTBSV)	solves a system of equations; complex triangular band coefficient matrix
F06PLF (STPSV/DTPSV)	solves a system of equations; real triangular packed coefficient matrix
F06SLF (CTPSV/ZTPSV)	solves a system of equations; complex triangular packed coefficient matrix
F06PMF (SGER/DGER)	performs a rank-one update; real general matrix
F06SMF (CGERU/ZGERU)	performs a rank-one update; complex general matrix (unconjugated vector)
F06SNF (CGERC/ZGERC)	performs a rank-one update; complex general matrix (conjugated vector)
F06PPF (SSYR/DSYR)	performs a rank-one update; real symmetric matrix
F06SPF (CHER/ZHER)	performs a rank-one update; complex Hermitian matrix
F06PQF (SSPR/DSPR)	performs a rank-one update; real symmetric packed matrix
F06SQF (CHPR/ZHPR)	performs a rank-one update; complex Hermitian packed matrix
F06PRF (SSYR2/DSYR2)	performs a rank-two update; real symmetric matrix
F06SRF (CHER2/ZHER2)	performs a rank-two update; complex Hermitian matrix
F06PSF (SSPR2/DSPR2)	performs a rank-two update; real symmetric packed matrix
F06SSF (CHPR2/ZHPR2)	performs a rank-two update; complex Hermitian packed matrix

3.3.2 The Level-2 matrix routines

F06QFF	copies a real general or trapezoidal matrix
F06TFF	copies a complex general or trapezoidal matrix
F06QHF	loads a scalar into each element of a real general or trapezoidal matrix; a different scalar may be loaded into the diagonal elements
F06THF	loads a scalar into each element of a complex general or trapezoidal matrix; a different scalar may be loaded into the diagonal elements
F06QJF	applies a sequence of permutation matrices, represented by an integer array, to a real general matrix
F06VJF	applies a sequence of permutation matrices, represented by an integer array, to a complex general matrix
F06QKF	applies a sequence of permutation matrices, represented by a real array, to a real general matrix
F06VKF	applies a sequence of permutation matrices, represented by a real array, to a complex general matrix
F06QMF	applies a sequence of plane rotations, as a similarity transformation, to a real symmetric matrix
F06TMF	applies a sequence of plane rotations, as a similarity transformation, to a complex Hermitian matrix
F06QPF	applies a rank-one update to a real upper triangular matrix, maintaining upper triangular form
F06TPF	applies a rank-one update to a complex upper triangular matrix, maintaining upper triangular form
F06QQF	performs a QR factorization of a real upper triangular matrix augmented by an additional full row

F06TQF performs a QR factorization of a complex upper triangular matrix augmented by an additional full row

F06QRF applies a sequence of plane rotations, from either the left or the right, to reduce a real upper Hessenberg matrix to upper triangular form

F06TRF applies a sequence of plane rotations, from either the left or the right, to reduce a complex upper Hessenberg matrix to upper triangular form

F06QSF applies a sequence of plane rotations, from either the left or the right, to reduce a real upper spiked matrix to upper triangular form

F06TSF applies a sequence of plane rotations, from either the left or the right, to reduce a complex upper spiked matrix to upper triangular form

F06QTF applies a given sequence of plane rotations, from either the left or the right, to a real upper triangular matrix and reduces the resulting matrix back to upper triangular form by applying plane rotations from the other side

F06TTF applies a given sequence of plane rotations, from either the left or the right, to a complex upper triangular matrix and reduces the resulting matrix back to upper triangular form by applying plane rotations from the other side

F06QVF applies a given sequence of plane rotations, from either the left or the right, to a real upper triangular matrix to give an upper Hessenberg matrix

F06TVF applies a given sequence of plane rotations, from either the left or the right, to a complex upper triangular matrix to give an upper Hessenberg matrix

F06QWF applies a given sequence of plane rotations, from either the left or the right, to a real upper triangular matrix to give an upper spiked matrix

F06TWF applies a given sequence of plane rotations, from either the left or the right, to a complex upper triangular matrix to give an upper spiked matrix

F06QXF applies a given sequence of plane rotations, from either the left or the right, to a real general matrix

F06TXF applies a given sequence of plane rotations with real cosines, from either the left or the right, to a complex general matrix

F06TYF applies a given sequence of plane rotations with real sines, from either the left or the right, to a complex general matrix

F06VXF applies a given sequence of real plane rotations, from either the left or the right, to a complex general matrix

F06RAF computes a norm, or the element of largest absolute value of a real general matrix

F06UAF computes a norm, or the element of largest absolute value of a complex general matrix

F06RBF computes a norm, or the element of largest absolute value of a real band matrix

F06UBF computes a norm, or the element of largest absolute value of a complex band matrix

F06RCF computes a norm, or the element of largest absolute value of a real symmetric matrix

F06UCF computes a norm, or the element of largest absolute value of a complex Hermitian matrix

F06RDF computes a norm, or the element of largest absolute value of a real symmetric matrix stored in packed form

F06UDF computes a norm, or the element of largest absolute value of a complex Hermitian matrix stored in packed form

F06REF computes a norm, or the element of largest absolute value of a real symmetric band matrix

F06UEF computes a norm, or the element of largest absolute value of a complex Hermitian band matrix

F06RJF computes a norm, or the element of largest absolute value of a real general trapezoidal matrix

F06UJF computes a norm, or the element of largest absolute value of a complex general trapezoidal matrix

F06RKF computes a norm, or the element of largest absolute value of a real triangular matrix stored in packed form

F06UKF	computes a norm, or the element of largest absolute value of a complex triangular matrix stored in packed form
F06RLF	computes a norm, or the element of largest absolute value of a real triangular band matrix
F06ULF	computes a norm, or the element of largest absolute value of a complex triangular band matrix
F06RMF	computes a norm, or the element of largest absolute value of a real Hessenberg matrix
F06UMF	computes a norm, or the element of largest absolute value of a complex Hessenberg matrix
F06UFF	computes a norm, or the element of largest absolute value of a complex symmetric matrix
F06UGF	computes a norm, or the element of largest absolute value of a complex symmetric matrix stored in packed form
F06UHF	computes a norm, or the element of largest absolute value of a complex symmetric band matrix

3.4 The Level-3 Matrix-matrix Routines

The Level-3 routines perform matrix-matrix operations, such as forming the product of two matrices.

3.4.1 The BLAS Level-3 matrix-matrix routines

F06YAF (SGEMM/DGEMM)	computes a matrix-matrix product; two real rectangular matrices
F06ZAF (CGEMM/ZGEMM)	computes a matrix-matrix product; two complex rectangular matrices
F06YCF (SSYMM/DSYMM)	computes a matrix-matrix product; one real symmetric matrix, one real rectangular matrix
F06ZCF (CHEMM/ZHEMM)	computes a matrix-matrix product; one complex Hermitian matrix, one complex rectangular matrix
F06YFF (STRMM/DTRMM)	computes a matrix-matrix product; one real triangular matrix, one real rectangular matrix
F06ZFF (CTRMM/ZTRMM)	computes a marix-matrix product; one complex triangular matrix, one complex rectangular matrix
F06YJF (STRSM/DTRSM)	solves a system of equations with multiple right-hand sides, real triangular coefficient matrix
F06ZJF (CTRSM/ZTRSM)	solves a system of equations with multiple right-hand sides, complex triangular coefficient matrix
F06YPF (SSYRK/DSYRK)	performs a rank-k update of a real symmetric matrix
F06ZPF (CHERK/ZHERK)	performs a rank-k update of a complex hermitian matrix
F06YRF (SSYR2K/DSYR2K)	performs a rank-$2k$ update of a real symmetric matrix
F06ZRF (CHER2K/ZHER2K)	performs a rank-$2k$ update of a complex Hermitian matrix
F06ZTF (CSYMM/ZSYMM)	computes a matrix-matrix product: one complex symmetric matrix, one complex rectangular matrix
F06ZUF (CSYRK/ZSYRK)	performs a rank-k update of a complex symmetric matrix
F06ZWF (CSYR2K/ZSYR2K)	performs a rank-$2k$ update of a complex symmetric matrix

4 Description of the F06 Routines

In this section we describe the purpose of each routine and give information on the parameter lists, where appropriate indicating their general nature. Usually the association between the routine arguments and the mathematical variables is obvious and in such cases a description of the argument is omitted.

Within each section, the parameter lists for all routines are presented, followed by the purpose of the routines and information on the parameter lists.

For those routines that meet the specification of the BLAS, the parameter lists indicate the single precision BLAS name, but this should be substituted by the double precision BLAS name in double precision implementations (see Sections 3.1–3.4).

Within each section routines are listed in alphabetic order of the fifth character in the routine name, so that corresponding real and complex routines may have adjacent entries.

4.1 The Level-0 Scalar Routines

The scalar routines have no array arguments.

4.1.1 The BLAS Level-0 scalar routine

```
SUBROUTINE      F06AAF   ( A,B,C,S )
ENTRY           srotg    ( A,B,C,S )
real                       A,B,C,S
```

F06AAF generates the parameters c and s of a Givens rotation as defined by equations (4) and (5), from given a and b. On exit, A is overwritten by d and B is overwritten by z.

4.1.2 The F06 scalar routines

```
SUBROUTINE      F06BAF   ( A,B,C,S )
real                       A,B,C,S

SUBROUTINE      F06CAF   ( A,B,C,S )
complex                    A,B,  S
real                           C

SUBROUTINE      F06CBF   ( A,B,C,S )
complex                    A,B,C
real                             S

SUBROUTINE      F06BCF   (   T,C,S )
real                         T,C,S

SUBROUTINE      F06CCF   (   T,C,S )
complex                      T,  S
real                           C

SUBROUTINE      F06CDF   (   T,C,S )
complex                      T,C
real                             S

SUBROUTINE      F06BEF   ( JOB,X,Y,Z,C,S )
CHARACTER*1                JOB
real                           X,Y,Z,C,S

SUBROUTINE      F06BHF   (     X,Y,Z,C,S )
real                           X,Y,Z,C,S

SUBROUTINE      F06CHF   (     X,Y,Z,C,S )
complex                        X,Y,Z,  S
real                                 C

real FUNCTION   F06BLF   ( A,B,FAIL )
real                       A,B
LOGICAL                        FAIL

complex FUNCTION F06CLF  ( A,B,FAIL )
complex                    A,B
LOGICAL                        FAIL

real FUNCTION   F06BMF   ( SCALE,SSQ )
real                       SCALE,SSQ

real FUNCTION   F06BNF   ( A,B )
real                       A,B

real FUNCTION   F06BPF   ( X,Y,Z )
real                       X,Y,Z
```

F06BAF, **F06CAF** and **F06CBF** generate the parameters c and s of a Givens rotation as defined by equations (6), (7) and their complex equivalents, from given a and b. On exit, A is overwritten by d and B is overwritten by t.

F06BCF, **F06CCF** and **F06CDF** recover the parameters c and s of a plane rotation from a given value of t.

F06BEF generates the parameters c and s of a Jacobi rotation from given x, y and z (see equation (8)). The input parameter JOB controls the choice of rotation as follows:

JOB = 'B', then $c \geq 1/\sqrt{2}$,

JOB = 'S', then $0 \leq c \leq 1/\sqrt{2}$,

JOB = 'M', then $|a| \geq |b|$.

On exit, a and b are overwritten on X and Z, and t is overwritten on Y.

F06BHF and **F06CHF** apply a similarity plane rotation to a two by two symmetric or Hermitian matrix defined by x, y and z. X, Y and Z are overwritten by the transformed elements.

F06BLF and **F06CLF** return the value a/b, unless overflow would occur. If overflow would occur then the value zero is returned when $a = 0$ and a value *big*, defined as follows, is returned otherwise. For F06BLF *big* is defined as

$$big = flmax.\mathrm{sign}(a/b)$$

and for F06CLF *big* is defined as

$$big = flmax.(\mathrm{sign}(\mathrm{Re}(a/b)) + i.\mathrm{sign}(\mathrm{Im}(a/b))),$$

where *flmax* is the reciprocal of the value returned by X02AMF and sign(a/b) is taken as sign(a) when $b = 0$. The argument FAIL is returned as *false* when overflow would not occur and is returned as *true* otherwise.

F06BMF returns the value $scale.\sqrt{sumsq}$. This routine is intended to be used following either of the routines F06FJF or F06KJF.

F06BNF returns the value $(a^2 + b^2)^{1/2}$, for given a and b.

F06BPF returns an eigenvalue of a two by two symmetric matrix. The eigenvalue λ is given by

$$\lambda = z - y/(f + \mathrm{sign}(f).(1 + f^2)^{1/2}), \quad \text{where } f = (x - z)/(2y).$$

When $y = 0$ then $\lambda = z$.

4.2 The Level-1 Vector Routines

The vector routines all have one or more one-dimensional arrays as arguments, each representing a vector.

In the non-sparse case the length of each vector, n, is represented by the argument N, and the routines may be called with non-positive values of N, in which case the routine returns immediately except for the functions, which set the function value to zero before returning.

In addition to the argument N, each array argument is also associated with an **increment** argument that immediately follows the array argument, and whose name consists of the three characters INC, followed by the name of the array. For example, a vector x will be represented by the two arguments X, INCX. The increment argument is the spacing (stride) in the array for which the elements of the vector occur. For instance, if INCX=2, then the elements of x are in locations X(1),X(3),. . .,X(2*N−1) of the array X and the intermediate locations X(2),X(4),. . .,X(2*N−2) are not referenced.

Thus when INCX >0, the vector element x_i is in the array element X(1+(i − 1)*INCX). When INCX $\leq$ 0 the elements are stored in the reverse order so that the vector element x_i is in the array element X(1 − (n − i) ∗ INCX) and hence, in particular, the element x_n is in X(1). The declared length of the array X in the calling (sub)program must be at least (1 + (N − 1) ∗ |INCX|).

Non-positive increments are permitted only for those routines that have more than one array argument. While zero increments are formally permitted for such routines, their use in Chapter F06 is strongly discouraged since the effect may be implementation dependent. There will usually be an alternative routine, with a simplified parameter list, to achieve the required purpose.

In the sparse case the routines are all concerned with operations on two sparse n element vectors x and y. The vector x is stored in a dense (compressed) one-dimensional array X containing only the interesting (usually non-zero) elements of x, while y is stored in full uncompressed form in an n element array Y. The vector x is represented by the three arguments NZ, X and INDX, where NZ is the number of interesting elements of x and INDX is a one-dimensional (index) array such that

$$x(\mathrm{INDX}(i)) = \mathrm{X}(i), \quad i = 1, 2, \ldots, \mathrm{NZ}.$$

The vector y is represented only by the argument Y; no increment arguments are included.

Non-positive values of NZ are permitted, in which case the routine returns immediately except for functions, which set the function value to zero before returning. For those routines where Y is an output argument **the values in the array INDX must be distinct**; violating this condition may yield incorrect results.

4.2.1 The BLAS Level-1 vector routines

```
real FUNCTION       F06EAF    ( N,        X,INCX,Y,INCY )
real                sdot
ENTRY               sdot      ( N,        X,INCX,Y,INCY )
INTEGER                         N,          INCX,INCY
real                                      X(*),  Y(*)

complex FUNCTION    F06GAF    ( N,        X,INCX,Y,INCY )
complex             cdotu
ENTRY               cdotu     ( N,        X,INCX,Y,INCY )
INTEGER                         N,          INCX,  INCY
complex                                   X(*),  Y(*)

complex FUNCTION    F06GBF    ( N,        X,INCX,Y,INCY )
complex             cdotc
ENTRY               cdotc     ( N,        X,INCX,Y,INCY )
INTEGER                         N,          INCX,  INCY
complex                                   X(*),  Y(*)

SUBROUTINE          F06ECF    ( N,ALPHA, X,INCX,Y,INCY )
ENTRY               saxpy     ( N,ALPHA, X,INCX,Y,INCY )
INTEGER                         N,          INCX,  INCY
real                              ALPHA, X(*),  Y(*)

SUBROUTINE          F06GCF    ( N,ALPHA, X,INCX,Y,INCY )
ENTRY               caxpy     ( N,ALPHA, X,INCX,Y,INCY )
INTEGER                         N,          INCX,  INCY
complex                           ALPHA, X(*),  Y(*)

SUBROUTINE          F06EDF    ( N,ALPHA, X,INCX )
ENTRY               sscal     ( N,ALPHA, X,INCX )
INTEGER                         N,         INCX
real                              ALPHA, X(*)

SUBROUTINE          F06GDF    ( N,ALPHA, X,INCX )
ENTRY               cscal     ( N,ALPHA, X,INCX )
INTEGER                         N,         INCX
complex                           ALPHA, X(*)

SUBROUTINE          F06JDF    ( N,ALPHA, X,INCX )
ENTRY               csscal    ( N,ALPHA, X,INCX )
INTEGER                         N,         INCX
real                              ALPHA
complex                                  X(*)
```

```
SUBROUTINE         F06EFF     ( N,     X,INCX,Y,INCY )
ENTRY              scopy      ( N,     X,INCX,Y,INCY )
INTEGER                         N,       INCX,  INCY
real                                   X(*),  Y(*)

SUBROUTINE         F06GFF     ( N,     X,INCX,Y,INCY )
ENTRY              ccopy      ( N,     X,INCX,Y,INCY )
INTEGER                         N,       INCX,  INCY
complex                                X(*),  Y(*)

SUBROUTINE         F06EGF     ( N,     X,INCX,Y,INCY )
ENTRY              sswap      ( N,     X,INCX,Y,INCY )
INTEGER                         N,       INCX,  INCY
real                                   X(*),  Y(*)

SUBROUTINE         F06GGF     ( N,     X,INCX,Y,INCY )
ENTRY              cswap      ( N,     X,INCX,Y,INCY )
INTEGER                         N,       INCX,  INCY
complex                                X(*),  Y(*)

real FUNCTION      F06EJF     ( N,     X,INCX )
real               snrm2
ENTRY              snrm2      ( N,     X,INCX )
INTEGER                         N,       INCX
real                                   X(*)

real FUNCTION      F06JJF     ( N,     X,INCX )
real               scnrm2
ENTRY              scnrm2     ( N,     X,INCX )
INTEGER                         N,       INCX
complex                                X(*)

real FUNCTION      F06EKF     ( N,     X,INCX )
real               sasum
ENTRY              sasum      ( N,     X,INCX )
INTEGER                         N,       INCX
real                                   X(*)

real FUNCTION      F06JKF     ( N,     X,INCX )
real               scasum
ENTRY              scasum     ( N,     X,INCX )
INTEGER                         N,       INCX
complex                                X(*)

INTEGER FUNCTION   F06JLF     ( N,     X,INCX )
INTEGER            isamax
ENTRY              isamax     ( N,     X,INCX )
INTEGER                         N,       INCX
real                                   X(*)

INTEGER FUNCTION   F06JMF     ( N,     X,INCX )
INTEGER            icamax
ENTRY              icamax     ( N,     X,INCX )
INTEGER                         N,       INCX
complex                                X(*)

SUBROUTINE         F06EPF     ( N,     X,INCX,Y,INCY,C,S )
ENTRY              srot       ( N,     X,INCX,Y,INCY,C,S )
INTEGER                         N,       INCX,  INCY
real                                   X(*),  Y(*),  C,S
```

```
real FUNCTION        F06ERF    ( NZ,      X,INDX,Y )
real                 sdoti
ENTRY                sdoti     ( NZ,      X,INDX,Y )
INTEGER                          NZ,        INDX(*)
real                                      X(*),  Y(*)

complex FUNCTION     F06GRF    ( NZ,      X,INDX,Y )
complex              cdotui
ENTRY                cdotui    ( NZ,      X,INDX,Y )
INTEGER                          NZ,        INDX(*)
complex                                   X(*),  Y(*)

complex FUNCTION     F06GSF    ( NZ,      X,INDX,Y )
complex              cdotci
ENTRY                cdotci    ( NZ,      X,INDX,Y )
INTEGER                          NZ,        INDX(*)
complex                                   X(*),  Y(*)

SUBROUTINE           F06ETF    ( NZ,ALPHA,X,INDX,Y )
ENTRY                saxpyi    ( NZ,ALPHA,X,INDX,Y )
INTEGER                          NZ,        INDX(*)
real                                ALPHA,X(*),  Y(*)

SUBROUTINE           F06GTF    ( NZ,ALPHA,X,INDX,Y )
ENTRY                caxpyi    ( NZ,ALPHA,X,INDX,Y )
INTEGER                          NZ,        INDX(*)
complex                             ALPHA,X(*),  Y(*)

SUBROUTINE           F06EUF    ( NZ, Y,   X,INDX )
ENTRY                sgthr     ( NZ, Y,   X,INDX )
INTEGER                          NZ,        INDX(*)
real                                 Y(*),X(*)

SUBROUTINE           F06GUF    ( NZ, Y,   X,INDX )
ENTRY                cgthr     ( NZ, Y,   X,INDX )
INTEGER                          NZ,        INDX(*)
complex                              Y(*),X(*)

SUBROUTINE           F06EVF    ( NZ, Y,   X,INDX )
ENTRY                sgthrz    ( NZ, Y,   X,INDX )
INTEGER                          NZ,        INDX(*)
real                                 Y(*),X(*)

SUBROUTINE           F06GVF    ( NZ, Y,   X,INDX )
ENTRY                cgthrz    ( NZ, Y,   X,INDX )
INTEGER                          NZ,        INDX(*)
complex                              Y(*),X(*)

SUBROUTINE           F06EWF    ( NZ,      X,INDX,Y )
ENTRY                ssctr     ( NZ,      X,INDX,Y )
INTEGER                          NZ,        INDX(*)
real                                      X(*),  Y(*)

SUBROUTINE           F06GWF    ( NZ,      X,INDX,Y )
ENTRY                csctr     ( NZ,      X,INDX,Y )
INTEGER                          NZ,        INDX(*)
complex                                   X(*),  Y(*)

SUBROUTINE           F06EXF    ( NZ,      X,INDX,Y, C,S )
ENTRY                sroti     ( NZ,      X,INDX,Y, C,S )
INTEGER                          NZ,        INDX(*)
real                                      X(*),Y(*),C,S
```

F06EAF, **F06GAF**, **F06ERF** and **F06GRF** return the dot product $x^T y$.

F06GBF and **F06GSF** return the dot product $x^H y$, where x^H denotes the complex conjugate of x^T.

F06ECF, **F06GCF**, **F06ETF** and **F06GTF** perform the operation

$$y \leftarrow \alpha x + y,$$

often called an **axpy** operation.

F06EDF, **F06GDF** and **F06JDF** perform the operation

$$x \leftarrow \alpha x.$$

F06EFF, **F06GFF**, **F06EWF** and **F06GWF** perform the operation

$$y \leftarrow x.$$

F06EGF and **F06GGF** perform the operation

$$x \Leftrightarrow y,$$

that is x and y are swapped.

F06EJF and **F06JJF** return the value $\|x\|_2$ defined by

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}.$$

F06EKF returns the value $\|x\|_1$ defined by

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|.$$

F06JKF returns the value *asum* defined by

$$asum = \sum_{i=1}^{n} (|\operatorname{Re}(x_i)| + |\operatorname{Im}(x_i)|).$$

F06JLF returns the first index j such that

$$|x_j| = \max_i |x_i|.$$

F06JMF returns the first index j such that

$$|\operatorname{Re}(x_j)| + |\operatorname{Im}(x_j)| = \max_i (|\operatorname{Re}(x_i)| + |\operatorname{Im}(x_i)|).$$

F06EPF and **F06EXF** performs the plane rotation

$$\begin{pmatrix} x^T \\ y^T \end{pmatrix} \leftarrow \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x^T \\ y^T \end{pmatrix}.$$

F06EUF and **F06GUF** perform the operation

$$x \leftarrow y.$$

F06EVF and **F06GVF** perform the operations

$$\begin{aligned} x &\leftarrow y \\ y &\leftarrow 0. \end{aligned}$$

4.2.2 The F06 Level-1 vector routines

```
real FUNCTION       F06FAF    ( N,J,TOLX,X,  INCX,TOLY,Y,INCY )
INTEGER                         N,J,         INCX,      INCY
real                                TOLX,X(*),     TOLY,Y(*)

SUBROUTINE          F06DBF    ( N,CONST, X,  INCX )
INTEGER                         N,           INCX
INTEGER                           CONST, X(*)

SUBROUTINE          F06FBF    ( N,CONST, X,  INCX )
INTEGER                         N,           INCX
real                              CONST, X(*)

SUBROUTINE          F06HBF    ( N,CONST, X,  INCX )
INTEGER                         N,           INCX
complex                           CONST, X(*)

SUBROUTINE          F06FCF    ( N,D,INCD,X,  INCX )
INTEGER                         N, INCD,     INCX
real                              D(*),  X(*)

SUBROUTINE          F06HCF    ( N,D,INCD,X,  INCX)
INTEGER                         N, INCD,     INCX
complex                           D(*),  X(*)

SUBROUTINE          F06KCF    ( N,D,INCD,X,  INCX )
INTEGER                         N, INCD,     INCX
real                              D(*)
complex                                  X(*)

SUBROUTINE          F06FDF    ( N,ALPHA, X,  INCX,Y,INCY )
INTEGER                         N,           INCX, INCY
real                              ALPHA, X(*),     Y(*)

SUBROUTINE          F06HDF    ( N,ALPHA, X,  INCX,Y,INCY )
INTEGER                         N,           INCX, INCY
complex                           ALPHA, X(*),     Y(*)

SUBROUTINE          F06KDF    ( N,ALPHA, X,  INCX,Y,INCY )
INTEGER                         N,           INCX, INCY
real                              ALPHA
complex                                  X(*),     Y(*)

SUBROUTINE          F06DFF    ( N,       X,  INCX,Y,INCY )
INTEGER                         N,           INCX, INCY
INTEGER                                  X(*),     Y(*)

SUBROUTINE          F06KFF    ( N,       X,  INCX,Y,INCY )
INTEGER                         N,           INCX, INCY
real                                     X(*)
complex                                            Y(*)

SUBROUTINE          F06FGF    ( N,       X,  INCX )
INTEGER                         N,           INCX
real                                     X(*)

SUBROUTINE          F06HGF    ( N,       X,  INCX )
INTEGER                         N,           INCX
complex                                  X(*)
```

```
SUBROUTINE          F06FJF   ( N,        X,  INCX,SCALE,SUMSQ )
INTEGER                        N,            INCX
real                                     X(*),    SCALE,SUMSQ

SUBROUTINE          F06KJF   ( N,        X,  INCX,SCALE,SUMSQ )
INTEGER                        N,            INCX
complex                                  X(*)
real                                              SCALE,SUMSQ

real FUNCTION       F06FKF   ( N,D,INCD,X,  INCX )
INTEGER                        N, INCD,     INCX
real                             D(*), X(*)

SUBROUTINE          F06FLF   ( N,        X,  INCX,XMAX,XMIN )
INTEGER                        N,            INCX
real                                     X(*),    XMAX,XMIN

INTEGER FUNCTION    F06KLF   ( N,        X,  INCX,TOL )
INTEGER                        N,            INCX
real                                     X(*),    TOL

SUBROUTINE          F06FPF   ( N,        X,  INCX,Y,INCY,C,S )
INTEGER                        N,            INCX, INCY
real                                     X(*),    Y(*), C,S

SUBROUTINE          F06HPF   ( N,        X,  INCX,Y,INCY,C,S )
INTEGER                        N,            INCX, INCY
complex                                  X(*),    Y(*), C,S

SUBROUTINE          F06KPF   ( N,        X,  INCX,Y,INCY,C,S )
INTEGER                        N,            INCX, INCY
complex                                  X(*),    Y(*)
real                                                    C,S

SUBROUTINE          F06FQF   ( PIVOT,DIRECT,N,ALPHA,X,INCX,C,   S )
CHARACTER*1                    PIVOT,DIRECT
INTEGER                                   N,        INCX
real                                        ALPHA,X(*),  C(*),S(*)

SUBROUTINE          F06HQF   ( PIVOT,DIRECT,N,ALPHA,X,INCX,C,   S )
CHARACTER*1                    PIVOT,DIRECT
INTEGER                                   N,        INCX
complex                                     ALPHA,X(*),       S(*)
real                                                     C(*)

SUBROUTINE          F06FRF   ( N,ALPHA,X,INCX,TOL,ZETA )
INTEGER                        N,       INCX
real                              ALPHA,X(*),  TOL,ZETA

SUBROUTINE          F06HRF   ( N,ALPHA,X,INCX,TOL,THETA )
INTEGER                        N,       INCX
complex                          ALPHA,X(*),       THETA
real                                           TOL

SUBROUTINE          F06FSF   ( N,ALPHA,X,INCX,TOL,Z1 )
INTEGER                        N,       INCX
real                             ALPHA,X(*),  TOL,Z1

SUBROUTINE          F06FTF   ( N,DELTA,Y,INCY,ZETA, Z,INCZ )
INTEGER                        N,        INCY,        INCZ
real                             DELTA,Y(*),  ZETA, Z(*)

SUBROUTINE          F06HTF   ( N,DELTA,Y,INCY,THETA,Z,INCZ )
INTEGER                        N,        INCY,        INCZ
complex                          DELTA,Y(*),  THETA,Z(*)
```

```
SUBROUTINE      F06FUF   ( N,Z,INCZ,ZETA,DELTA,Y,INCY )
INTEGER                    N, INCZ,            INCY
real                         Z(*), ZETA,DELTA,Y(*)
```

F06FAF returns the value of the cosine of the angle between the vectors x and y, defined as

$$\text{F06FAF} = \frac{x^T y}{\|x\|_2 \|y\|_2},$$

where $\|x\|_2 = \sqrt{x^T x}$. If the input argument J is such that $1 \le \text{J} \le \text{N}$ then y is taken as

$$y = e_{\text{J}}, 1 \le \text{J} \le \text{N},$$

where e_{J} is the Jth column of the unit matrix and in this case Y is not referenced. If $\|x\|_2 \le \text{TOLX}$ then F06FAF is returned as 2.0 and if $\|x\|_2 \le \text{TOLY}$ then F06FAF is returned as -2.0, otherwise F06FAF is returned in the range $[-1.0, 1.0]$. If either TOLX or TOLY are negative then zero is used in place of the respective tolerance.

F06DBF, **F06FBF** and **F06HBF** perform the operation

$$x \leftarrow \alpha e,$$

where e is the vector $e^T = (11\ldots 1)$.

F06FCF, **F06HCF** and **F06KCF** perform the operation

$$x \leftarrow Dx,$$

where D is a diagonal matrix, $D = \text{diag}(d_i)$.

F06FDF, **F06HDF** and **F06KDF** perform the operation

$$y \leftarrow \alpha x.$$

F06DFF and **F06KFF** perform the operation

$$y \leftarrow x.$$

F06FGF and **F06HGF** perform the operation

$$x \leftarrow -x.$$

F06FJF and **F06KJF** return the values *scl* and *ssq* given by

$$scl^2.ssq = scale^2.sumsq + \|x\|_2,$$

where for F06FJF,

$$\|x\|_2^2 = x^T x = x_1^2 + x_2^2 + \ldots + x_n^2$$

and for F06KJF,

$$\|x\|_2^2 = x^H x = |x_1|^2 + |x_2|^2 + \ldots + |x_n|^2.$$

The values of *scl* and *ssq* are overwritten on *scale* and *sumsq*, and either of these routines can be followed by routine F06BMF to compute the value $scale.\sqrt{sumsq}$. These routines are intended for the safe computation of the Euclidean lengths of vectors and matrices. Before entry, *scale* and *sumsq* are assumed to satisfy

$$0 \le scale, \quad 1 \le sumsq.$$

On exit from F06FJF, *scl* and *ssq* will then satisfy,

$$scl = \max_i(scale, |x_i|), \quad 1 \le ssq \le sumsq + n$$

and from F06KJF,

$$scl = \max_i(scale, |\operatorname{Re}(x_i)|, |\operatorname{Im}(x_i)|), \quad 1 \le ssq \le sumsq + 2n.$$

F06FKF returns the weighted Euclidean length $\|x_D\|$ defined as

$$\|x_D\| = \|D^{1/2}x\|_2 = \left(\sum_{i=1}^{n} d_i x_i^2\right)^{1/2}$$

where D is the diagonal matrix $D = \mathrm{diag}(d_i)$. The elements of D must satisfy $d_i \geq 0$.

F06FLF returns the values *xmax* and *xmin* given by

$$xmax = \max_i |x_i|, \quad xmin = \min_i |x_i|.$$

F06KLF returns the value $(k-1)$, where k is the smallest integer for which

$$|x_k| \leq tol.\max(|x_1|, |x_2|, \ldots, |x_{k-1}|).$$

If no such k exists then F06KLF returns the value n. If *tol* is less than zero on entry, then the value *eps*, where *eps* is the ***machine precision***, is used in place of *tol*. Note that *tol* is unchanged on exit. Note also that k is the index of the first negligible element in x and that $k = 1$ only if $x_1 = 0$.

F06FPF performs the symmetric plane rotation

$$\begin{pmatrix} x^T \\ y^T \end{pmatrix} \leftarrow \begin{pmatrix} c & s \\ s & -c \end{pmatrix} \begin{pmatrix} x^T \\ y^T \end{pmatrix}.$$

F06HPF performs the plane rotation

$$\begin{pmatrix} x^T \\ y^T \end{pmatrix} \leftarrow \begin{pmatrix} c & s \\ -\bar{s} & \bar{c} \end{pmatrix} \begin{pmatrix} x^T \\ y^T \end{pmatrix}.$$

Note that this differs slightly from the form given in equation (9).

F06KPF performs the plane rotation

$$\begin{pmatrix} x^T \\ y^T \end{pmatrix} \leftarrow \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x^T \\ y^T \end{pmatrix}.$$

where x and y are complex, but c and s are real.

F06FQF and **F06HQF** generate the parameters of a sequence of plane rotations. Denoting the product of the plane rotation matrices by P, the matrix P is such that

$$\begin{pmatrix} \alpha \\ 0 \end{pmatrix} \leftarrow P \begin{pmatrix} \alpha \\ x \end{pmatrix},$$

when PIVOT = 'F' and DIRECT = 'F', or when PIVOT = 'V' and DIRECT = 'B', and

$$\begin{pmatrix} 0 \\ \alpha \end{pmatrix} \leftarrow P \begin{pmatrix} x \\ \alpha \end{pmatrix},$$

when PIVOT = 'F' and DIRECT = 'B', or when PIVOT = 'V' and DIRECT = 'F'.

When PIVOT = 'F' (Fixed pivot) and DIRECT = 'F' (Forward sequence) then P is given as the sequence

$$P = P_n, P_{n-1}, \ldots, P_1,$$

where P_k is a plane rotation matrix for the $(1, k+1)$ plane designed to annihilate the kth element of x.

When PIVOT = 'V' (Variable pivot) and DIRECT = 'B' or 'b' (Backward sequence) then P is given as the sequence

$$P = P_1, P_2, \ldots, P_n,$$

where P_k is a plane rotation matrix for the $(k, k+1)$ plane designed to annihilate the kth element of x.

When PIVOT = 'F' and DIRECT = 'B' then P is given as the sequence

$$P = P_1, P_2, \ldots, P_n,$$

where P_k is a plane rotation matrix of the $(k, n-1)$ plane designed to annihilate the kth element of x.

When PIVOT = 'V' and DIRECT = 'F' then P is given as the sequence

$$P = P_n, P_{n-1}, \ldots, P_1,$$

where P_k is a plane rotation matrix of the $(k, k+1)$ plane designed to annihilate the kth element of x.

The two by two plane rotation part of P_k has the form given by equation (1) for F06FQF and (10) (with c real) for F06HQF. The cosine and sine that define P_k are returned in C(k) and S(k) respectively and the tangent, $t_k = \mathrm{S}(k)/\mathrm{C}(k)$, is overwritten on the element of X corresponding to x_k.

Note that for routine F06HQF, if the imaginary part of α is supplied as zero, then the imaginary part of α will also be zero on return.

F06HRF generates the parameters θ and z of a complex Householder transformation as described in Section 2.2.4. The elements of z are overwritten on x and β is overwritten on α. Note that $\mathrm{Im}(\beta) = 0$. If x is such that

$$\max_i(|\mathrm{Re}(x_i)|, |\mathrm{Im}(x_i)|) \leq \max(eps.\max(|\mathrm{Re}(\alpha)|, |\mathrm{Im}(\alpha)|), tol),$$

where *eps* is the ***machine precision***, then θ is returned such that $\mathrm{Re}(\theta) \leq 0$, as described at the end of Section 2.2.4, otherwise θ is such that

$$\theta = \zeta + i.\mathrm{Im}(\mu), \quad i = \sqrt{-1}$$

with

$$1 \leq \zeta \leq \sqrt{2}.$$

F06FSF generates the parameters ζ and z of a real Householder transformation of the form (14), where μ satisfies (15). The elements of z are overwritten on x and β is overwritten on α. If the elements of x are all zero or are all less than $tol.|\alpha|$, then ζ is returned as zero, otherwise ζ satisfies

$$1 \leq \zeta \leq 2.$$

If *tol* is outside the range [0, 1], then the value zero is used in place of *tol*, but *tol* is unchanged on exit.

F06FRF generates the parameters ζ and z of a real Householder transformation of the form (14), where μ satisfies (16). The elements of z are overwritten on x and β is overwritten on α. If the elements x satisfy

$$\max_i |x_i| \leq \max(eps.|\alpha|, tol),$$

where *eps* is the ***machine precision***, then ζ is returned as zero, otherwise ζ satisfies

$$1 \leq \zeta \leq \sqrt{2}.$$

F06FUF, **F06FTF** and **F06HTF** perform elementary reflections given by

$$\begin{pmatrix} \delta \\ y \end{pmatrix} \leftarrow P \begin{pmatrix} \delta \\ y \end{pmatrix},$$

where P is an elementary reflector. F06FUF is intended for use in conjunction with routine F06FSF, and F06FTF and F06HTF are intended for use in conjunction with routines F06FRF and F06HRF respectively. Note that F06HTF can be used to perform the transformation

$$\begin{pmatrix} \delta \\ y \end{pmatrix} \leftarrow P^H \begin{pmatrix} \delta \\ y \end{pmatrix},$$

by calling F06HTF with CONJG(THETA) in place of THETA.

4.3 The Level-2 Matrix-vector Routines

The matrix-vector routines all have one array argument representing a matrix; usually this is a two-dimensional array but in some cases the matrix is represented by a one-dimensional array.

The size of the matrix is determined by the arguments M and N for an m by n rectangular matrix; and by the argument N for an n by n symmetric, Hermitian, or triangular matrix. Note that it is permissible to call the routines with M or N = 0, in which case the routines exit immediately without referencing their array arguments. For band matrices, the bandwidth is determined by the arguments KL and KU for a rectangular matrix with kl sub-diagonals and ku super-diagonals; and by the argument K for a symmetric, Hermitian, or triangular matrix with k sub-diagonals and/or super-diagonals.

The description of the matrix consists either of the array name (A) followed by the first dimension of the array as declared in the calling (sub)program (LDA), when the matrix is being stored in a two-dimensional array; or the array name (AP) alone when the matrix is being stored as a (packed) vector. In the former case the actual array must contain at least $((n-1)d+l)$ elements, where d is the first dimension of the array, $d \geq l$, and $l = m$ for arrays representing general matrices, $l = n$ for arrays representing symmetric, Hermitian and triangular matrices, $l = kl + ku + 1$ for arrays representing general band matrices and $l = k+1$ for symmetric, Hermitian and triangular band matrices. For one-dimensional arrays representing matrices (**packed storage**) the actual array must contain at least $\frac{1}{2}n(n+1)$ elements.

You may wish to be aware that Chapter F01 provides some utility routines for conversion between storage formats. (See Section 3.3.)

As with the vector routines, vectors are represented by one-dimensonal arrays together with a corresponding increment argument (see Section 4.2). The only difference is that for these routines a zero increment is not permitted.

When the vector x consists of k elements then the declared length of the array X in the calling (sub)program must be at least $(1+(k-1)*|\mathrm{INCX}|)$.

The arguments that specify options are character arguments with the names TRANS, UPLO and DIAG. TRANS is used by the matrix-vector product routines as follows:

Value	**Meaning**
'N'	Operate with the matrix
'T'	Operate with the transpose of the matrix
'C'	Operate with the conjugate transpose of the matrix

In the real case the values 'T', 't', 'C' and 'c' have the same meaning.

UPLO is used by the Hermitian, symmetric, and triangular matrix routines to specify whether the upper or lower triangle is being referenced as follows:

Value	**Meaning**
'U'	Upper triangle
'L'	Lower triangle

DIAG is used by the triangular matrix routines to specify whether or not the matrix is unit triangular, as follows:

Value	**Meaning**
'U'	Unit triangular
'N'	Non-unit triangular

When DIAG is supplied as 'U' the diagonal elements are not referenced.

It is worth noting that actual character arguments in Fortran may be longer than the corresponding dummy arguments. So that, for example, the value 'T' for TRANS may be passed as 'TRANSPOSE'.

The routines for real symmetric and complex Hermitian matrices allow for the matrix to be stored in either the upper (UPLO = 'U') or lower triangle (UPLO = 'L') of a two-dimensional array, or to be packed in a one-dimensional array. In the latter case the upper triangle may be packed sequentially column by column (UPLO = 'U'), or the lower triangle may be packed sequentially column by column

(UPLO = 'L'). Note that for real symmetric matrices packing the upper triangle by column is equivalent to packing the lower triangle by rows, and packing the lower triangle by columns is equivalent to packing the upper triangle by rows. (For complex Hermitian matrices the only difference is that the off-diagonal elements are conjugated.)

For triangular matrices the argument UPLO serves to define whether the matrix is upper (UPLO = 'U') or lower (UPLO = 'L') triangular. In packed storage the triangle has to be packed by column.

The band matrix routines allow storage so that the jth column of the matrix is stored in the jth column of the Fortran array. For a general band matrix the diagonal of the matrix is stored in the $(ku+1)$th row of the array. For a Hermitian or symmetric matrix either the upper triangle (UPLO = 'U') may be stored in which case the leading diagonal is in the $(k+1)$th row of the array, or the lower triangle (UPLO = 'L') may be stored in which case the leading diagonal is in the first row of the array. For an upper triangular band matrix (UPLO = 'U') the leading diagonal is in the $(k+1)$th row of the array and for a lower triangular band matrix (UPLO = 'L') the leading diagonal is in the first row.

For a Hermitian matrix the imaginary parts of the diagonal elements are of course zero and thus the imaginary parts of the corresponding Fortran array elements need not be set, but are assumed to be zero.

For packed triangular matrices the same storage layout is used whether or not DIAG = 'U', i.e., space is left for the diagonal elements even if those array elements are not referenced.

Throughout the following sections A^H denotes the complex conjugate of A^T.

4.3.1 The Level-2 BLAS matrix-vector routines

```
SUBROUTINE    F06PAF   ( TRANS,M,N,        ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
ENTRY         sgemv    ( TRANS,M,N,        ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
CHARACTER*1              TRANS
INTEGER                        M,N,                LDA,       INCX,         INCY
real                                       ALPHA,A(LDA,*),X(*),  BETA,Y(*)

SUBROUTINE    F06SAF   ( TRANS,M,N,        ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
ENTRY         cgemv    ( TRANS,M,N,        ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
CHARACTER*1              TRANS
INTEGER                        M,N,                LDA,       INCX,         INCY
complex                                    ALPHA,A(LDA,*),X(*),  BETA,Y(*)

SUBROUTINE    F06PBF   ( TRANS,M,N,KL,KU,ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
ENTRY         sgbmv    ( TRANS,M,N,KL,KU,ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
CHARACTER*1              TRANS
INTEGER                        M,N,KL,KU,          LDA,       INCX,         INCY
real                                       ALPHA,A(LDA,*),X(*),  BETA,Y(*)

SUBROUTINE    F06SBF   ( TRANS,M,N,KL,KU,ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
ENTRY         cgbmv    ( TRANS,M,N,KL,KU,ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
CHARACTER*1              TRANS
INTEGER                        M,N,KL,KU,          LDA,       INCX,         INCY
complex                                    ALPHA,A(LDA,*),X(*),  BETA,Y(*)

SUBROUTINE    F06PCF   ( UPLO,    N,       ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
ENTRY         ssymv    ( UPLO,    N,       ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
CHARACTER*1              UPLO
INTEGER                           N,               LDA,       INCX,         INCY
real                                       ALPHA,A(LDA,*),X(*),  BETA,Y(*)

SUBROUTINE    F06SCF   ( UPLO,    N,       ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
ENTRY         chemv    ( UPLO,    N,       ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
CHARACTER*1              UPLO
INTEGER                           N,               LDA,       INCX,         INCY
complex                                    ALPHA,A(LDA,*),X(*),  BETA,Y(*)
```

```
SUBROUTINE    F06PDF     ( UPLO,   N,K,    ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
ENTRY         ssbmv      ( UPLO,   N,K,    ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
CHARACTER*1                UPLO
INTEGER                            N,K,           LDA,     INCX,       INCY
real                                       ALPHA,A(LDA,*),X(*),   BETA,Y(*)

SUBROUTINE    F06SDF     ( UPLO,   N,K,    ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
ENTRY         chbmv      ( UPLO,   N,K,    ALPHA,A,LDA,    X,INCX,BETA,Y,INCY )
CHARACTER*1                UPLO
INTEGER                            N,K,           LDA,     INCX,       INCY
complex                                    ALPHA,A(LDA,*),X(*),   BETA,Y(*)

SUBROUTINE    F06PEF     ( UPLO,   N,      ALPHA,AP,       X,INCX,BETA,Y,INCY )
ENTRY         sspmv      ( UPLO,   N,      ALPHA,AP,       X,INCX,BETA,Y,INCY )
CHARACTER*1                UPLO
INTEGER                            N,                        INCX,       INCY
real                                       ALPHA,AP(*),    X(*),   BETA,Y(*)

SUBROUTINE    F06SEF     ( UPLO,   N,      ALPHA,AP,       X,INCX,BETA,Y,INCY )
ENTRY         chpmv      ( UPLO,   N,      ALPHA,AP,       X,INCX,BETA,Y,INCY )
CHARACTER*1                UPLO
INTEGER                            N,                        INCX,       INCY
complex                                    ALPHA,AP(*),    X(*),   BETA,Y(*)

SUBROUTINE    F06PFF     ( UPLO,TRANS,DIAG,N,     A,LDA,    X,INCX )
ENTRY         strmv      ( UPLO,TRANS,DIAG,N,     A,LDA,    X,INCX )
CHARACTER*1                UPLO,TRANS,DIAG
INTEGER                                   N,        LDA,      INCX
real                                              A(LDA,*),X(*)

SUBROUTINE    F06SFF     ( UPLO,TRANS,DIAG,N,     A,LDA,    X,INCX )
ENTRY         ctrmv      ( UPLO,TRANS,DIAG,N,     A,LDA,    X,INCX )
CHARACTER*1                UPLO,TRANS,DIAG
INTEGER                                   N,        LDA,      INCX
complex                                           A(LDA,*),X(*)

SUBROUTINE    F06PGF     ( UPLO,TRANS,DIAG,N,K,   A,LDA,    X,INCX )
ENTRY         stbmv      ( UPLO,TRANS,DIAG,N,K,   A,LDA,    X,INCX )
CHARACTER*1                UPLO,TRANS,DIAG
INTEGER                                   N,K,      LDA,      INCX
real                                              A(LDA,*),X(*)

SUBROUTINE    F06SGF     ( UPLO,TRANS,DIAG,N,K,   A,LDA,    X,INCX )
ENTRY         ctbmv      ( UPLO,TRANS,DIAG,N,K,   A,LDA,    X,INCX )
CHARACTER*1                UPLO,TRANS,DIAG
INTEGER                                   N,K,      LDA,      INCX
complex                                           A(LDA,*),X(*)

SUBROUTINE    F06PHF     ( UPLO,TRANS,DIAG,N,     AP,       X,INCX )
ENTRY         stpmv      ( UPLO,TRANS,DIAG,N,     AP,       X,INCX )
CHARACTER*1                UPLO,TRANS,DIAG
INTEGER                                   N,                  INCX
real                                              AP(*),    X(*)

SUBROUTINE    F06SHF     ( UPLO,TRANS,DIAG,N,     AP,       X,INCX )
ENTRY         ctpmv      ( UPLO,TRANS,DIAG,N,     AP,       X,INCX )
CHARACTER*1                UPLO,TRANS,DIAG
INTEGER                                   N,                  INCX
complex                                           AP(*),    X(*)
```

```
SUBROUTINE      F06PJF    ( UPLO,TRANS,DIAG,N,     A,LDA,    X,INCX )
ENTRY           strsv     ( UPLO,TRANS,DIAG,N,     A,LDA,    X,INCX )
CHARACTER*1                 UPLO,TRANS,DIAG
INTEGER                                   N,         LDA,      INCX
real                                               A(LDA,*),X(*)

SUBROUTINE      F06SJF    ( UPLO,TRANS,DIAG,N,     A,LDA,    X,INCX )
ENTRY           ctrsv     ( UPLO,TRANS,DIAG,N,     A,LDA,    X,INCX )
CHARACTER*1                 UPLO,TRANS,DIAG
INTEGER                                   N,         LDA,      INCX
complex                                            A(LDA,*),X(*)

SUBROUTINE      F06PKF    ( UPLO,TRANS,DIAG,N,K,   A,LDA,    X,INCX )
ENTRY           stbsv     ( UPLO,TRANS,DIAG,N,K,   A,LDA,    X,INCX )
CHARACTER*1                 UPLO,TRANS,DIAG
INTEGER                                   N,K,       LDA,      INCX
real                                               A(LDA,*),X(*)

SUBROUTINE      F06SKF    ( UPLO,TRANS,DIAG,N,K,   A,LDA,    X,INCX )
ENTRY           ctbsv     ( UPLO,TRANS,DIAG,N,K,   A,LDA,    X,INCX )
CHARACTER*1                 UPLO,TRANS,DIAG
INTEGER                                   N,K,       LDA,      INCX
complex                                            A(LDA,*),X(*)

SUBROUTINE      F06PLF    ( UPLO,TRANS,DIAG,N,     AP,       X,INCX )
ENTRY           stpsv     ( UPLO,TRANS,DIAG,N,     AP,       X,INCX )
CHARACTER*1                 UPLO,TRANS,DIAG
INTEGER                                   N,                   INCX
real                                               AP(*),    X(*)

SUBROUTINE      F06SLF    ( UPLO,TRANS,DIAG,N,     AP,       X,INCX )
ENTRY           ctpsv     ( UPLO,TRANS,DIAG,N,     AP,       X,INCX )
CHARACTER*1                 UPLO,TRANS,DIAG
INTEGER                                   N,                   INCX
complex                                            AP(*),    X(*)

SUBROUTINE      F06PMF    (       M,N,ALPHA,                 X,INCX,Y,INCY,A,LDA )
ENTRY           sger      (       M,N,ALPHA,                 X,INCX,Y,INCY,A,LDA )
INTEGER                           M,N,                         INCX,  INCY,  LDA
real                                  ALPHA,                 X(*),  Y(*),  A(LDA,*)

SUBROUTINE      F06SMF    (       M,N,ALPHA,                 X,INCX,Y,INCY,A,LDA )
ENTRY           cgeru     (       M,N,ALPHA,                 X,INCX,Y,INCY,A,LDA )
INTEGER                           M,N,                         INCX,  INCY,  LDA
complex                               ALPHA,                 X(*),  Y(*),  A(LDA,*)

SUBROUTINE      F06SNF    (       M,N,ALPHA,                 X,INCX,Y,INCY,A,LDA )
ENTRY           cgerc     (       M,N,ALPHA,                 X,INCX,Y,INCY,A,LDA )
INTEGER                           M,N,                         INCX,  INCY,  LDA
complex                               ALPHA,                 X(*),  Y(*),  A(LDA,*)

SUBROUTINE      F06PPF    ( UPLO,   N,ALPHA,                 X,INCX,       A,LDA )
ENTRY           ssyr      ( UPLO,   N,ALPHA,                 X,INCX,       A,LDA )
CHARACTER*1                 UPLO
INTEGER                             N,                         INCX,         LDA
real                                  ALPHA,                 X(*),         A(LDA,*)

SUBROUTINE      F06SPF    ( UPLO,   N,ALPHA,                 X,INCX,       A,LDA )
ENTRY           cher      ( UPLO,   N,ALPHA,                 X,INCX,       A,LDA )
CHARACTER*1                 UPLO
INTEGER                             N,                         INCX,         LDA
real                                  ALPHA
complex                                                      X(*),         A(LDA,*)
```

```
SUBROUTINE    F06PQF   ( UPLO,  N,ALPHA,         X,INCX,       AP )
ENTRY         sspr     ( UPLO,  N,ALPHA,         X,INCX,       AP )
CHARACTER*1              UPLO
INTEGER                         N,                 INCX
real                              ALPHA,         X(*),         AP(*)

SUBROUTINE    F06SQF   ( UPLO,  N,ALPHA,         X,INCX,       AP )
ENTRY         chpr     ( UPLO,  N,ALPHA,         X,INCX,       AP )
CHARACTER*1              UPLO
INTEGER                         N,                 INCX
real                              ALPHA
complex                                          X(*),         AP(*)

SUBROUTINE    F06PRF   ( UPLO,  N,ALPHA,         X,INCX,Y,INCY,A,LDA )
ENTRY         ssyr2    ( UPLO,  N,ALPHA,         X,INCX,Y,INCY,A,LDA )
CHARACTER*1              UPLO
INTEGER                         N,                 INCX,  INCY,  LDA
real                              ALPHA,         X(*),  Y(*),  A(LDA,*)

SUBROUTINE    F06SRF   ( UPLO,  N,ALPHA,         X,INCX,Y,INCY,A,LDA )
ENTRY         cher2    ( UPLO,  N,ALPHA,         X,INCX,Y,INCY,A,LDA )
CHARACTER*1              UPLO
INTEGER                         N,                 INCX,  INCY,  LDA
complex                           ALPHA,         X(*),  Y(*),  A(LDA,*)

SUBROUTINE    F06PSF   ( UPLO,  N,ALPHA,         X,INCX,Y,INCY,AP )
ENTRY         sspr2    ( UPLO,  N,ALPHA,         X,INCX,Y,INCY,AP )
CHARACTER*1              UPLO
INTEGER                         N,                 INCX,  INCY
real                              ALPHA,         X(*),  Y(*),  AP(*)

SUBROUTINE    F06SSF   ( UPLO,  N,ALPHA,         X,INCX,Y,INCY,AP )
ENTRY         chpr2    ( UPLO,  N,ALPHA,         X,INCX,Y,INCY,AP )
CHARACTER*1              UPLO
INTEGER                         N,                 INCX,  INCY
complex                           ALPHA,         X(*),  Y(*),  AP(*)
```

F06PAF, **F06SAF**, **F06PBF** and **F06SBF** perform the operation

$$\begin{array}{ll} y \leftarrow \alpha Ax + \beta y, & \text{when TRANS = 'N'}, \\ y \leftarrow \alpha A^T x + \beta y, & \text{when TRANS = 'T'}, \\ y \leftarrow \alpha A^H x + \beta y, & \text{when TRANS = 'C'}, \end{array}$$

where A is a general matrix for F06PAF and F06SAF, and is a general band matrix for F06PBF and F06SBF.

F06PCF, **F06SCF**, **F06PEF**, **F06SEF**, **F06PDF** and **F06SDF** perform the operation

$$y \leftarrow \alpha Ax + \beta y$$

where A is symmetric and Hermitian for F06PCF and F06SCF respectively, is symmetric and Hermitian stored in packed form for F06PEF and F06SEF respectively, and is symmetric and Hermitian band for F06PDF and F06SDF.

F06PFF, **F06SFF**, **F06PHF**, **F06SHF**, **F06PGF** and **F06SGF** perform the operation

$$\begin{array}{ll} x \leftarrow Ax, & \text{when TRANS = 'N'}, \\ x \leftarrow A^T x, & \text{when TRANS = 'T'}, \\ x \leftarrow A^H x, & \text{when TRANS = 'C'}, \end{array}$$

where A is a triangular matrix for F06PFF and F06SFF, is a triangular matrix stored in packed form for F06PHF and F06SHF, and is a triangular band matrix for F06PGF and F06SGF.

F06PJF, **F06SJF**, **F06PLF**, **F06SLF**, **F06PKF** and **F06SKF** solve the equations

$$\begin{array}{ll} Ax = b, & \text{when TRANS = 'N'}, \\ A^T x = b, & \text{when TRANS = 'T'}, \\ A^H x = b, & \text{when TRANS = 'C'}, \end{array}$$

where A is a triangular matrix for F06PJF and F06SJF, is a triangular matrix stored in packed form for F06PLF and F06SLF, and is a triangular band matrix for F06PKF and F06SKF. The vector b must be supplied in the array X and is overwritten by the solution. It is important to note that no test for singularity is included in these routines.

F06PMF and **F06SMF** perform the operation

$$A \leftarrow \alpha xy^T + A,$$

where A is a general matrix.

F06SNF performs the operation

$$A \leftarrow \alpha xy^H + A,$$

where A is a general complex matrix.

F06PPF and **F06PQF** perform the operation

$$A \leftarrow \alpha xx^T + A,$$

where A is a symmetric matrix for F06PPF and is a symmetric matrix stored in packed form for F06PQF.

F06SPF and **F06SQF** perform the operation

$$A \leftarrow \alpha xx^H + A,$$

where A is an Hermitian matrix for F06SPF and is an Hermitian matrix stored in packed form for F06SQF.

F06PRF and **F06PSF** perform the operation

$$A \leftarrow \alpha xy^T + \alpha yx^T + A,$$

where A is a symmetric matrix for F06PRF and is a symmetric matrix stored in packed form for F06PSF.

F06SRF and **F06SSF** perform the operation

$$A \leftarrow \alpha xy^H + \bar{\alpha} yx^H + A,$$

where A is an Hermitian matrix for F06SRF and is an Hermitian matrix stored in packed form for F06SSF.

The following argument values are invalid:

Any value of the character arguments DIAG, TRANS, or UPLO whose meaning is not specified.

M < 0

N < 0

KL < 0

KU < 0

K < 0

LDA < M

LDA < KL + KU + 1

LDA < N for the routines involving full Hermitian, symmetric or triangular matrices

LDA < K+1 for the routines involving band Hermitian, symmetric or triangular matrices

INCX = 0

INCY = 0

If a routine is called with an invalid value then an error message is output, on the error message unit (see X04AAF), giving the name of the routine and the number of the first invalid argument, and execution is terminated.

4.4 The Level-2 Matrix Routines

The matrix routines have either one or two array arguments representing matrices; currently these are all two-dimensional arrays. When the array name A is used, the conventions for the size and description of the matrix are as for the matrix-vector routines described in Section 4.3. The alternative array name is B which is always followed by its first dimension as declared in the calling (sub)program (LDB).

The array B is used either as the second two-dimensional array argument, or to represent a matrix which is to be multiplied by a sequence of n, $m \times m$ permutation matrices. In this case B represents either an $m \times k$, or a $k \times m$ matrix depending upon whether the permutations are to be applied from the left or the right respectively. n, m and k are represented by the arguments N, M and K. The permutation matrices are represented by a single one-dimensional array argument PERM, which is either an integer or a real array.

Many of the routines in this section are concerned with applying sequences of plane rotations to matrices. For all these routines the sequence of plane rotations is represented by two one-dimensional array arguments, C and S, defining the cosines and sines respectively for each plane rotation (see Section 2.2.1 and Section 2.2.2). In most cases the plane rotations can be restricted to planes $k1$ through to $k2$ ($1 \leq k1 \leq k2 \leq m$ or n) defined by the integer arguments K1 and K2. If any of the above inequalities do not hold, then an immediate return is effected. In the descriptions P^H denotes the complex conjugate of P^T.

Vectors are again represented by one-dimensional array arguments (X or Y) together with a corresponding increment argument (INCX or INCY), which for the routines in this section must be positive.

The character arguments UPLO and TRANS have the same meaning as described in Section 4.3. Additionally five other character arguments are used to specify options in this section, with the names SIDE, PIVOT, DIRECT, NORM and MATRIX. SIDE is used by the permutation routines and many of the plane rotation routines as follows:

Value	**Meaning**
'L'	operate on the left-hand side
'R'	operate on the right-hand side

PIVOT and DIRECT are used together by some of the plane rotation routines. PIVOT is used as follows:

Value	**Meaning**
'B'	bottom (fixed) pivot
'T'	top (fixed) pivot
'V'	variable pivot

and DIRECT is used as follows:

Value	**Meaning**
'B'	backward sequence
'F'	forward sequence

NORM is used by the routines that return the norm of a matrix as follows:

Value	**Meaning**
'M'	$\max_{i,j} \lvert a_{i,j} \rvert$ (element of maximum absolute value. Not strictly a norm)
'1' or 'O'	$\lVert A \rVert_1$ (one norm of a matrix)
'I'	$\lVert A \rVert_\infty$ (infinity norm of a matrix)
'F' or 'E'	$\lVert A \rVert_F$ (Frobenius or Euclidean norm of a matrix)

MATRIX is used by some of the routines to determine the type of matrix represented by the corresponding array argument as follows:

Value	Meaning
'G'	general (rectangular or square) matrix
'U'	upper trapezoidal or triangular matrix
'L'	lower trapezoidal or triangular matrix

For the following routines, WORK must be an array of length M when NORM = 'I', and an array of length 1 otherwise:

```
F06RAF   F06RJF   F06UAF   F06UJF
```

For the following routines, WORK must be an array of length N when NORM = 'I', and an array of length 1 otherwise:

```
F06RBF   F06RKF   F06RLF   F06RMF   F06UBF   F06UKF
F06ULF   F06UMF
```

For the following routines, WORK must be an array of length N when NORM = 'I', '1' or 'O', and an array of length 1 otherwise:

```
F06RCF   F06RDF   F06REF   F06UCF   F06UDF   F06UEF
F06UFF   F06UGF   F06UHF
```

4.4.1 The F06 Level-2 matrix routines

```
SUBROUTINE      F06QFF   ( MATRIX,M,N,           A,LDA,    B,LDB )
CHARACTER*1                MATRIX
INTEGER                           M,N,            LDA,      LDB
real                                             A(LDA,*),B(LDB,*)

SUBROUTINE      F06TFF   ( MATRIX,M,N,           A,LDA,    B,LDB )
CHARACTER*1                MATRIX
INTEGER                           M,N,            LDA,      LDB
complex                                          A(LDA,*),B(LDB,*)

SUBROUTINE      F06QHF   ( MATRIX,M,N,CONST,DIAG,A,LDA )
CHARACTER*1                MATRIX
INTEGER                           M,N,           LDA
real                                  CONST,DIAG,A(LDA,*)

SUBROUTINE      F06THF   ( MATRIX,M,N,CONST,DIAG,A,LDA )
CHARACTER*1                MATRIX
INTEGER                           M,N,           LDA
complex                               CONST,DIAG,A(LDA,*)

SUBROUTINE      F06QJF   ( SIDE,TRANS,N,PERM,    K,B,LDB )
CHARACTER*1                SIDE,TRANS
INTEGER                               N,PERM(*),K,  LDB
real                                              B(LDB,*)

SUBROUTINE      F06VJF   ( SIDE,TRANS,N,PERM,    K,B,LDB )
CHARACTER*1                SIDE,TRANS
INTEGER                               N,PERM(*),K,  LDB
complex                                           B(LDB,*)

SUBROUTINE      F06QKF   ( SIDE,TRANS,N,PERM,    K,B,LDB )
CHARACTER*1                SIDE,TRANS
INTEGER                               N,         K,  LDB
real                                    PERM(*),  B(LDB,*)
```

```
SUBROUTINE      F06VKF   ( SIDE,TRANS,N,PERM,   K,B,LDB )
CHARACTER*1                SIDE,TRANS
INTEGER                              N,         K,  LDB
real                                   PERM(*)
complex                                             B(LDB,*)

SUBROUTINE      F06QMF   ( UPLO,PIVOT,DIRECT,N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                UPLO,PIVOT,DIRECT
INTEGER                                      N,K1,K2,            LDA
real                                                 C(*),S(*),A(LDA,*)

SUBROUTINE      F06TMF   ( UPLO,PIVOT,DIRECT,N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                UPLO,PIVOT,DIRECT
INTEGER                                      N,K1,K2,            LDA
real                                                 C(*)
complex                                                    S(*),A(LDA,*)

SUBROUTINE      F06QPF   ( N,ALPHA,X,INCX,Y,INCY,A,LDA,    C,    S )
INTEGER                    N,        INCX,  INCY,  LDA
real                         ALPHA,X(*),  Y(*),  A(LDA,*),C(*),S(*)

SUBROUTINE      F06TPF   ( N,ALPHA,X,INCX,Y,INCY,A,LDA,    C,    S )
INTEGER                    N,        INCX,  INCY,  LDA
complex                      ALPHA,X(*),  Y(*),  A(LDA,*),      S(*)
real                                                      C(*)

SUBROUTINE      F06QQF   ( N,ALPHA,X,INCX,        A,LDA,   C,    S )
INTEGER                    N,        INCX,          LDA
real                         ALPHA,X(*),          A(LDA,*),C(*),S(*)

SUBROUTINE      F06TQF   ( N,ALPHA,X,INCX,        A,LDA,   C,    S )
INTEGER                    N,        INCX,          LDA
complex                      ALPHA,X(*),          A(LDA,*),      S(*)
real                                                       C(*)

SUBROUTINE      F06QRF   ( SIDE,               N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                SIDE
INTEGER                                        N,K1,K2,            LDA
real                                                   C(*),S(*),A(LDA,*)

SUBROUTINE      F06TRF   ( SIDE,               N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                SIDE
INTEGER                                        N,K1,K2,            LDA
complex                                                C(*),       A(LDA,*)
real                                                         S(*)

SUBROUTINE      F06QSF   ( SIDE,               N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                SIDE
INTEGER                                        N,K1,K2,            LDA
real                                                   C(*),S(*),A(LDA,*)

SUBROUTINE      F06TSF   ( SIDE,               N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                SIDE
INTEGER                                        N,K1,K2,            LDA
real                                                   C(*)
complex                                                      S(*),A(LDA,*)

SUBROUTINE      F06QTF   ( SIDE,               N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                SIDE
INTEGER                                        N,K1,K2,            LDA
real                                                   C(*),S(*),A(LDA,*)
```

```
SUBROUTINE          F06TTF    ( SIDE,                   N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                     SIDE
INTEGER                                                 N,K1,K2,            LDA
real                                                           C(*)
complex                                                              S(*),A(LDA,*)

SUBROUTINE          F06QVF    ( SIDE,                   N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                     SIDE
INTEGER                                                 N,K1,K2,            LDA
real                                                           C(*),S(*),A(LDA,*)

SUBROUTINE          F06TVF    ( SIDE,                   N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                     SIDE
INTEGER                                                 N,K1,K2,            LDA
complex                                                        C(*),      A(LDA,*)
real                                                                 S(*)

SUBROUTINE          F06QWF    ( SIDE,                   N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                     SIDE
INTEGER                                                 N,K1,K2,            LDA
real                                                           C(*),S(*),A(LDA,*)

SUBROUTINE          F06TWF    ( SIDE,                   N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                     SIDE
INTEGER                                                 N,K1,K2,            LDA
real                                                           C(*)
complex                                                              S(*),A(LDA,*)

SUBROUTINE          F06QXF    ( SIDE,PIVOT,DIRECT,M,N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                     SIDE,PIVOT,DIRECT
INTEGER                                               M,N,K1,K2,            LDA
real                                                           C(*),S(*),A(LDA,*)

SUBROUTINE          F06TXF    ( SIDE,PIVOT,DIRECT,M,N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                     SIDE,PIVOT,DIRECT
INTEGER                                               M,N,K1,K2,            LDA
real                                                           C(*)
complex                                                              S(*),A(LDA,*)

SUBROUTINE          F06VXF    ( SIDE,PIVOT,DIRECT,M,N,K1,K2,C,S,        A,LDA )
CHARACTER*1                     SIDE,PIVOT,DIRECT
INTEGER                                               M,N,K1,K2,LDA
real                                                           C(*),S(*)
complex                                                                   A(LDA,*)

SUBROUTINE          F06TYF    ( SIDE,PIVOT,DIRECT,M,N,K1,K2,C,    S,    A,LDA )
CHARACTER*1                     SIDE,PIVOT,DIRECT
INTEGER                                               M,N,K1,K2,            LDA
complex                                                        C(*),      A(LDA,*)
real                                                                 S(*)

real FUNCTION       F06RAF    ( NORM,M,N,             A,LDA,      WORK)
CHARACTER*1                     NORM
INTEGER                              M,N,               LDA
real                                                  A(LDA,*),   WORK(*)

real FUNCTION       F06RBF    ( NORM,N,KL,KU,         A,LDA,      WORK)
CHARACTER*1                     NORM
INTEGER                              N,KL,KU,           LDA
real                                                  A(LDA,*),   WORK(*)

real FUNCTION       F06RCF    ( NORM,UPLO,N,          A,LDA,      WORK)
CHARACTER*1                     NORM,UPLO
INTEGER                                   N,            LDA
real                                                  A(LDA,*),   WORK(*)
```

```
real FUNCTION      F06RDF    ( NORM,UPLO,N,        AP,         WORK)
CHARACTER*1                    NORM,UPLO
INTEGER                                  N
real                                               AP(*),      WORK(*)

real FUNCTION      F06REF    ( NORM,UPLO,N,K,      A,LDA,      WORK)
CHARACTER*1                    NORM,UPLO
INTEGER                                  N,K,        LDA
real                                               A(LDA,*),   WORK(*)

real FUNCTION      F06RJF    ( NORM,UPLO,DIAG,M,N,A,LDA,       WORK)
CHARACTER*1                    NORM,UPLO,DIAG
INTEGER                                      M,N,  LDA
real                                               A(LDA,*),   WORK(*)

real FUNCTION      F06RKF    ( NORM,UPLO,DIAG,  N,AP,          WORK)
CHARACTER*1                    NORM,UPLO,DIAG
INTEGER                                         N
real                                               AP(*),      WORK(*)

real FUNCTION      F06RLF    ( NORM,UPLO,DIAG,  N,K,A,LDA,     WORK)
CHARACTER*1                    NORM,UPLO,DIAG
INTEGER                                         N,K,  LDA
real                                                 A(LDA,*),WORK(*)

real FUNCTION      F06RMF    ( NORM,            N,   A,LDA,    WORK)
CHARACTER*1                    NORM
INTEGER                                         N,      LDA
real                                                 A(LDA,*),WORK(*)

real FUNCTION      F06UAF    ( NORM,          M,N,   A,LDA,    WORK)
CHARACTER*1                    NORM
INTEGER                                       M,N
real                                                             WORK(*)
complex                                              A(LDA,*)

real FUNCTION      F06UBF    ( NORM,            N,KL,KU,A,LDA,WORK)
CHARACTER*1                    NORM
INTEGER                                         N,KL,KU,  LDA
real                                                             WORK(*)
complex                                                  A(LDA,*)

real FUNCTION      F06UCF    ( NORM,UPLO,       N,       A,LDA,WORK)
CHARACTER*1                    NORM,UPLO
INTEGER                                         N,          LDA
real                                                             WORK(*)
complex                                                  A(LDA,*)

real FUNCTION      F06UDF    ( NORM,UPLO,       N,       AP,   WORK)
CHARACTER*1                    NORM,UPLO
INTEGER                                         N
real                                                             WORK(*)
complex                                                  AP(*)

real FUNCTION      F06UEF    ( NORM,UPLO,       N,K,     A,LDA,WORK)
CHARACTER*1                    NORM,UPLO
INTEGER                                         N,K,        LDA
real                                                             WORK(*)
complex                                                  A(LDA,*)
```

```
real FUNCTION    F06UFF   ( NORM,UPLO,         N,      A,LDA,WORK)
CHARACTER*1                 NORM,UPLO
INTEGER                                        N,        LDA
real                                                          WORK(*)
complex                                                A(LDA,*)

real FUNCTION    F06UGF   ( NORM,UPLO,         N,      AP,    WORK)
CHARACTER*1                 NORM,UPLO
INTEGER                                        N
real                                                          WORK(*)
complex                                                AP(*)

real FUNCTION    F06UHF   ( NORM,UPLO,         N,K,    A,LDA,WORK )
CHARACTER*1                 NORM,UPLO
INTEGER                                        N,K,      LDA
real                                                          WORK(*)
complex                                                A(LDA,*)

real FUNCTION    F06UJF   ( NORM,UPLO,DIAG,M,N,        A,LDA,WORK)
CHARACTER*1                 NORM,UPLO,DIAG
INTEGER                                      M,N,        LDA
real                                                          WORK(*)
complex                                                A(LDA,*)

real FUNCTION    F06UKF   ( NORM,UPLO,DIAG,    N,      AP,    WORK)
CHARACTER*1                 NORM,UPLO,DIAG
INTEGER                                        N
real                                                          WORK(*)
complex                                                AP(*)

real FUNCTION    F06ULF   ( NORM,UPLO,DIAG,    N,K,    A,LDA,WORK)
CHARACTER*1                 NORM,UPLO,DIAG
INTEGER                                        N,K,      LDA
real                                                          WORK(*)
complex                                                A(LDA,*)

real FUNCTION    F06UMF   ( NORM,              N,      A,LDA,WORK)
CHARACTER*1                 NORM
INTEGER                                        N,        LDA
real                                                          WORK(*)
complex                                                A(LDA,*)
```

F06QFF and **F06TFF** perform the operation

$$B \leftarrow A$$

where A and B are $m \times n$ general (rectangular or square), or upper or lower trapezoidal or triangular matrices.

F06RAF, **F06UAF**, **F06RBF**, **F06UBF**, **F06RCF**, **F06UCF**, **F06RDF**, **F06UDF**, **F06REF**, **F06UEF**, **F06RJF**, **F06UJF**, **F06RKF**, **F06UKF**, **F06RLF**, **F06ULF**, **F06RMF**, **F06UMF**, **F06UFF**, **F06UGF** and **F06UHF** return one of the values *amax* or $\|A\|_1$ or $\|A\|_\infty$ or $\|A\|_F$ given by

$$\begin{aligned} amax &= \max_{i,j} |a_{ij}|, \\ \|A\|_1 &= \max_j \sum_{i=1}^{m} |a_{ij}|, \\ \|A\|_\infty &= \max_i \sum_{j=1}^{n} |a_{ij}|, \\ \|A\|_F &= \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2}, \end{aligned}$$

where A is either an $m \times n$ general rectangular or upper or lower trapezoidal matrix, or a square ($m = n$) band or symmetric or Hermitian or Hessenberg or upper or lower triangular matrix, or a combination of these (e.g. Hermitian band). When A is symmetric or Hermitian or triangular it may be supplied in packed form.

F06QHF and **F06THF** perform the operation

$$a_{ij} \leftarrow \begin{cases} diag, & i = j \\ const, & i \neq j \end{cases}$$

where A is an $m \times n$ general (rectangular or square), or upper or lower trapezoidal or triangular matrix.

F06QJF, **F06VJF**, **F06QKF** and **F06VKF** perform one of the operations

$$\begin{array}{ll} B \leftarrow PB & \text{when SIDE = 'L' and TRANS = 'N'}, \\ B \leftarrow P^T B & \text{when SIDE = 'L' and TRANS = 'T'}, \\ B \leftarrow BP & \text{when SIDE = 'R' and TRANS = 'N'}, \\ B \leftarrow BP^T & \text{when SIDE = 'R' and TRANS = 'T'}, \end{array}$$

where P is an $m \times m$ permutation matrix of the form

$$P = P_{1,i_1} P_{2,i_2} \ldots P_{n,i_n},$$

P_{j,i_j} being the permutation matrix that interchanges items j and i_j. That is, P_{j,i_j} is the unit matrix with rows and columns j and i_j interchanged. If $j = i_j$ then $P = I$. i_j must satisfy $1 \leq i_j \leq m$. P_{j,i_j} is represented by the jth element of the argument PERM such that PERM(j) = i_j.

When SIDE = 'L', B is an $m \times k$ matrix and when SIDE = 'R', B is a $k \times m$ matrix. Note that m is not actually an argument of these routines.

F06QMF and **F06TMF** perform the operations

$A \leftarrow PAP^T$ for F06QMF and

$A \leftarrow PAP^H$ for F06TMF,

where A is an $n \times n$ symmetric (Hermitian for F06TMF) matrix and P is an orthogonal (unitary for F06TMF) matrix consisting of a sequence of plane rotations, applied in planes $k1$ to $k2$. When DIRECT = 'F' then P is given by the sequence

$$P = P_{k2-1}, \ldots, P_{k1+1}, P_{k1}$$

and when DIRECT = 'B' then P is given by the sequence

$$P = P_{k1}, P_{k1+1}, \ldots, P_{k2-1},$$

where P_k is a plane rotation matrix for the $(k, k+1)$ plane when PIVOT = 'V', P_k is a plane rotation matrix for the $(k1, k+1)$ plane when PIVOT = 'T' and P_k is a plane rotation matrix for the $(k, k2)$ plane when PIVOT = 'B' or 'b'.

The two by two part of the plane rotations are assumed to be of the form given by equation (1) for F06QMF and (10) (with c real) for F06TMF. The cosine and sine that define P_k must be supplied in C(k) and S(k) respectively.

F06QPF and **F06TPF** perform the factorization

$$\alpha xy^T + U = QR,$$

where α is a scalar, x and y are n element vectors, U and R are $n\times$ upper triangular matrices and Q is an $n \times n$ orthogonal (unitary for F06TPF) matrix. For F06TPF, U must have real diagonal elements and R is returned with real diagonal elements. Q is formed as two sequences of plane rotations, P and S. P is a sequence of the form

$$P = P_1, P_2, \ldots, P_{n-1},$$

where P_k is a rotation for the (k, n) plane, chosen so that

$$Px = \beta e_n,$$

e_n being the last column of the unit matrix. S is a sequence of the form

$$S = S_{n-1}, S_{n-2}, \ldots, S_1,$$

where S_k is a rotation for the (k,n) plane, chosen so that

$$S(\alpha\beta e_n y^T + PU) = R$$

Q is given as

$Q^T = SP$ for F06QPF and as
$Q^H = DSP$ for F06TPF,

where D is a unitary diagonal matrix with a non-unit element only in d_n, which is chosen to make r_{nn} real and is returned in S(n).

The two by two part of the plane rotations are of the form of equation (1) for F06QPF and (10) (with c real) for F06TPF. The cosine and sine that define S_k are returned in C(k) and S(k) repectively and the tangent that defines the rotation P_k is returned in the element of X corresponding to x_k. (Routines F06BCF and F06CCF may be used to recover the cosine and sine from a given tangent.) The array Y is unchanged on exit.

U must be supplied in the $n \times n$ upper triangular part of the array A and is overwritten by R. In the case of F06TPF the imaginary parts of the diagonal elements of U must be supplied as zero.

F06QQF and **F06TQF** perform the factorization

$$\begin{pmatrix} U \\ \alpha x^T \end{pmatrix} = Q \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where *alpha* is a scalar, x is an n element vector, U and R are $n \times n$ upper triangular matrices and Q is an $(n+1) \times (n+1)$ orthogonal (unitary for F06TQF) matrix. For F06TQF, if U is supplied with real diagonal elements then the diagonal elements of R will also be real. Q is formed as a sequence of plane rotations

$Q^T = Q_n, \ldots, Q_2, Q_1$ for F06QQF
$Q^H = Q_n, \ldots, Q_2, Q_1$ for F06TQF,

where Q_k is a rotation for the $(k, n+1)$ plane.

The two by two part of the plane rotations are of the form of equation (1) for F06QQF and (10) (with c real) for F06TQF. The cosine, sine and tangent that define Q_k are returned respectively in C(k), S(k) and the element of X that corresponds to x_k. (Routines F06BCF and F06CCF may be used to recover the cosine and sine from a given tangent.)

U must be supplied in the $n \times n$ upper triangular part of the array A and is overwritten by R.

F06QRF and **F06TRF** perform one of the operations

$$\begin{array}{ll} R \leftarrow PH & \text{when SIDE = 'L'}, \\ \left.\begin{array}{l} R \leftarrow HP^T \text{ for F06QRF} \\ R \leftarrow HP^H \text{ for F06TRF} \end{array}\right\} & \text{when SIDE = 'R'}, \end{array}$$

where H is an $n \times n$ upper Hessenberg matrix, P is an $n \times n$ orthogonal (unitary for F06TRF) matrix and R is an $n \times n$ upper triangular matrix. H is assumed to have (possibly) non-zero sub-diagonal elements only in elements $h_{k+1,k}$ and these elements must be supplied in S(k), $k = k1, k1+1, \ldots, k2+1$. For F06TRF, H must have real sub-diagonal elements and R will be returned with real diagonal elements. The upper triangular part of H must be supplied in A and is overwritten by R.

When SIDE = 'L', P is given by

$P = P_{k2-1}, P_{k2-2}, \ldots, P_{k1}$ for F06QRF,
$P = D_{k2}, P_{k2-1}, P_{k2-2}, \ldots, P_{k1}$ for F06TRF,

and when SIDE = 'R', P is given by

$P = P_{k1}, P_{k1+1}, \ldots, P_{k2-1}$ for F06QRF,
$P = D_{k1}, P_{k1}, P_{k1+1}, \ldots, P_{k2-1}$ for F06TRF,

where P_k is a plane rotation matrix for the $(k, k+1)$ plane and D_k is a unitary diagonal matrix with a non-unit diagonal element only in d_k.

The two by two part of the plane rotations are of the form of equation (1) for F06QRF and (11) (with s real) for F06TRF. The cosine and sine that define P_k are returned in C(k) and S(k) respectively and, for F06TRF, d_k is returned in C($k2$).

F06QSF and **F06TSF** perform one of the operations

$$\begin{array}{ll} R \leftarrow PH & \text{when SIDE = 'L'}, \\ \left.\begin{array}{l} R \leftarrow HP^T \text{ for F06QSF} \\ R \leftarrow HP^H \text{ for F06TSF} \end{array}\right\} & \text{when SIDE = 'R'}, \end{array}$$

where H is an $n \times n$ upper spiked matrix, P is an $n \times n$ orthogonal (unitary for F06TSF) matrix and R is an $n \times n$ upper triangular matrix. When SIDE = 'L', H is assumed to have a row spike and have (possibly) non-zero sub-diagonal elements only in elements $h_{k2,k}$, $k = k1, k1+1, \ldots, k2-1$ and when SIDE = 'R', H is assumed to have a column spike and have (possibly) non-zero sub-diagonal elements only in elements $h_{k+1,k1}$, $k = k1, k1+1, \ldots, k2-1$. These spiked elements must be supplied in the elements S(k), $k = k1, k1+1, \ldots, k2-1$. For F06TSF, H must have real diagonal elements except in the position where the spike joins the diagonal, that is $h_{k2,k2}$ for a row spike and $h_{k1,k1}$ for a column spike, and R will be returned with real diagonal elements. The upper triangular part of H must be supplied in A and is overwritten by R.

When SIDE = 'L', P is given by

$P = P_{k2-1}, P_{k2-2}, \ldots, P_{k1}$ for F06QSF,
$P = D_{k2}, P_{k2-1}, P_{k2-2}, \ldots, P_{k1}$ for F06TSF,

where P_k is a plane rotation matrix for the $(k, k2)$ plane and when SIDE = 'R', P is given by

$P = P_{k1+1}, P_{k1+2}, \ldots, P_{k2-1}$ for F06QSF,
$P = D_{k1}, P_{k1+1}, P_{k1+2}, \ldots, P_{k2+1}$ for F06TSF,

where P_k is a plane rotation matrix for the $(k1, k)$ plane and D_k is a unitary diagonal matrix with a non-unit diagonal element only in d_k.

The two by two part of the plane rotations are of the form of equation (1) for F06QSF and (10) (with c real) for F06TSF. The cosine and sine that define P_k are returned in C(k) and S(k) respectively and, for F06TSF, d_k is returned in S($k2$).

F06QTF and **F06TTF** perform one of the operations

$$\left.\begin{array}{l} R \leftarrow PUQ^T \text{ for F06QTF} \\ R \leftarrow PUQ^H \text{ for F06TTF} \end{array}\right\} \text{ when SIDE = 'L'}$$

$$\left.\begin{array}{l} R \leftarrow QUP^T \text{ for F06QTF} \\ R \leftarrow QUP^H \text{ for F06TTF} \end{array}\right\} \text{ when SIDE = 'R'}$$

where U and R are $n \times n$ upper triangular matrices and P and Q are $n \times n$ orthogonal (unitary for F06TTF) matrices, with P given. When SIDE = 'L' then P is assumed to be given by

$$P = P_{k2-1}, P_{k2-2}, \ldots, P_{k1}$$

and Q is then given as

$$Q = Q_{k2-1}, Q_{k2-2}, \ldots, Q_{k1}$$

and when SIDE = 'R' then P is assumed to be given by

$$P = P_{k1}, P_{k1+1}, \ldots, P_{k2-1}$$

and Q is then given as

$$Q = Q_{k1}, Q_{k1+1}, \ldots, Q_{k2-1},$$

where P_k and Q_k are plane rotations matrices for the $(k, k+1)$ plane.

The two by two part of the plane rotations are of the form of equation (1) for F06QTF and (10) (with c real) for F06TTF. The cosine and sine that define P_k must be supplied in C(k) and S(k) respectively, and are overwritten by the cosine and sine that define Q_k.

The matrix U must be supplied in the upper triangular part of A and is overwritten by R.

F06QVF and **F06TVF** perform one of the operations

$$\begin{array}{ll} H \leftarrow PU & \text{when SIDE = 'L'}, \\ \left.\begin{array}{l} H \leftarrow UP^T \text{ for F06QVF} \\ H \leftarrow UP^H \text{ for F06TVF} \end{array}\right\} & \text{when SIDE = 'R'}, \end{array}$$

where U is an $n \times n$ upper triangular matrix, H is an $n \times n$ upper Hessenberg matrix and P is an $n \times n$ orthogonal (unitary for F06TVF) matrix. For F06TVF, U must be supplied with real diagonal elements and H will have real sub-diagonal elements. When SIDE = 'L' or 'l', then P is assumed to be given by

$$P = P_{k1}, P_{k1+1}, \ldots, P_{k2-1}$$

and when SIDE = 'R', then P is assumed to be given by

$$P = P_{k2-1}, P_{k2-2}, \ldots, P_{k1},$$

where P_k is a plane rotation matrix for the $(k, k+1)$ plane.

The two by two part of the plane rotations are of the form of equation (1) for F06QVF and (11) (with s real) for F06TVF. The cosine and sine that define P_k must be supplied in C(k) and S(k) respectively.

The matrix U must be supplied in the upper triangular part of A and is overwritten by the upper triangular part of H. The sub-diagonal elements of H, $h_{k+1,k}$, are returned in the elements S(k), $k = k1, k1+1, \ldots, k2-1$.

F06QWF and **F06TWF** perform one of the operations

$$\begin{array}{ll} H \leftarrow PU & \text{when SIDE = 'L'}, \\ \left.\begin{array}{l} H \leftarrow UP^T \text{ for F06QWF} \\ H \leftarrow UP^H \text{ for F06TWF} \end{array}\right\} & \text{when SIDE = 'R'}, \end{array}$$

where U is an $n \times n$ upper triangular matrix, H is an $n \times n$ upper spiked matrix and P is an $n \times n$ orthogonal (unitary for F06TWF) matrix. For F06TWF, U must be supplied with real diagonal elements and H will have real diagonal elements, except for the position where the spike joins the diagonal.

When SIDE = 'L', then P is assumed to be given by

$$P = P_{k1}, P_{k1+1}, \ldots, P_{k2-1},$$

where P_k is a plane rotation matrix for the $(k, k2)$ plane and when SIDE = 'R', then P is assumed to be given by

$$P = P_{k2-1}, P_{k2-2}, \ldots, P_{k1},$$

where P_k is a plane rotation matrix for the $(k1, k+1)$ plane.

The two by two part of the plane rotations are of the form of equation (1) for F06QWF and (10) for F06TWF. The cosine and sine that define P_k must be supplied in the elements C(k) and S(k) respectively.

The matrix U must be supplied in the upper triangular part of A and is overwritten by the upper triangular part of H. The sub-diagonal elements of H, $h_{k2,k}$ (row spike) when SIDE = 'L', and $h_{k+1,k1}$ (column spike) when SIDE = 'R' are returned in the elements S(k), $k = k1, k1+1, \ldots, k2-1$.

F06QXF, **F06TXF**, **F06TYF** and **F06VXF** perform the operation

$$\begin{array}{ll} A \leftarrow PA & \text{when SIDE = 'L'}, \\ \left.\begin{array}{l} A \leftarrow AP^T \text{ for F06QXF and F06VXF} \\ A \leftarrow AP^H \text{ for F06TXF and F06TYF} \end{array}\right\} & \text{when SIDE = 'R'}, \end{array}$$

where A is an m by n matrix and P is an orthogonal (unitary for F06TXF and F06TYF) matrix consisting of a sequence of plane rotations, applied in planes $k1$ to $k2$. When DIRECT = 'F' then P is given by the sequence

$$P = P_{k2-1}, \ldots, P_{k1+1}, P_{k1}$$

and when DIRECT = 'B' then P is given by the sequence

$$P = P_{k1}, P_{k1+1}, \ldots, P_{k2-1},$$

where P_k is a plane rotation matrix for the $(k, k+1)$ plane when PIVOT = 'V', P_k is a plane rotation matrix for the $(k1, k+1)$ plane, when PIVOT = 'T' and P_k is a plane rotation matrix for the $(k, k2)$ plane when PIVOT = 'B' or 'b'.

The two by two plane rotation part of P_k is assumed to be of the form given by equation (1) for F06QXF and F06VXF, (10) (with c real) for F06TXF and (11) (with s real) for F06TYF. The cosine and sine that define P_k must be supplied in C(k) and S(k) respectively.

4.5 The Level-3 Matrix-matrix Routines

The matrix-matrix routines all have either two or three arguments representing a matrix, one of which is an input-output argument, and in each case the arguments are two-dimensional arrays.

The sizes of the matrices are determined by one or more of the arguments M, N and K. The size of the input-output array is always determined by the arguments M and N for a rectangular m by n matrix, and by the argument N for a square n by n matrix. It is permissible to call the routines with M or N = 0, in which case the routines exit immediately without referencing their array arguments.

Many of the routines perform an operation of the form

$$C \leftarrow P + \beta C,$$

where P is the product of two matrices, or the sum of two such products. When the inner dimension of the matrix product is different from m or n it is denoted by K. Again it is permissible to call the routines with K = 0 and if M > 0, but K = 0, then the routines perform the operation

$$C \leftarrow \beta C.$$

As with the Level-2 routines (see Section 4) the description of the matrix consists of the array name (A or B or C) followed by the first dimension (LDA or LDB or LDC).

The arguments that specify options are character arguments with the names SIDE, TRANSA, TRANSB, TRANS, UPLO and DIAG. UPLO and DIAG have the same values and meanings as for the Level-2 routines (see Section 4.3); TRANSA, TRANSB and TRANS have the same values and meanings as TRANS in the Level-2 routines, where TRANSA and TRANSB apply to the matrices A and B respectively. SIDE is used by the routines as follows:

Value	**Meaning**
'L'	Multiply general matrix by symmetric, Hermitian or triangular matrix on the left
'R'	Multiply general matrix by symmetric, Hermitian or triangular matrix on the right

The storage conventions for matrices are as for the Level-2 routines (see Section 4.3).

4.5.1 The Level-3 BLAS matrix-matrix routines

```
SUBROUTINE    F06YAF   ( TRANSA,TRANSB,    M,N,K,ALPHA,A,LDA,   B,LDB,   BETA,C,LDC )
ENTRY         sgemm    ( TRANSA,TRANSB,    M,N,K,ALPHA,A,LDA,   B,LDB,   BETA,C,LDC )
CHARACTER*1              TRANSA,TRANSB
INTEGER                                    M,N,K,         LDA,      LDB,          LDC
real                                           ALPHA,A(LDA,*),B(LDB,*),BETA,C(LDC,*)

SUBROUTINE    F06ZAF   ( TRANSA,TRANSB,    M,N,K,ALPHA,A,LDA,   B,LDB,   BETA,C,LDC )
ENTRY         cgemm    ( TRANSA,TRANSB,    M,N,K,ALPHA,A,LDA,   B,LDB,   BETA,C,LDC )
CHARACTER*1              TRANSA,TRANSB
INTEGER                                    M,N,K,         LDA,      LDB,          LDC
complex                                        ALPHA,A(LDA,*),B(LDB,*),BETA,C(LDC,*)
```

```
SUBROUTINE    F06YCF  ( SIDE,UPLO,                 M,N,  ALPHA,A,LDA,      B,LDB,     BETA,C,LDC )
ENTRY         ssymm   ( SIDE,UPLO,                 M,N,  ALPHA,A,LDA,      B,LDB,     BETA,C,LDC )
CHARACTER*1             SIDE,UPLO
INTEGER                                            M,N,        LDA,        LDB,            LDC
real                                                     ALPHA,A(LDA,*),B(LDB,*),BETA,C(LDC,*)

SUBROUTINE    F06ZCF  ( SIDE,UPLO,                 M,N,  ALPHA,A,LDA,      B,LDB,     BETA,C,LDC )
ENTRY         chemm   ( SIDE,UPLO,                 M,N,  ALPHA,A,LDA,      B,LDB,     BETA,C,LDC )
CHARACTER*1             SIDE,UPLO
INTEGER                                            M,N,        LDA,        LDB,            LDC
complex                                                  ALPHA,A(LDA,*),B(LDB,*),BETA,C(LDC,*)

SUBROUTINE    F06ZTF  ( SIDE,UPLO,                 M,N,  ALPHA,A,LDA,      B,LDB,     BETA,C,LDC )
ENTRY         csymm   ( SIDE,UPLO,                 M,N,  ALPHA,A,LDA,      B,LDB,     BETA,C,LDC )
CHARACTER*1             SIDE,UPLO
INTEGER                                            M,N,        LDA,        LDB,            LDC
complex                                                  ALPHA,A(LDA,*),B(LDB,*),BETA,C(LDC,*)

SUBROUTINE    F06YFF  ( SIDE,UPLO,TRANSA,DIAG,M,N,  ALPHA,A,LDA,      B,LDB )
ENTRY         strmm   ( SIDE,UPLO,TRANSA,DIAG,M,N,  ALPHA,A,LDA,      B,LDB )
CHARACTER*1             SIDE,UPLO,TRANSA,DIAG
INTEGER                                      M,N,        LDA,        LDB
real                                                ALPHA,A(LDA,*),B(LDB,*)

SUBROUTINE    F06ZFF  ( SIDE,UPLO,TRANSA,DIAG,M,N,  ALPHA,A,LDA,      B,LDB )
ENTRY         ctrmm   ( SIDE,UPLO,TRANSA,DIAG,M,N,  ALPHA,A,LDA,      B,LDB )
CHARACTER*1             SIDE,UPLO,TRANSA,DIAG
INTEGER                                      M,N,        LDA,        LDB
complex                                             ALPHA,A(LDA,*),B(LDB,*)

SUBROUTINE    F06YJF  ( SIDE,UPLO,TRANSA,DIAG,M,N,  ALPHA,A,LDA,      B,LDB )
ENTRY         strsm   ( SIDE,UPLO,TRANSA,DIAG,M,N,  ALPHA,A,LDA,      B,LDB )
CHARACTER*1             SIDE,UPLO,TRANSA,DIAG
INTEGER                                      M,N,        LDA,        LDB
real                                                ALPHA,A(LDA,*),B(LDB,*)

SUBROUTINE    F06ZJF  ( SIDE,UPLO,TRANSA,DIAG,M,N,  ALPHA,A,LDA,      B,LDB )
ENTRY         ctrsm   ( SIDE,UPLO,TRANSA,DIAG,M,N,  ALPHA,A,LDA,      B,LDB )
CHARACTER*1             SIDE,UPLO,TRANSA,DIAG
INTEGER                                      M,N,        LDA,        LDB
complex                                             ALPHA,A(LDA,*),B(LDB,*)

SUBROUTINE    F06YPF  ( UPLO,TRANS,        N,K,ALPHA,A,LDA,                   BETA,C,LDC )
ENTRY         ssyrk   ( UPLO,TRANS,        N,K,ALPHA,A,LDA,                   BETA,C,LDC )
CHARACTER*1             UPLO,TRANS
INTEGER                                    N,K,        LDA,                         LDC
real                                           ALPHA,A(LDA,*),                BETA,C(LDC,*)

SUBROUTINE    F06ZPF  ( UPLO,TRANS,        N,K,ALPHA,A,LDA,                   BETA,C,LDC )
ENTRY         cherk   ( UPLO,TRANS,        N,K,ALPHA,A,LDA,                   BETA,C,LDC )
CHARACTER*1             UPLO,TRANS
INTEGER                                    N,K,        LDA,                         LDC
real                                           ALPHA,                         BETA
complex                                              A(LDA,*),                     C(LDC,*)

SUBROUTINE    F06ZUF  ( UPLO,TRANS,        N,K,ALPHA,A,LDA,                   BETA,C,LDC )
ENTRY         csyrk   ( UPLO,TRANS,        N,K,ALPHA,A,LDA,                   BETA,C,LDC )
CHARACTER*1             UPLO,TRAN
INTEGER                                    N,K,        LDA,                         LDC
complex                                        ALPHA,A(LDA,*),                BETA,C(LDC,*)

SUBROUTINE    F06YRF  ( UPLO,TRANS,        N,K,ALPHA,A,LDA,      B,LDB,       BETA,C,LDC )
ENTRY         ssyr2k  ( UPLO,TRANS,        N,K,ALPHA,A,LDA,      B,LDB,       BETA,C,LDC )
CHARACTER*1             UPLO,TRANS
INTEGER                                    N,K,        LDA,        LDB,             LDC
real                                           ALPHA,A(LDA,*),B(LDB,*),BETA,C(LDC,*)
```

```
SUBROUTINE      F06ZRF   ( UPLO,TRANS,     N,K,ALPHA,A,LDA,   B,LDB,    BETA,C,LDC )
ENTRY           cher2k   ( UPLO,TRANS,     N,K,ALPHA,A,LDA,   B,LDB,    BETA,C,LDC )
CHARACTER*1                UPLO,TRANS
INTEGER                                    N,K,        LDA,      LDB,          LDC
real                                                                      BETA
complex                                       ALPHA,A(LDA,*),B(LDB,*),    C(LDC,*)

SUBROUTINE      F06ZWF   ( UPLO,TRANS,     N,K,ALPHA,A,LDA,   B,LDB,    BETA,C,LDC )
ENTRY           csyr2k   ( UPLO,TRANS,     N,K,ALPHA,A,LDA,   B,LDB,    BETA,C,LDC )
CHARACTER*1                UPLO,TRANS
INTEGER                                    N,K,        LDA,      LDB,          LDC
complex                                       ALPHA,A(LDA,*),B(LDB,*),BETA,C(LDC,*)
```

F06YAF and **F06ZAF** perform the operation indicated in the following table:

	TRANSA = 'N'	TRANSA = 'T'	TRANSA = 'C'
TRANSB = 'N'	$C \leftarrow \alpha AB + \beta C$	$C \leftarrow \alpha A^T B + \beta C$	$C \leftarrow \alpha A^H B + \beta C$
	A is $m \times k$, B is $k \times n$	A is $k \times m$, B is $k \times n$	A is $k \times m$, B is $k \times n$
TRANSB = 'T'	$C \leftarrow \alpha AB^T + \beta C$	$C \leftarrow \alpha A^T B^T + \beta C$	$C \leftarrow \alpha A^H B^T + \beta C$
	A is $m \times k$, B is $n \times k$	A is $k \times m$, B is $n \times k$	A is $k \times m$, B is $n \times k$
TRANSB = 'C'	$C \leftarrow \alpha AB^H + \beta C$	$C \leftarrow \alpha A^T B^H + \beta C$	$C \leftarrow \alpha A^H B^H + \beta C$
	A is $m \times k$, B is $n \times k$	A is $k \times m$, B is $n \times k$	A is $k \times m$, B is $n \times k$

where A and B are general matrices and C is a general m by n matrix.

F06YCF, **F06ZCF** and **F06ZTF** perform the operation indicated in the following table:

SIDE = 'L'	SIDE = 'R'
$C \leftarrow \alpha AB + \beta C$	$C \leftarrow \alpha BA + \beta C$
A is $m \times m$	B is $m \times n$
B is $m \times n$	A is $n \times n$

where A is symmetric for F06YCF and F06ZTF and is Hermitian for F06ZCF, B is a general matrix and C is a general m by n matrix.

F06YFF and **F06ZFF** perform the operation indicated in the following table:

	TRANSA = 'N'	TRANSA = 'T'	TRANSA = 'C'
SIDE = 'L'	$B \leftarrow \alpha AB$	$B \leftarrow \alpha A^T B$	$B \leftarrow \alpha A^H B$
	A is triangular $m \times m$	A is triangular $m \times m$	A is triangular $m \times m$
SIDE = 'R'	$B \leftarrow \alpha BA$	$B \leftarrow \alpha BA^T$	$B \leftarrow \alpha BA^H$
	A is triangular $n \times n$	A is triangular $n \times n$	A is triangular $n \times n$

where B is a general m by n matrix.

F06YJF and **F06ZJF** solve the equations, indicated in the following table, for X:

	TRANSA = 'N'	TRANSA = 'T'	TRANSA = 'C'
SIDE = 'L'	$AX = \alpha B$	$A^T X = \alpha B$	$A^H X = \alpha B$
	A is triangular $m \times m$	A is triangular $m \times m$	A is triangular $m \times m$
SIDE = 'R'	$XA = \alpha B$	$XA^T = \alpha B$	$XA^H = \alpha B$
	A is triangular $n \times n$	A is triangular $n \times n$	A is triangular $n \times n$

where B is a general m by n matrix. The m by n solution matrix X is overwritten on the array B. It is important to note that no test for singularity is included in these routines.

F06YPF, **F06ZPF** and **F06ZUF** perform the operation indicated in the following table:

	TRANS = 'N'	TRANS = 'T'	TRANS = 'C'
F06YPF	$C \leftarrow \alpha AA^T + \beta C$	$C \leftarrow \alpha A^T A + \beta C$	$C \leftarrow \alpha A^T A + \beta C$
F06ZUF	$C \leftarrow \alpha AA^T + \beta C$	$C \leftarrow \alpha A^T A + \beta C$	–
F06ZPF	$C \leftarrow \alpha AA^H + \beta C$	–	$C \leftarrow \alpha A^H A + \beta C$
	A is $n \times k$	A is $k \times n$	A is $k \times n$

where A is a general matrix and C is an n by n symmetric matrix for F06YPF and F06ZUF, and is an n by n Hermitian matrix for F06ZPF.

F06YRF, **F06ZRF** and **F06ZWF** perform the operation indicated in the following table:

	TRANS = 'N'	TRANS = 'T'	TRANS = 'C'
F06YRF	$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$	$C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$	$C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$
F06ZWF	$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$	$C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$	–
F06ZRF	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C$	–	$C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C$
	A and B are $n \times k$	A and B are $k \times n$	A and B are $k \times n$

where A and B are general matrices and C is an n by n symmetric matrix for F06YRF and F06ZWF, and is an n by n Hermitian matrix for F06ZPF.

The following values of arguments are invalid:

Any value of the character arguments SIDE, TRANSA, TRANSB, TRANS, UPLO or DIAG, whose meaning is not specified.

M < 0

N < 0

K < 0

LDA < the number of rows in the matrix A

LDB < the number of rows in the matrix B

LDC < the number of rows in the matrix C

If a routine is called with an invalid value then an error message is output, on the error message unit (see X04AAF), giving the name of the routine and the number of the first invalid argument, and execution is terminated.

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'.

F06QGF F06VGF

6 Indexes of BLAS routines

Real Matrices		
BLAS single precision	BLAS double precision	NAG
ISAMAX	IDAMAX	F06JLF
SASUM	DASUM	F06EKF
SAXPY	DAXPY	F06ECF
SAXPYI	DAXPYI	F06ETF
SCASUM	DCASUM	F06JKF
SCNRM2	DCNRM2	F06JJF
SCOPY	DCOPY	F06EFF
SDOT	DDOT	F06EAF
SDOTI	DDOTI	F06ERF
SGBMV	DGBMV	F06PBF
SGEMM	DGEMM	F06YAF
SGEMV	DGEMV	F06PAF
SGER	DGER	F06PMF
SGTHR	DGTHR	F06EUF
SGTHRZ	DGTHRZ	F06EVF
SNRM2	DNRM2	F06EJF
SROT	DROT	F06EPF
SROTG	DROTG	F06AAF
SROTI	DROTI	F06EXF
SSBMV	DSBMV	F06PDF
SSCAL	DSCAL	F06EDF
SSCTR	DSCTR	F06EWF
SSPMV	DSPMV	F06PEF
SSPR	DSPR	F06PQF
SSPR2	DSPR2	F06PSF
SSWAP	DSWAP	F06EGF
SSYMM	DSYMM	F06YCF
SSYMV	DSYMV	F06PCF
SSYR	DSYR	F06PPF
SSYR2	DSYR2	F06PRF
SSYR2K	DSYR2K	F06YRF
SSYRK	DSYRK	F06YPF
STBMV	DTBMV	F06PGF
STBSV	DTBSV	F06PKF
STPMV	DTPMV	F06PHF
STPSV	DTPSV	F06PLF
STRMM	DTRMM	F06YFF
STRMV	DTRMV	F06PFF
STRSM	DTRSM	F06YJF
STRSV	DTRSV	F06PJF

Complex Matrices		
BLAS single precision	BLAS double precision	NAG
ICAMAX	IZAMAX	F06JMF
CAXPY	ZAXPY	F06GCF
CAXPYI	ZAXPYI	F06GTF
CCOPY	ZCOPY	F06GFF
CDOTC	ZDOTC	F06GBF
CDOTCI	ZDOTCI	F06GSF
CDOTU	ZDOTU	F06GAF
CDOTUI	ZDOTUI	F06GRF
CGBMV	ZGBMV	F06SBF
CGEMM	ZGEMM	F06ZAF
CGEMV	ZGEMV	F06SAF
CGERC	ZGERC	F06SNF
CGERU	ZGERU	F06SMF
CGTHR	ZGTHR	F06GUF
CGTHRZ	ZGTHRZ	F06GVF
CHBMV	ZHBMV	F06SDF
CHEMM	ZHEMM	F06ZCF
CHEMV	ZHEMV	F06SCF
CHER	ZHER	F06SPF
CHER2	ZHER2	F06SRF
CHER2K	ZHER2K	F06ZRF
CHERK	ZHERK	F06ZPF
CHPMV	ZHPMV	F06SEF
CHPR	ZHPR	F06SQF
CHPR2	ZHPR2	F06SSF
CSCAL	ZSCAL	F06GDF
CSCTR	ZSCTR	F06GWF
CSSCAL	ZSSCAL	F06JDF
CSWAP	ZSWAP	F06GGF
CSYMM	ZSYMM	F06ZTF
CSYR2K	ZSYR2K	F06ZWF
CSYRK	ZSYRK	F06ZUF
CTBMV	ZTBMV	F06SGF
CTBSV	ZTBSV	F06SKF
CTPMV	ZTPMV	F06SHF
CTPSV	ZTPSV	F06SLF
CTRMM	ZTRMM	F06ZFF
CTRMV	ZTRMV	F06SFF
CTRSM	ZTRSM	F06ZJF
CTRSV	ZTRSV	F06SJF

7 References

[1] Dodson D S and Grimes R G (1982) Remark on Algorithm 539 *ACM Trans. Math. Software* **8** 403–404

[2] Dodson D S, Grimes R G and Lewis J G (1991) Sparse extensions to the Fortran basic linear algebra subprograms *ACM Trans. Math. Software* **17** 253–263

[3] Dongarra J J, Moler C B, Bunch J R and Stewart G W (1979) *LINPACK Users' Guide* SIAM, Philadelphia

[4] Dongarra J J, Du Croz J J, Hammarling S and Hanson R J (1988) An extended set of FORTRAN basic linear algebra subprograms *ACM Trans. Math. Software* **14** 1–32

[5] Dongarra J J, Du Croz J J, Duff I S and Hammarling S (1990) A set of Level 3 basic linear algebra subprograms *ACM Trans. Math. Software* **16** 1–28

[6] Golub G H and Van Loan C F (1989) *Matrix Computations* Johns Hopkins University Press (2nd Edition), Baltimore

[7] Lawson C L, Hanson R J, Kincaid D R and Krogh F T (1979) Basic linear algebra subprograms for Fortran usage *ACM Trans. Math. Software* **5** 308–325

Chapter F07 – Linear Equations (LAPACK)

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
F07ADF	15	(SGETRF/DGETRF) *LU* factorization of real m by n matrix
F07AEF	15	(SGETRS/DGETRS) Solution of real system of linear equations, multiple right-hand sides, matrix already factorized by F07ADF
F07AGF	15	(SGECON/DGECON) Estimate condition number of real matrix, matrix already factorized by F07ADF
F07AHF	15	(SGERFS/DGERFS) Refined solution with error bounds of real system of linear equations, multiple right-hand sides
F07AJF	15	(SGETRI/DGETRI) Inverse of real matrix, matrix already factorized by F07ADF
F07ARF	15	(CGETRF/ZGETRF) *LU* factorization of complex m by n matrix
F07ASF	15	(CGETRS/ZGETRS) Solution of complex system of linear equations, multiple right-hand sides, matrix already factorized by F07ARF
F07AUF	15	(CGECON/ZGECON) Estimate condition number of complex matrix, matrix already factorized by F07ARF
F07AVF	15	(CGERFS/ZGERFS) Refined solution with error bounds of complex system of linear equations, multiple right-hand sides
F07AWF	15	(CGETRI/ZGETRI) Inverse of complex matrix, matrix already factorized by F07ARF
F07BDF	15	(SGBTRF/DGBTRF) *LU* factorization of real m by n band matrix
F07BEF	15	(SGBTRS/DGBTRS) Solution of real band system of linear equations, multiple right-hand sides, matrix already factorized by F07BDF
F07BGF	15	(SGBCON/DGBCON) Estimate condition number of real band matrix, matrix already factorized by F07BDF
F07BHF	15	(SGBRFS/DGBRFS) Refined solution with error bounds of real band system of linear equations, multiple right-hand sides
F07BRF	15	(CGBTRF/ZGBTRF) *LU* factorization of complex m by n band matrix
F07BSF	15	(CGBTRS/ZGBTRS) Solution of complex band system of linear equations, multiple right-hand sides, matrix already factorized by F07BRF
F07BUF	15	(CGBCON/ZGBCON) Estimate condition number of complex band matrix, matrix already factorized by F07BRF
F07BVF	15	(CGBRFS/ZGBRFS) Refined solution with error bounds of complex band system of linear equations, multiple right-hand sides
F07FDF	15	(SPOTRF/DPOTRF) Cholesky factorization of real symmetric positive-definite matrix
F07FEF	15	(SPOTRS/DPOTRS) Solution of real symmetric positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07FDF
F07FGF	15	(SPOCON/DPOCON) Estimate condition number of real symmetric positive-definite matrix, matrix already factorized by F07FDF
F07FHF	15	(SPORFS/DPORFS) Refined solution with error bounds of real symmetric positive-definite system of linear equations, multiple right-hand sides
F07FJF	15	(SPOTRI/DPOTRI) Inverse of real symmetric positive-definite matrix, matrix already factorized by F07FDF
F07FRF	15	(CPOTRF/ZPOTRF) Cholesky factorization of complex Hermitian positive-definite matrix
F07FSF	15	(CPOTRS/ZPOTRS) Solution of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07FRF

F07FUF	15	(CPOCON/ZPOCON) Estimate condition number of complex Hermitian positive-definite matrix, matrix already factorized by F07FRF
F07FVF	15	(CPORFS/ZPORFS) Refined solution with error bounds of complex Hermitian positive-definite system of linear equations, multiple right-hand sides
F07FWF	15	(CPOTRI/ZPOTRI) Inverse of complex Hermitian positive-definite matrix, matrix already factorized by F07FRF
F07GDF	15	(SPPTRF/DPPTRF) Cholesky factorization of real symmetric positive-definite matrix, packed storage
F07GEF	15	(SPPTRS/DPPTRS) Solution of real symmetric positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07GDF, packed storage
F07GGF	15	(SPPCON/DPPCON) Estimate condition number of real symmetric positive-definite matrix, matrix already factorized by F07GDF, packed storage
F07GHF	15	(SPPRFS/DPPRFS) Refined solution with error bounds of real symmetric positive-definite system of linear equations, multiple right-hand sides, packed storage
F07GJF	15	(SPPTRI/DPPTRI) Inverse of real symmetric positive-definite matrix, matrix already factorized by F07GDF, packed storage
F07GRF	15	(CPPTRF/ZPPTRF) Cholesky factorization of complex Hermitian positive-definite matrix, packed storage
F07GSF	15	(CPPTRS/ZPPTRS) Solution of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, matrix already factorized by F07GRF, packed storage
F07GUF	15	(CPPCON/ZPPCON) Estimate condition number of complex Hermitian positive-definite matrix, matrix already factorized by F07GRF, packed storage
F07GVF	15	(CPPRFS/ZPPRFS) Refined solution with error bounds of complex Hermitian positive-definite system of linear equations, multiple right-hand sides, packed storage
F07GWF	15	(CPPTRI/ZPPTRI) Inverse of complex Hermitian positive-definite matrix, matrix already factorized by F07GRF, packed storage
F07HDF	15	(SPBTRF/DPBTRF) Cholesky factorization of real symmetric positive-definite band matrix
F07HEF	15	(SPBTRS/DPBTRS) Solution of real symmetric positive-definite band system of linear equations, multiple right-hand sides, matrix already factorized by F07HDF
F07HGF	15	(SPBCON/DPBCON) Estimate condition number of real symmetric positive-definite band matrix, matrix already factorized by F07HDF
F07HHF	15	(SPBRFS/DPBRFS) Refined solution with error bounds of real symmetric positive-definite band system of linear equations, multiple right-hand sides
F07HRF	15	(CPBTRF/ZPBTRF) Cholesky factorization of complex Hermitian positive-definite band matrix
F07HSF	15	(CPBTRS/ZPBTRS) Solution of complex Hermitian positive-definite band system of linear equations, multiple right-hand sides, matrix already factorized by F07HRF
F07HUF	15	(CPBCON/ZPBCON) Estimate condition number of complex Hermitian positive-definite band matrix, matrix already factorized by F07HRF
F07HVF	15	(CPBRFS/ZPBRFS) Refined solution with error bounds of complex Hermitian positive-definite band system of linear equations, multiple right-hand sides
F07MDF	15	(SSYTRF/DSYTRF) Bunch–Kaufman factorization of real symmetric indefinite matrix
F07MEF	15	(SSYTRS/DSYTRS) Solution of real symmetric indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07MDF

F07MGF	15	(SSYCON/DSYCON) Estimate condition number of real symmetric indefinite matrix, matrix already factorized by F07MDF
F07MHF	15	(SSYRFS/DSYRFS) Refined solution with error bounds of real symmetric indefinite system of linear equations, multiple right-hand sides
F07MJF	15	(SSYTRI/DSYTRI) Inverse of real symmetric indefinite matrix, matrix already factorized by F07MDF
F07MRF	15	(CHETRF/ZHETRF) Bunch–Kaufman factorization of complex Hermitian indefinite matrix
F07MSF	15	(CHETRS/ZHETRS) Solution of complex Hermitian indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07MRF
F07MUF	15	(CHECON/ZHECON) Estimate condition number of complex Hermitian indefinite matrix, matrix already factorized by F07MRF
F07MVF	15	(CHERFS/ZHERFS) Refined solution with error bounds of complex Hermitian indefinite system of linear equations, multiple right-hand sides
F07MWF	15	(CHETRI/ZHETRI) Inverse of complex Hermitian indefinite matrix, matrix already factorized by F07MRF
F07NRF	15	(CSYTRF/ZSYTRF) Bunch–Kaufman factorization of complex symmetric matrix
F07NSF	15	(CSYTRS/ZSYTRS) Solution of complex symmetric system of linear equations, multiple right-hand sides, matrix already factorized by F07NRF
F07NUF	15	(CSYCON/ZSYCON) Estimate condition number of complex symmetric matrix, matrix already factorized by F07NRF
F07NVF	15	(CSYRFS/ZSYRFS) Refined solution with error bounds of complex symmetric system of linear equations, multiple right-hand sides
F07NWF	15	(CSYTRI/ZSYTRI) Inverse of complex symmetric matrix, matrix already factorized by F07NRF
F07PDF	15	(SSPTRF/DSPTRF) Bunch–Kaufman factorization of real symmetric indefinite matrix, packed storage
F07PEF	15	(SSPTRS/DSPTRS) Solution of real symmetric indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07PDF, packed storage
F07PGF	15	(SSPCON/DSPCON) Estimate condition number of real symmetric indefinite matrix, matrix already factorized by F07PDF, packed storage
F07PHF	15	(SSPRFS/DSPRFS) Refined solution with error bounds of real symmetric indefinite system of linear equations, multiple right-hand sides, packed storage
F07PJF	15	(SSPTRI/DSPTRI) Inverse of real symmetric indefinite matrix, matrix already factorized by F07PDF, packed storage
F07PRF	15	(CHPTRF/ZHPTRF) Bunch–Kaufman factorization of complex Hermitian indefinite matrix, packed storage
F07PSF	15	(CHPTRS/ZHPTRS) Solution of complex Hermitian indefinite system of linear equations, multiple right-hand sides, matrix already factorized by F07PRF, packed storage
F07PUF	15	(CHPCON/ZHPCON) Estimate condition number of complex Hermitian indefinite matrix, matrix already factorized by F07PRF, packed storage
F07PVF	15	(CHPRFS/ZHPRFS) Refined solution with error bounds of complex Hermitian indefinite system of linear equations, multiple right-hand sides, packed storage
F07PWF	15	(CHPTRI/ZHPTRI) Inverse of complex Hermitian indefinite matrix, matrix already factorized by F07PRF, packed storage
F07QRF	15	(CSPTRF/ZSPTRF) Bunch–Kaufman factorization of complex symmetric matrix, packed storage
F07QSF	15	(CSPTRS/ZSPTRS) Solution of complex symmetric system of linear equations, multiple right-hand sides, matrix already factorized by F07QRF, packed storage

F07QUF	15	(CSPCON/ZSPCON) Estimate condition number of complex symmetric matrix, matrix already factorized by F07QRF, packed storage
F07QVF	15	(CSPRFS/ZSPRFS) Refined solution with error bounds of complex symmetric system of linear equations, multiple right-hand sides, packed storage
F07QWF	15	(CSPTRI/ZSPTRI) Inverse of complex symmetric matrix, matrix already factorized by F07QRF, packed storage
F07TEF	15	(STRTRS/DTRTRS) Solution of real triangular system of linear equations, multiple right-hand sides
F07TGF	15	(STRCON/DTRCON) Estimate condition number of real triangular matrix
F07THF	15	(STRRFS/DTRRFS) Error bounds for solution of real triangular system of linear equations, multiple right-hand sides
F07TJF	15	(STRTRI/DTRTRI) Inverse of real triangular matrix
F07TSF	15	(CTRTRS/ZTRTRS) Solution of complex triangular system of linear equations, multiple right-hand sides
F07TUF	15	(CTRCON/ZTRCON) Estimate condition number of complex triangular matrix
F07TVF	15	(CTRRFS/ZTRRFS) Error bounds for solution of complex triangular system of linear equations, multiple right-hand sides
F07TWF	15	(CTRTRI/ZTRTRI) Inverse of complex triangular matrix
F07UEF	15	(STPTRS/DTPTRS) Solution of real triangular system of linear equations, multiple right-hand sides, packed storage
F07UGF	15	(STPCON/DTPCON) Estimate condition number of real triangular matrix, packed storage
F07UHF	15	(STPRFS/DTPRFS) Error bounds for solution of real triangular system of linear equations, multiple right-hand sides, packed storage
F07UJF	15	(STPTRI/DTPTRI) Inverse of real triangular matrix, packed storage
F07USF	15	(CTPTRS/ZTPTRS) Solution of complex triangular system of linear equations, multiple right-hand sides, packed storage
F07UUF	15	(CTPCON/ZTPCON) Estimate condition number of complex triangular matrix, packed storage
F07UVF	15	(CTPRFS/ZTPRFS) Error bounds for solution of complex triangular system of linear equations, multiple right-hand sides, packed storage
F07UWF	15	(CTPTRI/ZTPTRI) Inverse of complex triangular matrix, packed storage
F07VEF	15	(STBTRS/DTBTRS) Solution of real band triangular system of linear equations, multiple right-hand sides
F07VGF	15	(STBCON/DTBCON) Estimate condition number of real band triangular matrix
F07VHF	15	(STBRFS/DTBRFS) Error bounds for solution of real band triangular system of linear equations, multiple right-hand sides
F07VSF	15	(CTBTRS/ZTBTRS) Solution of complex band triangular system of linear equations, multiple right-hand sides
F07VUF	15	(CTBCON/ZTBCON) Estimate condition number of complex band triangular matrix
F07VVF	15	(CTBRFS/ZTBRFS) Error bounds for solution of complex band triangular system of linear equations, multiple right-hand sides

Chapter F07

Linear Equations (LAPACK)

Contents

1 Scope of the Chapter

This chapter provides routines for the solution of systems of simultaneous linear equations, and associated computations. It provides routines for:

- matrix factorizations
- solution of linear equations
- estimating matrix condition numbers
- computing error bounds for the solution of linear equations
- matrix inversion

Routines are provided for both *real* and *complex* data.

For a general introduction to the solution of systems of linear equations, you should turn first to the F04 Chapter Introduction. The decision trees, at the end of the F04 Chapter Introduction, direct you to the most appropriate routines in Chapter F04 or F07, for solving your particular problem. In particular, Chapter F04 contains *Black Box* routines which enable some standard types of problem to be solved by a call to a single routine. Where possible, routines in Chapter F04 call F07 routines to perform the necessary computational tasks.

The routines in this chapter (F07) handle only *dense* and *band* matrices (not matrices with more specialized structures, or general sparse matrices).

The routines in this chapter have all been derived from the LAPACK project (see Anderson *et al.* [1]). They have been designed to be efficient on a wide range of high-performance computers, without compromising efficiency on conventional serial machines.

2 Background to the Problems

This section is only a brief introduction to the numerical solution of systems of linear equations. Consult a standard textbook for a more thorough discussion, for example Golub and Van Loan [2].

2.1 Notation

We use the standard notation for a system of simultaneous linear equations:

$$Ax = b \tag{1}$$

where A is the *coefficient matrix*, b is the *right-hand side*, and x is the *solution*. A is assumed to be a square matrix of order n.

If there are several right-hand sides, we write

$$AX = B \tag{2}$$

where the columns of B are the individual right-hand sides, and the columns of X are the corresponding solutions.

We also use the following notation, both here and in the routine documents:

$\hat{x}$	a *computed* solution to $Ax = b$, (which usually differs from the exact solution x because of round-off error)
$r = b - A\hat{x}$	the *residual* corresponding to the computed solution $\hat{x}$
$\lVert x\rVert_\infty = \max_i \lvert x_i\rvert$	the infinity-norm of the vector x
$\lVert A\rVert_\infty = \max_i \sum_j \lvert a_{ij}\rvert$	the infinity-norm of the vector A
$\lvert x\rvert$	the vector with elements $\lvert x_i\rvert$
$\lvert A\rvert$	the matrix with elements $\lvert a_{ij}\rvert$

Inequalities of the form $|A| \leq |B|$ are interpreted componentwise, that is $|a_{ij}| \leq |b_{ij}|$ for all i, j.

2.2 Matrix Factorizations

If A is upper or lower triangular, $Ax = b$ can be solved by a straightforward process of backward or forward substitution.

Otherwise, the solution is obtained after first factorizing A, as follows:

General matrices (*LU* factorization with partial pivoting):

$$A = PLU$$

where P is a permutation matrix, L is lower-triangular with diagonal elements equal to 1, and U is upper-triangular; the permutation matrix P (which represents row interchanges) is needed to ensure numerical stability.

Symmetric positive-definite matrices (Cholesky factorization):

$$A = U^T U \text{ or } A = LL^T$$

where U is upper triangular and L is lower triangular.

Symmetric indefinite matrices (Bunch–Kaufman factorization):

$$A = PUDU^T P^T \text{ or } A = PLDL^T P^T$$

where P is a permutation matrix, U is upper triangular, L is lower triangular, and D is a block diagonal matrix with diagonal blocks of order 1 or 2; U and L have diagonal elements equal to 1, and have 2 by 2 unit matrices on the diagonal corresponding to the 2 by 2 blocks of D. The permutation matrix P (which represents symmetric row-and-column interchanges) and the 2 by 2 blocks in D are needed to ensure numerical stability. If A is in fact positive-definite, no interchanges are needed and the factorization reduces to $A = UDU^T$ or $A = LDL^T$ with diagonal D, which is simply a variant form of the Cholesky factorization.

2.3 Solution of Systems of Equations

Given one of the above matrix factorizations, it is straightforward to compute a solution to $Ax = b$ by solving two subproblems, as shown below, first for y and then for x. Each subproblem consists essentially of solving a triangular system of equations by forward or backward substitution; the permutation matrix P and the block diagonal matrix D introduce only a little extra complication:

General matrices (*LU* factorization):

$$\begin{array}{c} Ly = P^T b \\ Ux = y \end{array}$$

Symmetric positive-definite matrices (Cholesky factorization):

$$\begin{array}{c} U^T y = b \\ Ux = y \end{array} \text{ or } Ly = bL^T x = y$$

Symmetric indefinite matrices (Bunch–Kaufman factorization):

$$\begin{array}{c} PUDy = b \\ U^T P^T x = y \end{array} \text{ or } \begin{array}{c} PLDy = b \\ L^T P^T x = y \end{array}$$

2.4 Sensitivity and Error Analysis

2.4.1 Normwise error bounds

Frequently in practical problems, the data A and b are not known exactly, and it is then important to understand how uncertainties or perturbations in the data can affect the solution.

If x is the exact solution to $Ax = b$, and $x + \delta x$ is the exact solution to a perturbed problem $(A + \delta A)(x + \delta x) = (b + \delta b)$, then:

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|}\right) + \ldots \text{(2nd order terms)}$$

where $\kappa(A)$ is the *condition number* of A defined by:

$$\kappa(A) = ||A||.||A^{-1}||. \tag{3}$$

In other words, relative errors in A or b may be amplified in x by a factor $\kappa(A)$. Section 2.4.2 discusses how to compute or estimate $\kappa(A)$.

Similar considerations apply when we study the effects of *rounding errors* introduced by computation in finite precision. The effects of rounding errors can be shown to be equivalent to perturbations in the original data, such that $\frac{||\delta A||}{||A||}$ and $\frac{||\delta b||}{||b||}$ are usually at most $p(n)\epsilon$, where ϵ is the ***machine precision*** and $p(n)$ is an increasing function of n which is seldom larger than $10n$ (although in theory it can be as large as 2^{n-1}).

In other words, the computed solution $\hat{x}$ is the exact solution of a linear system $(A+\delta A)\hat{x} = b+\delta b$ which is close to the original system in a normwise sense.

2.4.2 Estimating condition numbers

The previous section has emphasized the usefulness of the quantity $\kappa(A)$ in understanding the sensitivity of the solution of $Ax = b$. To compute the value of $\kappa(A)$ from equation (3) is more expensive than solving $Ax = b$ in the first place. Hence it is standard practice to *estimate* $\kappa(A)$, in either the 1-norm or the ∞-norm, by a method which only requires $O(n^2)$ additional operations, assuming that a suitable factorization of A is available.

The method used in this chapter is Higham's modification of Hager's method [3]. It yields an estimate which is never larger than the true value, but which seldom falls short by more than a factor of 3 (although artificial examples can be constructed where it is much smaller). This is acceptable since it is the order of magnitude of $\kappa(A)$ which is important rather than its precise value.

Because $\kappa(A)$ is infinite if A is singular, the routines in this chapter actually return the *reciprocal* of $\kappa(A)$.

2.4.3 Componentwise error bounds

A disadvantage of normwise error bounds is that they do not reflect any special structure in the data A and b – that is, a pattern of elements which are known to be zero – and the bounds are dominated by the largest elements in the data.

Componentwise error bounds overcome these limitations. Instead of the normwise relative error, we can bound the relative error in *each component* of A and b:

$$\max_{ijk}\left(\frac{|\delta a_{ij}|}{|a_{ij}|}, \frac{|\delta b_k|}{|b_k|}\right) \le \omega$$

where the *componentwise backward error bound* ω is given by:

$$\omega = \max_i \frac{|r_i|}{(|A|.|\hat{x}| + |b|)_i}.$$

Routines are provided in this chapter which compute ω, and also compute a *forward error bound* which is sometimes much sharper than the normwise bound given earlier:

$$\frac{||x-\hat{x}||_\infty}{||x||_\infty} \le \frac{||\,|A^{-1}|.|r|\,||_\infty}{||x||_\infty}.$$

Care is taken when computing this bound to allow for rounding errors in computing r. The norm $||\,|A^{-1}|.|r|\,||_\infty$ is estimated cheaply (without computing A^{-1}) by a modification of the method used to estimate $\kappa(A)$.

2.4.4 Iterative refinement of the solution

If $\hat{x}$ is an approximate computed solution to $Ax = b$, and r is the corresponding residual, then a procedure for *iterative refinement* of $\hat{x}$ can be defined as follows, starting with $x_0 = \hat{x}$:

for $i = 0, 1, \ldots$, until convergence

compute $r_i = b - Ax_i$
solve $Ad_i = r_i$
compute $x_{i+1} = x_i + d_i$

In Chapter F04, routines are provided which perform this procedure using ***additional precision*** to compute r, and are thus able to reduce the *forward error* to the level of ***machine precision***.

The routines in this chapter do *not* use ***additional precision*** to compute r, and cannot guarantee a small forward error, but can guarantee a *small backward error* (except in rare cases when A is very ill-conditioned, or when A and x are sparse in such a way that $|A|.|x|$ has a zero or very small component). The iterations continue until the backward error has been reduced as much as possible; usually only one iteration is needed, and at most five iterations are allowed.

2.5 Matrix Inversion

It is seldom necessary to compute an explicit inverse of a matrix. In particular, do *not* attempt to solve $Ax = b$ by first computing A^{-1} and then forming the matrix-vector product $x = A^{-1}b$; the procedure described in Section 2.3 is more efficient and more accurate.

However, routines are provided for the rare occasions when an inverse is needed, using one of the factorizations described in Section 2.2.

2.6 Packed Storage

Routines which handle symmetric matrices are usually designed so that they use either the upper or lower triangle of the matrix; it is not necessary to store the whole matrix. If the upper or lower triangle is stored conventionally in the upper or lower triangle of a 2-dimensional array, the remaining elements of the array can be used to store other useful data. However, that is not always convenient, and if it is important to economize on storage, the upper or lower triangle can be stored in a 1-dimensional array of length $n(n+1)/2$ – in other words, the storage is almost halved.

This storage format is referred to as *packed storage;* it is described in Section 3.3.2. It may also be used for triangular matrices.

Routines designed for packed storage perform the same number of arithmetic operations as routines which use conventional storage, but they are usually less efficient, especially on high-performance computers, so there is then a trade-off between storage and efficiency.

2.7 Band Matrices

A *band* matrix is one whose non-zero elements are confined to a relatively small number of sub-diagonals or super-diagonals on either side of the main diagonal. Algorithms can take advantage of bandedness to reduce the amount of work and storage required. The storage scheme used for band matrices is described in Section 3.3.3.

The LU factorization for general matrices, and the Cholesky factorization for symmetric positive-definite matrices both preserve bandedness. Hence routines are provided which take advantage of the band structure when solving systems of linear equations.

The Cholesky factorization preserves bandedness in a very precise sense: the factor U or L has the same number of super-diagonals or sub-diagonals as the original matrix. In the LU factorization, the row-interchanges modify the band structure: if A has k_l sub-diagonals and k_u super-diagonals, then L is not a band matrix but still has at most k_l non-zero elements below the diagonal in each column; and U has at most $k_l + k_u$ super-diagonals.

The Bunch–Kaufman factorization does not preserve bandedness, because of the need for symmetric row-and-column permutations; hence no routines are provided for symmetric indefinite band matrices.

The inverse of a band matrix does not in general have a band structure, so no routines are provided for computing inverses of band matrices.

2.8 Block Algorithms

Many of the routines in this chapter use what is termed a *block algorithm*. This means that at each major step of the algorithm a *block* of rows or columns is updated, and most of the computation is performed by matrix-matrix operations on these blocks. The matrix-matrix operations are performed by calls to the Level 3 BLAS (see Chapter F06), which are the key to achieving high performance on many modern computers. See Golub and Van Loan [2] or Anderson *et al.* [1] for more about block algorithms.

The performance of a block algorithm varies to some extent with the ***blocksize*** – that is, the number of rows or columns per block. This is a machine-dependent parameter, which is set to a suitable value when the library is implemented on each range of machines. Users of the library do not normally need to be aware of what value is being used. Different block sizes may be used for different routines. Values in the range 16 to 64 are typical.

On more conventional machines there is often no advantage from using a block algorithm, and then the routines use an *unblocked* algorithm (effectively a blocksize of 1), relying solely on calls to the Level 2 BLAS (see Chapter F06 again).

The only situation in which a user needs some awareness of the block size is when it affects the amount of workspace to be supplied to a particular routine. This is discussed in Section 3.4.3.

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 Available Routines

Tables 1 and 2 in Section 3.5 show the routines which are provided for performing different computations on different types of matrices. Table 1 shows routines for real matrices; Table 2 shows routines for complex matrices. Each entry in the table gives the NAG routine name, the LAPACK single precision name, and the LAPACK double precision name (see Section 3.2).

Routines are provided for the following types of matrix:

general

general band

symmetric or Hermitian positive-definite

symmetric or Hermitian positive-definite (packed storage)

symmetric or Hermitian positive-definite band

symmetric or Hermitian indefinite

symmetric or Hermitian indefinite (packed storage)

triangular

triangular (packed storage)

triangular band

For each of the above types of matrix (except where indicated), routines are provided to perform the following computations:

(a) (except for triangular matrices) factorize the matrix (see Section 2.2).
(b) solve a system of linear equations, using the factorization (see Section 2.3).
(c) estimate the condition number of the matrix, using the factorization (see Section 2.4.2); these routines also require the norm of the original matrix (except when the matrix is triangular) which may be computed by a routine in Chapter F06.
(d) refine the solution and compute forward and backward error bounds (see Section 2.4.3 and Section 2.4.4); these routines require the original matrix and right-hand side, as well as the factorization returned from (a) and the solution returned from (b).
(e) (except for band matrices) invert the matrix, using the factorization (see Section 2.5).

Thus, to solve a particular problem, it is usually necessary to call two or more routines in succession. This is illustrated in the example programs in the routine documents.

3.2 NAG Names and LAPACK Names

As well as the NAG routine name (beginning F07-), Tables 1 and 2 show the LAPACK routine names in both single and double precision.

The routines may be called either by their NAG names or by their LAPACK names. When using a single precision implementation of the NAG Library, the single precision form of the LAPACK name must be used (beginning with S- or C-); when using a double precision implementation of the NAG Library, the double precision form of the LAPACK name must be used (beginning with D- or Z-).

References to F07 routines in the Manual normally include the LAPACK single and double precision names, in that order – for example, F07ADF (SGETRF/DGETRF).

The LAPACK routine names follow a simple scheme (which is similar to that used for the BLAS in Chapter F06). Each name has the structure **XYYZZZ**, where the components have the following meanings:

– the initial letter **X** indicates the data type (real or complex) and precision:

S – real, single precision (in Fortran 77, `REAL`)
D – real, double precision (in Fortran 77, `DOUBLE PRECISION`)
C – complex, single precision (in Fortran 77, `COMPLEX`)
Z – complex, double precision (in Fortran 77, `COMPLEX*16` or `DOUBLE COMPLEX`)

– the 2nd and 3rd letters **YY** indicate the type of the matrix A (and in some cases its storage scheme):

GE – general
GB – general band
PO – symmetric or Hermitian positive-definite
PP – symmetric or Hermitian positive-definite (packed storage)
PB – symmetric or Hermitian positive-definite band
SY – symmetric indefinite
SP – symmetric indefinite (packed storage)
HE – (complex) Hermitian indefinite
HP – (complex) Hermitian indefinite (packed storage)
TR – triangular
TP – triangular (packed storage)
TB – triangular band

– the last 3 letters **ZZZ** indicate the computation performed:

TRF – triangular factorization
TRS – solution of linear equations, using the factorization
CON – estimate condition number
RFS – refine solution and compute error bounds
TRI – compute inverse, using the factorization

Thus the routine SGETRF performs a triangular factorization of a real general matrix in a single precision implementation of the Library; the corresponding routine in a double precision implementation is DGETRF.

Some sections of the routine documents – Section 2 (Specification) and Section 9.1 (Example program) – print the LAPACK name in ***bold italics***, according to the NAG convention of using bold italics for precision-dependent terms – for example, ***sgetrf***, which should be interpreted as either SGETRF (in single precision) or DGETRF (in double precision).

3.3 Matrix Storage Schemes

In this chapter the following different storage schemes are used for matrices:

- conventional storage in a 2-dimensional array;
- packed storage for symmetric, Hermitian or triangular matrices;

– band storage for band matrices;

These storage schemes are compatible with those used in Chapter F06 (especially in the BLAS) and Chapter F08, but different schemes for packed or band storage are used in a few older routines in Chapters F01, F02, F03 and F04.

In the examples below, $*$ indicates an array element which need not be set and is not referenced by the routines. The examples illustrate only the relevant leading rows and columns of the arrays; array arguments may of course have additional rows or columns, according to the usual rules for passing array arguments in Fortran 77.

3.3.1 Conventional storage

The default scheme for storing matrices is the obvious one: a matrix A is stored in a 2-dimensional array A, with matrix element a_{ij} stored in array element A(i,j).

If a matrix is **triangular** (upper or lower, as specified by the argument UPLO), only the elements of the relevant triangle are stored; the remaining elements of the array need not be set. Such elements are indicated by $*$ in the examples below. For example, when $n = 4$:

UPLO	**Triangular matrix** A	**Storage in array A**
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

Routines which handle **symmetric** or **Hermitian** matrices allow for either the upper or lower triangle of the matrix (as specified by UPLO) to be stored in the corresponding elements of the array; the remaining elements of the array need not be set. For example, when $n = 4$:

UPLO	**Hermitian matrix** A	**Storage in array A**
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} \\ \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

3.3.2 Packed Storage

Symmetric, Hermitian or triangular matrices may be stored more compactly, if the relevant triangle (again as specified by UPLO) is packed by columns in a 1-dimensional array. In Chapters F07 and F08, arrays which hold matrices in packed storage, have names ending in P. So:

if UPLO = 'U', a_{ij} is stored in AP$(i + j(j-1)/2)$ for $i \le j$;

if UPLO = 'L', a_{ij} is stored in AP$(i + (2n-j)(j-1)/2)$ for $j \le i$.

For example:

UPLO	Triangle of matrix A	Packed storage in array AP
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$a_{11}\ \underbrace{a_{12}a_{22}}\ \underbrace{a_{13}a_{23}a_{33}}\ \underbrace{a_{14}a_{24}a_{34}a_{44}}$
'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}a_{21}a_{31}a_{41}}\ \underbrace{a_{22}a_{32}a_{42}}\ \underbrace{a_{33}a_{43}}\ a_{44}$

Note that for real symmetric matrices, packing the upper triangle by columns is equivalent to packing the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the upper triangle by rows. (For complex Hermitian matrices, the only difference is that the off-diagonal elements are conjugated.)

3.3.3 Band storage

A band matrix with k_l sub-diagonals and k_u super-diagonals may be stored compactly in a 2-dimensional array with $k_l + k_u + 1$ rows and n columns. Columns of the matrix are stored in corresponding columns of the array, and diagonals of the matrix are stored in rows of the array. This storage scheme should be used in practice only if k_l, $k_u \ll n$, although the routines in Chapters F07 and F08 work correctly for all values of k_l and k_u. In Chapters F07 and F08 arrays which hold matrices in band storage have names ending in B.

To be precise, a_{ij} is stored in AB$(k_u + 1 + i - j, j)$ for $\max(1, j - k_u) \leq i \leq \min(n, j + k_l)$. For example, when $n = 5$, $k_l = 2$ and $k_u = 1$:

Band matrix A	Band storage in array AB
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

The elements marked * in the upper left and lower right corners of the array AB need not be set, and are not referenced by the routines.

Note. when a general band matrix is supplied for LU factorization, space must be allowed to store an additional k_l super-diagonals, generated by fill-in as a result of row interchanges. This means that the matrix is stored according to the above scheme, but with $k_l + k_u$ super-diagonals.

Triangular band matrices are stored in the same format, with either $k_l = 0$ if upper triangular, or $k_u = 0$ if lower triangular.

For symmetric or Hermitian band matrices with k sub-diagonals or super-diagonals, only the upper or lower triangle (as specified by UPLO) need be stored:

if UPLO = 'U', a_{ij} is stored in AB$(k + 1 + i - j, j)$ for $\max(1, j - k) \leq i \leq j$;

if UPLO = 'L', a_{ij} is stored in AB$(1 + i - j, j)$ for $j \leq i \leq \min(n, j + k)$.

For example, when $n = 5$ and $k = 2$:

UPLO	Hermitian band matrix A	Band storage in array A
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} & \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} & a_{35} \\ & \bar{a}_{24} & \bar{a}_{34} & a_{44} & a_{45} \\ & & \bar{a}_{35} & \bar{a}_{45} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & & \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} & \bar{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \bar{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

Note that different storage schemes for band matrices are used by some routines in Chapters F01, F02, F03 and F04.

3.3.4 Unit triangular matrices

Some routines in this chapter have an option to handle unit triangular matrices (that is, triangular matrices with diagonal elements = 1). This option is specified by an argument DIAG. If DIAG = 'U' (Unit triangular), the diagonal elements of the matrix need not be stored, and the corresponding array elements are not referenced by the routines. The storage scheme for the rest of the matrix (whether conventional, packed or band) remains unchanged.

3.3.5 Real diagonal elements of complex matrices

Complex Hermitian matrices have diagonal elements that are by definition purely real. In addition, complex triangular matrices which arise in Cholesky factorization are defined by the algorithm to have real diagonal elements.

If such matrices are supplied as input to routines in this chapter, the imaginary parts of the diagonal elements are not referenced, but are assumed to be zero. If such matrices are returned as output by the routines, the computed imaginary parts are explicitly set to zero.

3.4 Parameter Conventions

3.4.1 Option parameters

Most routines in this chapter have one or more option parameters, of type CHARACTER. The descriptions in Section 5 of the routine documents refer only to upper-case values (for example 'U' or 'L'); however, in every case, the corresponding lower-case characters may be supplied (with the same meaning). Any other value is illegal.

A longer character string can be passed as the actual parameter, making the calling program more readable, but only the first character is significant. (This is a feature of Fortran 77.) For example:

```
CALL SGETRS ('Transpose', . . . )
```

3.4.2 Problem dimensions

It is permissible for the problem dimensions (for example, M, N or NRHS) to be passed as zero, in which case the computation (or part of it) is skipped. Negative dimensions are regarded as an error.

3.4.3 Length of work arrays

A few routines implementing block algorithms require workspace sufficient to hold one block of rows or columns of the matrix if they are to achieve optimum levels of performance — for example, workspace of size $n \times nb$, where nb is the optimum block size. In such cases, the actual declared length of the work array must be passed as a separate parameter LWORK, which immediately follows WORK in the parameter-list.

The routine will still perform correctly when less workspace is provided: it uses the largest block size allowed by the amount of workspace supplied, as long as this is likely to give better performance than the unblocked algorithm. On exit, WORK(1) contains the minimum value of LWORK which would allow the routine to use the optimum block size; this value of LWORK can be used for subsequent runs.

If LWORK indicates that there is insufficient workspace to perform the unblocked algorithm, this is regarded as an illegal value of LWORK, and is treated like any other illegal parameter value (see Section 3.4.4).

If you are in doubt how much workspace to supply and are concerned to achieve optimum performance, supply a generous amount (assume a block size of 64, say), and then examine the value of WORK(1) on exit.

3.4.4 Error-handling and the diagnostic parameter INFO

Routines in this chapter do not use the usual NAG Library error-handling mechanism, involving the parameter IFAIL. Instead they have a diagnostic parameter INFO. (Thus they preserve complete compatibility with the LAPACK specification.)

Whereas IFAIL is an *Input/Output* parameter and must be set before calling a routine, INFO is purely an *Output* parameter and need not be set before entry.

INFO indicates the success or failure of the computation, as follows:

INFO = 0: successful termination

INFO > 0: failure in the course of computation, control returned to the calling program

If the routine document specfies that the routine may terminate with INFO>0, then it is **essential** to **test INFO on exit** from the routine. (This corresponds to a *soft failure* in terms of the usual NAG error-handling terminology.) No error message is output.

All routines check that input parameters such as N or LDA or option parameters of type CHARACTER have permitted values. If an illegal value of the ith parameter is detected, INFO is set to $-i$, a message is output, and execution of the program is terminated. (This corresponds to a *hard failure* in the usual NAG terminology.)

3.5 Tables of Available Routines

Routines for real matrices					
Type of matrix and storage scheme	**factorize**	**solve**	**condition number**	**error estimate**	**invert**
general	F07ADF SGETRF DGETRF	F07AEF SGETRS DGETRS	F07AGF SGECON DGECON	F07AHF SGERFS DGERFS	F07AJF SGETRI DGETRI
general band	F07BDF SGBTRF DGBTRF	F07BEF SGBTRS DGBTRS	F07BGF SGBCON DGBCON	F07BHF SGBRFS DGBRFS	
symmetric positive-definite	F07FDF SPOTRF DPOTRF	F07FEF SPOTRS DPOTRS	F07FGF SPOCON DPOCON	F07FHF SPORFS DPORFS	F07FJF SPOTRI DPOTRI
symmetric positive-definite (packed storage)	F07GDF SPPTRF DPPTRF	F07GEF SPPTRS DPPTRS	F07GGF SPPCON DPPCON	F07GHF SPPRFS DPPRFS	F07GJF SPPTRI DPPTRI
symmetric positive-definite band	F07HDF SPBTRF DPBTRF	F07HEF SPBTRS DPBTRS	F07HGF SPBCON DPBCON	F07HHF SPBRFS DPBRFS	
symmetric indefinite	F07MDF SSYTRF DSYTRF	F07MEF SSYTRS DSYTRS	F07MGF SSYCON DSYCON	F07MHF SSYRFS DSYRFS	F07MJF SSYTRI DSYTRI
symmetric indefinite (packed storage)	F07PDF SSPTRF DSPTRF	F07PEF SSPTRS DSPTRS	F07PGF SSPCON DSPCON	F07PHF SSPRFS DSPRFS	F07PJF SSPTRI DSPTRI
triangular		F07TEF STRTRS DTRTRS	F07TGF STRCON DTRCON	F07THF STRRFS DTRRFS	F07TJF STRTRI DTRTRI
triangular (packed storage)		F07UEF STPTRS DTPTRS	F07UGF STPCON DTPCON	F07UHF STPRFS DTPRFS	F07UJF STPTRI DTPTRI
triangular band		F07VEF STBTRS DTBTRS	F07VGF STBCON DTBCON	F07VHF STBRFS DTBRFS	

Table 1

Each entry gives:

the NAG routine name
the LAPACK routine name in a single precision implementation
the LAPACK routine name in a double precision implementation

Routines for complex matrices					
Type of matrix and storage scheme	**factorize**	**solve**	**condition number**	**error estimate**	**invert**
general	F07ARF CGETRF ZGETRF	F07ASF CGETRS ZGETRS	F07AUF CGECON ZGECON	F07AVF CGERFS ZGERFS	F07AWF CGETRI ZGETRI
general band	F07BRF CGBTRF ZGBTRF	F07BSF CGBTRS ZGBTRS	F07BUF CGBCON ZGBCON	F07BVF CGBRFS ZGBRFS	
Hermitian positive-definite	F07FRF CPOTRF ZPOTRF	F07FSF CPOTRS ZPOTRS	F07FUF CPOCON ZPOCON	F07FVF CPORFS ZPORFS	F07FWF CPOTRI ZPOTRI
Hermitian positive-definite (packed storage)	F07GRF CPPTRF ZPPTRF	F07GSF CPPTRS ZPPTRS	F07GUF CPPCON ZPPCON	F07GVF CPPRFS ZPPRFS	F07GWF CPPTRI ZPPTRI
Hermitian positive-definite band	F07HRF CPBTRF ZPBTRF	F07HSF CPBTRS ZPBTRS	F07HUF CPBCON ZPBCON	F07HVF CPBRFS ZPBRFS	
Hermitian indefinite	F07MRF CHETRF ZHETRF	F07MSF CHETRS ZHETRS	F07MUF CHECON ZHECON	F07MVF CHERFS ZHERFS	F07MWF CHETRI ZHETRI
symmetric indefinite	F07NRF CSYTRF ZSYTRF	F07NSF CSYTRS ZSYTRS	F07NUF CSYCON ZSYCON	F07NVF CSYRFS ZSYRFS	F07NWF CSYTRI ZSYTRI
Hermitian indefinite (packed storage)	F07PRF CHPTRF ZHPTRF	F07PSF CHPTRS ZHPTRS	F07PUF CHPCON ZHPCON	F07PVF CHPRFS ZHPRFS	F07PWF CHPTRI ZHPTRI
symmetric indefinite (packed storage)	F07QRF CSPTRF ZSPTRF	F07QSF CSPTRS ZSPTRS	F07QUF CSPCON ZSPCON	F07QVF CSPRFS ZSPRFS	F07QWF CSPTRI ZSPTRI
triangular		F07TSF CTRTRS ZTRTRS	F07TUF CTRCON ZTRCON	F07TVF CTRRFS ZTRRFS	F07TWF CTRTRI ZTRTRI
triangular (packed storage)		F07USF CTPTRS ZTPTRS	F07UUF CTPCON ZTPCON	F07UVF CTPRFS ZTPRFS	F07UWF CTPTRI ZTPTRI
triangular band		F07VSF CTBTRS ZTBTRS	F07VUF CTBCON ZTBCON	F07VVF CTBRFS ZTBRFS	

Table 2

Each entry gives:

The NAG routine name
the LAPACK routine name in a single precision implementation
the LAPACK routine name in a double precision implementation

4 Indexes of LAPACK routines

Real Matrices		
LAPACK single precision	LAPACK double precision	NAG
SGBCO■	DGBCO■	F07BGF
SGBRFS	DGBRFS	F07BHF
SGBTRF	DGBTRF	F07BDF
SGBTRS	DGBTRS	F07BEF
SGECO■	DGECO■	F07AGF
SGERFS	DGERFS	F07AHF
SGETRF	DGETRF	F07ADF
SGETRI	DGETRI	F07AJF
SGETRS	DGETRS	F07AEF
SPBCO■	DPBCO■	F07HGF
SPBRFS	DPBRFS	F07HHF
SPBTRF	DPBTRF	F07HDF
SPBTRS	DPBTRS	F07HEF
SPOCO■	DPOCO■	F07FGF
SPORFS	DPORFS	F07FHF
SPOTRF	DPOTRF	F07FDF
SPOTRI	DPOTRI	F07FJF
SPOTRS	DPOTRS	F07FEF
SPPCO■	DPPCO■	F07GGF
SPPRFS	DPPRFS	F07GHF
SPPTRF	DPPTRF	F07GDF
SPPTRI	DPPTRI	F07GJF
SPPTRS	DPPTRS	F07GEF
SSPCO■	DSPCO■	F07PGF
SSPRFS	DSPRFS	F07PHF
SSPTRF	DSPTRF	F07PDF
SSPTRI	DSPTRI	F07PJF
SSPTRS	DSPTRS	F07PEF
SSYCO■	DSYCO■	F07MGF
SSYRFS	DSYRFS	F07MHF
SSYTRF	DSYTRF	F07MDF
SSYTRI	DSYTRI	F07MJF
SSYTRS	DSYTRS	F07MEF
STBCO■	DTBCO■	F07VGF
STBRFS	DTBRFS	F07VHF
STBTRS	DTBTRS	F07VEF
STPCO■	DTPCO■	F07UGF
STPRFS	DTPRFS	F07UHF
STPTRI	DTPTRI	F07UJF
STPTRS	DTPTRS	F07UEF
STRCO■	DTRCO■	F07TGF
STRRFS	DTRRFS	F07THF
STRTRI	DTRTRI	F07TJF
STRTRS	DTRTRS	F07TEF

Complex Matrices		
LAPACK single precision	LAPACK double precision	NAG
CGBCO■	ZGBCO■	F07BUF
CGBRFS	ZGBRFS	F07BVF
CGBTRF	ZGBTRF	F07BRF
CGBTRS	ZGBTRS	F07BSF
CGECO■	ZGECO■	F07AUF
CGERFS	ZGERFS	F07AVF
CGETRF	ZGETRF	F07ARF
CGETRI	ZGETRI	F07AWF
CGETRS	ZGETRS	F07ASF
CHECO■	ZHECO■	F07MUF
CHERFS	ZHERFS	F07MVF
CHETRF	ZHETRF	F07MRF
CHETRI	ZHETRI	F07MWF
CHETRS	ZHETRS	F07MSF
CHPCO■	ZHPCO■	F07PUF
CHPRFS	ZHPRFS	F07PVF
CHPTRF	ZHPTRF	F07PRF
CHPTRI	ZHPTRI	F07PWF
CHPTRS	ZHPTRS	F07PSF
CPBCO■	ZPBCO■	F07HUF
CPBRFS	ZPBRFS	F07HVF
CPBTRF	ZPBTRF	F07HRF
CPBTRS	ZPBTRS	F07HSF
CPOCO■	ZPOCO■	F07FUF
CPORFS	ZPORFS	F07FVF
CPOTRF	ZPOTRF	F07FRF
CPOTRI	ZPOTRI	F07FWF
CPOTRS	ZPOTRS	F07FSF
CPPCO■	ZPPCO■	F07GUF
CPPRFS	ZPPRFS	F07GVF
CPPTRF	ZPPTRF	F07GRF
CPPTRI	ZPPTRI	F07GWF
CPPTRS	ZPPTRS	F07GSF
CSPCO■	ZSPCO■	F07QUF
CSPRFS	ZSPRFS	F07QVF
CSPTRF	ZSPTRF	F07QRF
CSPTRI	ZSPTRI	F07QWF
CSPTRS	ZSPTRS	F07QSF
CSYCO■	ZSYCO■	F07■UF
CSYRFS	ZSYRFS	F07■VF
CSYTRF	ZSYTRF	F07■RF
CSYTRI	ZSYTRI	F07■WF
CSYTRS	ZSYTRS	F07■SF
CTBCO■	ZTBCO■	F07VUF
CTBRFS	ZTBRFS	F07VVF
CTBTRS	ZTBTRS	F07VSF
CTPCO■	ZTPCO■	F07UUF
CTPRFS	ZTPRFS	F07UVF
CTPTRI	ZTPTRI	F07UWF
CTPTRS	ZTPTRS	F07USF
CTRCO■	ZTRCO■	F07TUF
CTRRFS	ZTRRFS	F07TVF
CTRTRI	ZTRTRI	F07TWF
CTRTRS	ZTRTRS	F07TSF

Table 3

5 References

[1] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A, Ostrouchov S and Sorensen D (1995) *LAPACK Users' Guide* (2nd Edition) SIAM, Philadelphia

[2] Golub G H and Van Loan C F (1989) *Matrix Computations* Johns Hopkins University Press (2nd Edition), Baltimore

[3] Higham N J (1988) Algorithm 674: Fortran codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation *ACM Trans. Math. Software* **14** 381–396

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

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
F08AEF	16	(SGEQRF/DGEQRF) *QR* factorization of real general rectangular matrix
F08AFF	16	(SORGQR/DORGQR) Form all or part of orthogonal *Q* from *QR* factorization determined by F08AEF or F08BEF
F08AGF	16	(SORMQR/DORMQR) Apply orthogonal transformation determined by F08AEF or F08BEF
F08AHF	16	(SGELQF/DGELQF) *LQ* factorization of real general rectangular matrix
F08AJF	16	(SORGLQ/DORGLQ) Form all or part of orthogonal *Q* from *LQ* factorization determined by F08AHF
F08AKF	16	(SORMLQ/DORMLQ) Apply orthogonal transformation determined by F08AHF
F08ASF	16	(CGEQRF/ZGEQRF) *QR* factorization of complex general rectangular matrix
F08ATF	16	(CUNGQR/ZUNGQR) Form all or part of unitary *Q* from *QR* factorization determined by F08ASF or F08BSF
F08AUF	16	(CUNMQR/ZUNMQR) Apply unitary transformation determined by F08ASF or F08BSF
F08AVF	16	(CGELQF/ZGELQF) *LQ* factorization of complex general rectangular matrix
F08AWF	16	(CUNGLQ/ZUNGLQ) Form all or part of unitary *Q* from *LQ* factorization determined by F08AVF
F08AXF	16	(CUNMLQ/ZUNMLQ) Apply unitary transformation determined by F08AVF
F08BEF	16	(SGEQPF/DGEQPF) *QR* factorization of real general rectangular matrix with column pivoting
F08BSF	16	(CGEQPF/ZGEQPF) *QR* factorization of complex general rectangular matrix with column pivoting
F08FCF	19	(SSYEVD/DSYEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, using divide and conquer
F08FEF	16	(SSYTRD/DSYTRD) Orthogonal reduction of real symmetric matrix to symmetric tridiagonal form
F08FFF	16	(SORGTR/DORGTR) Generate orthogonal transformation matrix from reduction to tridiagonal form determined by F08FEF
F08FGF	16	(SORMTR/DORMTR) Apply orthogonal transformation determined by F08FEF
F08FQF	19	(CHEEVD/ZHEEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, using divide and conquer
F08FSF	16	(CHETRD/ZHETRD) Unitary reduction of complex Hermitian matrix to real symmetric tridiagonal form
F08FTF	16	(CUNGTR/ZUNGTR) Generate unitary transformation matrix from reduction to tridiagonal form determined by F08FSF
F08FUF	16	(CUNMTR/ZUNMTR) Apply unitary transformation matrix determined by F08FSF
F08GCF	19	(SSPEVD/DSPEVD) All eigenvalues and optionally all eigenvectors of real symmetric matrix, packed storage, using divide and conquer
F08GEF	16	(SSPTRD/DSPTRD) Orthogonal reduction of real symmetric matrix to symmetric tridiagonal form, packed storage
F08GFF	16	(SOPGTR/DOPGTR) Generate orthogonal transformation matrix from reduction to tridiagonal form determined by F08GEF

F08GGF	16	(SOPMTR/DOPMTR) Apply orthogonal transformation determined by F08GEF
F08GQF	19	(CHPEVD/ZHPEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian matrix, packed storage, using divide and conquer
F08GSF	16	(CHPTRD/ZHPTRD) Unitary reduction of complex Hermitian matrix to real symmetric tridiagonal form, packed storage
F08GTF	16	(CUPGTR/ZUPGTR) Generate unitary transformation matrix from reduction to tridiagonal form determined by F08GSF
F08GUF	16	(CUPMTR/ZUPMTR) Apply unitary transformation matrix determined by F08GSF
F08HCF	19	(SSBEVD/DSBEVD) All eigenvalues and optionally all eigenvectors of real symmetric band matrix, using divide and conquer
F08HEF	16	(SSBTRD/DSBTRD) Orthogonal reduction of real symmetric band matrix to symmetric tridiagonal form
F08HQF	19	(CHBEVD/ZHBEVD) All eigenvalues and optionally all eigenvectors of complex Hermitian band matrix, using divide and conquer
F08HSF	16	(CHBTRD/ZHBTRD) Unitary reduction of complex Hermitian band matrix to real symmetric tridiagonal form
F08JCF	19	(SSTEVD/DSTEVD) All eigenvalues and optionally all eigenvectors of real symmetric tridiagonal matrix, using divide and conquer
F08JEF	16	(SSTEQR/DSTEQR) All eigenvalues and eigenvectors of real symmetric tridiagonal matrix, reduced from real symmetric matrix using implicit QL or QR
F08JFF	16	(SSTERF/DSTERF) All eigenvalues of real symmetric tridiagonal matrix, root-free variant of QL or QR
F08JGF	16	(SPTEQR/DPTEQR) All eigenvalues and eigenvectors of real symmetric positive-definite tridiagonal matrix, reduced from real symmetric positive-definite matrix
F08JJF	16	(SSTEBZ/DSTEBZ) Selected eigenvalues of real symmetric tridiagonal matrix by bisection
F08JKF	16	(SSTEIN/DSTEIN) Selected eigenvectors of real symmetric tridiagonal matrix by inverse iteration, storing eigenvectors in real array
F08JSF	16	(CSTEQR/ZSTEQR) All eigenvalues and eigenvectors of real symmetric tridiagonal matrix, reduced from complex Hermitian matrix, using implicit QL or QR
F08JUF	16	(CPTEQR/ZPTEQR) All eigenvalues and eigenvectors of real symmetric positive-definite tridiagonal matrix, reduced from complex Hermitian positive-definite matrix
F08JXF	16	(CSTEIN/ZSTEIN) Selected eigenvectors of real symmetric tridiagonal matrix by inverse iteration, storing eigenvectors in complex array
F08KEF	16	(SGEBRD/DGEBRD) Orthogonal reduction of real general rectangular matrix to bidiagonal form
F08KFF	16	(SORGBR/DORGBR) Generate orthogonal transformation matrices from reduction to bidiagonal form determined by F08KEF
F08KGF	16	(SORMBR/DORMBR) Apply orthogonal transformations from reduction to bidiagonal form determined by F08KEF
F08KSF	16	(CGEBRD/ZGEBRD) Unitary reduction of complex general rectangular matrix to bidiagonal form
F08KTF	16	(CUNGBR/ZUNGBR) Generate unitary transformation matrices from reduction to bidiagonal form determined by F08KSF
F08KUF	16	(CUNMBR/ZUNMBR) Apply unitary transformations from reduction to bidiagonal form determined by F08KSF
F08LEF	19	(SGBBRD/DGBBRD) Reduction of real rectangular band matrix to upper bidiagonal form
F08LSF	19	(CGBBRD/ZGBBRD) Reduction of complex rectangular band matrix to upper bidiagonal form
F08MEF	16	(SBDSQR/DBDSQR) SVD of real bidiagonal matrix reduced from real general matrix

F08MSF	16	(CBDSQR/ZBDSQR) SVD of real bidiagonal matrix reduced from complex general matrix
F08NEF	16	(SGEHRD/DGEHRD) Orthogonal reduction of real general matrix to upper Hessenberg form
F08NFF	16	(SORGHR/DORGHR) Generate orthogonal transformation matrix from reduction to Hessenberg form determined by F08NEF
F08NGF	16	(SORMHR/DORMHR) Apply orthogonal transformation matrix from reduction to Hessenberg form determined by F08NEF
F08NHF	16	(SGEBAL/DGEBAL) Balance real general matrix
F08NJF	16	(SGEBAK/DGEBAK) Transform eigenvectors of real balanced matrix to those of original matrix supplied to F08NHF
F08NSF	16	(CGEHRD/ZGEHRD) Unitary reduction of complex general matrix to upper Hessenberg form
F08NTF	16	(CUNGHR/ZUNGHR) Generate unitary transformation matrix from reduction to Hessenberg form determined by F08NSF
F08NUF	16	(CUNMHR/ZUNMHR) Apply unitary transformation matrix from reduction to Hessenberg form determined by F08NSF
F08NVF	16	(CGEBAL/ZGEBAL) Balance complex general matrix
F08NWF	16	(CGEBAK/ZGEBAK) Transform eigenvectors of complex balanced matrix to those of original matrix supplied to F08NVF
F08PEF	16	(SHSEQR/DHSEQR) Eigenvalues and Schur factorization of real upper Hessenberg matrix reduced from real general matrix
F08PKF	16	(SHSEIN/DHSEIN) Selected right and/or left eigenvectors of real upper Hessenberg matrix by inverse iteration
F08PSF	16	(CHSEQR/ZHSEQR) Eigenvalues and Schur factorization of complex upper Hessenberg matrix reduced from complex general matrix
F08PXF	16	(CHSEIN/ZHSEIN) Selected right and/or left eigenvectors of complex upper Hessenberg matrix by inverse iteration
F08QFF	16	(STREXC/DTREXC) Reorder Schur factorization of real matrix using orthogonal similarity transformation
F08QGF	16	(STRSEN/DTRSEN) Reorder Schur factorization of real matrix, form orthonormal basis of right invariant subspace for selected eigenvalues, with estimates of sensitivities
F08QHF	16	(STRSYL/DTRSYL) Solve real Sylvester matrix equation $AX + XB = C$, A and B are upper quasi-triangular or transposes
F08QKF	16	(STREVC/DTREVC) Left and right eigenvectors of real upper quasi-triangular matrix
F08QLF	16	(STRSNA/DTRSNA) Estimates of sensitivities of selected eigenvalues and eigenvectors of real upper quasi-triangular matrix
F08QTF	16	(CTREXC/ZTREXC) Reorder Schur factorization of complex matrix using unitary similarity transformation
F08QUF	16	(CTRSEN/ZTRSEN) Reorder Schur factorization of complex matrix, form orthonormal basis of right invariant subspace for selected eigenvalues, with estimates of sensitivities
F08QVF	16	(CTRSYL/ZTRSYL) Solve complex Sylvester matrix equation $AX + XB = C$, A and B are upper triangular or conjugate-transposes
F08QXF	16	(CTREVC/ZTREVC) Left and right eigenvectors of complex upper triangular matrix
F08QYF	16	(CTRSNA/ZTRSNA) Estimates of sensitivities of selected eigenvalues and eigenvectors of complex upper triangular matrix
F08SEF	16	(SSYGST/DSYGST) Reduction to standard form of real symmetric-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$, B factorized by F07FDF
F08SSF	16	(CHEGST/ZHEGST) Reduction to standard form of complex Hermitian-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$, B factorized by F07FRF

F08TEF	16	(SSPGST/DSPGST) 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
F08TSF	16	(CHPGST/ZHPGST) 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
F08UEF	19	(SSBGST/DSBGST) Reduction of real symmetric-definite banded generalized eigenproblem $Ax = \lambda Bx$ to standard form $Cy = \lambda y$, such that C has the same bandwidth as A
F08UFF	19	(SPBSTF/DPBSTF) Computes a split Cholesky factorization of real symmetric positive-definite band matrix A
F08USF	19	(CHBGST/ZHBGST) Reduction of complex Hermitian-definite banded generalized eigenproblem $Ax = \lambda Bx$ to standard form $Cy = \lambda y$, such that C has the same bandwidth as A
F08UTF	19	(CPBSTF/ZPBSTF) Computes a split Cholesky factorization of complex Hermitian positive-definite band matrix A

Chapter F08

Least-squares and Eigenvalue Problems (LAPACK)

Contents

1 Scope of the Chapter

This chapter provides routines for the solution of linear least-squares problems, eigenvalue problems and singular value problems, as well as associated computations. It provides routines for:

- solution of linear least-squares problems
- solution of symmetric eigenvalue problems
- solution of nonsymmetric eigenvalue problems
- solution of singular value problems
- solution of generalized symmetric-definite eigenvalue problems
- matrix factorizations associated with the above problems
- estimating condition numbers of eigenvalues and eigenvectors
- estimating the numerical rank of a matrix
- solution of the Sylvester matrix equation

Routines are provided for both *real* and *complex* data.

For a general introduction to the solution of linear least-squares problems, you should turn first to the the F04 Chapter Introduction. The decision trees, at the end of the the F04 Chapter Introduction, direct you to the most appropriate routines in Chapter F04 or Chapter F08. Chapter F04 contains *Black Box* routines which enable standard linear least-squares problems to be solved by a call to a single routine.

For a general introduction to eigenvalue and singular value problems, you should turn first to the the F02 Chapter Introduction. The decision trees, at the end of the the F02 Chapter Introduction, direct you to the most appropriate routines in Chapter F02. Chapter F02 contains *Black Box* routines which enable some standard types of problem to be solved by a call to a single routine. Often routines in Chapter F02 call Chapter F08 routines to perform the necessary computational tasks. However, divide and conquer algorithms for symmetric (Hermitian) eigenvalue problem are available only in this chapter and they can be considered as *Black Box* routines.

The routines in this chapter (F08) handle only *dense*, *band*, *tridiagonal* and *Hessenberg* matrices (not matrices with more specialized structures, or general sparse matrices). The decision trees in Section 4 direct you to the most appropriate routines in Chapter F08.

The routines in this chapter have all been derived from the LAPACK project (see Anderson *et al.* [1]). They have been designed to be efficient on a wide range of high-performance computers, without compromising efficiency on conventional serial machines.

It is not expected that every user will need to read all of the following sections, but rather will pick out those sections relevant to their particular problem.

2 Background to the Problems

This section is only a brief introduction to the numerical solution of linear least-squares problems, eigenvalue and singular value problems. Consult a standard textbook for a more thorough discussion, for example Golub and Van Loan [4].

2.1 Linear Least-squares Problems

The *linear least-squares problem* is

$$\underset{x}{\text{minimize}} \|b - Ax\|_2, \tag{1}$$

where A is an m by n matrix, b is a given m element vector and x is the n element solution vector.

In the most usual case $m \geq n$ and $\text{rank}(A) = n$, so that A has *full rank* and in this case the solution to problem (1) is unique; the problem is also referred to as finding a *least-squares solution* to an *overdetermined* system of linear equations.

When $m < n$ and $\text{rank}(A) = m$, there are an infinite number of solutions x which exactly satisfy $b - Ax = 0$. In this case it is often useful to find the unique solution x which minimizes $\|x\|_2$, and the problem is referred to as finding a *minimum-norm solution* to an *underdetermined* system of linear equations.

In the general case when we may have $\text{rank}(A) < \min(m, n)$ – in other words, A may be *rank-deficient* – we seek the *minimum-norm least-squares* solution x which minimizes both $\|x\|_2$ and $\|b - Ax\|_2$.

This chapter (F08) contains computational routines that can be combined with routines in Chapter F07 to solve these linear least-squares problems. Chapter F04 contains Black Box routines to solve these linear least-squares problems in standard cases. The next two sections discuss the factorizations that can be used in the solution of linear least-squares problems.

2.2 Orthogonal Factorizations and Least-squares Problems

A number of routines are provided for factorizing a general rectangular m by n matrix A, as the product of an *orthogonal* matrix (*unitary* if complex) and a *triangular* (or possibly trapezoidal) matrix.

A real matrix Q is *orthogonal* if $Q^TQ = I$; a complex matrix Q is *unitary* if $Q^HQ = I$. Orthogonal or unitary matrices have the important property that they leave the two-norm of a vector invariant, so that

$$\|x\|_2 = \|Qx\|_2, \text{ if } Q \text{ is orthogonal or unitary.}$$

They usually help to maintain numerical stability because they do not amplify rounding errors.

Orthogonal factorizations are used in the solution of linear least-squares problems. They may also be used to perform preliminary steps in the solution of eigenvalue or singular value problems, and are useful tools in the solution of a number of other problems.

2.2.1 QR factorization

The most common, and best known, of the factorizations is the QR *factorization* given by

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}, \text{ if } m \geq n,$$

where R is an n by n upper triangular matrix and Q is an m by m orthogonal (or unitary) matrix. If A is of full rank n, then R is non-singular. It is sometimes convenient to write the factorization as

$$A = (Q_1\; Q_2)\begin{pmatrix} R \\ 0 \end{pmatrix}$$

which reduces to

$$A = Q_1R,$$

where Q_1 consists of the first n columns of Q, and Q_2 the remaining $m - n$ columns.

If $m < n$, R is trapezoidal, and the factorization can be written

$$A = Q\,(R_1\; R_2)\,, \text{if } m < n,$$

where R_1 is upper triangular and R_2 is rectangular.

The QR factorization can be used to solve the linear least-squares problem (1) when $m \geq n$ and A is of full rank, since

$$\|b - Ax\|_2 = \|Q^Tb - Q^TAx\|_2 = \left\| \begin{matrix} c_1 - Rx \\ c_2 \end{matrix} \right\|,$$

where

$$c \equiv \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} Q_1^Tb \\ Q_2^Tb \end{pmatrix} = Q^Tb;$$

and c_1 is an n element vector. Then x is the solution of the upper triangular system

$$Rx = c_1.$$

The residual vector r is given by

$$r = b - Ax = Q\begin{pmatrix} 0 \\ c_2 \end{pmatrix}.$$

The residual sum of squares $\|r\|_2^2$ may be computed without forming r explicitly, since

$$\|r\|_2 = \|b - Ax\|_2 = \|c_2\|_2.$$

2.2.2 LQ factorization

The *LQ factorization* is given by

$$A = (L\ \ 0)Q = (L\ \ 0)\begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = LQ_1, \text{ if } m \le n,$$

where L is m by m lower triangular, Q is n by n orthogonal (or unitary), Q_1 consists of the first m rows of Q, and Q_2 the remaining $n - m$ rows.

The LQ factorization of A is essentially the same as the QR factorization of A^T (A^H if A is complex), since

$$A = (L\ \ 0)\,Q \Leftrightarrow A^T = Q^T \begin{pmatrix} L^T \\ 0 \end{pmatrix}.$$

The LQ factorization may be used to find a minimum norm solution of an underdetermined system of linear equations $Ax = b$ where A is m by n with $m < n$ and has rank m. The solution is given by

$$x = Q^T \begin{pmatrix} L^{-1}b \\ 0 \end{pmatrix}.$$

2.2.3 QR factorization with column pivoting

To solve a linear least-squares problem (1) when A is not of full rank, or the rank of A is in doubt, we can perform either a QR factorization with column pivoting or a singular value decomposition.

The *QR factorization with column pivoting* is given by

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} P^T, \quad m \ge n,$$

where Q and R are as before and P is a (real) permutation matrix, chosen (in general) so that

$$|r_{11}| \ge |r_{22}| \ge \ldots \ge |r_{nn}|$$

and moreover, for each k,

$$|r_{kk}| \ge \|R_{k:j,j}\|_2 \quad \text{for } j = k+1, \ldots, n.$$

If we put

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

where R_{11} is the leading k by k upper triangular submatrix of R then, in exact arithmetic, if $\text{rank}(A) = k$, the whole of the submatrix R_{22} in rows and columns $k+1$ to n would be zero. In numerical computation, the aim must be to determine an index k, such that the leading submatrix R_{11} is well-conditioned, and R_{22} is negligible, so that

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \simeq \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}.$$

Then k is the effective rank of A. See Golub and Van Loan [4] for a further discussion of numerical rank determination.

The so-called basic solution to the linear least-squares problem (1) can be obtained from this factorization as

$$x = P\begin{pmatrix} R_{11}^{-1}\hat{c}_1 \\ 0 \end{pmatrix},$$

where $\hat{c}_1$ consists of just the first k elements of $c = Q^T b$.

2.3 The Singular Value Decomposition

The *singular value decomposition* (SVD) of an m by n matrix A is given by

$$A = U\Sigma V^T, \quad (A = U\Sigma V^H \text{ in the complex case})$$

where U and V are orthogonal (unitary) and Σ is an m by n diagonal matrix with real diagonal elements, σ_i, such that

$$\sigma_1 \ge \sigma_2 \ge \ldots \sigma_{\min(m,n)} \ge 0.$$

The σ_i are the *singular values* of A and the first $\min(m,n)$ columns of U and V are the *left* and *right singular vectors* of A. The singular values and singular vectors satisfy

$$Av_i = \sigma_i u_i \text{ and } A^T u_i = \sigma_i v_i \text{ (or } A^H u_i = \sigma_i v_i)$$

where u_i and v_i are the ith columns of U and V respectively.

The computation proceeds in the following stages.

(1) The matrix A is reduced to bidiagonal form $A = U_1 B V_1^T$ if A is real ($A = U_1 B V_1^H$ if A is complex), where U_1 and V_1 are orthogonal (unitary if A is complex), and B is real and upper bidiagonal when $m \geq n$ and lower bidiagonal when $m < n$, so that B is nonzero only on the main diagonal and either on the first superdiagonal (if $m \geq n$) or the first subdiagonal (if $m < n$).

(2) The SVD of the bidiagonal matrix B is computed as $B = U_2 \Sigma V_2^T$, where U_2 and V_2 are orthogonal and Σ is diagonal as described above. The singular vectors of A are then $U = U_1 U_2$ and $V = V_1 V_2$.

If $m \gg n$, it may be more efficient to first perform a QR factorization of A, and then compute the SVD of the n by n matrix R, since if $A = QR$ and $R = U\Sigma V^T$, then the SVD of A is given by $A = (QU)\Sigma V^T$.

Similarly, if $m \ll n$, it may be more efficient to first perform an LQ factorization of A.

2.4 The Singular Value Decomposition and Least-squares Problems

The SVD may be used to find a minimum norm solution to a (possibly) rank-deficient linear least-squares problem (1). The effective rank, k, of A can be determined as the number of singular values which exceed a suitable threshold. Let $\hat{\Sigma}$ be the leading k by k submatrix of Σ, and $\hat{V}$ be the matrix consisting of the first k columns of V. Then the solution is given by

$$x = \hat{V}\hat{\Sigma}^{-1}\hat{c}_1,$$

where $\hat{c}_1$ consists of the first k elements of $c = U^T b = U_2^T U_1^T b$.

2.5 Symmetric Eigenvalue Problems

The *symmetric eigenvalue problem* is to find the *eigenvalues*, λ, and corresponding *eigenvectors*, $z \neq 0$, such that

$$Az = \lambda z, \quad A = A^T, \quad \text{where } A \text{ is real.}$$

For the *Hermitian eigenvalue problem* we have

$$Az = \lambda z, \quad A = A^H, \quad \text{where } A \text{ is complex.}$$

For both problems the eigenvalues λ are real.

When all eigenvalues and eigenvectors have been computed, we write

$$A = Z\Lambda Z^T \text{ (or } A = Z\Lambda Z^H \text{ if complex),}$$

where Λ is a diagonal matrix whose diagonal elements are the eigenvalues, and Z is an orthogonal (or unitary) matrix whose columns are the eigenvectors. This is the classical *spectral factorization* of A.

The basic task of the symmetric eigenproblem routines is to compute values of λ and, optionally, corresponding vectors z for a given matrix A. This computation proceeds in the following stages.

(1) The real symmetric or complex Hermitian matrix A is reduced to *real tridiagonal form* T. If A is real symmetric this decomposition is $A = QTQ^T$ with Q orthogonal and T symmetric tridiagonal. If A is complex Hermitian, the decomposition is $A = QTQ^H$ with Q unitary and T, as before, *real* symmetric tridiagonal.

(2) Eigenvalues and eigenvectors of the real symmetric tridiagonal matrix T are computed. If all eigenvalues and eigenvectors are computed, this is equivalent to factorizing T as $T = S\Lambda S^T$, where S is orthogonal and Λ is diagonal. The diagonal entries of Λ are the eigenvalues of T, which are also the eigenvalues of A, and the columns of S are the eigenvectors of T; the eigenvectors of A are the columns of $Z = QS$, so that $A = Z\Lambda Z^T$ ($Z\Lambda Z^H$ when A is complex Hermitian).

This chapter now supports three primary algorithms for computing eigenvalues and eigenvectors of real symmetric matrices and complex Hermitian matrices. They are:

(i) the divide and conquer algorithm;
(ii) the QR algorithm;
(iii) bisection followed by inverse iteration.

The divide and conquer algorithm is generally more efficient than the traditional QR algorithm and is recommended for computing all eigenvalues and eigenvectors. Furthermore, eigenvalues and eigenvectors can be obtained by calling one single routine in the case of the divide and conquer algorithm. In general, more than one routine has to be called if the QR algorithm or bisection followed by inverse iteration is used.

2.6 Generalized Symmetric-Definite Eigenvalue Problems

This section is concerned with the solution of the generalized eigenvalue problems $Az = \lambda Bz$, $ABz = \lambda z$, and $BAz = \lambda z$, where A and B are real symmetric or complex Hermitian and B is positive-definite. Each of these problems can be reduced to a standard symmetric eigenvalue problem, using a Cholesky factorization of B as either $B = LL^T$ or $B = U^TU$ (LL^H or U^HU in the Hermitian case).

With $B = LL^T$, we have

$$Az = \lambda Bz \Rightarrow (L^{-1}AL^{-T})(L^Tz) = \lambda(L^Tz).$$

Hence the eigenvalues of $Az = \lambda Bz$ are those of $Cy = \lambda y$, where C is the symmetric matrix $C = L^{-1}AL^{-T}$ and $y = L^Tz$. In the complex case C is Hermitian with $C = L^{-1}AL^{-H}$ and $y = L^Hz$.

Table 1 summarizes how each of the three types of problem may be reduced to standard form $Cy = \lambda y$, and how the eigenvectors z of the original problem may be recovered from the eigenvectors y of the reduced problem. The table applies to real problems; for complex problems, transposed matrices must be replaced by conjugate-transposes.

	Type of problem	Factorization of B	Reduction	Recovery of eigenvectors
1.	$Az = \lambda Bz$	$B = LL^T$	$C = L^{-1}AL^{-T}$	$z = L^{-T}y$
		$B = U^TU$	$C = U^{-T}AU^{-1}$	$z = U^{-1}y$
2.	$ABz = \lambda z$	$B = LL^T$	$C = L^TAL$	$z = L^{-T}y$
		$B = U^TU$	$C = UAU^T$	$z = U^{-1}y$
3.	$BAz = \lambda z$	$B = LL^T$	$C = L^TAL$	$z = Ly$
		$B = U^TU$	$C = UAU^T$	$z = U^Ty$

Table 1
Reduction of generalized symmetric-definite eigenproblems to standard problems

When the generalized symmetric-definite problem has been reduced to the corresponding standard problem $Cy = \lambda y$, this may then be solved using the routines described in the previous section. No special routines are needed to recover the eigenvectors z of the generalized problem from the eigenvectors y of the standard problem, because these computations are simple applications of Level 2 or Level 3 BLAS (see Chapter F06).

2.7 Packed Storage for Symmetric Matrices

Routines which handle symmetric matrices are usually designed so that they use either the upper or lower triangle of the matrix; it is not necessary to store the whole matrix. If either the upper or lower triangle is stored conventionally in the upper or lower triangle of a two-dimensional array, the remaining elements of the array can be used to store other useful data. However, that is not always convenient, and if it is important to economize on storage, the upper or lower triangle can be stored in a one-dimensional array of length $n(n + 1)/2$; that is, the storage is almost halved.

This storage format is referred to as *packed storage*; it is described in Section 3.3.

Routines designed for packed storage are usually less efficient, especially on high-performance computers, so there is a trade-off between storage and efficiency.

2.8 Band Matrices

A *band* matrix is one whose elements are confined to a relatively small number of sub-diagonals or super-diagonals on either side of the main diagonal. Algorithms can take advantage of bandedness to reduce the amount of work and storage required. The storage scheme for band matrices is described in Section 3.3.

If the problem is the generalized symmetric definite eigenvalue problem $Az = \lambda Bz$ and the matrices A and B are additionally banded, the matrix C as defined in Section 2.6 is, in general, full. We can reduce the problem to a banded standard problem by modifying the definition of C thus:

$$C = X^T AX, \quad \text{where } X = U^{-1}Q \text{ or } L^{-T}Q,$$

where Q is an orthogonal matrix chosen to ensure that C has bandwidth no greater than that of A.

A further refinement is possible when A and B are banded, which halves the amount of work required to form C. Instead of the standard Cholesky factorization of B as U^TU or LL^T, we use a *split Cholesky* factorization $B = S^TS$, where

$$S = \begin{pmatrix} U_{11} & \\ M_{21} & L_{22} \end{pmatrix}$$

with U_{11} upper triangular and L_{22} lower triangular of order approximately $n/2$; S has the same bandwidth as B.

2.9 Nonsymmetric Eigenvalue Problems

The *nonsymmetric eigenvalue problem* is to find the *eigenvalues*, λ, and corresponding *eigenvectors*, $v \neq 0$, such that

$$Av = \lambda v.$$

More precisely, a vector v as just defined is called a *right eigenvector* of A, and a vector $u \neq 0$ satisfying

$$u^TA = \lambda u^T \quad (u^HA = \lambda u^H \text{ when } u \text{ is complex})$$

is called a *left eigenvector* of A.

A real matrix A may have complex eigenvalues, occurring as complex conjugate pairs.

This problem can be solved via the *Schur factorization* of A, defined in the real case as

$$A = ZTZ^T,$$

where Z is an orthogonal matrix and T is an upper quasi-triangular matrix with 1 by 1 and 2 by 2 diagonal blocks, the 2 by 2 blocks corresponding to complex conjugate pairs of eigenvalues of A. In the complex case, the Schur factorization is

$$A = ZTZ^H,$$

where Z is unitary and T is a complex upper triangular matrix.

The columns of Z are called the *Schur vectors*. For each k $(1 \leq k \leq n)$, the first k columns of Z form an orthonormal basis for the *invariant subspace* corresponding to the first k eigenvalues on the diagonal of T. Because this basis is orthonormal, it is preferable in many applications to compute Schur vectors rather than eigenvectors. It is possible to order the Schur factorization so that any desired set of k eigenvalues occupy the k leading positions on the diagonal of T.

The two basic tasks of the nonsymmetric eigenvalue routines are to compute, for a given matrix A, all n values of λ and, if desired, their associated right eigenvectors v and/or left eigenvectors u, and the Schur factorization.

These two basic tasks can be performed in the following stages.

(1) A general matrix A is reduced to *upper Hessenberg form* H which is zero below the first subdiagonal. The reduction may be written $A = QHQ^T$ with Q orthogonal if A is real, or $A = QHQ^H$ with Q unitary if A is complex.

(2) The upper Hessenberg matrix H is reduced to Schur form T, giving the Schur factorization $H = STS^T$ (for H real) or $H = STS^H$ (for H complex). The matrix S (the Schur vectors of H) may optionally be computed as well. Alternatively S may be postmultiplied into the matrix Q determined in stage 1, to give the matrix $Z = QS$, the Schur vectors of A. The eigenvalues are obtained from the diagonal elements or diagonal blocks of T.

(3) Given the eigenvalues, the eigenvectors may be computed in two different ways. Inverse iteration can be performed on H to compute the eigenvectors of H, and then the eigenvectors can be multiplied by the matrix Q in order to transform them to eigenvectors of A. Alternatively the eigenvectors of T can be computed, and optionally transformed to those of H or A if the matrix S or Z is supplied.

The accuracy with which eigenvalues can be obtained can often be improved by *balancing* a matrix. This is discussed further in Section 2.11.6 below.

2.10 The Sylvester Equation

The Sylvester equation is a matrix equation of the form

$$AX + XB = C,$$

where A, B, and C are given matrices with A being m by m, B an n by n matrix and C, and the solution matrix X, m by n matrices. The solution of a special case of this equation occurs in the computation of the condition number for an invariant subspace, but a combination of routines in this chapter allows the solution of the general Sylvester equation.

2.11 Error and Perturbation Bounds and Condition Numbers

In this section we discuss the effects of rounding errors in the solution process and the effects of uncertainties in the data, on the solution to the problem. A number of the routines in this chapter return information, such as condition numbers, that allow these effects to be assessed. First we discuss some notation used in the error bounds of later sections.

The bounds usually contain the factor $p(n)$ (or $p(m,n)$), which grows as a function of the matrix dimension n (or matrix dimensions m and n). It measures how errors can grow as a function of the matrix dimension, and represents a potentially different function for each problem. In practice, it usually grows just linearly; $p(n) \leq 10n$ is often true, although generally only much weaker bounds can be actually proved. We normally describe $p(n)$ as a 'modestly growing' function of n. For detailed derivations of various $p(n)$, see [4] and [6].

For linear equation (see Chapter F07) and least-squares solvers, we consider bounds on the relative error $\|x-\hat{x}\|/\|x\|$ in the computed solution $\hat{x}$, where x is the true solution. For eigenvalue problems we consider bounds on the error $|\lambda_i - \hat{\lambda}_i|$ in the ith computed eigenvalue $\hat{\lambda}_i$, where λ_i is the true ith eigenvalue. For singular value problems we similarly consider bounds $|\sigma_i - \hat{\sigma}_i|$.

Bounding the error in computed eigenvectors and singular vectors $\hat{v}_i$ is more subtle because these vectors are not unique: even though we restrict $\|\hat{v}_i\|_2 = 1$ and $\|v_i\|_2 = 1$, we may still multiply them by arbitrary constants of absolute value 1. So to avoid ambiguity we bound the *angular difference* between $\hat{v}_i$ and the true vector v_i, so that

$$\begin{aligned}\theta(v_i,\hat{v}_i) &= \text{acute angle between } v_i \text{ and } \hat{v}_i \\ &= \arccos|v_i^H \hat{v}_i|.\end{aligned} \qquad (2)$$

When $\theta(v_i,\hat{v}_i)$ is small, we can choose a constant α with absolute value 1 so that $\|\alpha v_i - \hat{v}_i\|_2 \approx \theta(v_i,\hat{v}_i)$.

In addition to bounds for individual eigenvectors, bounds can be obtained for the spaces spanned by collections of eigenvectors. These may be much more accurately determined than the individual eigenvectors which span them. These spaces are called *invariant subspaces* in the case of eigenvectors, because if v is any vector in the space, Av is also in the space, where A is the matrix. Again, we will use angle to measure the difference between a computed space $\hat{S}$ and the true space S:

$$\begin{aligned}\theta(S,\hat{S}) &= \text{acute angle between } S \text{ and } \hat{S} \\ &= \max_{\substack{s\in S \\ s\neq 0}} \min_{\substack{\hat{s}\in\hat{S} \\ \hat{s}\neq 0}} \theta(s,\hat{s}) \quad \text{or} \quad \max_{\substack{\hat{s}\in\hat{S} \\ \hat{s}\neq 0}} \min_{\substack{s\in S \\ s\neq 0}} \theta(s,\hat{s})\end{aligned} \qquad (3)$$

$\theta(S,\hat{S})$ may be computed as follows. Let S be a matrix whose columns are orthonormal and span S. Similarly let $\hat{S}$ be an orthonormal matrix with columns spanning $\hat{S}$. Then

$$\theta(S,\hat{S}) = \arccos\sigma_{\min}(S^H\hat{S}).$$

Finally, we remark on the accuracy of the bounds when they are large. Relative errors like $\|\hat{x}-x\|/\|x\|$ and angular errors like $\theta(\hat{v}_i,v_i)$ are only of interest when they are much less than 1. Some stated bounds are not strictly true when they are close to 1, but rigorous bounds are much more complicated and supply little extra information in the interesting case of small errors. These bounds are indicated by using the symbol $\lesssim$, or 'approximately less than', instead of the usual $\leq$. Thus, when these bounds are close to 1 or greater, they indicate that the computed answer may have no significant digits at all, but do not otherwise bound the error.

2.11.1 Least-squares problems

The conventional error analysis of linear least-squares problems goes as follows. The problem is to find the x minimizing $\|Ax - b\|_2$. Let $\hat{x}$ be the solution computed using one of the methods described above. We discuss the most common case, where A is overdetermined (i.e., has more rows than columns) and has full rank.

Then the computed solution $\hat{x}$ has a small normwise backward error. In other words $\hat{x}$ minimizes $\|(A+E)\hat{x} - (b+f)\|_2$, where

$$\max\left(\frac{\|E\|_2}{\|A\|_2}, \frac{\|f\|_2}{\|b\|_2}\right) \leq p(n)\epsilon$$

and $p(n)$ is a modestly growing function of n and ϵ is the machine precision. Let $\kappa_2(A) = \sigma_{\max}(A)/\sigma_{\min}(A)$, $\rho = \|Ax - b\|_2$, and $\sin(\theta) = \rho/\|b\|_2$. Then if $p(n)\epsilon$ is small enough, the error $\hat{x} - x$ is bounded by

$$\frac{\|x - \hat{x}\|_2}{\|x\|_2} \lesssim p(n)\epsilon\left\{\frac{2\kappa_2(A)}{\cos(\theta)} + \tan(\theta)\kappa_2^2(A)\right\}.$$

If A is rank-deficient, the problem can be *regularized* by treating all singular values less than a user-specified threshold as exactly zero. See [4] for error bounds in this case, as well as for the underdetermined case.

The solution of the overdetermined, full-rank problem may also be characterized as the solution of the linear system of equations

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}\begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}.$$

By solving this linear system (see Chapter F07) componentwise error bounds can also be obtained [2].

2.11.2 The singular value decomposition

The usual error analysis of the SVD algorithm is as follows [4].

The computed SVD, $\hat{U}\hat{\Sigma}\hat{V}^T$, is nearly the exact SVD of $A+E$, i.e., $A+E = (\hat{U}+\delta\hat{U})\hat{\Sigma}(\hat{V}+\delta\hat{V})$ is the true SVD, so that $\hat{U}+\delta\hat{U}$ and $\hat{V}+\delta\hat{V}$ are both orthogonal, where $\|E\|_2/\|A\|_2 \leq p(m,n)\epsilon$, $\|\delta\hat{U}\| \leq p(m,n)\epsilon$, and $\|\delta\hat{V}\| \leq p(m,n)\epsilon$. Here $p(m,n)$ is a modestly growing function of m and n and ϵ is the machine precision. Each computed singular value $\hat{\sigma}_i$ differs from the true σ_i by an amount satisfying the bound

$$|\hat{\sigma}_i - \sigma_i| \leq p(m,n)\epsilon\sigma_1.$$

Thus large singular values (those near σ_1) are computed to high relative accuracy and small ones may not be.

The angular difference between the computed left singular vector $\hat{u}_i$ and the true u_i satisfies the approximate bound

$$\theta(\hat{u}_i, u_i) \lesssim \frac{p(m,n)\epsilon\|A\|_2}{\mathrm{gap}_i}$$

where

$$\mathrm{gap}_i = \min_{j \neq i}|\sigma_i - \sigma_j|$$

is the *absolute gap* between σ_i and the nearest other singular value. Thus, if σ_i is close to other singular values, its corresponding singular vector u_i may be inaccurate. The same bound applies to the computed right singular vector $\hat{v}_i$ and the true vector v_i. The gaps may be easily obtained from the computed singular values.

Let $\hat{S}$ be the space spanned by a collection of computed left singular vectors $\{\hat{u}_i, i \in I\}$, where I is a subset of the integers from 1 to n. Let S be the corresponding true space. Then

$$\theta(\hat{S}, S) \lesssim \frac{p(m,n)\epsilon\|A\|_2}{\mathrm{gap}_I}.$$

where

$$\text{gap}_I = \min\{|\sigma_i - \sigma_j| \text{ for } i \in I,\ j \notin I\}$$

is the absolute gap between the singular values in I and the nearest other singular value. Thus, a cluster of close singular values which is far away from any other singular value may have a well determined space $\hat{S}$ even if its individual singular vectors are ill-conditioned. The same bound applies to a set of right singular vectors $\{\hat{v}_i, i \in I\}$.

In the special case of bidiagonal matrices, the singular values and singular vectors may be computed much more accurately [3]. A bidiagonal matrix B has nonzero entries only on the main diagonal and the diagonal immediately above it (or immediately below it). Reduction of a dense matrix to bidiagonal form B can introduce additional errors, so the following bounds for the bidiagonal case do not apply to the dense case.

Using the routines in this chapter, each computed singular value of a bidiagonal matrix is accurate to nearly full relative accuracy, no matter how tiny it is, so that

$$|\hat{\sigma}_i - \sigma_i| \le p(m,n)\epsilon\sigma_i.$$

The computed left singular vector $\hat{u}_i$ has an angular error at most about

$$\theta(\hat{u}_i, u_i) \lesssim \frac{p(m,n)\epsilon}{\text{relgap}_i}$$

where

$$\text{relgap}_i = \min_{j \ne i} |\sigma_i - \sigma_j| / (\sigma_i + \sigma_j)$$

is the *relative gap* between σ_i and the nearest other singular value. The same bound applies to the right singular vector $\hat{v}_i$ and v_i. Since the relative gap may be much larger than the absolute gap, this error bound may be much smaller than the previous one. The relative gaps may be easily obtained from the computed singular values.

2.11.3 The symmetric eigenproblem

The usual error analysis of the symmetric eigenproblem is as follows [5].

The computed eigendecomposition $\hat{Z}\hat{\Lambda}\hat{Z}^T$ is nearly the exact eigendecomposition of $A+E$, i.e., $A+E = (\hat{Z}+\delta\hat{Z})\hat{\Lambda}(\hat{Z}+\delta\hat{Z})^T$ is the true eigendecomposition so that $\hat{Z}+\delta\hat{Z}$ is orthogonal, where $\|E\|_2/\|A\|_2 \le p(n)\epsilon$ and $\|\delta\hat{Z}\|_2 \le p(n)\epsilon$ and $p(n)$ is a modestly growing function of n and ϵ is the machine precision. Each computed eigenvalue $\hat{\lambda}_i$ differs from the true λ_i by an amount satisfying the bound

$$|\hat{\lambda}_i - \lambda_i| \le p(n)\epsilon\|A\|_2.$$

Thus large eigenvalues (those near $\max_i|\lambda_i| = \|A\|_2$) are computed to high relative accuracy and small ones may not be.

The angular difference between the computed unit eigenvector $\hat{z}_i$ and the true z_i satisfies the approximate bound

$$\theta(\hat{z}_i, z_i) \lesssim \frac{p(n)\epsilon\|A\|_2}{\text{gap}_i}$$

if $p(n)\epsilon$ is small enough, where

$$\text{gap}_i = \min_{j \ne i} |\lambda_i - \lambda_j|$$

is the *absolute gap* between λ_i and the nearest other eigenvalue. Thus, if λ_i is close to other eigenvalues, its corresponding eigenvector z_i may be inaccurate. The gaps may be easily obtained from the computed eigenvalues.

Let $\hat{S}$ be the invariant subspace spanned by a collection of eigenvectors $\{\hat{z}_i, i \in I\}$, where I is a subset of the integers from 1 to n. Let S be the corresponding true subspace. Then

$$\theta(\hat{S}, S) \lesssim \frac{p(n)\epsilon\|A\|_2}{\text{gap}_I}$$

where

$$\text{gap}_I = \min\{|\lambda_i - \lambda_j| \text{ for } i \in I,\ j \notin I\}$$

is the absolute gap between the eigenvalues in I and the nearest other eigenvalue. Thus, a cluster of close eigenvalues which is far away from any other eigenvalue may have a well determined invariant subspace $\hat{S}$ even if its individual eigenvectors are ill-conditioned.

In the special case of a real symmetric tridiagonal matrix T, routines in this chapter can compute the eigenvalues and eigenvectors much more accurately. See Anderson *et al.*[1] for further details.

2.11.4 The generalized symmetric-definite eigenproblem

The three types of problem to be considered are $A - \lambda B$, $AB - \lambda I$ and $BA - \lambda I$. In each case A and B are real symmetric (or complex Hermitian) and B is positive-definite. We consider each case in turn, assuming that routines in this chapter are used to transform the generalized problem to the standard symmetric problem, followed by the solution of the the symmetric problem. In all cases

$$\mathrm{gap}_i = \min_{j \neq i} |\lambda_i - \lambda_j|$$

is the *absolute gap* between λ_i and the nearest other eigenvalue.

(1) $A - \lambda B$. The computed eigenvalues $\hat{\lambda}_i$ can differ from the true eigenvalues λ_i by an amount

$$|\hat{\lambda}_i - \lambda_i| \lesssim p(n)\epsilon \|B^{-1}\|_2 \|A\|_2.$$

The angular difference between the computed eigenvector $\hat{z}_i$ and the true eigenvector z_i is

$$\theta(\hat{z}_i, z_i) \lesssim \frac{p(n)\epsilon \|B^{-1}\|_2 \|A\|_2 (\kappa_2(B))^{1/2}}{\mathrm{gap}_i}.$$

(2) $AB - \lambda I$ or $BA - \lambda I$. The computed eigenvalues $\hat{\lambda}_i$ can differ from the true eigenvalues λ_i by an amount

$$|\hat{\lambda}_i - \lambda_i| \lesssim p(n)\epsilon \|B\|_2 \|A\|_2.$$

The angular difference between the computed eigenvector $\hat{z}_i$ and the true eigenvector z_i is

$$\theta(\hat{z}_i, z_i) \lesssim \frac{q(n)\epsilon \|B\|_2 \|A\|_2 (\kappa_2(B))^{1/2}}{\mathrm{gap}_i}.$$

These error bounds are large when B is ill-conditioned with respect to inversion ($\kappa_2(B)$ is large). It is often the case that the eigenvalues and eigenvectors are much better conditioned than indicated here. One way to get tighter bounds is effective when the diagonal entries of B differ widely in magnitude, as for example with a *graded matrix*.

(1) $A - \lambda B$. Let $D = \mathrm{diag}(b_{11}^{-1/2}, \ldots, b_{nn}^{-1/2})$ be a diagonal matrix. Then replace B by DBD and A by DAD in the above bounds.

(2) $AB - \lambda I$ or $BA - \lambda I$. Let $D = \mathrm{diag}(b_{11}^{-1/2}, \ldots, b_{nn}^{-1/2})$ be a diagonal matrix. Then replace B by DBD and A by $D^{-1}AD^{-1}$ in the above bounds.

Further details can be found in Anderson *et al.* [1].

2.11.5 The nonsymmetric eigenproblem

The nonsymmetric eigenvalue problem is more complicated than the symmetric eigenvalue problem. In this section, we just summarize the bounds. Further details can be found in Anderson *et al.* [1].

We let $\hat{\lambda}_i$ be the ith computed eigenvalue and λ_i the ith true eigenvalue. Let $\hat{v}_i$ be the corresponding computed right eigenvector, and v_i the true right eigenvector (so $Av_i = \lambda_i v_i$). If I is a subset of the integers from 1 to n, we let λ_I denote the average of the selected eigenvalues: $\lambda_I = (\sum_{i \in I} \lambda_i)/(\sum_{i \in I} 1)$, and similarly for $\hat{\lambda}_I$. We also let S_I denote the subspace spanned by $\{v_i, i \in I\}$; it is called a right invariant subspace because if v is any vector in S_I then Av is also in S_I. $\hat{S}_I$ is the corresponding computed subspace.

The algorithms for the nonsymmetric eigenproblem are normwise backward stable: they compute the exact eigenvalues, eigenvectors and invariant subspaces of slightly perturbed matrices $A + E$, where

$\|E\| \le p(n)\epsilon\|A\|$. Some of the bounds are stated in terms of $\|E\|_2$ and others in terms of $\|E\|_F$; one may use $p(n)\epsilon$ for either quantity.

Routines are provided so that, for each $(\hat{\lambda}_i,\ \hat{v}_i)$ pair the two values s_i and sep_i, or for a selected subset I of eigenvalues the values s_I and sep_I can be obtained, for which the error bounds in Table 2 are true for sufficiently small $\|E\|$, (which is why they are called asymptotic):

Simple eigenvalue	$\lvert\hat{\lambda}_i - \lambda_i\rvert \lesssim \|E\|_2/s_i$
Eigenvalue cluster	$\lvert\hat{\lambda}_I - \lambda_I \lesssim \|E\|_2/s_I$
Eigenvector	$\theta(\hat{\vartheta}_i, \vartheta_i) \lesssim \|E\|_F/sep_i$
Invariant subspace	$\theta(\hat{S}_I, S_I) \lesssim \|E\|_F/sep_I$

Table 2
Asymptotic error bounds for the nonsymmetric eigenproblem

If the problem is ill-conditioned, the asymptotic bounds may only hold for extremely small $\|E\|$. The global error bounds of Table 3 are guaranteed to hold for all $\|E\|_F < s \times sep/4$:

Simple eigenvalue	$\lvert\hat{\lambda}_i - \lambda_i\rvert \le n\|E\|_2/s_i$	Holds for all E
Eigenvalue cluster	$\lvert\hat{\lambda}_I - \lambda_I\rvert \le 2\|E\|_2/s_I$	Requires $\|E\|_F < s_I \times sep_I/4$
Eigenvector	$\theta(\hat{\vartheta}_i, \vartheta_i) \le \arctan(2\|E\|_F/(sep_i - 4\|E\|_F/s_i))$	Requires $\|E\|_F < s_i \times sep_i/4$
Invariant subspace	$\theta(\hat{S}_I, S_I) \le \arctan(2\|E\|_F/(sep_I - 4\|E\|_F/s_I))$	Requires $\|E\|_F < s_I \times sep_I/4$

Table 3
Global error bounds for the nonsymmetric eigenproblem

2.11.6 Balancing and condition

There are two preprocessing steps one may perform on a matrix A in order to make its eigenproblem easier. The first is *permutation*, or reordering the rows and columns to make A more nearly upper triangular (closer to Schur form): $A' = PAP^T$, where P is a permutation matrix. If A' is permutable to upper triangular form (or close to it), then no floating-point operations (or very few) are needed to reduce it to Schur form. The second is *scaling* by a diagonal matrix D to make the rows and columns of A' more nearly equal in norm: $A'' = DA'D^{-1}$. Scaling can make the matrix norm smaller with respect to the eigenvalues, and so possibly reduce the inaccuracy contributed by roundoff (see Chapter, II/11 of [7]). We refer to these two operations as *balancing*.

Permuting has no effect on the condition numbers or their interpretation as described previously. Scaling, however, does change their interpretation and further details can be found in Anderson *et al.* [1].

2.12 Block Algorithms

A number of the routines in this chapter use what is termed a *block algorithm*. This means that at each major step of the algorithm a *block* of rows or columns is updated, and much of the computation is performed by matrix-matrix operations on these blocks. The matrix-matrix operations are performed by calls to the Level 3 BLAS (see Chapter F06), which are the key to achieving high performance on many modern computers. In the case of the QR algorithm for reducing an upper Hessenberg matrix to Schur form, a multishift strategy is used in order to improve performance. See Golub and Van Loan [4] or Anderson *et al.* [1] for more about block algorithms and the multishift strategy.

The performance of a block algorithm varies to some extent with the ***blocksize*** – that is, the number of rows or columns per block. This is a machine-dependent parameter, which is set to a suitable value when the library is implemented on each range of machines. Users of the library do not normally need to be aware of what value is being used. Different block sizes may be used for different routines. Values in the range 16 to 64 are typical.

On more conventional machines there is often no advantage from using a block algorithm, and then the routines use an *unblocked* algorithm (effectively a block size of 1), relying solely on calls to the Level 2 BLAS (see Chapter F06 again).

The only situation in which a user needs some awareness of the block size is when it affects the amount of workspace to be supplied to a particular routine. This is discussed in Section 3.4.3.

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 Available Routines

The tables in the following subsections show the routines which are provided for performing different computations on different types of matrices. Each entry in the table gives the NAG routine name, the LAPACK single precision name, and the LAPACK double precision name (see Section 3.2).

For many computations it is necessary to call two or more routines in sequence some commonly required sequences of routines are indicated below; an asterisk (*) against a routine name means that the sequence of calls is illustrated in the example program for that routine. (But remember that Black Box routines for the same computations may be provided in Chapter F02 or Chapter F04.)

3.1.1 Orthogonal factorizations

Routines are provided for QR factorization (with and without column pivoting), and for LQ factorization (without pivoting only), of a general real or complex rectangular matrix.

The factorization routines do not form the matrix Q explicitly, but represent it as a product of elementary reflectors (see Section 3.3.6). Additional routines are provided to generate all or part of Q explicitly if it is required, or to apply Q in its factored form to another matrix (specifically to compute one of the matrix products QC, Q^TC, CQ or CQ^T with Q^T replaced by Q^H if C and Q are complex.

	Factorize without pivoting	**Factorize with pivoting**	**Generate Matrix Q**	**Apply matrix Q**
QR factorization, real matrices	F08AEF SGEQRF DGEQRF	F08BEF SGEQPF DGEQPF	F08AFF SORGQR DORGQR	F08AGF SORMQR DORMQR
LQ factorization, real matrices	F08AHF SGELQF DGELQF		F08AJF SORGLQ DORGLQ	F08AKF SORMLQ DORMLQ
QR factorization, complex matrices	F08ASF CGEQRF ZGEQRF	F08BSF CGEQPF ZGEQPF	F08ATF CUNGQR ZUNMQR	F08AUF CUNMQR ZUNGQR
LQ factorization, complex matrices	F08AVF CGELQF ZGELQF		F08AWF CUNGLQ ZUNGLQ	F08AXF CUNMQL ZUNMLQ

To solve linear least-squares problems, as described in Section 2.2.1 or Section 2.2.3, routines based on the QR factorization can be used:

real data, full-rank problem	F08AEF*, F08AGF, F06YJF
complex data, full-rank problem	F08ASF*, F08AUF, F06ZJF
real data, rank-deficient problem	F08BEF*, F08AGF, F06YJF
complex data, rank-deficient problem	F08BSF*, F08AUF, F06ZJF

To find the minimum norm solution of under-determined systems of linear equations, as described in Section 2.2.2, routines based on the LQ factorization can be used:

real data, full-rank problem	F08AHF*, F06YJF, F08AKF
complex data, full-rank problem	F08AVF*, F06ZJF, F08AXF

3.1.2 Singular value problems

Routines are provided to reduce a general real or complex rectangular matrix A to real bidiagonal form B by an orthogonal transformation $A = QBP^T$ (or by a unitary transformation $A = QBP^H$ if A is complex). Different routines allow a full matrix A to be stored conventionally (see Section 3.3.1), or a band matrix to use band storage (see Section 3.3.3).

The routines for reducing full matrices do not form the matrix Q or P explicitly; additional routines are provided to generate all or part of them, or to apply them to another matrix, as with the routines for orthogonal factorizations. Explicit generation of Q or P is required before using the bidiagonal QR algorithm to compute left or right singular vectors of A.

The routines for reducing band matrices have options to generate Q or P if required.

Further routines are provided to compute all or part of the singular value decomposition of a real bidiagonal matrix; the same routines can be used to compute the singular value decomposition of a real or complex matrix that has been reduced to bidiagonal form.

	Reduce to bidiagonal form	**Generate matrix Q or P^T**	**Apply matrix Q or P**	**Reduce band matrix to bidiagonal form**	**SVD of bidiagonal form (QR algorithm)**
real matrices	F08KEF SGEBRD DGEBRD	F08KFF SORGBR DORGBR	F08KGF SORMBR DORMBR	F08LEF SGBBRD DGBBRD	F08MEF SBDSQR DBDSQR
complex matrices	F08KSF CGEBRD ZGEBRD	F08KTF CUNGBR ZUNGBR	F08KUF CUNMBR ZUNMBR	F08LSF CGBBRD ZGBBRD	F08MSF CBDSQR ZBDSQR

To compute the singular values and vectors of a rectangular matrix, as described in Section 2.3, use the following sequence of calls:

Rectangular matrix (standard storage)

real matrix, singular values and vectors	F08KEF, F08KFF*, F08MEF
complex matrix, singular values and vectors	F08KSF, F08KTF*, F08MSF

Rectangular matrix (banded)

real matrix, singular values and vectors	F08LEF, F08MEF
complex matrix, singular values and vectors	F08LSF, F08MSF

To use the singular value decomposition to solve a linear least-squares problem, as described in Section 2.4, the following routines are required:

real data:	F08KEF, F08KGF, F08KFF, F08MEF, F06YAF
complex data:	F08KSF, F08KUF, F08KTF, F08MSF, F06ZAF

3.1.3 Symmetric eigenvalue problems

Routines are provided to reduce a real symmetric or complex Hermitian matrix A to real tridiagonal form T by an orthogonal similarity transformation $A = QTQ^T$ (or by a unitary transformation $A = QTQ^H$ if A is complex). Different routines allow a full matrix A to be stored conventionally (see Section 3.3.1) or in packed storage (see Section 3.3.2); or a band matrix to use band storage (see Section 3.3.3).

The routines for reducing full matrices do not form the matrix Q explicitly; additional routines are provided to generate Q, or to apply it to another matrix, as with the routines for orthogonal factorizations. Explicit generation of Q is required before using the QR algorithm to find all the eigenvectors of A; application of Q to another matrix is required after eigenvectors of T have been found by inverse iteration, in order to transform them to eigenvectors of A.

The routines for reducing band matrices have an option to generate Q if required.

	Reduce to tridiagonal form	Generate matrix Q	Apply matrix Q
real symmetric matrices	F08FEF SSYTRD DSYTRD	F08FFF SORGTR DORGTR	F08FGF SORMTR DORMTR
real symmetric matrices (packed storage)	F08GEF SSPTRD DSPTRD	F08GFF SOPGTR DOPGTR	F08GGF SOPMTR DOPMTR
real symmetric band matrices	F08HEF SSBTRD DSBTRD		
complex Hermitian matrices	F08FSF CHETRD ZHETRD	F08FTF CUNGTR ZUNGTR	F08FUF CUNMTR ZUNMTR
complex Hermitian matrices (packed storage)	F08GSF CHPTRD ZHPTRD	F08GTF CUPGTR ZUPGTR	F08GUF CUPMTR ZUPMTR
complex Hermitian band matrices	F08HSF CHBTRD ZHBTRD		

A variety of routines are provided to compute eigenvalues and eigenvectors of the real symmetric tridiagonal matrix T, some computing all eigenvalues and eigenvectors, some computing selected eigenvalues and eigenvectors. The same routines can be used to compute eigenvalues and eigenvectors of a real symmetric or complex Hermitian matrix which has been reduced to tridiagonal form.

Eigenvalues and eigenvectors of real symmetric tridiagonal matrices:

The original (non-reduced) matrix is Real or Complex Hermitian

all eigenvalues (root-free QR algorithm)	F08JFF
all eigenvalues (root-free QR algorithm called by divide and conquer)	F08JCF
selected eigenvalues (bisection)	F08JJF

The original (non-reduced) matrix is Real

all eigenvalues and eigenvectors (QR algorithm)	F08JEF
all eigenvalues and eigenvectors (divide and conquer)	F08JCF
all eigenvalues and eigenvectors (positive-definite case)	F08JGF
selected eigenvectors (inverse iteration)	F08JKF

The original (non-reduced) matrix is Complex Hermitian

all eigenvalues and eigenvectors (QR algorithm)	F08JSF
all eigenvalues and eigenvectors (positive-definite case)	F08JUF
selected eigenvectors (inverse iteration)	F08JXF

The following sequences of calls may be used to compute various combinations of eigenvalues and eigenvectors, as described in Section 2.5.

Sequences for computing eigenvalues and eigenvectors

Real Symmetric matrix (standard storage)

all eigenvalues and eigenvectors (using divide and conquer)	F08FCF
all eigenvalues and eigenvectors (using QR algorithm)	F08FEF, F08FFF*, F08JEF
selected eigenvalues and eigenvectors (bisection and inverse iteration)	F08FEF, F08JJF, F08JKF, F08FGF*

Real Symmetric matrix (packed storage)

all eigenvalues and eigenvectors (using divide and conquer)	F08GCF
all eigenvalues and eigenvectors (using QR algorithm)	F08GEF, F08GFF*, F08JEF
selected eigenvalues and eigenvectors (bisection and inverse iteration)	F08GEF, F08JJF, F08JKF, F08GGF*

Real Symmetric banded matrix

all eigenvalues and eigenvectors (using divide and conquer)	F08HCF
all eigenvalues and eigenvectors (using QR algorithm)	F08HEF*, F08JEF

Complex Hermitian matrix (standard storage)

all eigenvalues and eigenvectors (using divide and conquer)	F08FQF
all eigenvalues and eigenvectors (using QR algorithm)	F08FSF, F08FTF*, F08JSF
selected eigenvalues and eigenvectors (bisection and inverse iteration)	F08FSF, F08JJF, F08JXF, F08FUF*

Complex Hermitian matrix (packed storage)

all eigenvalues and eigenvectors (using divide and conquer)	F08GQF
all eigenvalues and eigenvectors (using QR algorithm)	F08GSF, F08GTF*, F08JSF
selected eigenvalues and eigenvectors (bisection and inverse iteration)	F08GSF, F08JJF, F08JXF, F08GUF*

Complex Hermitian banded matrix

all eigenvalues and eigenvectors (using divide and conquer)	F08HQF
all eigenvalues and eigenvectors (using QR algorithm)	F08HSF*, F08JSF

3.1.4 Generalized symmetric-definite eigenvalue problems

Routines are provided for reducing each of the problems $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$ to an equivalent standard eigenvalue problem $Cy = \lambda y$. Different routines allow the matrices to be stored either conventionally or in packed storage. The positive-definite matrix B must first be factorized using a routine from Chapter F07. There is also a routine which reduces the problem $Ax = \lambda Bx$ where A and B are banded, to an equivalent banded standard eigenvalue problem; this uses a split Cholesky factorization for which a routine in Chapter F08 is provided.

	Reduce to standard problem	**Reduce to standard problem (packed storage)**	**Reduce to standard problem (band matrices)**
real symmetric matrices	F08SEF SSYGST DSYGST	F08TEF SSPGST DSPGST	F08UEF SSBGST DSBGST
complex Hermitian matrices	F08SSF CHEGST ZHEGST	F08TSF CHPGST ZHPGST	F08USF CHBGST ZHBGST

The equivalent standard problem can then be solved using the routines discussed in Section 3.1.3. For example, to compute all the eigenvalues, the following routines must be called:

real symmetric-definite problem	F07FDF, F08SEF*, F08FEF, F08JFF
real symmetric-definite problem, packed storage	F07GDF, F08TEF*, F08GEF, F08JFF
real symmetric-definite banded problem	F08UFF*, F08UEF*, F08HEF, F08JFF
complex Hermitian-definite problem	F07FRF, F08SSF*, F08FSF, F08JFF
complex Hermitian-definite problem, packed storage	F07GRF, F08TSF*, F08GSF, F08JFF
complex Hermitian-definite banded problem	F08UTF*, F08USF*, F08HSF, F08JFF

If eigenvectors are computed, the eigenvectors of the equivalent standard problem must be transformed back to those of the original generalized problem, as indicated in Section 2.6; routines from Chapter F06 may be used for this.

3.1.5 Nonsymmetric eigenvalue problems

Routines are provided to reduce a general real or complex matrix A to upper Hessenberg form H by an orthogonal similarity transformation $A = QHQ^T$ (or by a unitary transformation $A = QHQ^H$ if A is complex).

These routines do not form the matrix Q explicitly; additional routines are provided to generate Q, or to apply it to another matrix, as with the routines for orthogonal factorizations. Explicit generation of Q is required before using the QR algorithm on H to compute the Schur vectors; application of Q to another matrix is needed after eigenvectors of H have been computed by inverse iteration, in order to transform them to eigenvectors of A.

Routines are also provided to balance the matrix before reducing it to Hessenberg form, as described in Section 2.11.6. Companion routines are required to transform Schur vectors or eigenvectors of the balanced matrix to those of the original matrix.

	Reduce to Hessenberg form	**Generate matrix Q**	**Apply matrix Q**	**Balance**	**Backtransform vectors after balancing**
real matrices	F08NEF SGEHRD DGEHRD	F08NFF SORGHR DORGHR	F08NGF SORMHR DORMHR	F08NHF SGEBAL DGEBAL	F08NJF SGEBAK DGEBAK
complex matrices	F08NSF CGEHRD ZGEHRD	F08NTF CUNGHR ZUNGHR	F08NUF CUNMHR ZUNMHR	F08NVF CGEBAL ZGEBAL	F08NWF CGEBAK ZGEBAK

Routines are provided to compute the eigenvalues and all or part of the Schur factorization of an upper Hessenberg matrix. Eigenvectors may be computed either from the upper Hessenberg form by inverse iteration, or from the Schur form by back-substitution; these approaches are equally satisfactory for computing individual eigenvectors, but the latter may provide a more accurate basis for a subspace spanned by several eigenvectors.

Additional routines estimate the sensitivities of computed eigenvalues and eigenvectors, as discussed in Section 2.11.5.

	Eigenvalues and Schur factorization (QR algorithm)	**Eigenvectors from Hessenberg form (inverse iteration)**	**Eigenvectors from Schur factorization**	**Sensitivities of eigenvalues and eigenvectors**
real matrices	F08PEF SHSEQR DHSEQR	F08PKF SHSEIN DHSEIN	F08QKF STREVC DTREVC	F08QLF STRSNA DTRSNA
complex matrices	F08PSF CHSEQR ZHSEQR	F08PXF CHSEIN ZHSEIN	F08QXF CTREVC ZTREVC	F08QYF CTRSNA ZTRSNA

Finally routines are provided for re-ordering the Schur factorization, so that eigenvalues appear in any desired order on the diagonal of the Schur form. The routines F08QFF and F08QTF simply swap two diagonal elements or blocks, and may need to be called repeatedly to achieve a desired order. The routines F08QGF and F08QUF perform the whole re-ordering process for the important special case where a specified cluster of eigenvalues is to appear at the top of the Schur form; if the Schur vectors are re-ordered at the same time, they yield an orthonormal basis of the invariant subspace corresponding to the specified cluster of eigenvalues. These routines can also compute the sensitivities of the cluster of eigenvalues and the invariant subspace.

	Reorder Schur factorization	Reorder Schur factorization, find basis of invariant subspace and estimate sensitivities
real matrices	F08QFF STREXC DTREXC	F08QGF STRSEN DTRSEN
complex matrices	F08QTF CTREXC ZTREXC	F08QUF CTRSEN ZTRSEN

The following sequences of calls may be used to compute various combinations of eigenvalues, Schur vectors and eigenvectors, as described in Section 2.9:

real matrix, all eigenvalues and Schur factorization	F08NEF, F08NFF*, F08PEF
real matrix, all eigenvalues and selected eigenvectors	F08NEF, F08PEF, F08PKF, F08NGF*
real matrix, all eigenvalues and eigenvectors (with balancing)	F08NHF*, F08NEF, F08NFF, F08PEF, F08PKF, F08NJF
complex matrix, all eigenvalues and Schur factorization	F08NSF, F08NTF*, F08PSF
complex matrix, all eigenvalues and selected eigenvectors	F08NSF, F08PSF, F08PXF, F08NUF*
complex matrix, all eigenvalues and eigenvectors (with balancing)	F08NVF*, F08NSF, F08NTF, F08PSF, F08PXF, F08NWF

3.1.6 Sylvester's equation

Routines are provided to solve the real or complex Sylvester equation $AX \pm XB = C$, where A and B are upper quasi-triangular if real, or upper triangular if complex. To solve the general form of Sylvester's equation in which A and B are general square matrices, A and B must be reduced to upper (quasi-) triangular form by the Schur factorization,using routines described in Section 3.1.5. For more details, see the documents for the routines listed below.

	solve Sylvester's equation
real matrices	F08QHF STRSYL DTRSYL
complex matrices	F08QVF CTRSYL ZTRSYL

3.2 NAG Names and LAPACK Names

As well as the NAG routine name (beginning F08-), the tables in Section 3.1 show the LAPACK routine names in both single and double precision.

The routines may be called either by their NAG names or by their LAPACK names. When using a single precision implementation of the NAG Library, the single precision form of the LAPACK name must be used (beginning with S- or C-); when using a double precision implementation of the NAG Library, the double precision form of the LAPACK name must be used (beginning with D- or Z-).

References to F08 routines in the Manual normally include the LAPACK single and double precision names, in that order – for example F08AEF (SGEQRF/DGEQRF). The LAPACK routine names follow a simple scheme (which is similar to that used for the BLAS in Chapter F06). Each name has the structure **XYYZZZ**, where the components have the following meanings:

– the initial letter **X** indicates the data type (real or complex) and precision:

S – real, single precision (in Fortran 77, REAL)

D – real, double precision (in Fortran 77, DOUBLE PRECISION)

C – complex, single precision (in Fortran 77, `COMPLEX`)

Z – complex, double precision (in Fortran 77, `COMPLEX*16 or DOUBLE COMPLEX`)

– the 2nd and 3rd letters **YY** indicate the type of the matrix A (and in some cases its storage scheme):

BD – bidiagonal

GB – general band

GE – general

HS – upper Hessenberg

OP – (real) orthogonal (packed storage)

UP – (complex) unitary (packed storage)

OR – (real) orthogonal

UN – (complex) unitary

PT – symmetric or Hermitian positive-definite tridiagonal

SB – (real) symmetric band

HB – (complex) Hermitian band

SP – symmetric (packed storage)

HP – Hermitian (packed storage)

ST – (real) symmetric tridiagonal

SY – symmetric

HE – Hermitian

TR – triangular (or quasi-triangular)

– the last 3 letters **ZZZ** indicate the computation performed. For example, QRF is a QR factorization.

Thus the routine SGEQRF performs a QR factorization of a real general matrix in a single precision implementation of the Library; the corresponding routine in a double precision implementation is DGEQRF.

Some sections of the routine documents – Section 2 (Specification) and Section 9.1 (Example program) – print the LAPACK name in ***bolditalics***, according to the NAG convention of using bold italics for precision-dependent terms – for example, ***sgeqrf***, which should be interpreted as either SGEQRF (in single precision) or DGEQRF (in double precision).

3.3 Matrix Storage Schemes

In this chapter the following storage schemes are used for matrices:

- conventional storage in a two-dimensional array;
- packed storage for symmetric or Hermitian matrices;
- packed storage for orthogonal or unitary matrices;
- band storage for general, symmetric or Hermitian band matrices;
- storage of bidiagonal, symmetric or Hermitian tridiagonal matrices in two one-dimensional arrays.

These storage schemes are compatible with those used in Chapter F06 and Chapter F07, but different schemes for packed, band and tridiagonal storage are used in a few older routines in Chapter F01, Chapter F02, Chapter F03 and Chapter F04.

In the examples below, $*$ indicates an array element which need not be set and is not referenced by the routines. The examples illustrate only the relevant leading rows and columns of the arrays; array arguments may of course have additional rows or columns, according to the usual rules for passing array arguments in Fortran 77.

3.3.1 Conventional storage

The default scheme for storing matrices is the obvious one: a matrix A is stored in a two-dimensional array A, with matrix element a_{ij} stored in array element A(i,j).

If a matrix is *triangular* (upper or lower, as specified by the argument UPLO when present), only the elements of the relevant triangle are stored; the remaining elements of the array need not be set. Such elements are indicated by $*$ in the examples below. For example, when $n = 4$:

UPLO	Triangular matrix A	Storage in array A
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

Similarly, if the matrix is upper Hessenberg, or if the matrix is quasi-upper triangular, elements below the first subdiagonal need not be set.

Routines that handle *symmetric* or *Hermitian* matrices allow for either the upper or lower triangle of the matrix (as specified by UPLO) to be stored in the corresponding elements of the array; the remaining elements of the array need not be set. For example, when $n = 4$:

UPLO	Hermitian matrix A	Storage in array A
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} \\ \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

3.3.2 Packed storage

Symmetric and Hermitian matrices may be stored more compactly, if the relevant triangle (again as specified by UPLO) is packed *by columns* in a one-dimensional array. In Chapter F07 and Chapter F08, arrays that hold matrices in packed storage, have argument names ending in 'P'. So:

if UPLO = 'U', a_{ij} is stored in AP$(i + j(j-1)/2)$ for $i \leq j$;

if UPLO = 'L', a_{ij} is stored in AP$(i + (2n-j)(j-1)/2)$ for $j \leq i$.

For example:

UPLO	Triangle of matrix A	Packed storage in array AP
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$a_{11}\ \underbrace{a_{12}a_{22}}\ \underbrace{a_{13}a_{23}a_{33}}\ \underbrace{a_{14}a_{24}a_{34}a_{44}}$
'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}a_{21}a_{31}a_{41}}\ \underbrace{a_{22}a_{32}a_{42}}\ \underbrace{a_{33}a_{43}}\ a_{44}$

Note that for symmetric matrices, packing the upper triangle by columns is equivalent to packing the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the upper triangle by rows. For Hermitian matrices, packing the upper triangle by columns is equivalent to packing the conjugate of the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the conjugate of the upper triangle by rows.

3.3.3 Band storage

A general m by n band matrix with k_l subdiagonals and k_u superdiagonals may be stored compactly in a two-dimensional array with $k_l + k_u + 1$ rows and n columns. Columns of the matrix are stored in corresponding columns of the array, and diagonals of the matrix are stored in rows of the array. This storage scheme should be used in practice only if $k_l, k_u \ll n$, although routines in Chapter F07 and Chapter F08 work correctly for all values of k_l and k_u. In Chapter F07 and Chapter F08, arrays that hold matrices in band storage have argument names ending in 'B'. So:

a_{ij} is stored in $\mathrm{AB}(k_u + 1 + i - j, j)$ for $\max(1, j - k_u) \le i \le \min(m, j + k_l)$.

For example, when $m = 6$, $n = 5$, $k_l = 2$ and $k_u = 1$:

general band matrix A	Band storage in array AB
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \\ & & & a_{64} & a_{65} \end{pmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} \\ a_{31} & a_{42} & a_{53} & a_{64} & * \end{matrix}$

A symmetric or Hermitian band matrix with k subdiagonals and superdiagonals may be stored more compactly in a two-dimensional array with $k + 1$ rows and n columns. Only the upper or lower triangle (as specified by UPLO) need to be stored. So:

if UPLO = 'U', a_{ij} is stored in $\mathrm{AB}(k + 1 + i - j, j)$ for $\max(1, j - k) \le i \le j$;

if UPLO = 'L', a_{ij} is stored in $\mathrm{AB}(1 + i - j, j)$ for $j \le i \le \min(n, j + k)$.

For example, when $n = 5$ and $k = 2$:

UPLO	Hermitian band matrix A	Band storage in array AB
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} & \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} & a_{35} \\ & \bar{a}_{24} & \bar{a}_{34} & a_{44} & a_{45} \\ & & \bar{a}_{35} & \bar{a}_{45} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & & \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} & \bar{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \bar{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

3.3.4 Tridiagonal and bidiagonal matrices

A symmetric tridiagonal or bidiagonal matrix is stored in two one-dimensional arrays, one of length n containing the diagonal elements, and one of length $n-1$ containing the off-diagonal elements. (Older routines in Chapter F02 store the off-diagonal elements in elements $2:n$ of a vector of length n.)

3.3.5 Real diagonal elements of complex matrices

Complex Hermitian matrices have diagonal matrices that are by definition purely real. In addition, some complex triangular matrices computed by F08 routines are defined by the algorithm to have real diagonal elements – in QR factorization, for example.

If such matrices are supplied as input to F08 routines, the imaginary parts of the diagonal elements are not referenced, but are assumed to be zero. If such matrices are returned as output by F08 routines, the computed imaginary parts are explicitly set to zero.

3.3.6 Representation of orthogonal or unitary matrices

A real orthogonal or complex unitary matrix (usually denoted Q) is often represented in the NAG Library as a product of *elementary reflectors* – also referred to as *elementary Householder matrices* (usually denoted H_i). For example,

$$Q = H_1 H_2 \ldots H_k.$$

Most users need not be aware of the details, because routines are provided to work with this representation, either to generate all or part of Q explicitly, or to multiply a given matrix by Q or Q^T (Q^H in the complex case) without forming Q explicitly.

Nevertheless, the following further details may occasionally be useful.

An elementary reflector (or elementary Householder matrix) H of order n is a unitary matrix of the form

$$H = I - \tau v v^H \tag{4}$$

where τ is a scalar, and v is an n element vector, with $|\tau|^2\|v\|_2^2 = 2 \times \mathrm{Re}(\tau)$; v is often referred to as the *Householder vector*. Often v has several leading or trailing zero elements, but for the purpose of this discussion assume that H has no such special structure.

There is some redundancy in the representation (4), which can be removed in various ways. The representation used in Chapter F08 and in LAPACK (which differs from those used in some of the routines in Chapter F01, Chapter F02, Chapter F04 and Chapter F06) sets $v_1 = 1$; hence v_1 need not be stored. In real arithmetic, $1 \le \tau \le 2$, except that $\tau = 0$ implies $H = I$.

In complex arithmetic, τ may be complex, and satisfies $1 \le \mathrm{Re}(\tau) \le 2$ and $|\tau - 1| \le 1$. Thus a complex H is not Hermitian (as it is in other representations), but it is unitary, which is the important property. The advantage of allowing τ to be complex is that, given an arbitrary complex vector x, H can be computed so that

$$H^H x = \beta(1, 0, \ldots, 0)^T$$

with *real* β. This is useful, for example, when reducing a complex Hermitian matrix to real symmetric tridiagonal form, or a complex rectangular matrix to real bidiagonal form.

3.4 Parameter Conventions

3.4.1 Option parameters

Most routines in this chapter have one or more option parameters, of type CHARACTER. The descriptions in Section 5 of the routine documents refer only to upper case values (for example 'U' or 'L'); however in every case, the corresponding lower case characters may be supplied (with the same meaning). Any other value is illegal.

A longer character string can be passed as the actual parameter, making the calling program more readable, but only the first character is significant. (This is a feature of Fortran 77.) For example:

```
CALL SSYTRD ('Upper', . . . )
```

3.4.2 Problem dimensions

It is permissible for the problem dimensions (for example, M or N) to be passed as zero, in which case the computation (or part of it) is skipped. Negative dimensions are regarded as an error.

3.4.3 Length of work arrays

A number of routines implementing block algorithms require workspace sufficient to hold one block of rows or columns of the matrix if they are to achieve optimum levels of performance – for example, workspace of size $n \times nb$, where nb is the optimum block size. In such cases, the actual declared length of the work array must be passed as a separate argument LWORK, which immediately follows WORK in the argument-list.

The routine will still perform correctly when less workspace is provided: it simply uses the largest block size allowed by the amount of workspace supplied, as long as this is likely to give better performance than the unblocked algorithm. On exit, WORK(1) contains the minimum value of LWORK which would allow the routine to use the optimum block size; this value of LWORK can be used for subsequent runs.

If LWORK indicates that there is insufficient workspace to perform the unblocked algorithm, this is regarded as an illegal value of LWORK, and is treated like any other illegal parameter value (see Section 3.4.4).

If you are in doubt how much workspace to supply and are concerned to achieve optimum performance, supply a generous amount (assume a block size of 64, say), and then examine the value of WORK(1) on exit.

3.4.4 Error-handling and the diagnostic parameter INFO

Routines in this chapter do not use the usual NAG Library error-handling mechanism, involving the parameter IFAIL. Instead they have a diagnostic parameter INFO. (Thus they preserve complete compatibility with the LAPACK specification.)

Whereas IFAIL is an *Input/Output* parameter and must be set before calling a routine, INFO is purely an *Output* parameter and need not be set before entry.

INFO indicates the success or failure of the computation, as follows:

INFO = 0: successful termination

INFO < 0: failure in the course of computation, control returned to the calling program

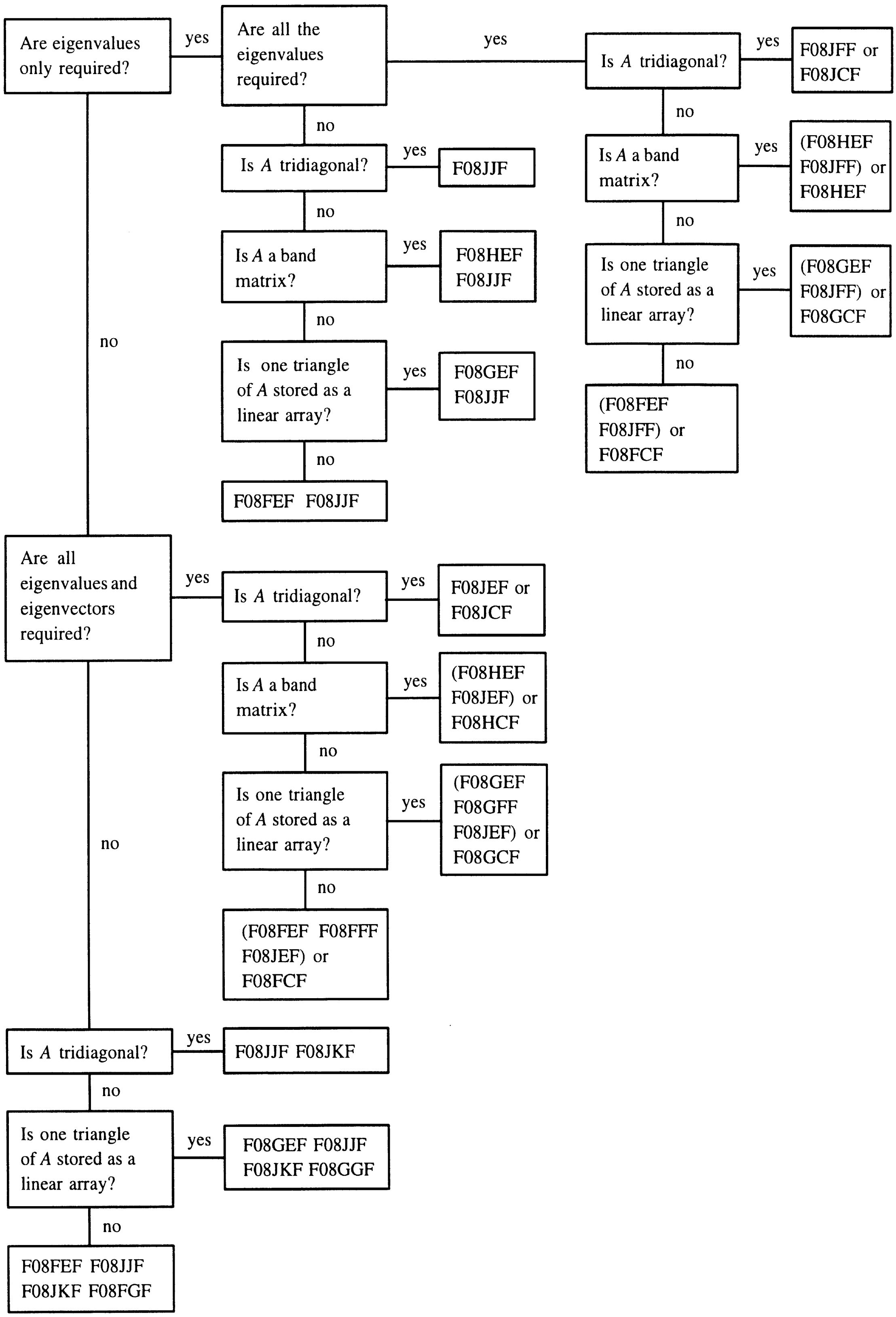
If the routine document specifies that the routine may terminate with INFO < 0, then it is **essential to test INFO on exit** from the routine. (This corresponds to a *soft failure* in terms of the usual NAG error-handling terminology.) No error message is output.

All routines check that input parameters such as N or LDA or option parameters of type CHARACTER have permitted values. If an illegal value of the ith parameter is detected, INFO is set to $-i$, a message is output, and execution of the program is terminated. (This corresponds to a *hard failure* in the usual NAG terminology.)

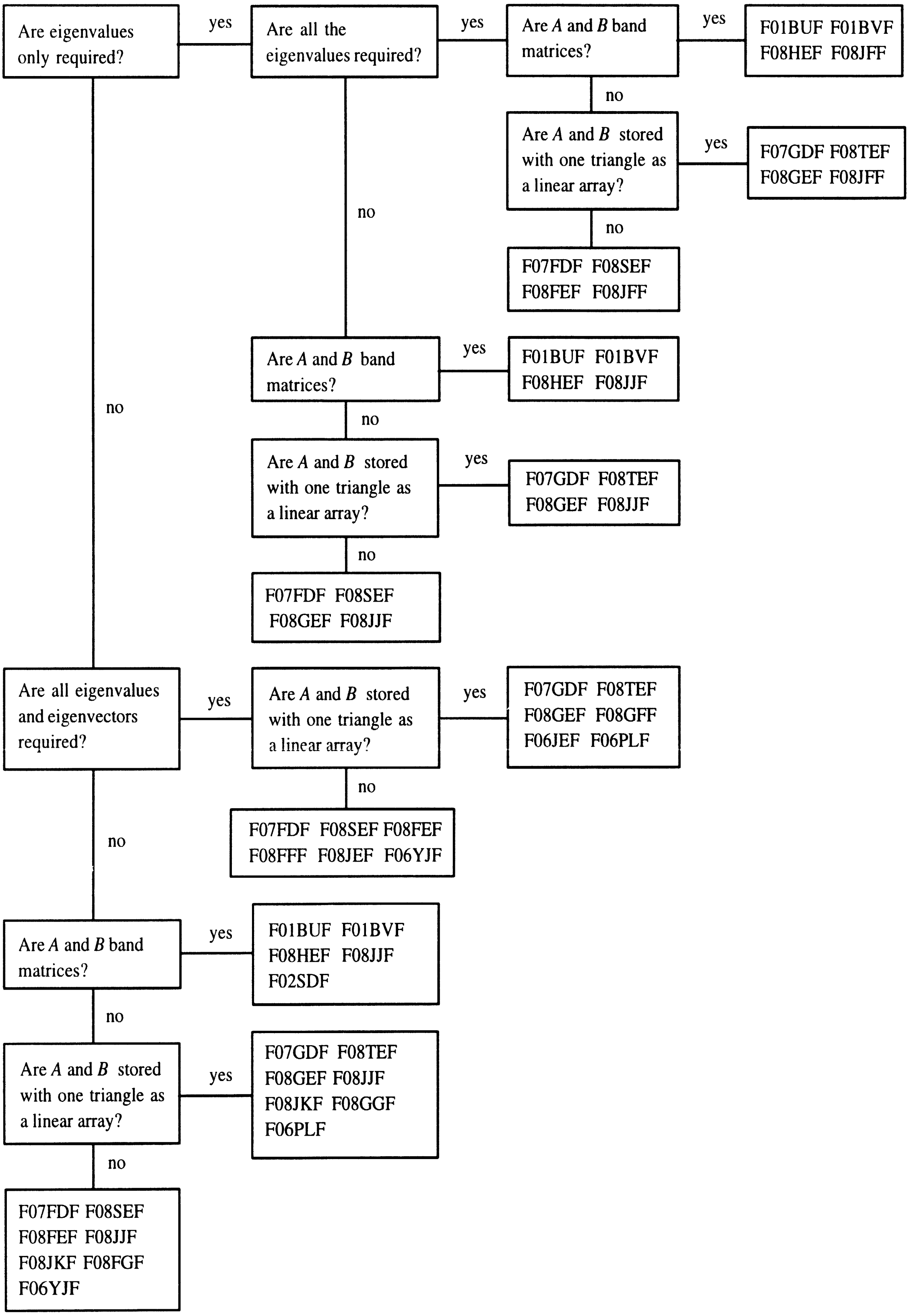
4 Decision Trees

4.1 General purpose routines (eigenvalues and eigenvectors)

Tree 1: Real Symmetric Matrices

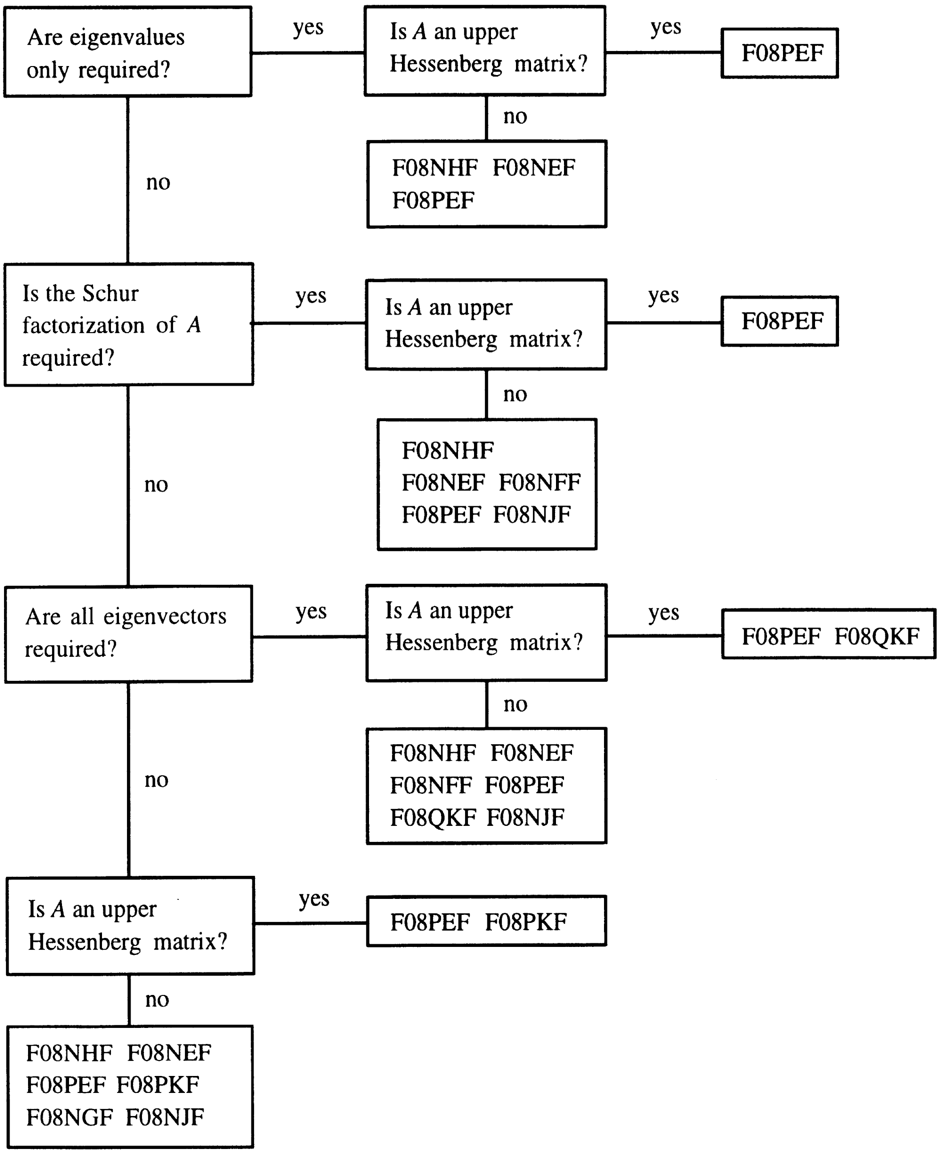


Tree 2: Real Generalized Symmetric-definite Eigenvalue Problems

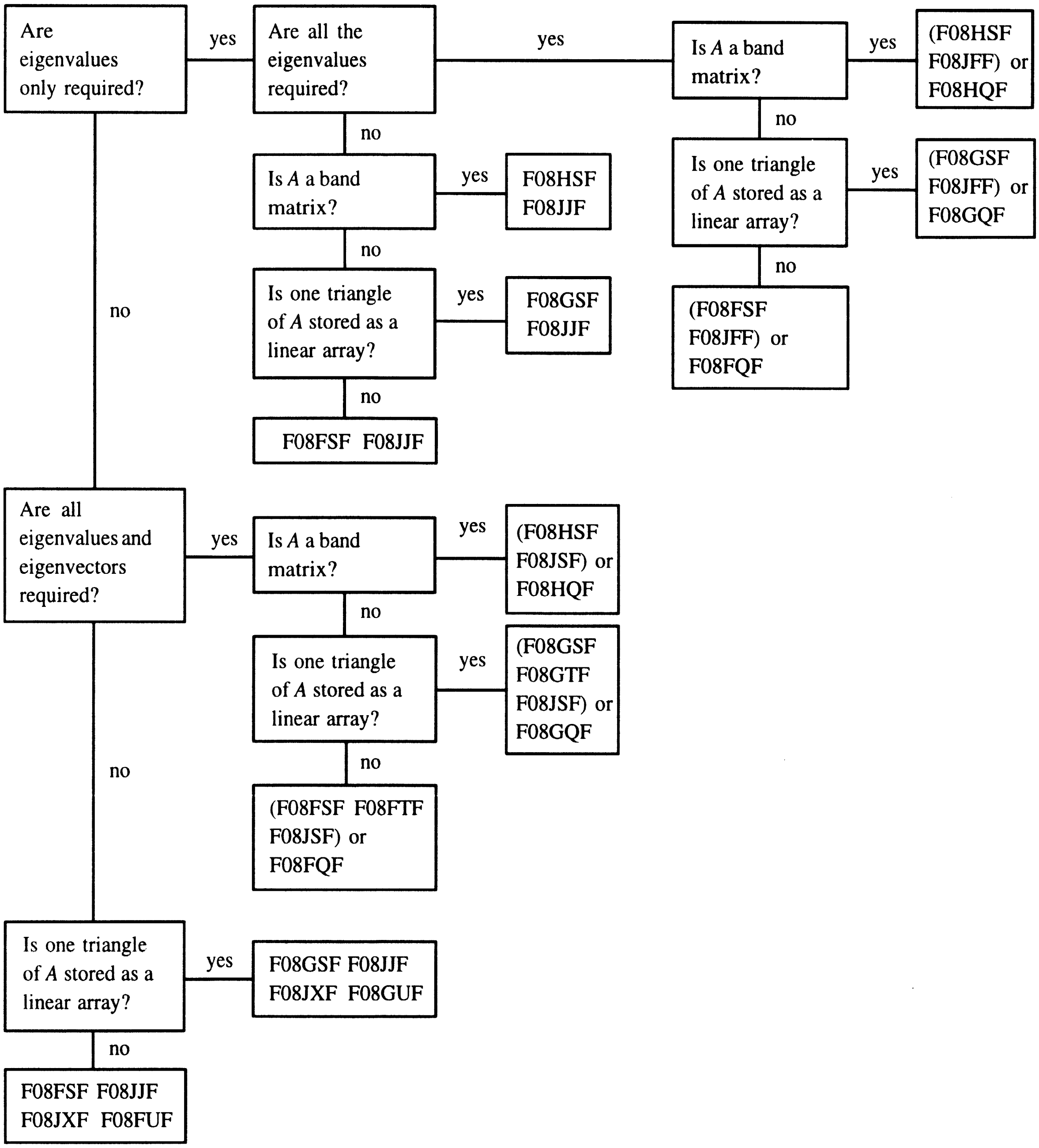


Note: the routines for band matrices only handle the problem $Ax = \lambda Bx$; the other routines handle all three types of problems ($Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$) except that, if the problem is $BAx = \lambda x$ and eigenvectors are required, F06PHF must be used instead of F06PLF, and F06YFF instead of F06YJF.

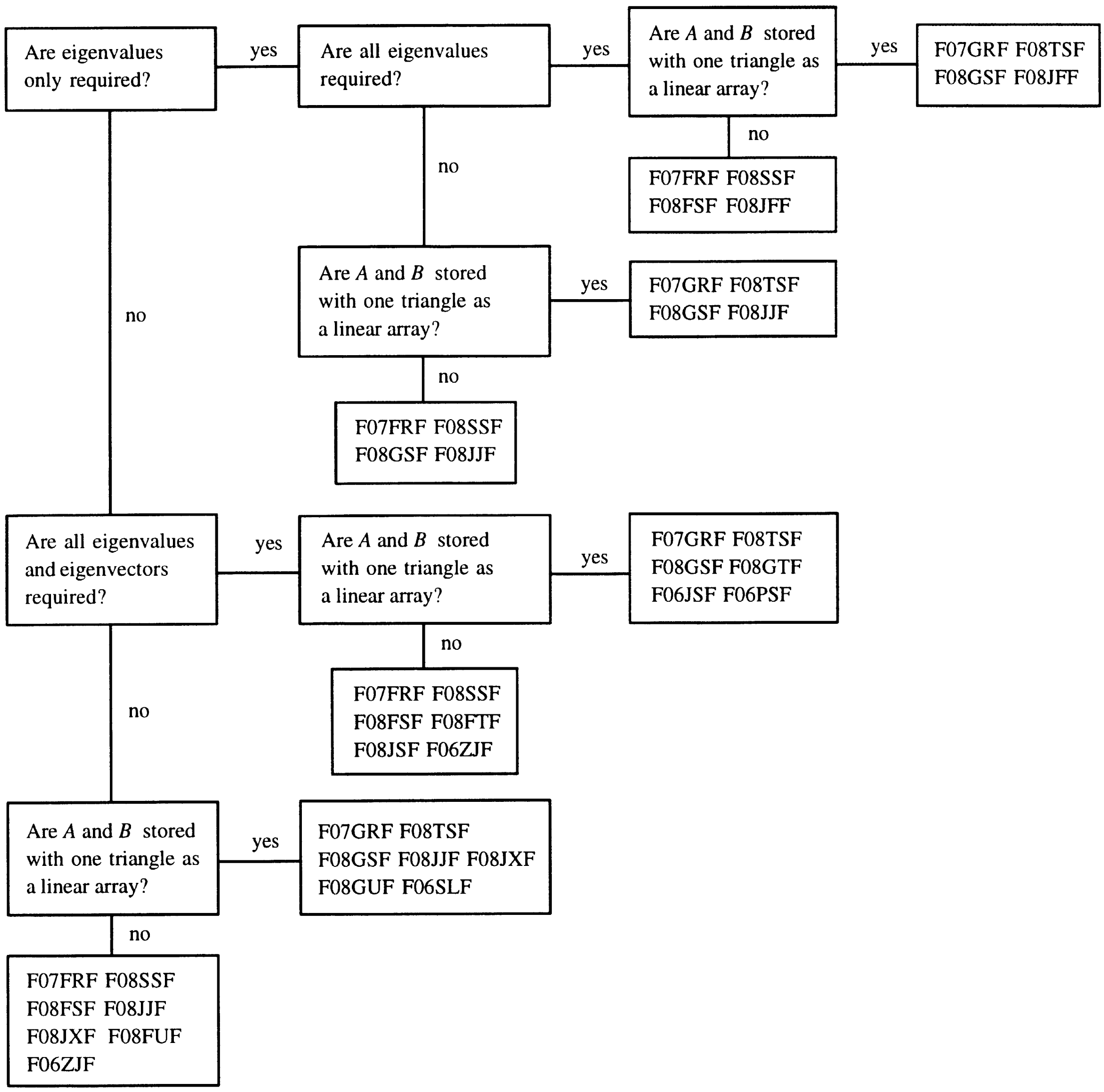
Tree 3: Real Nonsymmetric Matrices



Tree 4: Complex Hermitian Matrices

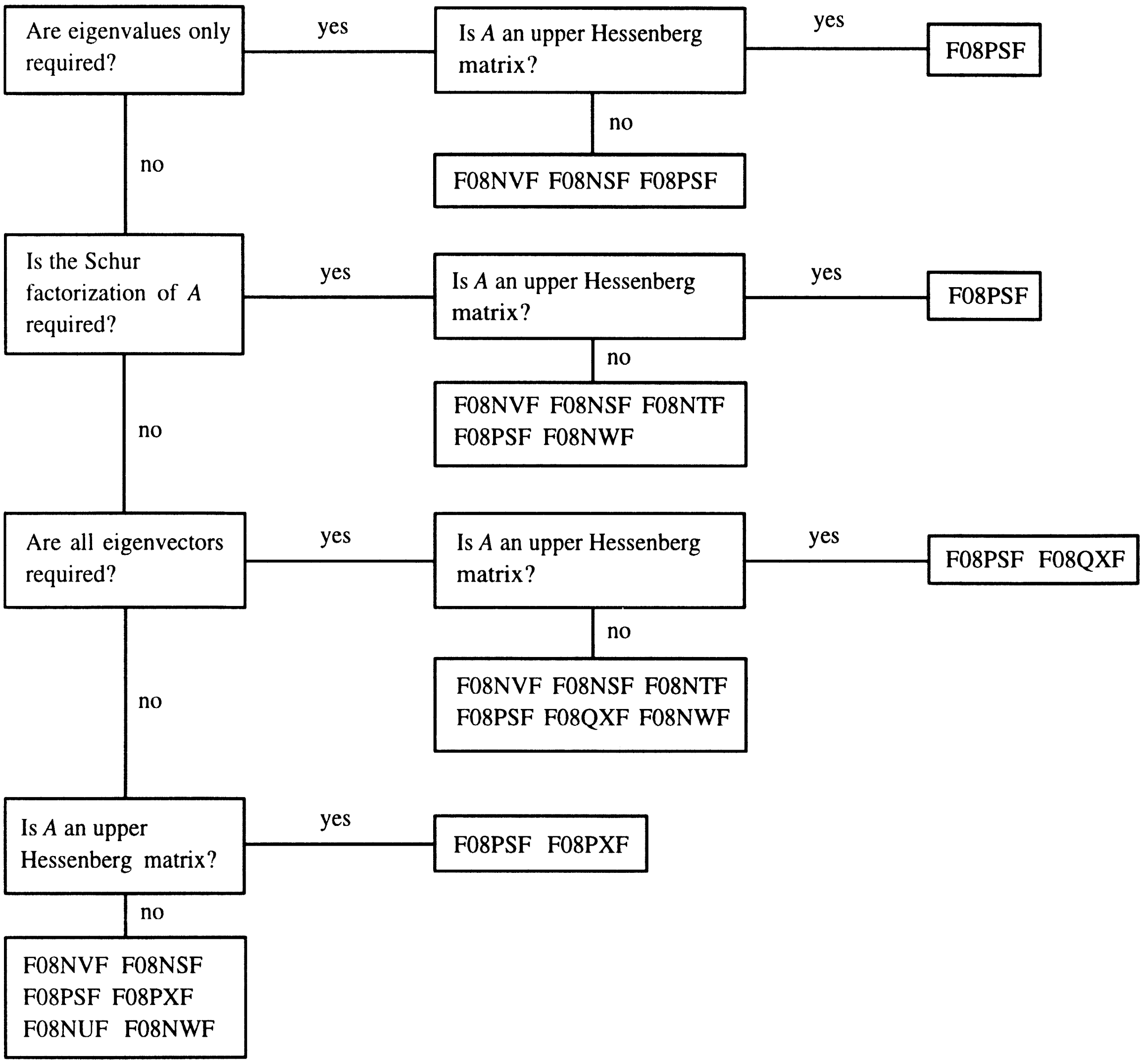


Tree 5: Complex Generalized Hermitian-definite Eigenvalue Problems

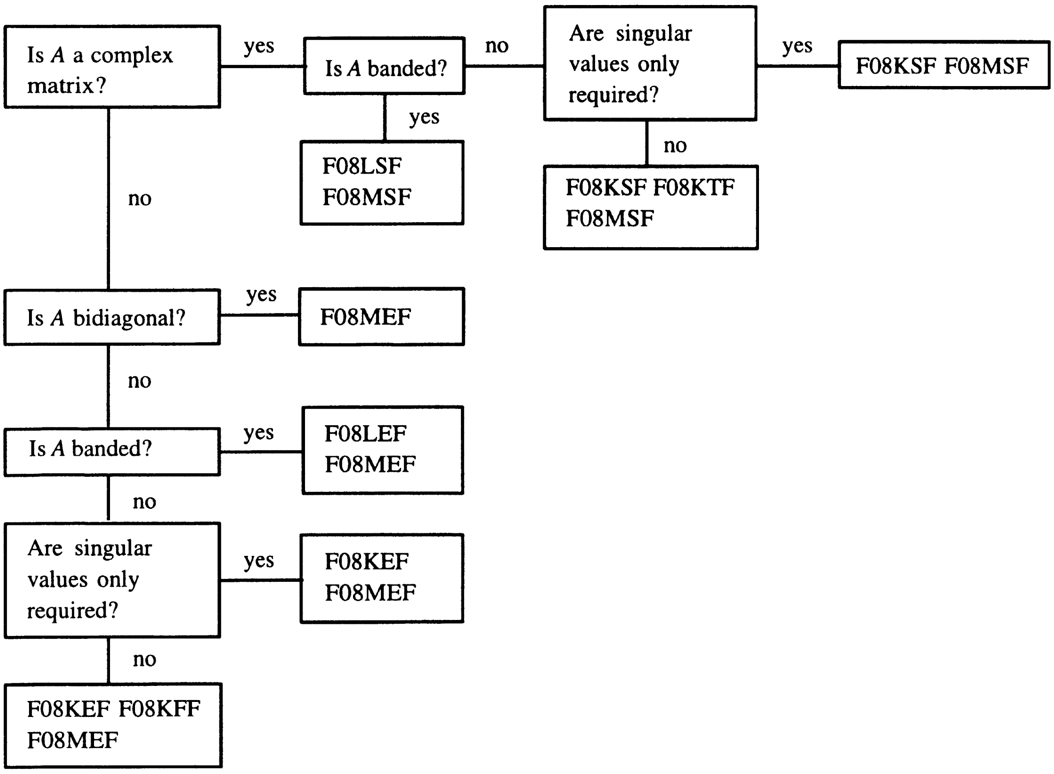


Note: the same routines are required for all three types of problem ($Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$) except that, if the problem is $BAx = \lambda x$ and eigenvectors are required, F06SHF must be used instead of F06SLF, and F06ZFF instead of F06ZJF.

Tree 6: Complex Nonhermitian Matrices



4.2 General purpose routines (singular value decomposition)



5 Indexes of LAPACK Routines

Real Matrices		
LAPACK single precision	LAPACK double precision	NAG
SBDSQR	DBDSQR	F08MEF
SGBBRD	DGBBRD	F08LEF
SGEBAK	DGEBAK	F08NJF
SGEBAL	DGEBAL	F08NHF
SGEBRD	DGEBRD	F08KEF
SGEHRD	DGEHRD	F08NEF
SGELQF	DGELQF	F08AHF
SGEQPF	DGEQPF	F08BEF
SGEQRF	DGEQRF	F08AEF
SHSEIN	DHSEIN	F08PKF
SHSEQR	DHSEQR	F08PEF
SOPGTR	DOPGTR	F08GFF
SOPMTR	DOPMTR	F08GGF
SORGBR	DORGBR	F08KFF
SORGHR	DORGHR	F08NFF
SORGLQ	DORGLQ	F08AJF
SORGQR	DORGQR	F08AFF
SORGTR	DORGTR	F08FFF
SORMBR	DORMBR	F08KGF
SORMHR	DORMHR	F08NGF
SORMLQ	DORMLQ	F08AKF
SORMQR	DORMQR	F08AGF
SORMTR	DORMTR	F08FGF
SPBSTF	DPBSTF	F08UFF
SPTEQR	DPTEQR	F08JGF
SSBEVD	DSBEVD	F08HCF
SSBGST	DSBGST	F08UEF
SSBTRD	DSBTRD	F08HEF
SSPEVD	DSPEVD	F08GCF
SSPGST	DSPGST	F08TEF
SSPTRD	DSPTRD	F08GEF
SSTEBZ	DSTEBZ	F08JJF
SSTEIN	DSTEIN	F08JKF
SSTEQR	DSTEQR	F08JEF
SSTERF	DSTERF	F08JFF
SSTEVD	DSTEVD	F08JCF
SSYEVD	DSYEVD	F08FCF
SSYGST	DSYGST	F08SEF
SSYTRD	DSYTRD	F08FEF
STREVC	DTREVC	F08QKF
STREXC	DTREXC	F08QFF
STRSEN	DTRSEN	F08QGF
STRSNA	DTRSNA	F08QLF
STRSYL	DTRSYL	F08QHF

Complex Matrices		
LAPACK single precision	LAPACK double precision	NAG
CBDSQR	ZBDSQR	F08MSF
CGBBRD	ZGBBRD	F08LSF
CGEBAK	ZGEBAK	F08NWF
CGEBAL	ZGEBAL	F08NVF
CGEBRD	ZGEBRD	F08KSF
CGEHRD	ZGEHRD	F08NSF
CGELQF	ZGELQF	F08AVF
CGEQPF	ZGEQPF	F08BSF
CGEQRF	ZGEQRF	F08ASF
CHBEVD	ZHBEVD	F08HQF
CHBGST	ZHBGST	F08USF
CHBTRD	ZHBTRD	F08HSF
CHEEVD	ZHEEVD	F08FQF
CHEGST	ZHEGST	F08SSF
CHETRD	ZHETRD	F08FSF
CHPEVD	ZHPEVD	F08GQF
CHPGST	ZHPGST	F08TSF
CHPTRD	ZHPTRD	F08GSF
CHSEIN	ZHSEIN	F08PXF
CHSEQR	ZHSEQR	F08PSF
CPBSTF	ZPBSTF	F08UTF
CPTEQR	ZPTEQR	F08JUF
CSTEIN	ZSTEIN	F08JXF
CSTEQR	ZSTEQR	F08JSF
CTREVC	ZTREVC	F08QXF
CTREXC	ZTREXC	F08QTF
CTRSEN	ZTRSEN	F08QUF
CTRSNA	ZTRSNA	F08QYF
CTRSYL	ZTRSYL	F08QVF
CUNGBR	ZUNGBR	F08KTF
CUNGHR	ZUNGHR	F08NTF
CUNGLQ	ZUNGLQ	F08AWF
CUNGQR	ZUNGQR	F08ATF
CUNGTR	ZUNGTR	F08FTF
CUNMBR	ZUNMBR	F08KUF
CUNMHR	ZUNMHR	F08NUF
CUNMLQ	ZUNMLQ	F08AXF
CUNMQR	ZUNMQR	F08AUF
CUNMTR	ZUNMTR	F08FUF
CUPGTR	ZUPGTR	F08GTF
CUPMTR	ZUPMTR	F08GUF

Table 4

6 Routines Withdrawn or Scheduled for Withdrawal

None since Mark 13.

7 References

[1] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A, Ostrouchov S and Sorensen D (1995) *LAPACK Users' Guide* (2nd Edition) SIAM, Philadelphia

[2] Arioli M, Duff I S and De Rijk P P M (1989) On the augmented system approach to sparse least-squares problems *Numer. Math.* **55** 667–684

[3] Demmel J W and Kahan W (1990) Accurate singular values of bidiagonal matrices *SIAM J. Sci. Statist. Comput.* **11** 873–912

[4] Golub G H and Van Loan C F (1996) *Matrix Computations* Johns Hopkins University Press (3rd Edition), Baltimore

[5] Parlett B N (1980) *The Symmetric Eigenvalue Problem* Prentice–Hall

[6] Wilkinson J H (1965) *The Algebraic Eigenvalue Problem* Oxford University Press, London

[7] Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation II, Linear Algebra* Springer–Verlag

Chapter F11 – Sparse Linear Algebra

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
F11BAF**	18	Real sparse nonsymmetric linear systems, set-up for F11BBF
F11BBF**	18	Real sparse nonsymmetric linear systems, preconditioned RGMRES, CGS or Bi-CGSTAB
F11BCF**	18	Real sparse nonsymmetric linear systems, diagnostic for F11BBF
F11BDF	19	Real sparse nonsymmetric linear systems, set-up for F11BEF
F11BEF	19	Real sparse nonsymmetric linear systems, preconditioned RGMRES, CGS, Bi-CGSTAB or TFQMR method
F11BFF	19	Real sparse nonsymmetric linear systems, diagnostic for F11BEF
F11BRF	19	Complex sparse non-Hermitian linear systems, set-up for F11BSF
F11BSF	19	Complex sparse non-Hermitian linear systems, preconditioned RGMRES, CGS, Bi-CGSTAB or TFQMR method
F11BTF	19	Complex sparse non-Hermitian linear systems, diagnostic for F11BSF
F11DAF	18	Real sparse nonsymmetric linear systems, incomplete *LU* factorization
F11DBF	18	Solution of linear system involving incomplete *LU* preconditioning matrix generated by F11DAF
F11DCF	18	Solution of real sparse nonsymmetric linear system, RGMRES, CGS or Bi-CGSTAB method, preconditioner computed by F11DAF (Black Box)
F11DDF	18	Solution of linear system involving preconditioning matrix generated by applying SSOR to real sparse nonsymmetric matrix
F11DEF	18	Solution of real sparse nonsymmetric linear system, RGMRES, CGS or Bi-CGSTAB method, Jacobi or SSOR preconditioner (Black Box)
F11DNF	19	Complex sparse non-Hermitian linear systems, incomplete *LU* factorization
F11DPF	19	Solution of complex linear system involving incomplete *LU* preconditioning matrix generated by F11DNF
F11DQF	19	Solution of complex sparse non-Hermitian linear system, RGMRES, CGS, Bi-CGSTAB or TFQMR method, preconditioner computed by F11DNF (Black Box)
F11DRF	19	Solution of linear system involving preconditioning matrix generated by applying SSOR to complex sparse non-Hermitian matrix
F11DSF	19	Solution of complex sparse non-Hermitian linear system, RGMRES, CGS, Bi-CGSTAB or TFQMR method, Jacobi or SSOR preconditioner (Black Box)
F11GAF	17	Real sparse symmetric linear systems, set-up for F11GBF
F11GBF	17	Real sparse symmetric linear systems, preconditioned conjugate gradient or Lanczos
F11GCF	17	Real sparse symmetric linear systems, diagnostic for F11GBF
F11JAF	17	Real sparse symmetric matrix, incomplete Cholesky factorization
F11JBF	17	Solution of linear system involving incomplete Cholesky preconditioning matrix generated by F11JAF
F11JCF	17	Solution of real sparse symmetric linear system, conjugate gradient/Lanczos method, preconditioner computed by F11JAF (Black Box)
F11JDF	17	Solution of linear system involving preconditioning matrix generated by applying SSOR to real sparse symmetric matrix
F11JEF	17	Solution of real sparse symmetric linear system, conjugate gradient/Lanczos method, Jacobi or SSOR preconditioner (Black Box)
F11JNF	19	Complex sparse Hermitian matrix, incomplete Cholesky factorization
F11JPF	19	Solution of complex linear system involving incomplete Cholesky preconditioning matrix generated by F11JNF

F11JQF	19	Solution of complex sparse Hermitian linear system, conjugate gradient/Lanczos method, preconditioner computed by F11JNF (Black Box)
F11JRF	19	Solution of linear system involving preconditioning matrix generated by applying SSOR to complex sparse Hermitian matrix
F11JSF	19	Solution of complex sparse Hermitian linear system, conjugate gradient/Lanczos method, Jacobi or SSOR preconditioner (Black Box)
F11XAF	18	Real sparse nonsymmetric matrix vector multiply
F11XEF	17	Real sparse symmetric matrix vector multiply
F11XNF	19	Complex sparse non-Hermitian matrix vector multiply
F11XSF	19	Complex sparse Hermitian matrix vector multiply
F11ZAF	18	Real sparse nonsymmetric matrix reorder routine
F11ZBF	17	Real sparse symmetric matrix reorder routine
F11ZNF	19	Complex sparse non-Hermitian matrix reorder routine
F11ZPF	19	Complex sparse Hermitian matrix reorder routine

** This routine has been superseded, although it will be retained in the Library until at least Mark 21. See the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines' for details of the recommended replacement routine.

Chapter F11

Sparse Linear Algebra

Contents

1 Scope of the Chapter

This chapter provides routines for the solution of large sparse systems of simultaneous linear equations. These include **iterative** methods for real nonsymmetric and symmetric, complex non-Hermitian and Hermitian linear systems. Some further direct methods are currently available in Chapter F01 and Chapter F04, and will be added to this chapter at future marks.

For a wider selection of routines for sparse linear algebra, users are referred to the Harwell Sparse Matrix Library (available from NAG), especially for direct methods for solving linear systems (see Section 2.2).

2 Background to the Problems

This section is only a brief introduction to the solution of sparse linear systems. For a more detailed discussion see for example Duff *et al.* [2] for direct methods, or Barrett *et al.* [1] for iterative methods.

2.1 Sparse Matrices and Their Storage

A matrix A may be described as **sparse** if the number of zero elements is sufficiently large that it is worthwhile using algorithms which avoid computations involving zero elements.

If A is sparse, and the chosen algorithm requires the matrix coefficients to be stored, a significant saving in storage can often be made by storing only the non-zero elements. A number of different formats may be used to represent sparse matrices economically. These differ according to the amount of storage required, the amount of indirect addressing required for fundamental operations such as matrix–vector products, and their suitability for vector and/or parallel architectures. For a survey of some of these storage formats see Barrett *et al.* [1].

Some of the routines in this chapter have been designed to be independent of the matrix storage format. This allows users to choose their own preferred format, or to avoid storing the matrix altogether. Other routines are the so-called **black-boxes**, which are easier to use, but are based on fixed storage formats. Two such formats are currently provided. These are known as coordinate storage (CS) format and symmetric coordinate storage (SCS) format.

2.1.1 Coordinate storage (CS) format

This storage format represents a sparse nonsymmetric matrix A, with NNZ non-zero elements, in terms of three one-dimensional arrays — a real or complex array A and two integer arrays IROW and ICOL. These arrays are all of dimension at least NNZ. A contains the non-zero elements themselves, while IROW and ICOL store the corresponding row and column indices respectively.

For example, the matrix

$$A = \begin{pmatrix} 1 & 2 & -1 & -1 & -3 \\ 0 & -1 & 0 & 0 & -4 \\ 3 & 0 & 0 & 0 & 2 \\ 2 & 0 & 4 & 1 & 1 \\ -2 & 0 & 0 & 0 & 1 \end{pmatrix}$$

might be represented in the arrays A, IROW and ICOL as

A = (1,2,−1,−1,−3,−1,−4,3,2,2,4,1,1,−2,1)

IROW = (1,1,1,1,1,2,2,3,3,4,4,4,4,5,5)

ICOL = (1,2,3,4,5,2,5,1,5,1,3,4,5,1,5).

Notes

(i) The general format specifies no ordering of the array elements, but some routines may impose a specific ordering. For example, the non-zero elements may be required to be ordered by increasing row index and by increasing column index within each row, as in the example above. A utility routine is provided to order the elements appropriately.

(ii) With this storage format it is possible to enter duplicate elements. These may be interpreted in various ways (raising an error, ignoring all but the first entry, all but the last, or summing, for example).

2.1.2 Symmetric coordinate storage (SCS) format

This storage format is suitable for symmetric and Hermitian matrices, and is identical to the CS format described in Section 2.1.1, except that only the lower triangular non-zero elements are stored. Thus, for example, the matrix

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 & -1 & 2 \\ 1 & 5 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & -1 \\ 0 & 2 & 1 & 3 & 1 & 0 \\ -1 & 0 & 0 & 1 & 4 & 0 \\ 2 & 0 & -1 & 0 & 0 & 3 \end{pmatrix}$$

might be represented in the arrays A, IROW and ICOL as

A = (4,1,5,2,2,1,3,−1,1,4,2,−1,3)

IROW = (1,2,2,3,4,4,4,5,5,5,6,6,6)

ICOL = (1,1,2,3,2,3,4,1,4,5,1,3,6).

2.2 Direct Methods

Direct methods for the solution of the linear algebraic system

$$Ax = b \tag{1}$$

aim to determine the solution vector x in a fixed number of arithmetic operations, which is determined a priori by the number of unknowns. For example, an LU factorization of A followed by forward and backward substitution is a direct method for (1).

If the matrix A is sparse it is possible to design direct methods which exploit the sparsity pattern and are therefore much more computationally efficient than the algorithms in Chapter F07, which in general take no account of sparsity. However, if the matrix is very large and sparse, then **iterative** methods (see Section 2.3) are generally more efficient still.

This chapter currently provides direct methods for sparse real nonsymmetric, complex non-Hermitian, real symmetric positive-definite and complex Hermitian positive-definite systems. Further direct methods may be found in Chapter F01, Chapter F04 and Chapter F07, and will be introduced into Chapter F11 at a future mark. For more details see Section 3.4.

2.3 Iterative Methods

In contrast to the direct methods discussed in Section 2.2, **iterative** methods for (1) approach the solution through a sequence of approximations until some user-specified termination criterion is met or until some predefined maximum number of iterations has been reached. The number of iterations required for convergence is not generally known in advance, as it depends on the accuracy required, and on the matrix A — its sparsity pattern, conditioning and eigenvalue spectrum.

Faster convergence can often be achieved using a **preconditioner** (Golub and Van Loan [5], Barrett *et al.* [1]). A preconditioner maps the original system of equations onto a different system

$$\bar{A}\bar{x} = \bar{b}, \tag{2}$$

which hopefully exhibits better convergence characteristics: for example, the condition number of the matrix $\bar{A}$ may be better than that of A, or it may have eigenvalues of greater multiplicity.

An unsuitable preconditioner or no preconditioning at all may result in a very slow rate or lack of convergence. However, preconditioning involves a trade-off between the reduction in the number of iterations required for convergence and the additional computational costs per iteration. Also, setting up a preconditioner may involve non-negligible overheads. The application of preconditioners to real nonsymmetric, complex non-Hermitian, real symmetric and complex Hermitian systems of equations is further considered in Section 2.4 and Section 2.5.

2.4 Iterative Methods for Real Nonsymmetric and Complex Non-Hermitian Linear Systems

Many of the most effective iterative methods for the solution of (1) lie in the class of non-stationary **Krylov subspace methods** [1]. For real nonsymmetric and complex non-Hermitian matrices this class includes:

the restarted generalized minimum residual (RGMRES) method [10];

the conjugate gradient squared (CGS) method [12];

the polynomial stabilized bi-conjugate gradient (Bi-CGSTAB (ℓ)) method [13], [11];

the transpose-free quasi-minimal residual method (TFQMR) [3], [4].

Here we just give a brief overview of these algorithms as implemented in Chapter F11. For full details see the routine documents for F11BDF and F11BRF.

RGMRES is based on the Arnoldi method, which explicitly generates an orthogonal basis for the Krylov subspace span$\{A^k r_0\}$, $k = 0, 1, 2, \ldots$, where r_0 is the initial residual. The solution is then expanded onto the orthogonal basis so as to minimize the residual norm. For real nonsymmetric and complex non-Hermitian matrices the generation of the basis requires a 'long' recurrence relation, resulting in prohibitive computational and storage costs. RGMRES limits these costs by restarting the Arnoldi process from the latest available residual every m iterations. The value of m is chosen in advance and is fixed throughout the computation. Unfortunately, an optimum value of m cannot easily be predicted.

CGS is a development of the bi-conjugate gradient method where the nonsymmetric Lanczos method is applied to reduce the coefficient matrix to tridiagonal form: two bi-orthogonal sequences of vectors are generated starting from the initial residual r_0 and from the *shadow residual* $\hat{r}_0$ corresponding to the arbitrary problem $A^H\hat{x} = \hat{b}$, where $\hat{b}$ is chosen so that $r_0 = \hat{r}_0$. In the course of the iteration, the residual and shadow residual $r_i = P_i(A)r_0$ and $\hat{r}_i = P_i(A^H)\hat{r}_0$ are generated, where P_i is a polynomial of order i, and bi-orthogonality is exploited by computing the vector product $\rho_i = (\hat{r}_i, r_i) = (P_i(A^H)\hat{r}_0, P_i(A)r_0) = (\hat{r}_0, P_i^2(A)r_0)$. Applying the 'contraction' operator $P_i(A)$ twice, the iteration coefficients can still be recovered without advancing the solution of the shadow problem, which is of no interest. The CGS method often provides fast convergence; however, there is no reason why the contraction operator should also reduce the once reduced vector $P_i(A)r_0$: this can lead to a highly irregular convergence.

Bi-CGSTAB (ℓ)) is similar to the CGS method. However, instead of generating the sequence $\{P_i^2(A)r_0\}$, it generates the sequence $\{Q_i(A)P_i(A)r_0\}$ where the $Q_i(A)$ are polynomials chosen to minimize the residual *after* the application of the contraction operator $P_i(A)$. Two main steps can be identified for each iteration: an OR (Orthogonal Residuals) step where a basis of order ℓ is generated by a Bi-CG iteration and an MR (Minimum Residuals) step where the residual is minimized over the basis generated, by a method akin to GMRES. For $\ell = 1$, the method corresponds to the Bi-CGSTAB method of van der Vorst [13]. For $\ell > 1$, more information about complex eigenvalues of the iteration matrix can be taken into account, and this may lead to improved convergence and robustness. However, as ℓ increases, numerical instabilities may arise.

The transpose-free quasi-minimal residual method (TFQMR) (Freund and Nachtigal [3], Freund [4]) is conceptually derived from the CGS method. The residual is minimised over the space of the residual vectors generated by the CGS iterations under the simplifying assumption that residuals are almost orthogonal. In practice, this is not the case but theoretical analysis has proved the validity of the method. This has the effect of remedying the rather irregular convergence behaviour with wild oscillations in the residual norm that can degrade the numerical performance and robustness of the CGS method. In general, the TFQMR method can be expected to converge at least as fast as the CGS method, in terms of number of iterations, although each iteration involves a higher operation count. When the CGS method exhibits irregular convergence, the TFQMR method can produce much smoother, almost monotonic convergence curves. However, the close relationship between the CGS and TFQMR method implies that the *overall* speed of convergence is similar for both methods. In some cases, the TFQMR method may converge faster than the CGS method.

Faster convergence can usually be achieved by using a **preconditioner**. A *left* preconditioner M^{-1} can be used by the RGMRES, CGS and TFQMR methods, such that $\bar{A} = M^{-1}A \sim I_n$ in (2), where I_n is the identity matrix of order n; a *right* preconditioner M^{-1} can be used by the Bi-CGSTAB (ℓ) method, such that $\bar{A} = AM^{-1} \sim I_n$. These are formal definitions, used only in the design of the algorithms; in practice, only the means to compute the matrix–vector products $v = Au$ and $v = A^H u$ (the latter only being required when an estimate of $\|A\|_1$ or $\|A\|_\infty$ is computed internally), and to solve the preconditioning equations $Mv = u$ are required, that is, explicit information about M, or its inverse is not required at any stage.

Preconditioning matrices M are typically based on incomplete factorizations [8], or on the approximate inverses occurring in stationary iterative methods (see Young [14]). A common example is the **incomplete** LU **factorization**

$$M = PLDUQ = A - R$$

where L is lower triangular with unit diagonal elements, D is diagonal, U is upper triangular with unit diagonals, P and Q are permutation matrices, and R is a remainder matrix. A **zero-fill** incomplete LU factorization is one for which the matrix

$$S = P(L + D + U)Q$$

has the same pattern of non-zero entries as A. This is obtained by discarding any **fill** elements (non-zero elements of S arising during the factorization in locations where A has zero elements). Allowing some of these fill elements to be kept rather than discarded generally increases the accuracy of the factorization at the expense of some loss of sparsity. For further details see [1].

2.5 Iterative Methods for Real Symmetric and Complex Hermitian Linear Systems

Two of the best known iterative methods applicable to real symmetric and complex Hermitian linear systems are the conjugate gradient (CG) method [6], [5] and a Lanczos type method based on SYMMLQ [9]. The description of these methods given below is for the real symmetric case. The generalization to complex Hermitian matrices is straightforward.

For the CG method the matrix A should ideally be positive-definite. The application of CG to indefinite matrices may lead to failure, or to lack of convergence. The SYMMLQ method is suitable for both positive-definite and indefinite symmetric matrices. It is more robust than CG, but less efficient when A is positive-definite.

Both methods start from the residual $r_0 = b - Ax_0$, where x_0 is an initial estimate for the solution (often $x_0 = 0$), and generate an orthogonal basis for the Krylov subspace $\text{span}\{A^k r_0\}$, for $k = 0, 1, \ldots$, by means of three-term recurrence relations (Golub and Van Loan [5]). A sequence of symmetric tridiagonal matrices $\{T_k\}$ is also generated. Here and in the following, the index k denotes the iteration count. The resulting symmetric tridiagonal systems of equations are usually more easily solved than the original problem. A sequence of solution iterates $\{x_k\}$ is thus generated such that the sequence of the norms of the residuals $\{\|r_k\|\}$ converges to a required tolerance. Note that, in general, the convergence is not monotonic.

In exact arithmetic, after n iterations, this process is equivalent to an orthogonal reduction of A to symmetric tridiagonal form, $T_n = Q^T AQ$; the solution x_n would thus achieve exact convergence. In finite-precision arithmetic, cancellation and round-off errors accumulate causing loss of orthogonality. These methods must therefore be viewed as genuinely iterative methods, able to converge to a solution **within a prescribed tolerance**.

The orthogonal basis is not formed explicitly in either method. The basic difference between the two methods lies in the method of solution of the resulting symmetric tridiagonal systems of equations: the CG method is equivalent to carrying out an LDL^T (Cholesky) factorization whereas the Lanczos method (SYMMLQ) uses an LQ factorization.

A preconditioner for these methods must be **symmetric and positive-definite**, i.e., representable by $M = EE^T$, where M is non-singular, and such that $\bar{A} = E^{-1}AE^{-T} \sim I_n$ in (2), where I_n is the identity matrix of order n. These are formal definitions, used only in the design of the algorithms; in practice, only the means to compute the matrix-vector products $v = Au$ and to solve the preconditioning equations $Mv = u$ are required.

Preconditioning matrices M are typically based on incomplete factorizations [7], or on the approximate inverses occurring in stationary iterative methods (see Young [14]). A common example is the **incomplete**

Cholesky factorization

$$M = PLDL^T P^T = A - R$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements, D is diagonal and R is a remainder matrix. A **zero-fill** incomplete Cholesky factorization is one for which the matrix

$$S = P(L + D + L^T)P^T$$

has the same pattern of non-zero entries as A. This is obtained by discarding any **fill** elements (non-zero elements of S arising during the factorization in locations where A has zero elements). Allowing some of these fill elements to be kept rather than discarded generally increases the accuracy of the factorization at the expense of some loss of sparsity. For further details see [1].

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 Types of Routine Available

The routines available in this chapter divide essentially into three types: basic routines, utility routines and black-box routines.

Basic routines are grouped in suites of three, and implement the underlying iterative method. Each suite comprises a set-up routine, a solver, and a routine to return additional information. The solver routine is independent of the matrix storage format (indeed the matrix need not be stored at all) and the type of preconditioner. It uses **reverse communication**, i.e., it returns repeatedly to the calling program with the parameter IREVCM set to specified values which require the calling program to carry out a specific task (either to compute a matrix-vector product or to solve the preconditioning equation), to signal the completion of the computation or to allow the calling program to monitor the solution. Reverse communication has the following advantages.

(i) Maximum flexibility in the representation and storage of sparse matrices. All matrix operations are performed outside the solver routine, thereby avoiding the need for a complicated interface with enough flexibility to cope with all types of storage schemes and sparsity patterns. This also applies to preconditioners.

(ii) Enhanced user interaction: the progress of the solution can be closely monitored by the user and tidy or immediate termination can be requested. This is useful, for example, when alternative termination criteria are to be employed or in case of failure of the external routines used to perform matrix operations.

At present there are suites of basic routines for real symmetric and nonsymmetric systems, and for complex non-Hermitian systems.

Utility routines perform such tasks as initializing the preconditioning matrix M, solving linear systems involving M, or computing matrix-vector products, for particular preconditioners and matrix storage formats. Used in combination, basic routines and utility routines therefore provide iterative methods with a considerable degree of flexibility, allowing the user to select from different termination criteria, monitor the approximate solution, and compute various diagnostic parameters. The tasks of computing the matrix-vector products and dealing with the preconditioner are removed from the user, but at the expense of sacrificing some flexibility in the choice of preconditioner and matrix storage format.

Black-box routines call basic and utility routines in order to provide easy-to-use routines for particular preconditioners and sparse matrix storage formats. They are much less flexible than the basic routines, but do not use reverse communication, and may be suitable in many simple cases.

The structure of this chapter has been designed to cater for as many types of application as possible. If a black-box routine exists which is suitable for a given application you are recommended to use it. If you then decide you need some additional flexibility it is easy to achieve this by using basic and utility routines which reproduce the algorithm used in the black-box, but allow more access to algorithmic control parameters and monitoring. If you wish to use a preconditioner or storage format for which no utility routines are provided, you must call basic routines, and provide your own utility routines.

3.2 Iterative Methods for Real Nonsymmetric and Complex Non-Hermitian Linear Systems

The suite of basic routines F11BDF, F11BEF and F11BFF implements either RGMRES, CGS, Bi-CGSTAB(ℓ), or TFQMR, for the iterative solution of the real sparse nonsymmetric linear system $Ax = b$. These routines allow a choice of termination criteria and the norms used in them, allow monitoring of the approximate solution, and can return estimates of the norm of A and the largest singular value of the preconditioned matrix $\bar{A}$.

In general, it is not possible to recommend one of these methods in preference to another. RGMRES is popular, but requires the most storage, and can easily stagnate when the size m of the orthogonal basis is too small, or the preconditioner is not good enough. CGS can be the fastest method, but the computed residuals can exhibit instability which may greatly affect the convergence and quality of the solution. Bi-CGSTAB(ℓ) seems robust and reliable, but it can be slower than the other methods. TFQMR can be viewed as a more robust variant of the CGS method: it shares the CGS method speed but avoids the CGS fluctuations in the residual, which may give, rise to instability. Some further discussion of the relative merits of these methods can be found in [1].

The utility routines provided for real nonsymmetric matrices use the co-ordinate storage (CS) format described in Section 2.1.1. F11DAF computes a preconditioning matrix based on incomplete LU factorization, and F11DBF solves linear systems involving the preconditioner generated by F11DAF. The amount of fill-in occurring in the incomplete factorization can be controlled by specifying either the level of fill, or the drop tolerance. Partial or complete pivoting may optionally be employed, and the factorization can be modified to preserve row-sums.

F11DDF is similar to F11DBF, but solves linear systems involving the preconditioner corresponding to symmetric successive-over-relaxation (SSOR). The value of the relaxation parameter ω must currently be supplied by the user. Automatic procedures for choosing ω will be included in the chapter at a future mark.

F11XAF computes matrix-vector products for real nonsymmetric matrices stored in ordered CS format. An additional utility routine F11ZAF orders the non-zero elements of a real sparse nonsymmetric matrix stored in general CS format.

The black-box routine F11DCF makes calls to F11BDF, F11BEF, F11BFF, F11DBF and F11XAF, to solve a real sparse nonsymmetric linear system, represented in CS format, using RGMRES, CGS, Bi-CGSTAB(ℓ), or TFQMR, with incomplete LU preconditioning. F11DEF is similar, but has options for no preconditioning, Jacobi preconditioning or SSOR preconditioning.

For complex non-Hermitian sparse matrices there is an equivalent suite of routines. F11BRF, F11BSF and F11BTF are the basic routines which implement the same methods used for real nonsymmetric systems, namely RGMRES, CGS, Bi-CGSTAB(ℓ) and TFQMR, for the solution of complex sparse non-Hermitian linear systems. F11DNF and F11DPF are the complex equivalents of F11DAF and F11DBF, respectively, providing facilities for implementing ILU preconditioning. F11DRF implements a complex version of the SSOR preconditioner. Utility routines F11XNF and F11ZNF are provided for computing matrix-vector products and sorting the elements of complex sparse non-Hermitian matrices, respectively. Finally, the black-box routines F11DQF and F11DSF are complex equivalents of F11DCF and F11DEF.

3.3 Iterative Methods for Real Symmetric and Complex Hermitian Linear Systems

The suite of basic routines F11GAF, F11GBF and F11GCF implement either the conjugate gradient (CG) method, or a Lanczos method based on SYMMLQ, for the iterative solution of the real sparse symmetric linear system $Ax = b$. If A is known to be positive-definite the CG method should be chosen; the Lanczos method is more robust but less efficient for positive-definite matrices. These routines allow a choice of termination criteria and the norms used in them, allow monitoring of the approximate solution, and can return estimates of the norm of A and the largest singular value of the preconditioned matrix $\bar{A}$.

The utility routines provided for real symmetric matrices use the symmetric co-ordinate storage (SCS) format described in Section 2.1.2. F11JAF computes a preconditioning matrix based on incomplete Cholesky factorization, and F11JBF solves linear systems involving the preconditioner generated by F11JAF. The amount of fill-in occurring in the incomplete factorization can be controlled by specifying either the level of fill, or the drop tolerance. Diagonal Markowitz pivoting may optionally be employed, and the factorization can be modified to preserve row-sums.

F11JDF is similar to F11JBF, but solves linear systems involving the preconditioner corresponding to symmetric successive-over-relaxation (SSOR). The value of the relaxation parameter ω must currently be supplied by the user. Automatic procedures for choosing ω will be included in the chapter at a future mark.

F11XEF computes matrix-vector products for real symmetric matrices stored in ordered SCS format. An additional utility routine F11ZBF orders the non-zero elements of a real sparse symmetric matrix stored in general SCS format.

The black-box routine F11JCF makes calls to F11GAF, F11GBF, F11GCF, F11JBF and F11XEF, to solve a real sparse symmetric linear system, represented in SCS format, using a conjugate gradient or Lanczos method, with incomplete Cholesky preconditioning. F11JEF is similar, but has options for no preconditioning, Jacobi preconditioning or SSOR preconditioning.

For complex Hermitian sparse matrices there is an equivalent suite of utility and black-box routines, although the basic routines are not available at this Mark. F11JNF and F11JPF are the complex equivalents of F11JAF and F11JBF, respectively, providing facilities for implementing incomplete Cholesky preconditioning. F11JRF implements a complex version of the SSOR preconditioner. Utility routines F11XSF and F11ZPF are provided for computing matrix-vector products and sorting the elements of complex sparse Hermitian matrices, respectively. Finally, the black-box routines F11JQF and F11JSF provide easy-to-use implementations of the CG and SYMMLQ methods for complex Hermitian linear systems.

3.4 Direct Methods

Chapter F11 does not currently provide any routines **specifically** designed for direct solution of sparse linear systems. However, the arguments of the various preconditioner generation routines can be chosen in such a way as to produce a direct solution.

For example, routine F11DBF solves a linear system involving the incomplete LU preconditioning matrix

$$M = PLDUQ = A - R$$

generated by F11DAF, where P and Q are permutation matrices, L is lower triangular with unit diagonal elements, U is upper triangular with unit diagonal elements, D is diagonal and R is a remainder matrix.

If A is non-singular, a call to F11DAF with LFILL < 0 and DTOL $= 0.0$ results in a zero remainder matrix R and a **complete** factorization. A subsequent call to F11DBF will therefore result in a direct method for real sparse nonsymmetric systems.

If A is known to be symmetric positive-definite, F11JAF and F11JBF may similarly be used to give a direct solution. For further details see Section 8.4 of the document for F11JAF.

Complex non-Hermitian systems can be solved directly in the same way using F11DNF and F11DPF, while for complex Hermitian systems F11JNF and F11JPF may be used.

Some routines specifically designed for direct solution of sparse linear systems can currently be found in Chapter F01, Chapter F04 and Chapter F07. In particular, the following routines allow the direct solution of nonsymmetric systems:

Band	F07BDF and F07BEF
Almost block-diagonal	F01LHF and F04LHF
Tridiagonal	F01LEF and F04LEF or F04EAF
Sparse	F01BRF (or F01BSF) and F04AXF

and the following routines allow the direct solution of symmetric positive-definite systems:

Band	F07HDF and F07HEF
Variable band (skyline)	F01MCF and F04MCF
Tridiagonal	F04FAF

4 Index

5 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Those routines indicated by a dagger are still present at Mark 19, but will be omitted at a future date. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

F11BAF† F11BBF† F11BCF†

6 References

[1] Barrett R, Berry M, Chan T F, Demmel J, Donato J, Dongarra J, Eijkhout V, Pozo R, Romine C and van der Vorst H (1994) *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* SIAM, Philadelphia

[2] Duff I S, Erisman A M, Reid J K (1986) *Direct Methods for Sparse Matrices* Oxford University Press, London

[3] Freund R W and Nachtigal N (1991) QMR: a Quasi-Minimal Residual Method for Non-Hermitian Linear Systems *Numer. Math.* **60** 315–339

[4] Freund R W (1993) A Transpose-Free Quasi-Mimimal Residual Algorithm for Non-Hermitian Linear Sytems *SIAM J. Sci. Comput.* **14** 470–482

[5] Golub G H and Van Loan C F (1996) *Matrix Computations* Johns Hopkins University Press (3rd Edition), Baltimore

[6] Hestenes M and Stiefel E (1952) Methods of conjugate gradients for solving linear systems *J. Res. Nat. Bur. Stand.* **49** 409–436

[7] Meijerink J and van der Vorst H (1977) An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix *Math. Comput.* **31** 148–162

[8] Meijerink J and van der Vorst H (1981) Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems *J. Comput. Phys.* **44** 134–155

[9] Paige C C and Saunders M A (1975) Solution of sparse indefinite systems of linear equations *SIAM J. Numer. Anal.* **12** 617–629

[10] Saad Y and Schultz M (1986) GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **7** 856–869

[11] Sleijpen G L G and Fokkema D R (1993) BiCGSTAB(ℓ) for linear equations involving matrices with complex spectrum *ETNA* **1** 11–32

[12] Sonneveld P (1989) CGS, a fast Lanczos-type solver for nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **10** 36–52

[13] van der Vorst H (1989) Bi-CGSTAB, A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **13** 631–644

[14] Young D (1971) *Iterative Solution of Large Linear Systems* Academic Press, New York

Chapter G01 – Simple Calculations and Statistical Data

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G01AAF	4	Mean, variance, skewness, kurtosis, etc, one variable, from raw data
G01ABF	4	Mean, variance, skewness, kurtosis, etc, two variables, from raw data
G01ADF	4	Mean, variance, skewness, kurtosis, etc, one variable, from frequency table
G01AEF	4	Frequency table from raw data
G01AFF	4	Two-way contingency table analysis, with χ^2/Fisher's exact test
G01AGF	8	Lineprinter scatterplot of two variables
G01AHF	8	Lineprinter scatterplot of one variable against Normal scores
G01AJF	10	Lineprinter histogram of one variable
G01ALF	14	Computes a five-point summary (median, hinges and extremes)
G01ARF	14	Constructs a stem and leaf plot
G01ASF	14	Constructs a box and whisker plot
G01BJF	13	Binomial distribution function
G01BKF	13	Poisson distribution function
G01BLF	13	Hypergeometric distribution function
G01DAF	8	Normal scores, accurate values
G01DBF	12	Normal scores, approximate values
G01DCF	12	Normal scores, approximate variance-covariance matrix
G01DDF	12	Shapiro and Wilk's W test for Normality
G01DHF	15	Ranks, Normal scores, approximate Normal scores or exponential (Savage) scores
G01EAF	15	Computes probabilities for the standard Normal distribution
G01EBF	14	Computes probabilities for Student's t-distribution
G01ECF	14	Computes probabilities for χ^2 distribution
G01EDF	14	Computes probabilities for F-distribution
G01EEF	14	Computes upper and lower tail probabilities and probability density function for the beta distribution
G01EFF	14	Computes probabilities for the gamma distribution
G01EMF	15	Computes probability for the Studentized range statistic
G01EPF	15	Computes bounds for the significance of a Durbin–Watson statistic
G01ERF	16	Computes probability for von Mises distribution
G01EYF	14	Computes probabilities for the one-sample Kolmogorov–Smirnov distribution
G01EZF	14	Computes probabilities for the two-sample Kolmogorov–Smirnov distribution
G01FAF	15	Computes deviates for the standard Normal distribution
G01FBF	14	Computes deviates for Student's t-distribution
G01FCF	14	Computes deviates for the χ^2 distribution
G01FDF	14	Computes deviates for the F-distribution
G01FEF	14	Computes deviates for the beta distribution
G01FFF	14	Computes deviates for the gamma distribution
G01FMF	15	Computes deviates for the Studentized range statistic
G01GBF	14	Computes probabilities for the non-central Student's t-distribution
G01GCF	14	Computes probabilities for the non-central χ^2 distribution
G01GDF	14	Computes probabilities for the non-central F-distribution
G01GEF	14	Computes probabilities for the non-central beta distribution
G01HAF	14	Computes probability for the bivariate Normal distribution
G01HBF	15	Computes probabilities for the multivariate Normal distribution
G01JCF	14	Computes probability for a positive linear combination of χ^2 variables

G01JDF	15	Computes lower tail probability for a linear combination of (central) χ^2 variables
G01MBF	15	Computes reciprocal of Mills' Ratio
G01NAF	16	Cumulants and moments of quadratic forms in Normal variables
G01NBF	16	Moments of ratios of quadratic forms in Normal variables, and related statistics

Chapter G01

Simple Calculations and Statistical Data

Contents

1 Scope of the Chapter

This chapter covers three topics:

plots, descriptive statistics, and exploratory data analysis;

statistical distribution functions and their inverses;

testing for Normality and other distributions.

2 Background to the Problems

2.1 Plots, Descriptive Statistics and Exploratory Data Analysis

Plots and simple descriptive statistics are generally used for one of two purposes:

the presentation of data;

exploratory data analysis.

Exploratory data analysis (EDA) is used to pick out the important features of the data in order to guide the choice of appropriate models. EDA makes use of simple displays and summary statistics. These may suggest models or transformations of the data which can then be confirmed by further plots. The process is interactive between the user, the data, and the program producing the EDA displays. In a formal presentation of data, selected features of the data are displayed for others to examine. In this situation high quality graphics are often needed (for example, the NAG Graphics Library) but the character plots produced by routines in this chapter are usually adequate for EDA work.

The summary statistics consist of two groups. The first group are those based on moments; for example: mean, standard deviation, coefficient of skewness, and coefficient of kurtosis (sometimes called the 'excess of kurtosis', which has the value 0 for the Normal distribution). These statistics may be sensitive to extreme observations and some robust versions are available in Chapter G07. The second group of summary statistics are based on the order statistics, where the ith order statistic in a sample is the ith smallest observation in that sample. Examples of such statistics are: minimum, maximum, median and hinges.

In addition to summarizing the data by using suitable statistics the data can be displayed using tables and diagrams. Such data displays include: frequency tables, stem and leaf displays, box and whisker plots, histograms and scatter plots.

2.2 Statistical Distribution Functions and Their Inverses

Statistical distributions are commonly used in three problems:

evaluation of probabilities and expected frequencies for a distribution model;

testing of hypotheses about the variables being observed;

evaluation of confidence limits for parameters of fitted model, for example, the mean of a Normal distribution.

Random variables can be either discrete (i.e., they can take only a limited number of values) or continuous (i.e., can take any value in a given range). However, for a large sample from a discrete distribution an approximation by a continuous distribution, usually the Normal distribution, can be used. Distributions commonly used as a model for discrete random variables are the binomial, hypergeometric, and Poisson distributions. The binomial distribution arises when there is a fixed probability of a selected outcome as in sampling with replacement, the hypergeometric distribution is used in sampling from a finite population without replacement, and the Poisson distribution is often used to model counts.

Distributions commonly used as a model for continuous random variables are the Normal, gamma, and beta distributions. The Normal is a symmetric distribution whereas the gamma is skewed and only appropriate for non-negative values. The beta is for variables in the range [0,1] and may take many different shapes. For circular data, the 'equivalent' to the Normal distribution is the von Mises distribution. The assumption of the Normal distribution leads to procedures for testing and interval estimation based on the χ^2, F (variance ratio), and Student's t-distributions.

In the hypothesis testing situation, a statistic X with known distribution under the null hypothesis is evaluated, and the probability α of observing such a value or one more 'extreme' value is found. This probability (the significance) is usually then compared with a preassigned value (the significance level of the test), to decide whether the null hypothesis can be rejected in favour of an alternate hypothesis on the basis of the sample values. Many tests make use of those distributions derived from the Normal distribution as listed above, but for some tests specific distributions such as the Studentized range distribution and the distribution of the Durbin–Watson test have been derived. Non-parametric tests as given in Chapter G08, such as the Kolmogorov–Smirnov test, often use statistics with distributions specific to the test. The probability that the null hypothesis will be rejected when the simple alternate hypothesis is true (the power of the test) can be found from the non-central distribution.

The confidence interval problem requires the inverse calculation. In other words, given a probability α, the value x is to be found, such that the probability that a value not exceeding x is observed is equal to α. A confidence interval of size $1 - 2\alpha$, for the quantity of interest, can then be computed as a function of x and the sample values.

The required statistics for either testing hypotheses or constructing confidence intervals can be computed with the aid of routines in this chapter, and Chapter G02 (Regression), Chapter G04 (Analysis of Designed Experiments), Chapter G13 (Time Series), and Chapter E04 (Non-linear Least-squares Problems).

Pseudo-random numbers from many statistical distributions can be generated by routines in Chapter G05.

2.3 Testing for Normality and Other Distributions

Methods of checking that observations (or residuals from a model) come from a specified distribution, for example, the Normal distribution, are often based on order statistics. Graphical methods include the use of **probability plots**. These can be either *P-P* plots (probability–probability plots), in which the empirical probabilities are plotted against the theoretical probabilities for the distribution, or *Q-Q* plots (quantile–quantile plots), in which the sample points are plotted against the theoretical quantiles. *Q-Q* plots are more common, partly because they are invariant to differences in scale and location. In either case if the observations come from the specified distribution then the plotted points should roughly lie on a straight line.

If y_i is the ith smallest observation from a sample of size n (i.e., the ith order statistic) then in a *Q-Q* plot for a distribution with cumulative distribution function F, the value y_i is plotted against x_i, where $F(x_i) = (i - \alpha)/(n - 2\alpha + 1)$; a common value of α being $\frac{1}{2}$. For the Normal distribution, the *Q-Q* plot is known as a Normal probability plot.

The values x_i used in *Q-Q* plots can be regarded as approximations to the expected values of the order statistics. For a sample from a Normal distribution the expected values of the order statistics are known as **Normal scores** and for an exponential distribution they are known as **Savage scores**.

An alternative approach to probability plots are the more formal tests. A test for Normality is the Shapiro and Wilks W Test, which uses Normal scores. Other tests are the χ^2 goodness of fit test and the Kolmogorov–Smirnov test; both can be found in Chapter G08.

2.4 Distribution of Quadratic Forms

Many test statistics for Normally distributed data lead to quadratic forms in Normal variables. If X is a n-dimensional Normal variable with mean μ and variance-covariance matrix Σ then for an n by n matrix A the quadratic form is:

$$Q = X^T AX.$$

The distribution of Q depends on the relationship between A and Σ: if $A\Sigma$ is idempotent then the distribution of Q will be central or non-central χ^2 depending on whether μ is zero.

The distribution of other statistics may be derived as the distribution of linear combinations of quadratic forms, for example the Durbin–Watson test statistic, or as ratios of quadratic forms. In some cases rather than the distribution of these functions of quadratic forms the values of the moments may be all that is required.

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 following routines are recommended for the tasks described, for a full list of routines see the Chapter Contents.

3.1 Plots, Descriptive Statistics and Exploratory Data Analysis

Descriptive statistics:	
mean, standard deviation etc. from raw data	G01AAF
mean, standard deviation etc. from a frequency table	G01ADF
five-point summary	G01ALF
Data displays:	
frequency table produced from a set of data	G01AEF
histogram of one variable	G01AJF
box-whisker plot	G01ASF
stem and leaf	G01ARF
scatter plot of two variables	G01AGF

3.2 Distribution Functions and Their Inverses

Discrete distributions:	
binomial	G01BJF
Poisson	G01BKF
hypergeometric	G01BLF
Continuous distributions:	
Normal distribution	G01EAF
Student's t-distribution	G01EBF
χ^2 distribution	G01ECF
F (variance-ratio) distribution	G01EDF
beta distribution	G01EEF
gamma distribution	G01EFF
von Mises distribution	G01ERF
one sample Kolmogorov–Smirnov distribution	G01EYF
two sample Kolmogorov–Smirnov distribution	G01EZF
distribution of the Studentized range statistic	G01EMF
bounds for the significance of the Durbin–Watson statistic	G01EPF
Inverses of distribution functions:	
Normal distribution	G01FAF
Student's t	G01FBF
χ^2	G01FCF
F (variance-ratio)	G01FDF
beta	G01FEF
gamma	G01FFF
distribution of the Studentized range statistic	G01FMF

Note. The Student's t, χ^2, and F routines do not aim to achieve a high degree of accuracy, only about 4 or 5 significant figures, but this should be quite sufficient for hypothesis-testing. However, both the Student's t and the F distributions can be transformed to a beta distribution and the χ^2 distribution can be transformed to a gamma distribution, so a higher accuracy can be obtained by calls to the gamma or beta routines.

Non-central distributions:	
non-central Student's t-distribution	G01GBF
non-central χ^2 distribution	G01GCF
non-central F (variance-ratio) distribution	G01GDF
non-central beta distribution	G01GEF
Multivariate distributions:	
bivariate Normal distribution	G01HAF
multivariate Normal distribution	G01HBF
Distribution of functions of quadratic forms of Normal variables:	
non-central χ^2 distribution	G01GCF
positive linear combination of (non-central) χ^2 variables	G01JCF
linear combination of (central) χ^2 variables	G01JDF
moments of quadratic forms	G01NAF
moments of ratios of quadratic forms	G01NBF
Other related functions:	
reciprocal of Mills' ratio	G01MBF

3.3 Testing for Normality and Other Distributions

Calculation of ranks and scores	G01DHF
Normal probability plot	G01AHF
Calculation of the W Test for Normality	G01DDF
Calculation of Normal Scores, the expected value of the order statistics from a standard Normal sample	G01DAF
Calculation of the variance-covariance matrix of the order statistics from a standard Normal sample	G01DCF

Note. G01DHF computes either ranks, approximations to the Normal scores, Normal, or Savage scores for a given sample. G01DHF also gives the user control over how it handles tied observations. G01DAF computes the Normal scores for a given sample size to a requested accuracy; the scores are returned in ascending order. G01DAF can be used if either high accuracy is required or if Normal scores are required for many samples of the same size, in which case the user will have to sort the data or scores.

4 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'.

G01BAF	G01BBF	G01BCF	G01BDF	G01CAF	G01CBF
G01CCF	G01CDF	G01CEF			

5 References

[1] Hastings N A J and Peacock J B (1975) *Statistical Distributions* Butterworths

[2] Kendall M G and Stuart A (1969) *The Advanced Theory of Statistics (Volume 1)* Griffin (3rd Edition)

[3] Tukey J W (1977) *Exploratory Data Analysis* Addison-Wesley

Chapter G02 – Correlation and Regression Analysis

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G02BAF	4	Pearson product-moment correlation coefficients, all variables, no missing values
G02BBF	4	Pearson product-moment correlation coefficients, all variables, casewise treatment of missing values
G02BCF	4	Pearson product-moment correlation coefficients, all variables, pairwise treatment of missing values
G02BDF	4	Correlation-like coefficients (about zero), all variables, no missing values
G02BEF	4	Correlation-like coefficients (about zero), all variables, casewise treatment of missing values
G02BFF	4	Correlation-like coefficients (about zero), all variables, pairwise treatment of missing values
G02BGF	4	Pearson product-moment correlation coefficients, subset of variables, no missing values
G02BHF	4	Pearson product-moment correlation coefficients, subset of variables, casewise treatment of missing values
G02BJF	4	Pearson product-moment correlation coefficients, subset of variables, pairwise treatment of missing values
G02BKF	4	Correlation-like coefficients (about zero), subset of variables, no missing values
G02BLF	4	Correlation-like coefficients (about zero), subset of variables, casewise treatment of missing values
G02BMF	4	Correlation-like coefficients (about zero), subset of variables, pairwise treatment of missing values
G02BNF	4	Kendall/Spearman non-parametric rank correlation coefficients, no missing values, overwriting input data
G02BPF	4	Kendall/Spearman non-parametric rank correlation coefficients, casewise treatment of missing values, overwriting input data
G02BQF	4	Kendall/Spearman non-parametric rank correlation coefficients, no missing values, preserving input data
G02BRF	4	Kendall/Spearman non-parametric rank correlation coefficients, casewise treatment of missing values, preserving input data
G02BSF	4	Kendall/Spearman non-parametric rank correlation coefficients, pairwise treatment of missing values
G02BTF	14	Update a weighted sum of squares matrix with a new observation
G02BUF	14	Computes a weighted sum of squares matrix
G02BWF	14	Computes a correlation matrix from a sum of squares matrix
G02BXF	14	Computes (optionally weighted) correlation and covariance matrices
G02BYF	17	Computes partial correlation/variance-covariance matrix from correlation/variance-covariance matrix computed by G02BXF
G02CAF	4	Simple linear regression with constant term, no missing values
G02CBF	4	Simple linear regression without constant term, no missing values
G02CCF	4	Simple linear regression with constant term, missing values
G02CDF	4	Simple linear regression without constant term, missing values
G02CEF	4	Service routines for multiple linear regression, select elements from vectors and matrices
G02CFF	4	Service routines for multiple linear regression, re-order elements of vectors and matrices
G02CGF	4	Multiple linear regression, from correlation coefficients, with constant term

G02CHF	4	Multiple linear regression, from correlation-like coefficients, without constant term
G02DAF	14	Fits a general (multiple) linear regression model
G02DCF	14	Add/delete an observation to/from a general linear regression model
G02DDF	14	Estimates of linear parameters and general linear regression model from updated model
G02DEF	14	Add a new variable to a general linear regression model
G02DFF	14	Delete a variable from a general linear regression model
G02DGF	14	Fits a general linear regression model for new dependent variable
G02DKF	14	Estimates and standard errors of parameters of a general linear regression model for given constraints
G02DNF	14	Computes estimable function of a general linear regression model and its standard error
G02EAF	14	Computes residual sums of squares for all possible linear regressions for a set of independent variables
G02ECF	14	Calculates R^2 and C_P values from residual sums of squares
G02EEF	14	Fits a linear regression model by forward selection
G02FAF	14	Calculates standardized residuals and influence statistics
G02FCF	15	Computes Durbin–Watson test statistic
G02GAF	14	Fits a generalized linear model with Normal errors
G02GBF	14	Fits a generalized linear model with binomial errors
G02GCF	14	Fits a generalized linear model with Poisson errors
G02GDF	14	Fits a generalized linear model with gamma errors
G02GKF	14	Estimates and standard errors of parameters of a general linear model for given constraints
G02GNF	14	Computes estimable function of a generalized linear model and its standard error
G02HAF	13	Robust regression, standard M-estimates
G02HBF	13	Robust regression, compute weights for use with G02HDF
G02HDF	13	Robust regression, compute regression with user-supplied functions and weights
G02HFF	13	Robust regression, variance-covariance matrix following G02HDF
G02HKF	14	Calculates a robust estimation of a correlation matrix, Huber's weight function
G02HLF	14	Calculates a robust estimation of a correlation matrix, user-supplied weight function plus derivatives
G02HMF	14	Calculates a robust estimation of a correlation matrix, user-supplied weight function

Chapter G02

Correlation and Regression Analysis

Contents

1 Scope of the Chapter

This chapter is concerned with two techniques – correlation analysis and regression modelling – both of which are concerned with determining the inter-relationships among two or more variables.

Other chapters of the NAG Fortran Library which cover similar problems are E02 and E04. E02 routines may be used to fit linear models by criteria other than least-squares, and also for polynomial regression; E04 routines may be used to fit nonlinear models and linearly constrained linear models.

2 Background to the Problems

2.1 Correlation

2.1.1 Aims of correlation analysis

Correlation analysis provides a single summary statistic – the correlation coefficient – describing the strength of the **association** between two variables. The most common types of association which are investigated by correlation analysis are linear relationships, and there are a number of forms of linear correlation coefficients for use with different types of data.

2.1.2 Correlation coefficients

The (Pearson) product-moment correlation coefficients measure a linear relationship, while Kendall's tau and Spearman's rank order correlation coefficients measure monotonicity only. All three coefficients range from -1.0 to $+1.0$. A coefficient of zero always indicates that no **linear** relationship exists; a $+1.0$ coefficient implies a 'perfect' positive relationship (i.e., an increase in one variable is always associated with a corresponding increase in the other variable); and a coefficient of -1.0 indicates a 'perfect' negative relationship (i.e., an increase in one variable is always associated with a corresponding decrease in the other variable).

Consider the bivariate scattergrams in Figure 1: (a) and (b) show strictly linear functions for which the values of the product-moment correlation coefficient, and (since a linear function is also monotonic) both Kendall's tau and Spearman's rank order coefficients, would be $+1.0$ and -1.0 respectively. However, though the relationships in figures (c) and (d) are respectively monotonically increasing and monotonically decreasing, for which both Kendall's and Spearman's non-parametric coefficients would be $+1.0$ (in (c)) and -1.0 (in (d)), the functions are nonlinear so that the product-moment coefficients would not take such 'perfect' extreme values. There is no obvious relationship between the variables in figure (e), so all three coefficients would assume values close to zero, while in figure (f) though there is an obvious parabolic relationship between the two variables, it would not be detected by any of the correlation coefficients which would again take values near to zero; it is important therefore to examine scattergrams as well as the correlation coefficients.

In order to decide which type of correlation is the most appropriate, it is necessary to appreciate the different groups into which variables may be classified. Variables are generally divided into four types of scales: the nominal scale, the ordinal scale, the interval scale, and the ratio scale. The nominal scale is used only to categorise data; for each category a name, perhaps numeric, is assigned so that two different categories will be identified by distinct names. The ordinal scale, as well as categorising the observations, orders the categories. Each category is assigned a distinct identifying symbol, in such a way that the order of the symbols corresponds to the order of the categories. (The most common system for ordinal variables is to assign numerical identifiers to the categories, though if they have previously been assigned alphabetic characters, these may be transformed to a numerical system by any convenient method which preserves the ordering of the categories.) The interval scale not only categorises and orders the observations, but also quantifies the comparison between categories; this necessitates a common unit of measurement and an arbitrary zero-point. Finally, the ratio scale is similar to the interval scale, except that it has an **absolute** (as opposed to **arbitrary**) zero-point.

For a more complete discussion of these four types of scales, and some examples, the user is referred to Churchman and Ratoosh [2] and Hayes [7].

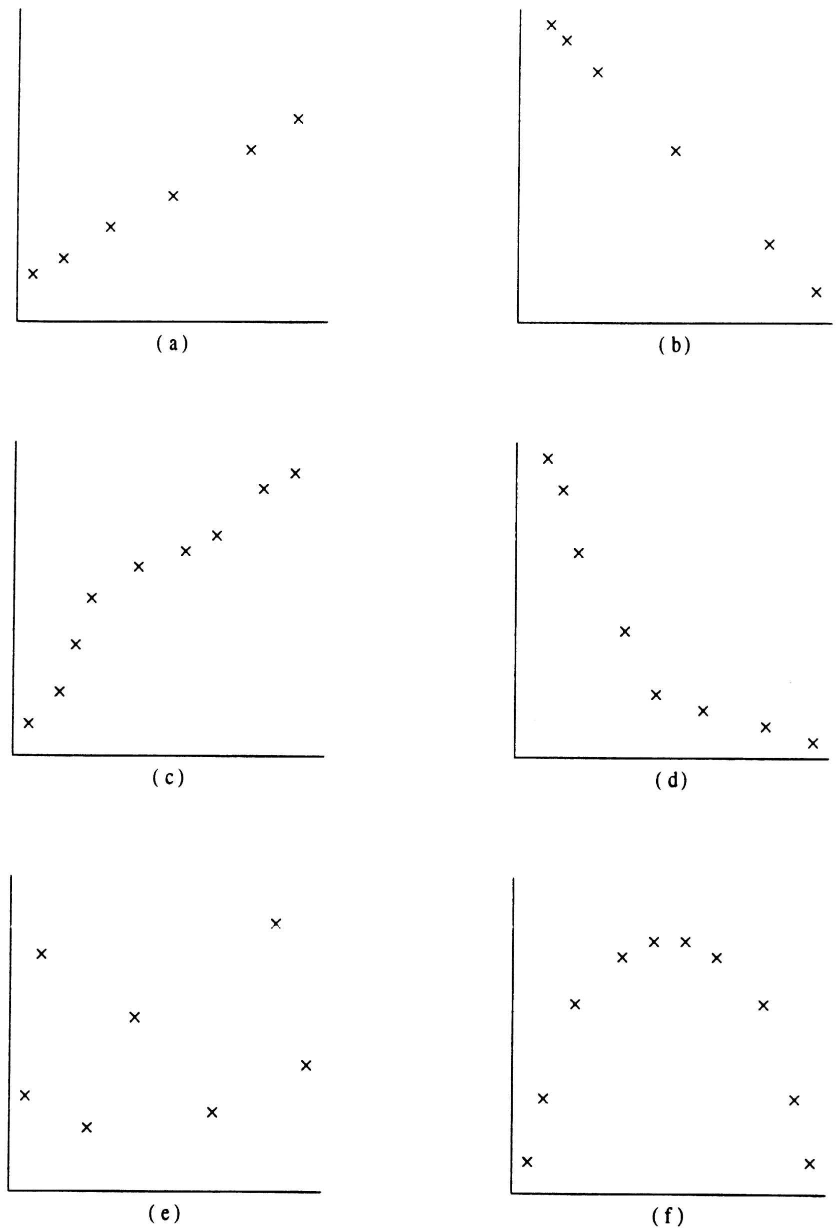


Figure 1

Product-moment correlation coefficients are used with variables which are interval (or ratio) scales; these coefficients measure the amount of spread about the linear least-squares equation. For a product-moment correlation coefficient, r, based on n pairs of observations, testing against the null hypothesis that there is no correlation between the two variables, the statistic

$$r\sqrt{\frac{n-2}{1-r^2}}$$

has a Student's t-distribution with $n-2$ degrees of freedom; its significance can be tested accordingly.

Ranked and ordinal scale data are generally analysed by non-parametric methods – usually either Spearman's or Kendall's tau rank-order correlation coefficients, which, as their names suggest, operate solely on the ranks, or relative orders, of the data values. Interval or ratio scale variables may also be validly analysed by non-parametric methods, but such techniques are statistically less powerful than a product-moment method. For a Spearman rank-order correlation coefficient, R, based on n pairs of observations, testing against the null hypothesis that there is no correlation between the two variables, for large samples the statistic

$$R\sqrt{\frac{n-2}{1-R^2}}$$

has approximately a Student's t-distribution with $n-2$ degrees of freedom, and may be treated accordingly. (This is similar to the product-moment correlation coefficient, r, see above.) Kendall's tau coefficient, based on n pairs of observations, has, for large samples, an approximately Normal distribution with mean zero and standard deviation

$$\sqrt{\frac{4n+10}{9n(n-1)}}$$

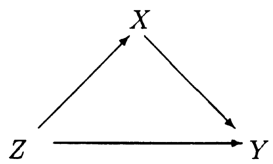
when tested against the null hypothesis that there is no correlation between the two variables; the coefficient should therefore be divided by this standard deviation and tested against the standard Normal distribution, N(0,1).

When the number of ordinal categories a variable takes is large, and the number of ties is relatively small, Spearman's rank-order correlation coefficients have advantages over Kendall's tau; conversely, when the number of categories is small, or there are a large number of ties, Kendall's tau is usually preferred. Thus when the ordinal scale is more or less continuous, Spearman's rank-order coefficients are preferred, whereas Kendall's tau is used when the data is grouped into a smaller number of categories; both measures do however include corrections for the occurrence of ties, and the basic concepts underlying the two coefficients are quite similar. The absolute value of Kendall's tau coefficient tends to be slightly smaller than Spearman's coefficient for the same set of data.

There is no authoritative dictum on the selection of correlation coefficients – particularly on the advisability of using correlations with ordinal data. This is a matter of discretion for the user.

2.1.3 Partial Correlation

The correlation coefficients described above measure the association between two variables ignoring any other variables in the system. Suppose there are three variables X, Y and Z as shown in the path diagram below.



The association between Y and Z is made up of the direct association between Y and Z and the association caused by the path through X, that is the association of both Y and Z with the third variable X. For example if Z and Y were cholesterol level and blood pressure and X were age since both blood pressure and cholesterol level may increase with age the correlation between blood pressure and cholesterol level eliminating the effect of age is required.

The correlation between two variables eliminating the effect of a third variable is know as the partial correlation. If ρ_{zy}, ρ_{zx} and ρ_{xy} represent the correlations between x, y and z then the partial correlation between Z and Y given X is:

$$\frac{\rho_{zy}-\rho_{zx}\rho_{xy}}{\sqrt{(1-\rho_{zx}^2)(1-\rho_{xy}^2)}},$$

The partial correlation is then estimated by using product-moment correlation coefficients.

In general, let a set of variables be partitioned into two groups Y and X with n_y variables in Y and n_x variables in X and let the variance-covariance matrix of all n_y+n_x variables be partitioned into

$$\left[\begin{array}{cc}\Sigma_{xx} & \Sigma_{yx}\\ \Sigma_{xy} & \Sigma_{yy}\end{array}\right].$$

Then the variance-covariance of Y conditional on fixed values of the X variables is given by:

$$\Sigma_{y|x} = \Sigma_{yy} - \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy}.$$

The partial correlation matrix is then computed by standardising $\Sigma_{y|x}$.

2.1.4 Robust estimation of correlation coefficients

The product-moment correlation coefficient can be greatly affected by the presence of a few extreme observations or outliers. There are robust estimation procedures which aim to decrease the effect of extreme values.

Mathematically these methods can be described as follows. A robust estimate of the variance-covariance matrix, C, can be written as:

$$C = \tau^2(A^TA)^{-1}$$

where τ^2 is a correction factor to give an unbiased estimator if the data is Normal and A is a lower triangular matrix. Let x_i be the vector of values for the ith observation and let $z_i = A(x_i - \theta)$, θ is a robust estimate of location, then θ and A are found as solutions to:

$$\frac{1}{n}\sum_{i=1}^{n} w(\|z_i\|_2)z_i = 0$$

and

$$\frac{1}{n}\sum_{i=1}^{n} w(\|z_i\|_2)z_iz_i^T - v(\|z_i\|_2)I = 0,$$

where $w(t)$, $u(t)$ and $v(t)$ are functions such that they return value 1 for reasonable values of t and decreasing values for large t. The correlation matrix can then be calculated from the variance-covariance matrix. If w, u, and v returned 1 for all values then the product-moment correlation coefficient would be calculated.

2.1.5 Missing values

When there are missing values in the data these may be handled in one of two ways. Firstly, if a case contains a missing observation for any variable, then that case is omitted in its entirety from all calculations; this may be termed **casewise** treatment of missing data. Secondly, if a case contains a missing observation for any variable, then the case is omitted from only those calculations involving the variable for which the value is missing; this may be called **pairwise** treatment of missing data. Pairwise deletion of missing data has the advantage of using as much of the data as possible in the computation of each coefficient. In extreme circumstances, however, it can have the disadvantage of producing coefficients which are based on a different number of cases, and even on different selections of cases or samples; furthermore, the correlation matrices formed in this way need not necessarily be positive-definite, a requirement for a correlation matrix. Casewise deletion of missing data generally causes fewer cases to be used in the calculation of the coefficients than does pairwise deletion. How great this difference is will obviously depend on the distribution of the missing data, both among cases and among variables.

Pairwise treatment does therefore use more information from the sample, but should not be used without careful consideration of the location of the missing observations in the data matrix, and the consequent effect of processing the missing data in that fashion.

2.2 Regression

2.2.1 Aims of regression modelling

In regression analysis the relationship between one specific random variable, the **dependent** or **response variable**, and one or more known variables, called the **independent variables** or **covariates**, is studied. This relationship is represented by a mathematical model, or an equation, which associates the dependent variable with the independent variables, together with a set of relevant assumptions. The independent variables are related to the dependent variable by a function, called the **regression function**, which involves a set of unknown **parameters**. Values of the parameters which give the best fit for a given set of data are obtained, these values are known as the **estimates** of the parameters.

The reasons for using a regression model are twofold. The first is to obtain a **description** of the relationship between the variables as an indicator of possible causality. The second reason is to **predict** the value of the dependent variable from a set of values of the independent variables. Accordingly, the most usual statistical problems involved in regression analysis are:

(i) to obtain best estimates of the unknown regression parameters;
(ii) to test hypotheses about these parameters;
(iii) to determine the adequacy of the assumed model; and
(iv) to verify the set of relevant assumptions.

2.2.2 Linear regression models

When the regression model is linear in the parameters (but not necessarily in the independent variables), then the regression model is said to be linear; otherwise the model is classified as nonlinear.

The most elementary form of regression model is the **simple linear regression** of the dependent variable, Y, on a single independent variable, x, which takes the form

$$E(Y) = \beta_0 + \beta_1 x \tag{1}$$

where $E(Y)$ is the expected or average value of Y and β_0 and β_1 are the parameters whose values are to be estimated, or, if the regression is required to pass through the origin (i.e., no constant term),

$$E(Y) = \beta_1 x \tag{2}$$

where β_1 is the only unknown parameter.

An extension of this is **multiple linear regression** in which the dependent variable, Y, is regressed on the p ($p > 1$) independent variables, $x_1, x_2, \ldots, x_p$, which takes the form

$$E(Y) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p \tag{3}$$

where $\beta_1, \beta_2, \ldots, \beta_p$ and β_0 are the unknown parameters.

A special case of multiple linear regression is **polynomial linear regression**, in which the p independent variables are in fact powers of the same single variable x (i.e., $x_j = x^j$, for $j = 1, 2, \ldots, p$).

In this case, the model defined by (3) becomes

$$E(Y) = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_p x^p. \tag{4}$$

There are a great variety of **nonlinear regression models** one of the most common is **exponential regression**, in which the equation may take the form

$$E(Y) = a + be^{cx}. \tag{5}$$

It should be noted that equation (4) represents a **linear** regression, since even though the equation is not linear in the independent variable, x, it is linear in the parameters $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$, whereas the regression model of equation (5) is **nonlinear**, as it is nonlinear in the parameters (a, b and c).

2.2.3 Fitting the regression model – least-squares estimation

The method used to determine values for the parameters is, based on a given set of data, to minimize the sums of squares of the differences between the observed values of the dependent variable and the values predicted by the regression equation for that set of data – hence the term **least-squares** estimation. For example, if a regression model of the type given by equation (3), viz.

$E(Y) = \beta_0 x_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$,
where $x_0 = 1$ for all observations,

is to be fitted to the n data points

$$\begin{array}{c} (x_{01}, x_{11}, x_{21}, \ldots, x_{p1}, y_1) \\ (x_{02}, x_{12}, x_{22}, \ldots, x_{p2}, y_2) \\ \vdots \\ (x_{0n}, x_{1n}, x_{2n}, \ldots, x_{pn}, y_n) \end{array} \tag{6}$$

such that

$$y_i = \beta_0 x_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_p x_{pi} + e_i, \quad i = 1, 2, \ldots, n$$

where e_i are unknown independent random errors with $E(e_i) = 0$ and $\text{var}(e_i) = \sigma^2$, σ^2 being a constant, then the method used is to calculate the estimates of the regression parameters $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ by minimizing

$$\sum_{i=1}^{n} e_i^2. \tag{7}$$

If the errors do not have constant variance, i.e.,

$$\text{var}(e_i) = \sigma_i^2 = \frac{\sigma^2}{w_i}$$

then **weighted least-squares** estimation is used in which

$$\sum_{i=1}^{n} w_i e_i^2$$

is minimized. For a more complete discussion of these least-squares regression methods, and details of the mathematical techniques used, see Draper and Smith [4] or Kendall and Stuart [9].

2.2.4 Regression models and designed experiments

One application of regression models is in the analysis of experiments. In this case the model relates the dependent variable to qualitative independent variables known as **factors**. Factors may take a number of different values known as **levels**. For example, in an experiment in which one of four different treatments is applied, the model will have one factor with four levels. Each level of the factor can be represented by a dummy variable taking the values 0 or 1. So in the example there are four dummy variables x_j, for $j = 1, 2, 3, 4$ such that:

$$x_{ij} = 1 \text{ if the } i\text{th observation received the } j\text{th treatment}$$
$$= 0 \text{ otherwise,}$$

along with a variable for the mean x_0:

$$x_{i0} = 1 \text{ for all } i.$$

If there were 7 observations the data would be:

Treatment	Y	x_0	x_1	x_2	x_3	x_4
1	y_1	1	1	0	0	0
2	y_2	1	0	1	0	0
2	y_3	1	0	1	0	0
3	y_4	1	0	0	1	0
3	y_5	1	0	0	1	0
4	y_6	1	0	0	0	1
4	y_7	1	0	0	0	1

Models which include factors are sometimes known as **General Linear (Regression) Models**. When dummy variables are used it is common for the model not to be of full rank. In the case above, the model would not be of full rank because:

$$x_{i4} = x_{i0} - x_{i1} - x_{i2} - x_{i3}, \text{ for } i = 1, 2, \ldots, 7.$$

This means that the effect of x_4 cannot be distinguished from the combined effect of x_0, x_1, x_2 and x_3. This is known as **aliasing**. In this situation, the aliasing can be deduced from the experimental design and as a result the model to be fitted; in such situations it is known as intrinsic aliasing. In the example above no matter how many times each treatment is replicated (other than 0) the aliasing will still be present. If the aliasing is due to a particular data set to which the model is to be fitted then it is known as extrinsic aliasing. If in the example above observation 1 was missing then the x_1 term would also be aliased. In general intrinsic aliasing may be overcome by changing the model, e.g. remove x_0 or x_1 from the model, or by introducing constraints on the parameters, e.g. $\beta_1 + \beta_2 + \beta_3 + \beta_4 = 0$.

If aliasing is present then there will no longer be a unique set of least-squares estimates for the parameters of the model but the fitted values will still have a unique estimate. Some linear functions of the parameters will also have unique estimates these are known as **estimable functions**. In the example given above the functions $(\beta_0 + \beta_1)$ and $(\beta_2 - \beta_3)$ are both estimable.

2.2.5 Selecting the regression model

In many situations there are several possible independent variables not all of which may be needed in the model. In order to select a suitable set of independent variables, two basic approaches can be used.

(a) All possible regressions

In this case all the possible combinations of independent variables are fitted and the one considered the best selected. To choose the best, two conflicting criteria have to be balanced. One is the fit of the model as measured by the residual sum of squares. This will decrease as more variables are added to the model. The second criterion is the desire to have a model with a small number of significant terms. To aid in the choice of model, statistics such as R^2, which gives the proportion of variation explained by the model, and C_p, which tries to balance the size of the residual sum of squares against the number of terms in the model, can be used.

(b) Stepwise model building

In stepwise model building the regression model is constructed recursively, adding or deleting the independent variables one at a time. When the model is built up the procedure is known as forward selection. The first step is to choose the single variable which is the best predictor. The second independent variable to be added to the regression equation is that which provides the best fit in conjunction with the first variable. Further variables are then added in this recursive fashion, adding at each step the optimum variable, given the other variables already in the equation. Alternatively, backward elimination can be used. This is when all variables are added and then the variables dropped one at a time, the variable dropped being the one which has the least effect on the fit of the model at that stage. There are also hybrid techniques which combine forward selection with backward elimination.

2.2.6 Examining the fit of the model

Having fitted a model two questions need to be asked: first, 'are all the terms in the model needed?' and second, 'is there some systematic lack of fit?'. To answer the first question either confidence intervals can be computed for the parameters or t-tests can be calculated to test hypotheses about the regression parameters – for example, whether the value of the parameter, β_k, is significantly different from a specified value, b_k (often zero). If the estimate of β_k is $\hat{\beta}_k$ and its standard error is $\mathrm{se}(\hat{\beta}_k)$ then the t-statistic is:

$$\frac{\hat{\beta}_k - b_k}{\sqrt{\mathrm{se}(\hat{\beta}_k)}}.$$

It should be noted that both the tests and the confidence intervals may not be independent. Alternatively F-tests based on the residual sums of squares for different models can also be used to test the significance of terms in the model. If model 1, giving residual sum of squares RSS_1 with degrees of freedom ν_1, is a sub-model of model 2, giving residual sum of squares RSS_2 with degrees of freedom ν_2, i.e., all terms in model 1 are also in model 2, then to test if the extra terms in model 2 are needed the F-statistic

$$F = \frac{(RSS_1 - RSS_2)/(\nu_1 - \nu_2)}{RSS_2/\nu_2}$$

may be used. These tests and confidence intervals require the additional assumption that the errors, e_i, are Normally distributed.

To check for systematic lack of fit the residuals, $r_i = y_i - \hat{y}_i$, where $\hat{y}_i$ is the fitted value, should be examined. If the model is correct then they should be random with no discernable pattern. Due to the way they are calculated the residuals do not have constant variance. Now the vector of fitted values can be written as a linear combination of the vector of observations of the dependent variable, y, $\hat{y} = Hy$. The variance-covariance matrix of the residuals is then $(I - H)\sigma^2$, I being the identity matrix. The diagonal elements of H, h_{ii}, can therefore be used to standardize the residuals. The h_{ii} are a measure of the effect of the ith observation on the fitted model and are sometimes known as **leverages**.

If the observations were taken serially the residuals may also be used to test the assumption of the independence of the e_i and hence the independence of the observations.

2.2.7 Computational methods

Let X be the n by p matrix of independent variables and y be the vector of values for the dependent variable. To find the least-squares estimates of the vector of parameters, $\hat{\beta}$, the QR decomposition of X is found, i.e.,

$$X = QR^*$$

where $R^* = \begin{pmatrix} R \\ 0 \end{pmatrix}$, R being a p by p upper triangular matrix and Q is a n by n orthogonal matrix. If R is of full rank then $\hat{\beta}$ is the solution to:

$$R\hat{\beta} = c_1$$

where $c = Q^T y$ and c_1 is the first p rows of c. If R is not of full rank, a solution is obtained by means of a singular value decomposition (SVD) of R,

$$R = Q_* \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} P^T,$$

where D is a k by k diagonal matrix with non-zero diagonal elements, k being the rank of R, and Q_* and P are p by p orthogonal matrices. This gives the solution

$$\hat{\beta} = P_1 D^{-1} Q_{*_1}^T c_1$$

P_1 being the first k columns of P and Q_{*_1} being the first k columns of Q_*.

This will be only one of the possible solutions. Other estimates may be obtained by applying constraints to the parameters. If weighted regression with a vector of weights w is required then both X and y are premultiplied by $w^{1/2}$.

The method described above will, in general, be more accurate than methods based on forming $(X^T X)$, (or a scaled version), and then solving the equations:

$$(X^T X)\hat{\beta} = X^T y.$$

2.2.8 Robust estimation

Least-squares regression can be greatly affected by a small number of unusual, atypical, or extreme observations. To protect against such occurrences, robust regression methods have been developed. These methods aim to give less weight to an observation which seems to be out of line with the rest of the data given the model under consideration. That is to seek to bound the influence. For a discussion of influence in regression, see Hampel *et al.* [6] and Huber [8].

There are two ways in which an observation for a regression model can be considered atypical. The values of the independent variables for the observation may be atypical or the residual from the model may be large.

The first problem of atypical values of the independent variables can be tackled by calculating weights for each observation which reflect how atypical it is, i.e., a strongly atypical observation would have a low weight. There are several ways of finding suitable weights; some are discussed in Hampel *et al.* [6].

The second problem is tackled by bounding the contribution of the individual e_i's to the criterion to be minimized. When minimizing (7) a set of linear equations is formed, the solution of which gives the least-squares estimates. The equations are:

$$\sum_{i=1}^{n} e_i x_{ij} = 0 \quad j = 0, 1, \ldots, k.$$

These equations are replaced by

$$\sum_{i=1}^{n} \psi(e_i/\sigma)x_{ij} = 0 \quad j = 0,1,\ldots,k, \tag{8}$$

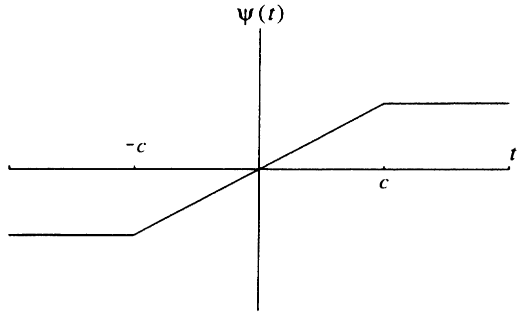


Figure 2

where σ^2 is the variance of the e_i's, and ψ is a suitable function which down weights large values of the standardized residuals e_i/σ. There are several suggested forms for ψ, one of which is Huber's function,

$$\psi(t) = \begin{cases} -c, \; t < c \\ t, \; |t| \leq c \\ c, \; t > c \end{cases} \tag{9}$$

The solution to (8) gives the M-estimates of the regression coefficients. The weights can be included in (8) to protect against both types of extreme value. The parameter σ can be estimated by the median absolute deviations of the residuals or as a solution to, in the unweighted case:

$$\sum_{i=1}^{n} \chi(e_i/\hat{\sigma}) = (n-k)\beta$$

where χ is a suitable function and β is a constant chosen to make the estimate unbiased. χ is often chosen to be $\psi^2/2$ where ψ is given in (9). Another form of robust regression is to minimize the sum of absolute deviations, i.e.,

$$\sum_{i=1}^{n} |e_i|.$$

For details of robust regression, see Hampel *et al.* [6] and Huber [8].

Robust regressions using least absolute deviations can be computed using routines in Chapter E02.

2.2.9 Generalized linear models

Generalized linear models are an extension of the general linear regression model discussed above. They allow a wide range of models to be fitted. These included certain non-linear regression models, logistic and probit regression models for binary data, and log-linear models for contingency tables. A generalized linear model consists of three basic components:

(a) A suitable distribution for the dependent variable Y. The following distributions are common:

  (i) Normal

  (ii) binomial

  (iii) Poisson

  (iv) gamma

  In addition to the obvious uses of models with these distributions it should be noted that the Poisson distribution can be used in the analysis of contingency tables while the gamma distribution can be used to model variance components. The effect of the choice of the distribution is to define the relationship between the expected value of Y, $E(Y) = \mu$, and its variance and so a generalized linear model with one of the above distributions may be used in a wider context when that relationship holds.

(b) A linear model $\eta = \sum \beta_j x_j$, η is known as a **linear predictor**.

(c) A link function $g(\cdot)$ between the expected value of Y and the **linear predictor**, $g(\mu) = \eta$. The following link functions are available:

For the binomial distribution ϵ, observing y out of t:

(i) logistic link: $\eta = \log\left(\frac{\mu}{t-\mu}\right)$;

(ii) probit link: $\eta = \Phi^{-1}(\frac{\mu}{t})$;

(iii) complementary log-log: $\eta = \log(-\log(1-\frac{\mu}{t}))$.

For the Normal, Poisson, and gamma distributions:

(i) exponent link: $\eta = \mu^a$, for a constant a;

(ii) identity link: $\eta = \mu$;

(iii) log link: $\eta = \log \mu$;

(iv) square root link: $\eta = \sqrt{\mu}$;

(v) reciprocal link: $\eta = \frac{1}{\mu}$.

For each distribution there is a **canonical link**. For the canonical link there exist sufficient statistics for the parameters. The canonical links are:

(i) Normal – identity;

(ii) binomial – logistic;

(iii) Poisson – logarithmic;

(iv) gamma – reciprocal.

For the general linear regression model described above the three components are:

(i) Distribution – Normal;

(ii) Linear model – $\sum \beta_j x_j$;

(iii) Link – identity.

The model is fitted by **maximum likelihood**; this is equivalent to least-squares in the case of the Normal distribution. The residual sums of squares used in regression models is generalized to the concept of **deviance**. The deviance is the logarithm of the ratio of the likelihood of the model to the full model in which $\hat{\mu}_i = y_i$ where $\hat{\mu}_i$ is the estimated value of μ_i. For the Normal distribution the deviance is the residual sum of squares. Except for the case of the Normal distribution with the identity link χ^2 and F tests based on the deviance are only approximate; also the estimates of the parameters will only be approximately Normally distributed. Thus only approximate z- or t-tests may be performed on the parameter values and approximate confidence intervals computed.

The estimates are found by using an **iterative weighted least-squares** procedure. This is equivalent to the Fisher scoring method in which the Hessian matrix used in the Newton–Raphson method is replaced by its expected value. In the case of canonical links the Fisher scoring method and the Newton–Raphson method are identical. Starting values for the iterative procedure are obtained by replacing the μ_i by y_i in the appropriate equations.

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 Correlation

3.1.1 Product-moment correlation

Let SS_x be the sum of squares of deviations from the mean, $\bar{x}$, for the variable x for a sample of size n, i.e.,

$$SS_x = \sum_{i=1}^{n} (x_i - \bar{x})^2$$

and let SC_{xy} be the cross-products of deviations from the means, $\bar{x}$ and $\bar{y}$, for the variables x and y for a sample of size n, i.e.,

$$SC_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}).$$

Then the sample covariance of x and y is

$$\operatorname{cov}(x,y) = \frac{SC_{xy}}{(n-1)}$$

and the product-moment correlation coefficent is

$$r = \frac{\operatorname{cov}(x,y)}{\sqrt{\operatorname{var}(x)\operatorname{var}(y)}} = \frac{SC_{xy}}{\sqrt{SS_x SS_y}}.$$

G02BUF computes the sample sums of squares and cross-products deviations from the means (optionally weighted). G02BTF updates the sample sums of squares and cross-products and deviations from the means by the addition/deletion of a (weighted) observation. G02BWF computes the product-moment correlation coefficients from the sample sums of squares and cross-products of deviations from the means. The three routines compute only the upper triangle of the correlation matrix which is stored in a one-dimensional array in packed form. G02BXF computes both the (optionally weighted) covariance matrix and the (optionally weighted) correlation matrix. These are returned in two-dimensional arrays. (Note that G02BTF and G02BUF can be used to compute the sums of squares from zero.)

G02BGF can be used to calculate the correlation coefficients for a subset of variables in the data matrix.

3.1.2 Product-moment correlation with missing values

If there are missing values then G02BUF and G02BXF, as described above, will allow casewise deletion by the user giving the observation zero weight (compared with unit weight for an otherwise unweighted computation).

Other routines also handle missing values in the calculation of unweighted product-moment correlation coefficients. Casewise exclusion of missing values is provided by G02BBF while pairwise omission of missing values is carried out by G02BCF. These two routines calculate a correlation matrix for all the variables in the data matrix; similar output but for only a selected subset of variables is provided by routines G02BHF and G02BJF respectively. As well as providing the Pearson product-moment correlation coefficients, these routines also calculate the means and standard deviations of the variables, and the matrix of sums of squares and cross-products of deviations from the means. For all four routines the user is free to select appropriate values for consideration as missing values, bearing in mind the nature of the data and the possible range of valid values. The missing values for each variable may be either different or alike and it is not necessary to specify missing values for all the variables.

3.1.3 Non-parametric correlation

There are five routines which perform non-parametric correlations, each of which is capable of producing both Spearman's rank-order and Kendall's tau correlation coefficients. The basic underlying concept of both these methods is to replace each observation by its corresponding rank or order within the observations on that variable, and the correlations are then calculated using these ranks.

It is obviously more convenient to order the observations and calculate the ranks for a particular variable just once, and to store these ranks for subsequent use in calculating all coefficients involving that variable; this does however require an amount of store of the same size as the original data matrix, which in some cases might be excessive. Accordingly, some routines calculate the ranks only once, and replace the input data matrix by the matrix of ranks, which are then also made available to the user on exit from the routine, while others preserve the data matrix and calculate the ranks a number of times within the routine; the ranks of the observations are not provided as output by routines which work in the latter way. The routines which overwrite the data matrix with the ranks are intended for possible use in two ways: firstly, if the data matrix is no longer required by the program once the correlation coefficients have been determined, then it is of no consequence that this matrix is replaced by the ranks, and secondly, if the original data is still required, the data can be copied into a second matrix, and this new matrix used in the routine, so that even though this second matrix is replaced by the ranks, the original data matrix is still accessible. If it is possible to arrange the program in such a way that the first technique can be used, then efficiency of timing is achieved with no additional storage, whereas in the second case, it is necessary to have a second matrix of the same size as the data matrix, which may not be acceptable in certain circumstances; in this case it is necessary to reach a compromise between efficiency of time and of storage, and this may well be dependent upon local conditions.

Routines G02BNF and G02BQF both calculate Kendall's tau and/or Spearman's rank-order correlation coefficients taking no account of missing values; G02BNF does so by calculating the ranks of each variable only once, and replacing the data matrix by the matrix of ranks, whereas G02BQF calculates the ranks of each variable several times. Routines G02BPF and G02BRF provide the same output, but treat missing values in a 'casewise' manner (see above); G02BPF calculates the ranks of each variable only once, and overwrites the data matrix of ranks, while G02BRF determines the ranks of each variable several times. For 'pairwise' omission of missing data (see above), the routine G02BSF provides Kendall and/or Spearman coefficients.

Since G02BNF and G02BPF order the observations and calculate the ranks of each variable only once, then if there are M variables involved, there are only M separate 'ranking' operations; this should be contrasted with the method used by routines G02BQF and G02BRF which perform $M(M-1)/2+1$ similar ranking operations. These ranking operations are by far the most time-consuming parts of these non-parametric routines, so for a matrix of as few as five variables, the time taken by one of the slower routines can be expected to be at least a factor of two slower than the corresponding efficient routine; as the number of variables increases, so this relative efficiency factor increases. Only one routine, G02BSF, is provided for pairwise missing values, and this routine carries out $M(M-1)$ separate rankings; since by the very nature of the pairwise method it is necessary to treat each pair of variables separately and rank them individually, it is impossible to reduce this number of operations, and so no alternative routine is provided.

3.1.4 Partial correlation

G02BYF computes a matrix of partial correlation coefficients from the correlation coefficients or variance-covariance matrix returned by G02BXF.

3.1.5 Robust correlation

G02HLF and G02HMF compute robust esimates of the variance-covariance matrix by solving the equations:

$$\frac{1}{n}\sum_{i=1}^{n} w(\|z_i\|_2)z_i = 0$$

and

$$\frac{1}{n}\sum_{i=1}^{n} u(\|z_i\|_2)z_i z_i^T - v(\|z_i\|_2)I = 0,$$

as described in Section 2.1.3 for user-supplied functions w and u. Two options are available for v, either $v(t)=1$ for all t or $v(t)=u(t)$.

G02HMF requires only the function w and u to be supplied while G02HLF also requires their derivatives. In general G02HLF will be considerably faster than G02HMF and should be used if derivatives are available.

G02HKF computes a robust variance-covariance matrix for the following functions:

$$\begin{aligned} u(t) &= \frac{a_u}{t^2} \text{ if } t < a_u^2 \\ u(t) &= 1 \text{ if } a_u^2 \leq t \leq b_u^2 \\ u(t) &= \frac{b_u}{t^2} \text{ if } t > b_u^2 \end{aligned}$$

and

$$\begin{aligned} w(t) &= 1 \text{ if } t \leq c_w \\ w(t) &= \frac{c_w}{t} \text{ if } t > c_w \end{aligned}$$

for constants a_u, b_u and c_w.

These functions solve a minimax space problem considered by Huber [8]. The values of a_u, b_u and c_w are calculated from the fraction of gross errors; see Hampel *et al.* [6] and Huber [8].

To compute a correlation matrix from the variance-covariance matrix G02BWF may be used.

3.2 Regression

3.2.1 Simple linear regression

Four routines are provided for simple linear regressions: G02CAF and G02CCF perform the simple linear regression with a constant term (equation (1) above), while G02CBF and G02CDF fit the simple linear regression with **no** constant term (equation (2) above). Two of these routines, G02CCF and G02CDF, take account of missing values, which the others do not. In these two routines, an observation is omitted if it contains a missing value for either the dependent or the independent variable; this is equivalent to both the casewise and pairwise methods, since both are identical when there are only two variables involved. Input to these routines consists of the raw data, and output includes the coefficients, their standard errors and t-values for testing the significance of the coefficients; the F-value for testing the overall significance of the regression is also given.

3.2.2 Multiple linear regression – general linear model

G02DAF fits a general linear regression model using the QR method and an SVD if the model is not of full rank. The results returned include: residual sum of squares, parameter estimates, their standard errors and variance-covariance matrix, residuals and leverages. There are also several routines to modify the model fitted by G02DAF and to aid in the interpretation of the model.

G02DCF adds or deletes an observation from the model.

G02DDF computes the parameter estimates, and their standard errors and variance-covariance matrix for a model that is modified by G02DCF, G02DEF or G02DFF.

G02DEF adds a new variable to a model.

G02DFF drops a variable from a model.

G02DGF fits the regression to a new dependent variable, i.e., keeping the same independent variables.

G02DKF calculates the estimates of the parameters for a given set of constraints, (e.g. parameters for the levels of a factor sum to zero), for a model which is not of full rank and the SVD has been used.

G02DNF calculates the estimate of an estimable function and its standard error.

Note. G02DEF also allows the user to initialize a model building process and then to build up the model by adding variables one at a time.

If the user wishes to use methods based on forming the cross-products/correlation matrix (i.e., $(X^T X)$ matrix) rather than the recomended use of G02DAF then the following routines should be used.

For regression through the origin (i.e., no constant) G02CHF preceded by:

G02BDF (no missing values, all variables)

G02BKF (no missing values, subset of variables)

G02BEF (casewise missing values, all variables)

G02BLF (casewise missing values, subset of variables)

G02BFF* (pairwise missing values, all variables)

G02BMF* (pairwise missing values, subset of variables)

For regression with intercept (i.e., with constant) G02CGF preceded by:

G02BAF (no missing values, all variables)

G02BGF (no missing values, subset of variables)

G02BBF (casewise missing values, all variables)

G02BHF (casewise missing values, subset of variables)

G02BCF* (pairwise missing values, all variables)

G02BJF* (pairwise missing values, subset of variables)

Note that the four routines using pairwise deletion of missing value (marked with $*$) should be used with great caution as the use of this method can lead to misleading results, particularly if a significant proportion of values are missing.

Both G02CHF and G02CGF require that the correlations/sums of squares involving the dependent variable must appear as the last row/column. Because the layout of the variables in a user's data array may not be arranged in this way, two routines, G02CEF and G02CFF, are provided for re-arranging the rows and columns of vectors and matrices. G02CFF simply re-orders the rows and columns while G02CEF forms smaller vectors and matrices from larger ones.

Output from G02CGF and G02CHF consists of the coefficients, their standard errors, R^2-values, t and F statistics.

3.2.3 Selecting regression models

To aid the selection of a regression model the following routines are available.

G02EAF computes the residual sums of squares for all possible regressions for a given set of dependent variables. The routine allows some variables to be forced into all regressions.

G02ECF computes the values of R^2 and C_p from the residual sums of squares as provided by G02EAF.

G02EEF enables the user to fit a model by forward selection. The user may call G02EEF a number of times. At each call the routine will calculate the changes in the residual sum of squares from adding each of the variables not already included in the model, select the variable which gives the largest change and then if the change in residual sum of squares meets the given criterion will add it to the model.

3.2.4 Residuals

G02FAF computes the following standardized residuals and measures of influence for the residuals and leverages produced by G02DAF:

(i) Internally studentized residual;
(ii) Externally studentized residual;
(iii) Cook's D statistic;
(iv) Atkinson's T statistic.

G02FCF computes the Durbin–Watson test statistic and bounds for its signficance to test for serial correlation in the errors, e_i.

3.2.5 Robust regression

For robust regression using M-estimates instead of least-squares the routine G02HAF will generally be suitable. G02HAF provides a choice of four ψ-functions (Huber's, Hampel's, Andrew's and Tukey's) plus two different weighting methods and the option not to use weights. If other weights or different ψ-functions are needed the routine G02HDF may be used. G02HDF requires the user to supply weights, if required, and also routines to calculate the ψ-function and, optionally, the χ-function. G02HBF can be used in calculating suitable weights. The routine G02HFF can be used after a call to G02HDF in order to calculate the variance-covariance estimate of the estimated regression coefficients.

For robust regression, using least absolute deviation, E02GAF can be used.

3.2.6 Generalized linear models

There are four routines for fitting generalized linear models. The output includes: the deviance, parameter estimates and their standard errors, fitted values, residuals and leverages. The routines are:

G02GAF – Normal distribution

G02GBF – binomial distribution

G02GCF – Poisson distribution

G02GDF – gamma distribution

While G02GAF can be used to fit linear regression models (i.e., by using an identity link) this is not recomended as G02DAF will fit these models more efficiently. G02GCF can be used to fit log-linear models to contingency tables.

In addition to the routines to fit the models there are two routines to aid the interpretation of the model if a model which is not of full rank has been fitted, i.e., aliasing is present.

G02GKF computes parameter estimates for a set of constraints, (e.g. sum of effects for a factor is zero), from the SVD solution provided by the fitting routine.

G02GNF calculates an estimate of an estimable function along with its standard error.

3.2.7 Polynomial regression and non-linear regression

No routines are currently provided in this chapter for polynomial regression. Users wishing to perform polynomial regressions do however have three alternatives: they can use the multiple linear regression routines, G02DAF, with a set of independent variables which are in fact simply the same single variable raised to different powers, or they can use the routine G04EAF to compute orthogonal polynomials which can then be used with G02DAF, or they can use the routines in Chapter E02 (Curve and Surface Fitting) which fit polynomials to sets of data points using the techniques of orthogonal polynomials. This latter course is to be preferred, since it is more efficient and liable to be more accurate, but in some cases more statistical information may be required than is provided by those routines, and it may be necessary to use the routines of this chapter.

More general nonlinear regression models may be fitted using the optimization routines in Chapter E04, which contains routines to minimize the function

$$\sum_{i=1}^{n} e_i^2$$

where the regression parameters are the variables of the minimization problem.

4 Index

Note. Only a selection of the routines available in this chapter appears on the following list. This selection should cover most applications and includes the recomended routines.

Product-moment correlation:	
unweighted/weighted correlation and covariance matrix	G02BXF
unweighted/weighted sum of squares and cross-products	G02BUF
update sum of squares and cross-products matrix	G02BTF
correlation matrix from sum of squares and cross-products matrix	G02BWF
unweighted on a subset of variables	G02BGF
unweighted with missing values	G02BBF
unweighted on a subset of variables with missing values	G02BHF
Non-parametric correlation:	
no missing observations, overwriting input data	G02BNF
missing observations, overwriting input data	G02BPF
Partial correlation:	
From correlation/variance-covariance matrix	G02BYF
Robust correlation:	

Huber's method	G02HKF
user-supplied weight function plus derivatives	G02HLF
user-supplied weight function only	G02HMF
Simple linear regression:	
simple linear regression	G02CAF
simple linear regression, no intercept	G02CBF
simple linear regression with missing values	G02CCF
simple linear regression, no intercept with missing values	G02CDF
Multiple linear regression/General linear model:	
general linear regression model	G02DAF
add/delete observation from model	G02DCF
add independent variable to model	G02DEF
delete independent variable from model	G02DFF
regression parameters from updated model	G02DDF
regression for new dependent variable	G02DGF
transform model parameters	G02DKF
computes estimable function	G02DNF
Selecting regression model:	
all possible regressions	G02EAF
R^2 and C_p statistics	G02ECF
forward selection	G02EEF
Residuals:	
standardized residuals and influence statistics	G02FAF
Durbin–Watson test	G02FCF
Robust regression:	
standard M-estimates	G02HAF
user supplies weight functions	G02HDF
Generalized linear models:	
Normal errors	G02GAF
binomial errors	G02GBF
Poisson errors	G02GCF
gamma errors	G02GDF
transform model parameters	G02GKF
computes estimable function	G02GNF

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'.

G02CJF

6 References

[1] Atkinson A C (1986) *Plots, Transformations and Regressions* Clarendon Press, Oxford

[2] Churchman C W and Ratoosh P (1959) *Measurement Definitions and Theory* Wiley

[3] Cook R D and Weisberg S (1982) *Residuals and Influence in Regression* Chapman and Hall

[4] Draper N R and Smith H (1985) *Applied Regression Analysis* Wiley (2nd Edition)

[5] Hammarling S (1985) The singular value decomposition in multivariate statistics *SIGNUM Newsl.* **20 (3)** 2–25

[6] Hampel F R, Ronchetti E M, Rousseeuw P J and Stahel W A (1986) *Robust Statistics. The Approach Based on Influence Functions* Wiley

[7] Hays W L (1970) *Statistics* Holt, Rinehart and Winston

[8] Huber P J (1981) *Robust Statistics* Wiley

[9] Kendall M G and Stuart A (1973) *The Advanced Theory of Statistics (Volume 2)* Griffin (3rd Edition)

[10] McCullagh P and Nelder J A (1983) *Generalized Linear Models* Chapman and Hall

[11] Searle S R (1971) *Linear Models* Wiley

[12] Siegel S (1956) *Non-parametric Statistics for the Behavioral Sciences* McGraw-Hill

[13] Weisberg S (1985) *Applied Linear Regression* Wiley

Chapter G03 – Multivariate Methods

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G03AAF	14	Performs principal component analysis
G03ACF	14	Performs canonical variate analysis
G03ADF	14	Performs canonical correlation analysis
G03BAF	15	Computes orthogonal rotations for loading matrix, generalized orthomax criterion
G03BCF	15	Computes Procrustes rotations
G03CAF	15	Computes maximum likelihood estimates of the parameters of a factor analysis model, factor loadings, communalities and residual correlations
G03CCF	15	Computes factor score coefficients (for use after G03CAF)
G03DAF	15	Computes test statistic for equality of within-group covariance matrices and matrices for discriminant analysis
G03DBF	15	Computes Mahalanobis squared distances for group or pooled variance-covariance matrices (for use after G03DAF)
G03DCF	15	Allocates observations to groups according to selected rules (for use after G03DAF)
G03EAF	16	Computes distance matrix
G03ECF	16	Hierarchical cluster analysis
G03EFF	16	K-means cluster analysis
G03EHF	16	Constructs dendrogram (for use after G03ECF)
G03EJF	16	Computes cluster indicator variable (for use after G03ECF)
G03FAF	17	Performs principal co-ordinate analysis, classical metric scaling
G03FCF	17	Performs non-metric (ordinal) multidimensional scaling
G03ZAF	15	Produces standardized values (z-scores) for a data matrix

Chapter G03

Multivariate Methods

Contents

1 Scope of the Chapter

This chapter is concerned with methods for studying multivariate data. A multivariate data set consists of several variables recorded on a number of objects or individuals. Multivariate methods can be classified as those that seek to examine the relationships between the variables (e.g. principal components), known as variable-directed methods, and those that seek to examine the relationships between the objects (e.g. cluster analysis), known as individual-directed methods.

Multiple regression is not included in this chapter as it involves the relationship of a single variable, known as the response variable, to the other variables in the data set, the explanatory variables. Routines for multiple regression are provided in Chapter G02.

2 Background to the Problems

2.1 Variable-directed Methods

Let the n by p data matrix consist of p variables, $x_1, x_2, \ldots, x_p$, observed on n objects or individuals. Variable-directed methods seek to examine the linear relationships between the p variables with the aim of reducing the dimensionality of the problem. There are different methods depending on the structure of the problem. **Principal component analysis** and **factor analysis** examine the relationships between all the variables. If the individuals are classified into groups then **canonical variate analysis** examines the between group structure. If the variables can be considered as coming from two sets then **canonical correlation analysis** examines the relationships between the two sets of variables. All four methods are based on an eigenvalue decomposition or a singular value decomposition (SVD) of an appropriate matrix.

The above methods may reduce the dimensionality of the data from the original p variables to a smaller number, k, of derived variables that adequately represent the data. In general these k derived variables will be unique only up to an **orthogonal rotation**. Therefore it may be useful to see if there exists suitable rotations of these variables that lead to a simple interpretation of the new variables in terms of the original variables.

2.1.1 Principal component analysis

Principal component analysis finds new variables which are linear combinations of the p observed variables so that they have maximum variation and are orthogonal (uncorrelated).

Let S be the p by p variance-covariance matrix of the n by p data matrix. A vector a_1 of length p is found such that:

$$a_1^T S a_1 \text{ is maximised subject to } a_1^T a_1 = 1.$$

The variable $z_1 = \sum_{i=1}^{p} a_{1i} x_i$ is known as the first principal component and gives the linear combination of the variables that gives the maximum variation. A second principal component, $z_2 = \sum_{i=1}^{p} a_{2i} x_i$, is found such that:

$$a_2^T S a_2 \text{ is maximised subject to } a_2^T a_2 = 1 \text{ and } a_2^T a_1 = 0.$$

This gives the linear combination of variables, orthogonal to the first principal component, that gives the maximum variation. Further principal components are derived in a similar way.

The vectors a_i, for $i = 1, 2, \ldots, p$ are the eigenvectors of the matrix S and associated with each eigenvector is the eigenvalue, γ_i^2. The value of $\gamma_i^2 / \sum \gamma_i^2$ gives the proportion of variation explained by the ith principal component. Alternatively the a_i can be considered as the right singular vectors in a SVD of a scaled mean centred data matrix. The singular values of the SVD are the γ_i-values.

Often fewer than p dimensions (principal components) are needed to represent most of the variation in the data. A test on the smaller eigenvalues can be used to investigate the number of dimensions needed.

The values of the principal component variables for the individuals are known as the principal component scores. These can be standardized so that the variance of these scores for each principal component is 1.0 or equal to the corresponding eigenvalue. The principal component scores correspond to the left-hand singular vectors in the SVD.

2.1.2 Factor analysis

Let the p variables have variance-covariance matrix Σ. The aim of factor analysis is to account for the covariances in these p variables in terms of a smaller number, k, of hypothetical variables, or factors, $f_1, f_2, \ldots, f_k$. These are assumed to be independent and to have unit variance. The relationship between the observed variables and the factors is given by the model

$$x_i = \sum_{j=1}^{k} \lambda_{ij} f_j + e_i \quad i = 1, 2, \ldots, p$$

where λ_{ij}, for $i = 1, 2, \ldots, p$, $j = 1, 2, \ldots, k$, are the factor loadings and e_i, for $i = 1, 2, \ldots, p$, are independent random variables with variances ψ_i. These represent the unique component of the variation of each observed variable. The proportion of variation for each variable accounted for by the factors is known as the communality.

The model for the variance-covariance matrix, Σ, can then be written as:

$$\Sigma = \Lambda\Lambda^T + \Psi,$$

where Λ is the matrix of the factor loadings, λ_{ij}, and Ψ is a diagonal matrix of the unique variances ψ_i.

If it is assumed that both the k factors and the e_i follow independent Normal distributions then the parameters of the model, Λ and Ψ, can be estimated by maximum likelihood as described by Lawley and Maxwell [7]. The computation of the maximum likelihood estimates is an iterative procedure which involves computing the eigenvalues and eigenvectors of the matrix

$$S^* = \Psi^{-1/2} S \Psi^{-1/2},$$

where S is the sample variance-covariance matrix. Alternatively the SVD of the matrix $R\Psi^{-1/2}$ can be used, where $R^T R = S$. When convergence has been achieved the estimates $\hat{\Lambda}$, of Λ, are obtained by scaling the eigenvectors of S^*. The use of maximum likelihood estimation means that likelihood ratio tests can be constructed to test for the number of factors required.

Having found the estimates of the parameters of the model, the estimates of the values of the factors for the individuals, the **factorscores**, can be computed. These involve the calculation of the **factor score coefficients**. Two common methods of computing factor score coefficients are the regression method and Bartlett's method. Bartlett's method gives unbiased estimates of the factor scores while estimates from the regression method are biased but have smaller variance than those from Bartlett's method; see Lawley and Maxwell [7].

2.1.3 Canonical variate analysis

If the individuals can be classified into one of g groups then canonical variate analysis finds the linear combinations of the p variables that maximize the ratio of the between group variation to the within-group variation. These variables are known as canonical variates. As the canonical variates provide discrimination between the groups the method is also known as **canonical discrimination**.

The canonical variates can be calculated from the eigenvectors of the within group sums of squares and cross-products matrix or from the SVD of the matrix

$$V = Q_x^T Q_g,$$

where Q_g is an orthogonal matrix that defines the groups and Q_x is the first p columns of the orthogonal matrix Q from the QR decompostion of the data matrix with the variable means subtracted. If the data matrix is not of full rank the Q_x matrix can be obtained from a SVD. If the SVD of V is

$$V = U_x \Delta U_g^T,$$

then the non-zero elements ($\delta_i > 0$) of the diagonal matrix Δ are the canonical correlations. The largest δ_i is called the **first canonical correlation** and associated with it is the first canonical variate.

The eigenvalues, γ_i^2, of the within-group sums of squares matrix are given by:

$$\gamma_i^2 = \frac{\delta_i^2}{1 - \delta_i^2}$$

and the value of $\pi_i = \gamma_i^2 / \sum \gamma_i^2$ gives the proportion of variation explained by the ith canonical variate. The values of the π_i give an indication as to how many canonical variates are needed to adequately describe the data, i.e., the dimensionality of the problem. The number of dimensions can be investigated by means of a test on the smaller canonical correlations.

The canonical variate loadings and the relationship between the original variables and the canonical variates are calculated from the matrix U_x. This matrix is scaled so that the canonical variates have unit variance.

2.1.4 Canonical correlation analysis

If the p variables can be considered as coming from two sets then canonical correlation analysis finds linear combinations of the variables in each set, known as canonical variates, such that the correlations between corresponding canonical variates for the two sets are maximized. Let the two sets of variables be denoted by x and y with p_x and p_y variables in each set respectively. Let the variance-covariance of the data set be

$$S = \begin{bmatrix} S_{xx} & S_{xy} \\ S_{yx} & S_{yy} \end{bmatrix}$$

and let

$$\Sigma = S_{yy}^{-1} S_{yx} S_{xx}^{-1} S_{xy}$$

then the canonical correlations can be calculated from the eigenvalues of the matrix Σ. Alternatively the canonical correlations can be calculated by means of a SVD of the matrix

$$V = Q_x^T Q_y,$$

where Q_x is the first p_x columns of the orthogonal matrix Q from the QR decompostion of the x-variables in the data matrix and Q_y is the first p_y columns of the Q matrix of the QR decomposition of the y-variables in the data matrix. In both cases the variable means are subtracted before the QR decomposition is computed. If either sets of variables is not of full rank an SVD can be used instead of the QR decomposition. If the SVD of V is

$$V = U_x \Delta U_y^T,$$

then the non-zero elements ($\delta_i > 0$) of the diagonal matrix Δ are the canonical correlations. The largest δ_i is called the **first canonical correlation** and associated with it is the first canonical variate. The eigenvalues, γ_i^2, of the matrix Σ are given by

$$\gamma_i^2 = \frac{\delta_i^2}{1 + \delta_i^2}.$$

The value of $\pi_i = \gamma_i^2 / \sum \gamma_i^2$ gives the proportion of variation explained by the ith canonical variate. The values of the π_i give an indication as to how many canonical variates are needed to adequately describe the data, i.e., the dimensionality of the problem; this can also be investigated by means of a test on the smaller values of the γ_i^2.

The relationship between the canonical variables and the original variables, the canonical variate loadings, can be computed from the U_x and U_y matrices.

2.1.5 Rotations

There are two principal reasons for using rotations. Either

(a) simplifying the structure to aid interpretation of derived variables, or
(b) comparing two or more data sets or sets of derived variables.

The most common type of rotations used for (a) are **orthogonal rotations**. If Λ is the p by k loading matrix from a variable-directed multivariate method, then the rotations are selected such that the elements, λ_{ij}^*, of the rotated loading matrix, Λ^*, are either relatively large or small. The rotations may be found by minimizing the criterion

$$V = \sum_{j=1}^{k} \sum_{i=1}^{p} (\lambda_{ij}^*)^4 - \frac{\gamma}{p} \sum_{j=1}^{k} \left(\sum_{i=1}^{p} (\lambda_{ij}^*)^2 \right)^2$$

where the constant, γ, gives a family of rotations, with $\gamma = 1$ giving **varimax rotations** and $\gamma = 0$ giving **quartimax** rotations.

For (b) **Procrustes** rotations are used. Let A and B be two l by m matrices, which can be considered as representing l points in m dimensions. One example is if A is the loading matrix from a variable-directed multivariate method and B is a hypothesised pattern matrix. In order to try to match the points in A and B there are three steps:

(i) translate so that centroids of both matrices are at the origin,
(ii) find a rotation that minimizes the sum of squared distances between corresponding points of the matrices,
(iii) scale the matrices.

For a more detailed description, see Krzanowski [6].

2.2 Individual-directed Methods

While dealing with the same n by p data matrix as variable-directed methods the emphasis is the n objects or individuals rather than the p variables. The methods are generally based on an n by n distance or dissimilarity matrix such that the (k,j)th element gives a measure of how 'far apart' individual k and j are. Alternatively, a similarity matrix can be used which measures how 'close' individuals are. The form of the measure of distance or similarity will depend upon the form of the p variables. For continuous variables it is usually assumed that some form of Euclidean distance is suitable. That is, for x_{ki} and x_{ji} measured for individuals k and j on variable i respectively, the contribution to distance between individuals k and j from variable i is given by

$$(x_{ki} - x_{ji})^2.$$

Often there will be a need to scale the variables to produce satisfactory distances. For discrete variables there are various measures of similarity or distance that can easily be computed. For example, for binary data a measure of similarity could be

1 – if the individuals take the same value,

0 – otherwise.

Given a measure of distance between individuals there are three basic tasks that can be performed.

(1) *Group the individuals;* that is, collect the individuals into groups so that those within a group are closer to each other than they are to members of another group.
(2) *Classify individuals;* that is, if some individuals are known to come from certain groups allocate individuals whose group membership is unknown to the nearest group.
(3) *Map the individuals;* that is, produce a multidimensional diagram in which the distances on the diagram represent the distances between the individuals.

In the above, (1) leads to cluster analysis, (2) leads to discriminant analysis and (3) leads to scaling methods.

2.2.1 Hierarchical cluster analysis

Approaches for cluster analysis can be classified into two types: hierarchical and non-hierarchical. Hierarchical cluster analysis produces a series of overlapping groups or clusters ranging from separate individuals to one single cluster. For example five individuals could be hierarchically clustered as follows.

Step 1	(1)	(2)	(3)	(4)	(5)
Step 2	(1,2)		(3,4)		(5)
Step 3	(1,2)		(3,4,5)		
Step 4	(1,2,3,4,5)				

The clusters at a level are constructed from the clusters at a previous level. There are two basic approaches to hierarchical cluster analysis: agglomerative methods which build up clusters starting from individuals until there is only one cluster, or divisive methods which start with a single cluster and split clusters until the individual level is reached. This chapter contains the more common agglomerative methods.

The stages in a hierarchical cluster analysis are usually as follows.

(a) Form a distance matrix
(b) Use selected criterion to form hierarchy.
(c) Print cluster information in the form of a dendrogram or use information to form a set of clusters.

These three stages will be considered in turn.

(a) Form distance matrix.

For the n by p data matrix X, a general measure of the distance between object j and object k, d_{jk}, is:

$$d_{jk} = \left(\sum_{i=1}^{p} D(x_{ji}/s_i, x_{ki}/s_i)\right)^{\alpha},$$

where x_{ji} and x_{ki} are the (j,i)th and (k,i)th elements of X, s_i is a standardization for the ith variable and $D(u,v)$ is a suitable function. Three common distances for continuous variables are:

(i) Euclidean distance: $D(u,v) = (u-v)^2$ and $\alpha = \frac{1}{2}$.

(ii) Euclidean squared distance: $D(u,v) = (u-v)^2$ and $\alpha = 1$.

(iii) Absolute distance (city block metric): $D(u,v) = |u-v|$ and $\alpha = 1$.

The common standardisations are the standard deviation and the range. For dichotomous variables there are a number of different measures (see Krzanowski [6] and Everitt [2]); these are usually easy to compute. If the individuals in a cluster analysis are themselves variables, then a suitable distance measure will be based on the correlation coefficient for continuous variables and contingency table statistics for discrete data.

(b) Form Hierarchy

Given a distance matrix for the n individuals, an agglomerative clustering methods produces a hierarchical tree by starting with n clusters each with a single individual and then at each of $n-1$ stages merging two clusters to form a larger cluster until all individuals are in a single cluster. At each stage the two clusters that are nearest are merged to form a new cluster and a new distance matrix is computed for the reduced number of clusters.

Methods differ as to how the distances between the new cluster and other clusters are computed. For three clusters i, j and k let n_i, n_j and n_k be the number of objects in each cluster and let d_{ij}, d_{ik} and d_{jk} be the distances between the clusters. If clusters j and k be merged to give cluster jk, then the distance from cluster i to cluster jk, $d_{i.jk}$, can be computed in the following ways.

(a) Single Link or nearest neighbour : $d_{i.jk} = \min(d_{ij}, d_{ik})$.

(b) Complete Link or furthest neighbour : $d_{i.jk} = \max(d_{ij}, d_{ik})$.

(c) Group average : $d_{i.jk} = \frac{n_j}{n_j+n_k}d_{ij} + \frac{n_k}{n_j+n_k}d_{ik}$.

(d) Centroid : $d_{i.jk} = \frac{n_j}{n_j+n_k}d_{ij} + \frac{n_k}{n_j+n_k}d_{ik} - \frac{n_j n_k}{(n_j+n_k)^2}d_{jk}$.

(e) Median : $d_{i.jk} = \frac{1}{2}d_{ij} + \frac{1}{2}d_{ik} - \frac{1}{4}d_{jk}$.

(f) Minimum variance : $d_{i.jk} = [(n_i+n_j)d_{ij} + (n_i+n_k)d_{ik} - n_i d_{jk}]/(n_i+n_j+n_k)$

For further details, see Everitt [2] or Krzanowski [6].

(c) Produce Dendrogram and Clusters

Hierarchical cluster analysis can be represented by a tree that shows at which distance the clusters merge. Such a tree is known as a dendrogram; see Everitt [2] and Krzanowski [6].

A simple example is

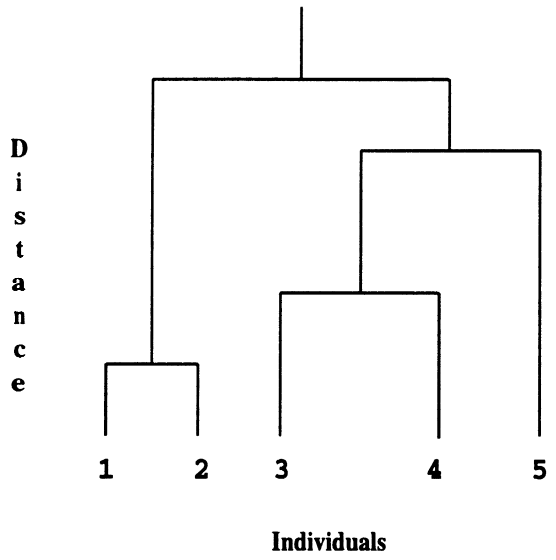


Figure 1

The end-points of the dendrogram represent the individuals that have been clustered.

Alternatively the information from the tree can be used to produced either a chosen number of clusters or the clusters that exist at a given distance. The latter is equivalent to taking the dendrogram and drawing a line across at a given distance to produce clusters.

2.2.2 Non-hierarchical clustering

Non-heirarchical cluster analysis usually forms a given number of clusters from the data. There is no requirement that if first $k-1$ and then k clusters were requested then the $k-1$ clusters would be formed from the k clusters.

Most non-hierarchical methods of cluster analysis seek to partition the set of individuals into a number of clusters so as to optimise a criterion. The number of clusters is usually specified prior to the analysis. One commonly used criterion is the within-cluster sum of squares. Given n individuals with p variables measured on each individual, x_{ij} for $i = 1,2,..., n$, $j = 1,2,...p$, the within-cluster sum of squares for K clusters is:

$$SS_c = \sum_{k=1}^{K} \sum_{i \in S_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2,$$

where S_k is the set of objects in the kth cluster and $\bar{x}_{kj}$ is the mean for the variable j over cluster k. Starting with an initial allocation of individuals to clusters the method then seeks to minimise SS_c by a series of re-allocations. This is often known as K-means clustering.

2.2.3 Discriminant analysis

Discriminant analysis is concerned with the **allocation** of objects to n_g groups on the basis of observations on those objects using an allocation rule. This rule is computed from observations coming from a **training** set in which group membership is known. The allocation rule is based on the distance between the object and an estimate of the location of the groups. If p variables are observed and the vector of means for the jth group in the training set are $\bar{x}_j$ then the usual measure of the distance of an observation, x_k, from the jth group mean is given by Mahalanobis distance:

$$D_{kj}^2 = (x_k - \bar{x}_j)^T S_*^{-1} (x_k - \bar{x}_j),$$

where S_* is either the within-group variance-covariance matrix, S_j, for the n_j objects in the jth group, or a pooled variance-covariance matrix, S, computed from all n objects from all groups where

$$S = \frac{\sum_{j=1}^{n_g} (n_j - 1) S_j}{(n - n_g)}.$$

If the within group variance-covariance matrices can be assumed to be equal then the pooled variance-covariance matrix can be used. This assumption can be tested using the test statistic:

$$G = C\left((n-n_g)\log|S| - \sum_{j=1}^{n_g}(n_j-1)\log|S_j|\right),$$

where

$$C = 1 - \frac{2p^2+3p-1}{6(p+1)(n_g-1)}\left(\sum_{j=1}^{n_g}\frac{1}{(n_j-1)} - \frac{1}{(n-n_g)}\right).$$

For large n, G is approximately distributed as a χ^2 variable with $\frac{1}{2}p(p+1)(n_g-1)$ degrees of freedom; see Morrison [8].

In addition to the distances a set of prior probabilities of group membership, π_j, for $j = 1,2,\ldots,n_g$, may be used. The prior probabilities reflect the user's view as to the likelihood of the objects coming from the different groups.

It is generally assumed that the p variables follow a multivariate Normal distribution with, for the jth group, mean μ_j and variance-covariance matrix Σ_j. If $p(x_k|\mu_j,\Sigma_j)$ is the probability of observing the observation x_k from group j, then the posterior probability of belonging to group j is

$$p(j|x_k,\mu_j,\Sigma_j) \propto p(x_k|\mu_j,\Sigma_j)\pi_j.$$

An observation is allocated to the group with the highest posterior probability.

In the **estimative** approach to discrimination the parameters μ_j and Σ_j in $p(j|x_k,\mu_j,\Sigma_j)$ are replaced by their estimates calculated from the training set. If it is assumed that the within-group variance-covariance matrices are equal then the **linear discriminant function** is obtained; otherwise if it is assumed that the variance-covariance matrices are unequal then the **quadratic discriminant function** is obtained.

In the Bayesian **predictive** approach a non-informative prior distribution is used for the parameters giving the posterior distribution for the parameters from the training set, X_t, of, $p(\mu_j,\Sigma_j|X_t)$. A predictive distribution is then obtained by integrating $p(j|x_k,\mu_j,\Sigma_j)p(\mu_j,\Sigma_j|X)$ over the parameter space. This predictive distribution, $p(x_k|X_t)$, then replaces $p(x_k|\mu_j,\Sigma_j)$ to give

$$p(j|x_k,\mu_j,\Sigma_j) \propto p(x_k|X_t)\pi_j.$$

In addition to allocating the objects to groups an atypicality index for each object and for each group can be computed. This represents the probability of obtaining an observation more typical of the group than that observed. A high value of the atypicality index for all groups indicates that the observation may in fact come from a group not represented in the training set.

Alternative approaches to discrimination are the use of canonical variates and logistic discrimination. Canonical variate analysis is described above and as it seeks to find the directions that best discriminate between groups these directions can also be used to allocate further observations. This can be viewed as an extension of **Fisher's linear discriminant function**. This approach does not assume that the data is Normally distributed, but Fisher's linear discriminant function may not perform well on non-Normal data. In the case of two groups, logistic regression can be performed with the response variable indicating the group allocation and the variables in the discriminant analysis being the explanatory variables. Allocation can then be made on the basis of the fitted response value. This is known as **logistic discrimination** and can be shown to be valid for a wide range of distributional assumptions.

2.2.4 Scaling methods

Scaling methods seek to represent the observed dissimilarities or distances between objects as distances between points in Euclidean space. For example if the distances between objects A, B and C were 3, 4 and 5 the distances could be represented exactly by three points in two-dimensional space. Only their relative positions would be important, the whole configuration of points could be rotated or shifted without effecting the distances between the points. If a one-dimensional representation was required the 'best' representation might give distances of $2\frac{1}{3}, 3\frac{1}{3}$ and $5\frac{2}{3}$, which may be an adequate representation. If the distances were 3, 4 and 8 then these distances could not be exactly represented in Euclidean space even in two dimensions; the best representation being the three points in a straight line giving distances 3, 4 and 7.

In practice the user of scaling methods has to decide upon the number of dimensions in which the data is to be represented. The smaller the number the easier it will be to assimilate the information. The chosen number of dimensions needs to give an adequate representation of the data but will often not give an exact representation because either the number of chosen dimensions is too small or the data cannot be represented in Euclidean space.

Two basic methods are available depending on the nature of the dissimilarities or distances being analysed. If the distances can be assumed to satisfy the metric inequality

$$d_{ij} \leq d_{ik} + d_{kj},$$

then the distances can be represented exactly by points in Euclidean space and the technique known as metric scaling, classical scaling or principal coordinate analysis can be used. This technique involves the computing of the eigenvalues of a matrix derived from the distance matrix. The eigenvectors corresponding to the k largest positive eigenvalues gives the best k dimensions in which to represent the objects. If there are negative eigenvalues then the distance matrix cannot be represented in Euclidean space.

Instead of the above approach of requiring the distances from the points to match the distances from the objects as closely as possible sometimes only a rank-order equivalence is required. That is, the ith largest distance between objects should, as far as possible, be represented by the ith largest distance between points. This would be appropriate when the dissimilarities are based on subjective rankings. For example if the objects were foods the a number of judges rank the foods for different qualities such as taste and texture the resulting distances would not necessarily obey the metric inequality but the rank order would be significant. Alternatively, by relaxing the requirement from matching distances to rank order equivalence only, the number of dimensions required to represent the distance matrix may be decreased. The requirement of rank-order equivalence leads to non-metric or ordinal multidimensional scaling. The criterion used to measure the closeness of the fitted distance matrix to the observed distance matrix is known as STRESS which is given by

$$\sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{i-1}(\hat{d}_{ij} - \tilde{d}_{ij})^2}{\sum_{i=1}^{n}\sum_{j=1}^{i-1}\hat{d}_{ij}^2}},$$

where $\hat{d}_{ij}^2$ is the Euclidean squared distance between the computed points i and j and $\tilde{d}_{ij}$ is the fitted distance obtained when $\hat{d}_{ij}$ is monotonically regressed on the observed distances d_{ij}, that is, $\tilde{d}_{ij}$ is monotonic relative to d_{ij} and is obtained from $\hat{d}_{ij}$ with the smallest number of changes. Thus STRESS is a measure of by how much the set of points preserve the order of the distances in the original distance matrix and non-metric multidimensional scaling seeks to find the set of points that minimize the STRESS.

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 following routines perform the computations for variable-directed methods.

G03AAF computes the principal components from an input data matrix. Results include tests on the eigenvalues, the principal component loadings, and the principal component scores.

G03ACF computes a canonical variate analysis from an input data matrix. Results include canonical correlations, tests on eigenvalues, canonical variate means, and canonical variate loadings.

G03ADF computes a canonical correlation analysis from a input data matrix. Results include tests on the eigenvalues and canonical variates loadings.

G03CAF computes maximum likelihood estimates of the parameters of the factor analysis model.

G03CCF computes the factor score coefficients from the results of G03CAF.

G03BAF computes orthogonal rotations, including varimax and equimax rotations.

G03BCF computes Procrustes rotations.

The following routines perform the computations for individual-directed methods.

Discriminant Analysis

G03DAF computes matrices for use in discriminant analysis and test statistics for use in testing the equality of within group variance-covariance matrices.

G03DBF computes Mahalanobis distances from the results of G03DAF.

G03DCF allocates observations to groups using allocation rules as described above. An atypicality index can also be computed. G03DCF uses the results of G03DAF.

Note also that G02GBF will fit a logistic regression model and can be used for logistic discrimination.

Cluster Analysis

G03EAF computes a distance matrix.

G03ECF computes heirarchical cluster analysis from a given distance matrix.

G03EHF computes a dendrogram from the results of G03ECF.

G03EJF computes a set of clusters from the results of G03ECF.

G03EFF computes non-heirarchical (K-means) cluster analysis.

Scaling Methods

G03FAF computes a principal co-ordinate analysis.

G03FCF computes non-metric multi-dimensional scaling.

The following service routine is also available:

G03ZAF computes a matrix of standardized variables from an input data matrix.

4 References

[1] Chatfield C and Collins A J (1980) *Introduction to Multivariate Analysis* Chapman and Hall

[2] Everitt B S (1974) *Cluster Analysis* Heinemann

[3] Gnanadesikan R (1977) *Methods for Statistical Data Analysis of Multivariate Observations* Wiley

[4] Hammarling S (1985) The singular value decomposition in multivariate statistics *SIGNUM Newsl.* **20 (3)** 2–25

[5] Kendall M G and Stuart A (1976) *The Advanced Theory of Statistics (Volume 3)* Griffin (3rd Edition)

[6] Krzanowski W J (1990) *Principles of Multivariate Analysis* Oxford University Press

[7] Lawley D N and Maxwell A E (1971) *Factor Analysis as a Statistical Method* Butterworths (2nd Edition)

[8] Morrison D F (1967) *Multivariate Statistical Methods* McGraw-Hill

Chapter G04 – Analysis of Variance

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G04AGF	8	Two-way analysis of variance, hierarchical classification, subgroups of unequal size
G04BBF	16	Analysis of variance, randomized block or completely randomized design, treatment means and standard errors
G04BCF	17	Analysis of variance, general row and column design, treatment means and standard errors
G04CAF	16	Analysis of variance, complete factorial design, treatment means and standard errors
G04DAF	17	Computes sum of squares for contrast between means
G04DBF	17	Computes confidence intervals for differences between means computed by G04BBF or G04BCF
G04EAF	17	Computes orthogonal polynomials or dummy variables for factor/classification variable

Chapter G04

Analysis of Variance

Contents

1 Scope of the Chapter

This chapter is concerned with methods for analysing the results of designed experiments. The range of experiments covered includes:

(1) single factor designs with equal sized blocks such as randomised complete block and balanced incomplete block designs,
(2) row and column designs such as Latin squares, and
(3) complete factorial designs.

Further designs may be analysed by combining the analyses provided by multiple calls to routines or by using general linear model routines provided in Chapter G02.

2 Background to the Problems

2.1 Experimental Designs

An experimental design consists of a plan for allocating a set of controlled conditions, the treatments, to subsets of the experimental material, the plots or units. Two examples are:

(a) In an experiment to examine the effects of different diets on the growth of chickens, the chickens were kept in pens and a different diet was fed to the birds in each pen. In this example the pens are the units and the different diets are the treatments.

(b) In an experiment to compare four materials for ware-loss, a sample from each of the materials is tested in a machine that simulates ware. The machine can take four samples at a time and a number of runs are made. In this experiment the treatments are the materials and the units are the samples from the materials.

In designing an experiment the following principles are important.

(1) Randomisation: Given the overall plan of the experiment, the final allocation of treatments to units is performed using a suitable random allocation. This avoids the possibility of a systematic bias in the allocation and gives a basis for the statistical analysis of the experiment.

(2) Replication: Each treatment should be 'observed' more than once. So in example (b) more than one sample from each material should be tested. Replication allows for an estimate of the variablity of the treatment effect to be measured.

(3) Blocking: In many situations the experimental material will not be homogeneous and there may be some form of systematic variation in the experimental material. In order to reduce the effect of systematic variation the material can be grouped into blocks so that units within a block are similar but there is variation between blocks. For example, in an animal experiment litters may be considered as blocks; in an industrial experiment it may be material from one production batch.

(4) Factorial designs: If more than one type of treatment is under consideration, for example the effect of changes in temperature and changes in pressure, a factorial design consists of looking at all combinations of temperature and pressure. The different types of treatment are known as factors and the different values of the factors that are considered in the experiment are known as levels. So if three temperatures and four different pressures were being considered, then factor 1 (temperature) would have 3 levels and factor 2 (pressure) would have four levels and the design would be a 3×4 factorial giving a total of 12 treatment combinations. This design has the advantage of being able to detect the interaction between factors, that is, the effect of the combination of factors.

The following are examples of standard experimental designs; in the descriptions, it is assumed that there are t treatments.

(1) Completely Randomised Design: There are no blocks and the treatments are allocated to units at random.

(2) Randomised Complete Block Design: The experimental units are grouped into b blocks of t units and each treatment occurs once in each block. The treatments are allocated to units within blocks at random.

(3) Latin Square Designs: The units can be represented as cells of a $t \times t$ square classified by rows and columns. The t rows and t columns represent sources of variation in the experimental material. The design allocates the treatments to the units so that each treatment occurs once in each row and each column.

(4) Balanced Incomplete Block Designs: The experimental units are grouped into b blocks of $k < t$ units. The treatments are allocated so that each treatment is replicated the same number of times and each treatment occurs in the same block with any other treatment the same number of times. The treatments are allocated to units within blocks at random.

(5) Complete Factorial Experiments: If there are t treatment combinations derived from the levels of all factors then either there are no blocks or the blocks are of size t units.

Other designs include: partially balanced incomplete block designs, split-plot designs, factorial designs with confounding, and fractional factorial designs. For further information on these designs, see Cochran and Cox [1], Davies [2] or John and Quenouille [4].

2.2 Analysis of Variance

The analysis of a designed experiment usually consists of two stages. The first is the computation of the estimate of variance of the underlying random variation in the experiment along with tests for the overall effect of treatments. This results in an analysis of variance (ANOVA) table. The second stage is a more detailed examination of the effect of different treatments either by comparing the difference in treatment means with an appropriate standard error or by the use of orthogonal contrasts.

The analysis assumes a linear model such as:

$$y_{ij} = \mu + \delta_i + \tau_l + e_{ij}$$

where y_{ij} is the observed value for unit j of block i, μ is the overall mean, δ_i is the effect of the ith block, τ_l is the effect of the lth treatment which has been applied to the unit, and e_{ij} is the random error term associated with this unit. The expected value of e_{ij} is zero and its variance is σ^2.

In the analysis of variance, the total variation, measured by the sum of squares of observations about the overall mean, is partitioned into the sum of squares due to blocks, the sum of squares due to treatments, and a residual or error sum of squares. This partition corresponds to the parameters β, τ and σ. In parallel to the partition of the sum of squares there is a partition of the degrees of freedom associated with the sums of squares. The total degrees of freedom is $n-1$, where n is the number of observations. This is partitioned into $b-1$ degrees of freedom for blocks, $t-1$ degrees of freedom for treatments, and $n-t-b+1$ degrees of freedom for the residual sum of squares. From these the mean squares can be computed as the sums of squares divided by their degrees of freedom. The residual mean square is an estimate of σ^2. An F-test for an overall effect of the treatments can be calculated as the ratio of the treatment mean square to the residual mean square.

For row and column designs the model is:

$$y_{ij} = \mu + \rho_i + \gamma_j + \tau_l + e_{ij}$$

where ρ_i is the effect of the ith row and γ_j is the effect of the jth column. Usually the rows and columns are orthogonal. In the analysis of variance the total variation is partitioned into rows, columns treatments and residual.

In the case of factorial experiments, the treatment sum of squares and degrees of freedom may be partitioned into main effects for the factors and interactions between factors. The main effect of a factor is the effect of the factor averaged over all other factors. The interaction between two factors is the additional effect of the combination of the two factors, over and above the additive effects of the two factors, averaged over all other factors. For a factorial experiment in blocks with two factors, A and B, in which the jth unit of the ith block received level l of factor A and level k of factor B the model is:

$$y_{ij} = \mu + \delta_i + (\alpha_l + \beta_k + \alpha\beta_{lk}) + e_{ij}$$

where α_l is the main effect of level l of factor a, β_k is the main effect of level k of factor B, and $\alpha\beta_{lk}$ is the interaction between level l of A and level k of B. Higher-order interactions can be defined in a similar way.

Once the signifcant treatment effects have been uncovered they can be further investigated by comparing the differences between the means with the appropriate standard error. Some of the assumptions of the analysis can be checked by examining the residuals.

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 Chapter contains routines that can handle a wide range of experimental designs plus routines for further analysis and a routine to compute dummy variables for use in a general linear model.

G04BBF computes the analysis of variance and treatment means with standard errors for any block design with equal sized blocks. The routine will handle both complete block designs and balanced and partially balanced incomplete block designs.

G04BCF computes the analysis of variance and treatment means with standard errors for a row and column designs such as a Latin square.

G04CAF computes the analysis of variance and treatment means with standard errors for a complete factorial experiment.

Other designs can be analysed by combinations of calls to G04BBF, G04BCF and G04CAF. The routines compute the residuals from the model specified by the design, so these can then be input as the response variable in a second call to one of the routines. For example a factorial experiment in a Latin square design can be analysed by first calling G04BCF to remove the row and column effects and then calling G04CAF with the residuals from G04BCF as the response variable to compute the ANOVA for the treatments. Another example would be to use both G02DAF and G04BBF to compute an analysis of covariance.

It is also possible to analyse factorial experiments in which some effects have been confounded with blocks or some fractional factorial experiments. For examples see Morgan [6].

For experiments with missing values, these values can be estimated by using the Healy and Westmacott procedure, see John and Quenouille [4]. This procedure involves starting with initial estimates for the missing values and then making adjustments based on the residuals from the analysis. The improved estimates are then used in further iterations of the process.

For designs that cannot be analysed by the above approach the routine G04EAF can be used to compute dummy variables from the classification variables or factors that define the design. These dummy variables can then be used with the general linear model routine G02DAF.

As well as the routines considered above the routine G04AGF computes the analysis of variance for a two strata nested design.

In addition to the routines for computing the means and the basic analysis of variance two routines are available for further analysis.

G04DAF computes the sum of squares for a user defined contrast between means. For example, if there are four treatments, the first is a control and the other three are different amounts of a chemical the contrasts that are the difference between no chemical and chemical and the linear effect of chemical could be defined. G04DAF could be used to compute the sums of squares for these contrasts from which the appropriate F-tests could be computed.

G04DBF computes simultaneous confidence intervals for the differences between means with the choice of different methods such as the Tukey–Kramer, Bonferron and Dunn–Sidak.

4 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'.

G04ADF G04AEF G04AFF

5 References

[1] Cochran W G and Cox G M (1957) *Experimental Designs* Wiley

[2] Davis O L (1978) *The Design and Analysis of Industrial Experiments* Longman

[3] John J A (1987) *Cyclic Designs* Chapman and Hall

[4] John J A and Quenouille M H (1977) *Experiments: Design and Analysis* Griffin

[5] Searle S R (1971) *Linear Models* Wiley

[6] Morgan G W (1993) Analysis of variance using the NAG Fortran Library: Examples from Cochran and Cox *NAG Technical Report TR 3/93* NAG Ltd, Oxford

---

Chapter G05 – Random Number Generators

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G05CAF	6	Pseudo-random real numbers, uniform distribution over (0,1)
G05CBF	6	Initialise random number generating routines to give repeatable sequence
G05CCF	6	Initialise random number generating routines to give non-repeatable sequence
G05CFF	6	Save state of random number generating routines
G05CGF	6	Restore state of random number generating routines
G05DAF	6	Pseudo-random real numbers, uniform distribution over (a, b)
G05DBF	6	Pseudo-random real numbers, (negative) exponential distribution
G05DCF	6	Pseudo-random real numbers, logistic distribution
G05DDF	6	Pseudo-random real numbers, Normal distribution
G05DEF	6	Pseudo-random real numbers, log-normal distribution
G05DFF	6	Pseudo-random real numbers, Cauchy distribution
G05DHF	6	Pseudo-random real numbers, χ^2 distribution
G05DJF	6	Pseudo-random real numbers, Student's t-distribution
G05DKF	6	Pseudo-random real numbers, F-distribution
G05DPF	8	Pseudo-random real numbers, Weibull distribution
G05DRF	15	Pseudo-random integer, Poisson distribution
G05DYF	6	Pseudo-random integer from uniform distribution
G05DZF	6	Pseudo-random logical (boolean) value
G05EAF	10	Set up reference vector for multivariate Normal distribution
G05EBF	6	Set up reference vector for generating pseudo-random integers, uniform distribution
G05ECF	6	Set up reference vector for generating pseudo-random integers, Poisson distribution
G05EDF	6	Set up reference vector for generating pseudo-random integers, binomial distribution
G05EEF	6	Set up reference vector for generating pseudo-random integers, negative binomial distribution
G05EFF	6	Set up reference vector for generating pseudo-random integers, hypergeometric distribution
G05EGF	8	Set up reference vector for univariate ARMA time series model
G05EHF	10	Pseudo-random permutation of an integer vector
G05EJF	10	Pseudo-random sample from an integer vector
G05EWF	8	Generate next term from reference vector for ARMA time series model
G05EXF	6	Set up reference vector from supplied cumulative distribution function or probability distribution function
G05EYF	6	Pseudo-random integer from reference vector
G05EZF	10	Pseudo-random multivariate Normal vector from reference vector
G05FAF	14	Generates a vector of random numbers from a uniform distribution
G05FBF	14	Generates a vector of random numbers from an (negative) exponential distribution
G05FDF	14	Generates a vector of random numbers from a Normal distribution
G05FEF	15	Generates a vector of pseudo-random numbers from a beta distribution
G05FFF	15	Generates a vector of pseudo-random numbers from a gamma distribution
G05FSF	16	Generates a vector of pseudo-random variates from von Mises distribution
G05GAF	16	Computes random orthogonal matrix
G05GBF	16	Computes random correlation matrix

G05HDF	15	Generates a realisation of a multivariate time series from a VARMA model

Chapter G05

Random Number Generators

Contents

1 Scope of the Chapter

This chapter is concerned with the generation of sequences of independent pseudo-random numbers from various distributions, and the generation of pseudo-random time series from specified time-series models.

2 Background to the Problems

A sequence of pseudo-random numbers is a sequence of numbers generated in some systematic way such that its statistical properties are as close as possible to those of true random numbers: for example, negligible correlation between consecutive numbers. The most common method used is a **multiplicative congruential** algorithm defined as:

$$n_i = (a \times n_{i-1}) \bmod m \tag{1}$$

The integers n_i are then divided by m to give uniformly distributed random numbers lying in the interval (0,1).

The NAG generator uses the values $a = 13^{13}$ and $m = 2^{59}$; for further details see G05CAF. This generator gives a **cycle length** (i.e., the number of random numbers before the sequence starts repeating itself) of 2^{57}. A good rule of thumb is never to use more numbers than the square root of the cycle length in any one experiment as the statistical properties are impaired. For closely related reasons, breaking numbers down into their bit patterns and using individual bits may cause trouble.

The sequence given in (1) needs an initial value n_0, known as the **seed**. The use of the same seed will lead to the same sequence of numbers. One method of obtaining the seed is to use the real-time clock; this will give a non-repeatable sequence. It is important to note that the statistical properties of the random numbers are only guaranteed within sequences and not between sequences. Repeated initialization will thus render the numbers obtained less rather than more independent.

Random numbers from other distributions may be obtained from the uniform random numbers by the use of transformations, rejection techniques, and for discrete distributions table based methods.

(a) Transformation methods

For a continuous random variable, if the cumulative distribution function (CDF) is $F(x)$ then for a uniform (0,1) random variate u, $y = F^{-1}(u)$ will have CDF $F(x)$. This method is only efficient in a few simple cases such as the exponential distribution with mean μ, in which case $F^{-1}(u) = -\mu \log u$. Other transformations are based on the joint distribution of several random variables. In the bivariate case, if v and w are random variates there may be a function g such that $y = g(v,w)$ has the required distribution; for example, the Student's t-distribution with n degrees of freedom in which v has a Normal distribution, w has a gamma distibution and $g(v,w) = v\sqrt{n/w}$.

(b) Rejection methods

Rejection techniques are based on the ability to easily generate random numbers from a distribution (called the envelope) similar to the distribution required. The value from the envelope distribution is then accepted as a random number from the required distribution with a certain probability; otherwise, it is rejected and a new number is generated from the envelope distribution.

(c) Table search methods

For discrete distributions, if the cumulative probabilities, $P_i = \text{Prob}(x \leq i)$, are stored in a table then, given u from a uniform (0,1) distribution, the table is searched for i such that $P_{i-1} < u \leq P_i$. The returned value i will have the required distribution. The table searching can be made faster by means of an index, see Ripley [4]. The effort required to set up the table and its index may be considerable, but the methods are very efficient when many values are needed from the same distribution.

In addition to random numbers from various distributions, random compound structures can be generated. These include random time series, random matrices and random samples.

The efficiency of a simulation exercise may often be increased by the use of variance reduction methods (see Morgan [3]). It is also worth considering whether a simulation is the best approach to solving the problem. For example, low-dimensional integrals are usually more efficiently calculated by routines in Chapter D01 rather than by Monte Carlo integration.

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 Design of the Chapter

All the generation routines call - directly or indirectly - an internal basic generator, which generates random numbers from a uniform distribution over (0,1). Thus a call to any generation routine will affect all subsequent random numbers produced by any other routine in the chapter. Despite this effect, the values will remain as independent as if the different sequences were produced separately.

Two utility routines are provided to initialize the basic generator:

G05CBF initializes it to a repeatable state, dependent on an integer parameter: two calls of G05CBF with the same parameter-value will result in the same subsequent sequences of random numbers.

G05CCF initializes it to a non-repeatable state, in such a way that different calls of G05CCF, either in the same run or different runs of the program, will almost certainly result in different subsequent sequences of random numbers.

As mentioned in Section 2, it is important to note that the statistical properties of pseudo-random numbers are only guaranteed within sequences and not between sequences. Repeated initialization will thus render the numbers obtained less rather than more independent. In a simple case there should be only one call to G05CBF or G05CCF, which should be before any call to an actual generation routine.

Two other utility routines, G05CFF and G05CGF, are provided to save or restore the state of the basic generator (including the seed of the multiplicative congruential method used by the basic generator). G05CFF and G05CGF can be used to produce two or more sequences of numbers simultaneously, where some are repeatable and some are not; for example, this can be used to simulate signal and noise. As their overheads are not negligible, numbers should be produced in batches when this technique is used. While they can be used to save the state of the internal generator between jobs, the two arrays must be restored accurately. The corresponding process between machines, while sometimes possible, is not advised.

3.2 Selection of Routine

For three of the commonest continuous distributions - uniform, exponential, and Normal - there is a choice between calling a function to return a single random number and calling a subroutine to fill an array with a sequence of random numbers; the latter is likely to be much more efficient on vector-processing machines.

Distribution	Function returning a single number	Subroutine returning an array of numbers
uniform over (0,1)	G05CAF	G05FAF
uniform over (a,b)	G05DAF	G05FAF
exponential	G05DBF	G05FBF
Normal	G05DDF	G05FDF

For two discrete distributions, the uniform and Poisson, there is a choice between routines that use indexed search tables, which are suitable for the generation of many variates from the distribution with the same parameters, and routines that are more efficient in the single call situation when the parameters may be changing.

Distribution	Single call	Set up table
discrete uniform	G05DYF	G05EBF
Poisson	G05DRF	G05ECF

G05EBF and G05ECF return a reference array which is then used by G05EYF.

The following distributions are also available. Those indicated can return more than one value per call.

(a) Continuous Distributions

Beta distribution (multiple)	G05FEF
Cauchy distribution	G05DFF
Chi-square distribution	G05DHF
F-distribution	G05DKF
Gamma distribution (multiple)	G05FFF
Logistic distribution	G05DCF
Lognormal distribution	G05DEF
Student's t-distribution	G05DJF
von Mises distribution	G05FSF
Weibull distribution	G05DPF

(b) Multivariate Distributions

Multivariate Normal distribution	G05EAF and G05EZF

(c) Discrete Distributions using table search

Binomial distribution	G05EDF
Hypergeometric distribution	G05EFF
Negative binomial distribution	G05EEF
User-supplied distribution	G05EXF

The above routines set up the table and index in a reference array; G05EYF can then be called to generate the random variate from the information in the reference array.

(d) Generation of Time Series

Univariate ARMA model, Normal errors	G05EGF and G05EWF
Vector ARMA model, Normal errors	G05HDF

(e) Sampling and Permutation

Random permutation of an integer vector	G05EHF
Random sample from an integer vector	G05EJF
Random logical value	G05DZF

(f) Random Matrices

Random orthogonal matrix	G05GAF
Random correlation matrix	G05GBF

3.3 Programming Advice

Take care when programming calls to those routines in this chapter which are functions. The reason is that different calls with the same parameters are intended to give different results.

For example, if you wish to assign to Z the difference between two successive random numbers generated by G05CAF, beware of writing

```
Z = G05CAF(X) - G05CAF(X)
```

It is quite legitimate for a Fortran compiler to compile zero, one or two calls to G05CAF; if two calls, they may be in either order (if zero or one calls are compiled, Z would be set to zero). A safe method to program this would be

```
X = G05CAF(X)
Y = G05CAF(Y)
Z = X-Y
```

Another problem that can occur is that an optimising compiler may move a call to a function out of a loop. Thus, the same value would be used for all iterations of the loop, instead of a different random number being generated at each iteration. If this problem occurs, consult an expert on your Fortran compiler.

All the routines in this chapter rely on information stored in common blocks, which must be saved between calls. This need not be a matter of concern unless a program is split into overlays; in such a case, the safest course is to ensure that the G05 routines are in the root overlay.

4 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have either been withdrawn or superseded. Those routines indicated by a dagger are still present at Mark 18, but will be omitted at a future date. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

G05DGF G05DLF G05DMF

5 References

[1] Dagpunar J (1988) *Principles of Random Variate Generation* Oxford University Press

[2] Knuth D E (1981) *The Art of Computer Programming (Volume 2)* Addison-Wesley (2nd Edition)

[3] Morgan B J T (1984) *Elements of Simulation* Chapman and Hall

[4] Ripley B D (1987) *Stochastic Simulation* Wiley

Chapter G07 – Univariate Estimation

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G07AAF	15	Computes confidence interval for the parameter of a binomial distribution
G07ABF	15	Computes confidence interval for the parameter of a Poisson distribution
G07BBF	15	Computes maximum likelihood estimates for parameters of the Normal distribution from grouped and/or censored data
G07BEF	15	Computes maximum likelihood estimates for parameters of the Weibull distribution
G07CAF	15	Computes t-test statistic for a difference in means between two Normal populations, confidence interval
G07DAF	13	Robust estimation, median, median absolute deviation, robust standard deviation
G07DBF	13	Robust estimation, M-estimates for location and scale parameters, standard weight functions
G07DCF	13	Robust estimation, M-estimates for location and scale parameters, user-defined weight functions
G07DDF	14	Computes a trimmed and winsorized mean of a single sample with estimates of their variance
G07EAF	16	Robust confidence intervals, one-sample
G07EBF	16	Robust confidence intervals, two-sample

Chapter G07

Univariate Estimation

Contents

1 Scope of the Chapter

This chapter deals with the estimation of unknown parameters of a univariate distribution. It includes both point and interval estimation using maximum likelihood and robust methods.

2 Background to the Problems

Statistical inference is concerned with the making of inferences about a **population** using the observed part of the population called a **sample**. The population can usually be described using a probability model which will be written in terms of some unknown **parameters**. For example, the hours of relief given by a drug may be assumed to follow a Normal distribution with mean μ and variance σ^2; it is then required to make inferences about the parameters, μ and σ^2, on the basis of an observed sample of relief times.

There are two main aspects of statistical inference: the **estimation** of the parameters and the **testing of hypotheses** about the parameters. In the example above, the values of the parameter σ^2 may be estimated and the hypothesis that $\mu \geq 3$ tested. This chapter is mainly concerned with estimation but the test of a hypothesis about a parameter is often closely linked to its estimation. Tests of hypotheses which are not linked closely to estimation are given in the chapter on non-parametric statistics (Chapter G08).

There are two types of estimation to be considered in this chapter: **point estimation** and **interval estimation**. Point estimation is when a single value is obtained as the best estimate of the parameter. However, as this estimate will be based on only one of a large number of possible samples, it can be seen that if a different sample were taken, a different estimate would be obtained. The distribution of the estimate across all the possible samples is known as the **sampling distribution**. The sampling distribution contains information on the performance of the estimator, and enables estimators to be compared. For example, a good estimator would have a sampling distribution with mean equal to the true value of the parameter; that is, it should be an **unbiased** estimator; also the variance of the sampling distribution should be as small as possible. When considering a parameter estimate it is important to consider its variability as measured by its variance, or more often the square root of the variance, the **standard error**.

The sampling distribution can be used to find interval estimates or confidence intervals for the parameter. A **confidence interval** is an interval calculated from the sample so that its distribution, as given by the sampling distribution, is such that it contains the true value of the parameter with a certain probability.

Estimates will be functions of the observed sample and these functions are known as **estimators**. It is usually more convenient for the estimator to be based on statistics from the sample rather than all the individuals observations. If these statistics contain all the relevant information then they are known as **sufficient statistics**. There are several ways of obtaining the estimators; these include least-squares, the method of moments, and **maximum likelihood**. Least-squares estimation requires no knowledge of the distributional form of the error apart from its mean and variance matrix, whereas the method of maximum likelihood is mainly applicable to situations in which the true distribution is known apart from the values of a finite number of unknown parameters. Note that under the assumption of Normality, the least-squares estimation is equivalent to the maximum likelihood estimation. Least squares is often used in regression analysis as described in Chapter G02, and maximum likelihood is described below.

Estimators derived from least-squares or maximum likelihood will often be greatly affected by the presence of extreme or unusual observations. Estimators that are designed to be less affected are known as **robust estimators**.

2.1 Maximum Likelihood Estimation

Let X_i be a univariate random variable with probability density function

$$f_{X_i}(x_i;\theta),$$

where θ is a vector of length p consisting of the unknown parameters. For example, a Normal distribution with mean θ_1 and standard deviation θ_2 has probability density function

$$\frac{1}{\sqrt{2\pi}\theta_2}\exp\left(-\frac{1}{2}\left(\frac{x_i-\theta_1}{\theta_2}\right)^2\right).$$

The likelihood for a sample of n independent observations is

$$\text{Like} = \prod_{i=1}^{n} f_{X_i}(x_i;\theta),$$

where x_i is the observed value of X_i. If each X_i has an identical distribution, this reduces to

$$\text{Like} = \prod_{i=1}^{n} f_X(x_i;\theta), \tag{1}$$

and the log-likelihood is

$$\log(\text{Like}) = L = \sum_{i=1}^{n} \log(f_X(x_i;\theta)). \tag{2}$$

The maximum likelihood estimates $(\hat{\theta})$ of θ are the values of θ that maximize (1) and (2). If the range of X is independent of the parameters, then $\hat{\theta}$ can usually be found as the solution to

$$\sum_{i=1}^{n} \frac{\partial}{\partial \hat{\theta}_j} \log(f_X(x_i;\hat{\theta})) = \frac{\partial L}{\partial \hat{\theta}_j} = 0, \quad j = 1, 2, \ldots, p. \tag{3}$$

Note that $\frac{\partial L}{\partial \theta_j}$ is known as the efficient score.

Maximum likelihood estimators possess several important properties.

(a) Maximum likelihood estimators are functions of the sufficient statistics.

(b) Maximum likelihood estimators are (under certain conditions) **consistent**. That is, the estimator converges in probability to the true value as the sample size increases. Note that for small samples the maximum likelihood estimator may be biased.

(c) For maximum likelihood estimators found as a solution to (3), subject to certain conditions, it follows that

$$E\left(\frac{\partial L}{\partial \theta}\right) = 0, \tag{4}$$

and

$$I(\theta) = -E\left(\frac{\partial^2 L}{\partial \theta^2}\right) = E\left(\left(\frac{\partial L}{\partial \theta}\right)^2\right), \tag{5}$$

and then that $\hat{\theta}$ is asymptotically Normal with mean vector θ_0 and variance-covariance matrix $I_{\theta_0}^{-1}$ where θ_0 denotes the true value of θ. The matrix I_θ is known as the information matrix and $I_{\theta_0}^{-1}$ is known as the Cramer–Rao lower bound for the variance of an estimator of θ.

For example, if we consider a sample, $x_1, x_2, \ldots, x_n$, of size n drawn from a Normal distribution with unknown mean μ and unknown variance σ_2 then we have

$$L = \log(\text{Like}(\mu,\sigma^2;x)) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \sum_{i=1}^{n}(x_i-\mu)^2/2\sigma^2$$

and thus

$$\frac{\partial L}{\partial \mu} = \sum_{i=1}^{n}(x_i-\mu)/\sigma^2$$

and

$$\frac{\partial L}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \sum_{i=1}^{n}(x_i-\mu)^2/2\sigma^4.$$

Then equating these two equations to zero and solving gives the maximum likelihood estimates

$$\hat{\mu} = \bar{x}$$

and

$$\hat{\sigma}^2 = \sum_{i=1}^{n}(x_i - \bar{x})^2/n.$$

These maximum likelihood estimates are asymptotically Normal with mean vector a, where

$$a^T = (\mu, \sigma^2),$$

and covariance matrix C. To obtain C we find the second derivatives of L with respect to μ and σ^2 as follows:

$$\frac{\partial^2 L}{\partial \mu^2} = -\frac{n}{\sigma^2}$$

$$\frac{\partial^2 L}{\partial (\sigma^2)^2} = \frac{n}{2\sigma^4} - \sum_{i=1}^{n}(x_i - \mu)^2/\sigma^6$$

$$\frac{\partial^2 L}{\partial \mu \partial \sigma^2} = \frac{\partial^2 L}{\partial \sigma^2 \partial \mu} = -\frac{n(\bar{x} - \mu)}{\sigma^4}.$$

Then

$$C^{-1} = -E\begin{pmatrix} \frac{\partial^2 L}{\partial \mu^2} & \frac{\partial^2 L}{\partial \sigma^2 \partial \mu} \\ \frac{\partial^2 L}{\partial \mu \partial \sigma^2} & \frac{\partial^2 L}{\partial (\sigma^2)^2} \end{pmatrix} = \begin{pmatrix} n/\sigma^2 & 0 \\ 0 & n/2\sigma^4 \end{pmatrix}$$

so that

$$C = \begin{pmatrix} \sigma^2/n & 0 \\ 0 & 2\sigma^4/n \end{pmatrix}.$$

To obtain an estimate of C the matrix may be evaluated at the maximum likelihood estimates.

It may not always be possible to find maximum likelihood estimates in a convenient closed form, and in these cases iterative numerical methods, such as the Newton–Raphson procedure or the EM algorithm (expectation maximisation), will be neccessary to compute the maximum likelihood estimates. Their asymptotic variances and covariances may then be found by substituting the estimates into the second derivatives. Note that it may be difficult to find the expected value of the second derivatives required for the variance-covariance matrix and in these cases the observed value of the second derivatives is often used.

The use of maximum likelihood estimation allows the construction of generalized likelihood ratio tests. If $\lambda = 2(l_1 - l_2)$ where l_1 is the maximised log-likelihood function for a model 1 and l_2 is the maximised log-likelihood function for a model 2, then under the hypothesis that model 2 is correct, 2λ is asymptotically distributed as a χ^2 variable with $p - q$ degrees of freedom. Consider two models in which model 1 has p parameters and model 2 is a sub-model (nested model) of model 1 with $q < p$ paramters, that is model 1 has an extra $p - q$ parameters. This result provides a useful method for performing hypothesis tests on the parameters. Alternatively, tests exist based on the asymptotic Normality of the estimator and the efficient score; see Cox and Hinkley [1], page 315.

2.2 Confidence Intervals

Suppose we can find a function, $t(x, \theta)$, whose distribution depends upon the sample x but not on the unknown parameter θ, and which is a monotonic (say decreasing) function in θ for each x, then we can find t_1 such that $P(t_1 \leq t(x, \theta)) = 1 - \alpha$ no matter what θ happens to be. The function $t(x, \theta)$ is known as a pivotal quantity. Since the function is monotonic the statement that $t_1 \leq t(x, \theta)$ may be rewritten as $\theta \geq \theta_1(x)$, see Figure 1. The statistic $\theta_1(x)$ will vary from sample to sample and if we assert that $\theta \geq \theta_1(x)$ for any sample values which arise, we will be right in a proportion $1 - \alpha$ of the cases, in the long run or on average. We call $\theta_1(x)$ a $1 - \alpha$ upper confidence limit for θ.

We have considered only an upper confidence limit. The above idea may be generalised to a two-sided confidence interval where two quantities, t_0 and t_1, are found such that for all θ, $\mathrm{P}(t_1 \leq t(x, \theta) \leq t_0) = 1 - \alpha$. This interval may be rewritten as $\theta_0(x) \leq \theta \leq \theta_1(x)$. Thus if we assert that θ lies in the interval $[\theta_0(x), \theta_1(x)]$ we will be right on average in $1 - \alpha$ proportion of the times under repeated sampling.

Hypothesis (significance) tests on the parameters may be used to find these confidence limits. For example, if we observe a value, k, from a binomial distribution, with known parameter n and unknown parameter

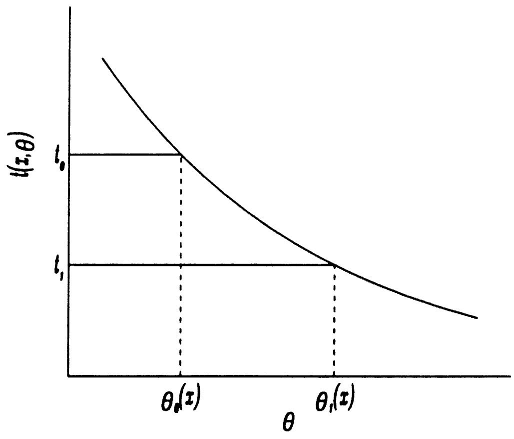


Figure 1

p, then to find the lower confidence limit we find p_l such that the probability that the null hypothesis H_0: $p = p_l$ (against the one sided alternative that $p > p_l$) will be rejected, is less than or equal to $\alpha/2$. Thus for a binomial random variable, B, with parameters n and p_l we require that $P(B \geq k) \leq \alpha/2$. The upper confidence limit, p_u, can be constructed in a similar way.

For large samples the asymptotic Normality of the maximum likelihood estimates discussed above is used to construct confidence intervals for the unknown parameters.

2.3 Robust Estimation

For particular cases the probability density function can be written as

$$f_{X_i}(x_i;\theta) = \frac{1}{\theta_2} g\left(\frac{x_i - \theta_1}{\theta_2}\right),$$

for a suitable function g; then θ_1 is known as a location parameter and θ_2, usually written as σ, is known as a scale parameter. This is true of the Normal distribution.

If θ_1 is a location parameter, as described above, then equation (3) becomes

$$\sum_{i=1}^{n} \psi\left(\frac{x_i - \hat{\theta}_1}{\hat{\sigma}}\right) = 0, \qquad (6)$$

where $\psi(z) = -\frac{d}{dz}\log(g(z))$.

For the scale parameter σ (or σ^2) the equation is

$$\sum_{i=1}^{n} \chi\left(\frac{x_i - \hat{\theta}_1}{\hat{\sigma}}\right) = n/2, \qquad (7)$$

where $\chi(z) = z\psi(z)/2$.

For the Normal distribution $\psi(z) = z$ and $\chi(z) = z^2/2$. Thus, the maximum likelihood estimates for θ_1 and σ^2 are the sample mean and variance with the n divisor respectively. As the latter is biased, (7) can be replaced by

$$\sum_{i=1}^{n} \chi\left(\frac{x_i - \hat{\theta}_1}{\hat{\sigma}}\right) = (n-1)\beta, \qquad (8)$$

where β is a suitable constant, which for the Normal χ function is $\frac{1}{2}$.

The influence of an observation on the estimates depends on the form of the ψ and χ functions. For a discussion of influence, see Hampel *et al.* [2] and Huber [3]. The influence of extreme values can be reduced by bounding the values of the ψ- and χ-functions. One suggestion due to Huber [3] is

$$\psi(z) = \begin{cases} -C, & z < -C \\ z, & |z| \leq C \\ C, & z > C. \end{cases}$$

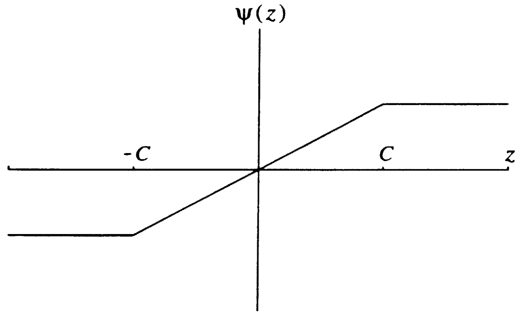


Figure 2

Redescending ψ-functions are often considered; these give zero values to $\psi(z)$ for large positive or negative values of z. Hampel [2] suggested

$$\psi(z) = \begin{cases} -\psi(-z) & \\ z, & 0 \leq z \leq h_1 \\ h_1, & h_1 \leq z \leq h_2 \\ h_1(h_3 - z)/(h_3 - h_2), & h_2 \leq z \leq h_3 \\ 0, & z > h_3. \end{cases}$$

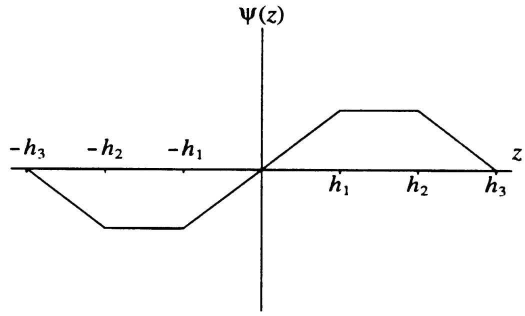


Figure 3

Usually a χ-function based on Huber's ψ-function is used: $\chi = \psi^2/2$. Estimators based on such bounded ψ-functions are known as M-estimators, and provide one type of robust estimator.

Other robust estimators for the location parameter are:

(i) the sample median,
(ii) the trimmed mean, i.e., the mean calculated after the extreme values have been removed from the sample,
(iii) the winsorized mean, i.e., the mean calculated after the extreme values of the sample have been replaced by other more moderate values from the sample.

For the scale parameter, alternative estimators are:

(i) the median absolute deviation scaled to produce an estimator which is unbiased in the case of data coming from a Normal distribution,
(ii) the winsorized variance, i.e., the variance calculated after the extreme values of the sample have been replaced by other more moderate values from the sample.

For a general discussion of robust estimation, see Hampel *et al.* [2] and Huber [3].

2.4 Robust Confidence Intervals

In Section 2.2 it was shown how tests of hypotheses can be used to find confidence intervals. That approach uses a parametric test that requires the assumption that the data used in the computation of the confidence has a known distribution. As an alternative, a more robust confidence interval can be found by replacing the parametric test by a non-parametric test. In the case of the confidence interval for the location parameter, a Wilcoxon test statistic can be used, and for the difference in location, computed from two samples, a Mann–Whitney test statistic can be used.

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.

Maximum likelihood estimation and confidence intervals.

G07AAF provides a confidence interval for the parameter p of the binomial distribution.

G07ABF provides a confidence interval for the mean parameter of the Poisson distribution.

G07BBF provides maximum likelihood estimates and their standard errors for the parameters of the Normal distribution from grouped and/or censored data.

G07BEF provides maximum likelihood estimates and their standard errors for the parameters of the Weibull distribution from data which may be right-censored.

G07CAF provides a t-test statistic to test for a difference in means between two Normal populations, together with a confidence interval for the difference between the means.

Robust estimation.

G07DBF provides M-estimates for location and, optionally, scale using four common forms of the ψ-function.

G07DCF produces the M-estimates for location and, optionally, scale but for user-supplied ψ- and χ-functions.

G07DAF provides the sample median, median absolute deviation, and the scaled value of the median absolute deviation.

G07DDF provides the trimmed mean and winsorized mean together with estimates of their variance based on a winsorized variance.

Robust Internal Estimation.

G07EAF produces a rank based confidence interval for locations.

G07EBF produces a rank based confidence interval for the difference in location between two populations.

4 References

[1] Cox D R and Hinkley D V (1974) *Theoretical Statistics* Chapman and Hall

[2] Hampel F R, Ronchetti E M, Rousseeuw P J and Stahel W A (1986) *Robust Statistics. The Approach Based on Influence Functions* Wiley

[3] Huber P J (1981) *Robust Statistics* Wiley

[4] Kendall M G and Stuart A (1973) *The Advanced Theory of Statistics (Volume 2)* Griffin (3rd Edition)

[5] Silvey S D (1975) *Statistical Inference* Chapman and Hall

Chapter G08 – Nonparametric Statistics

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G08AAF	8	Sign test on two paired samples
G08ACF	8	Median test on two samples of unequal size
G08AEF	8	Friedman two-way analysis of variance on k matched samples
G08AFF	8	Kruskal–Wallis one-way analysis of variance on k samples of unequal size
G08AGF	14	Performs the Wilcoxon one-sample (matched pairs) signed rank test
G08AHF	14	Performs the Mann–Whitney U test on two independent samples
G08AJF	14	Computes the exact probabilities for the Mann–Whitney U statistic, no ties in pooled sample
G08AKF	14	Computes the exact probabilities for the Mann–Whitney U statistic, ties in pooled sample
G08ALF	15	Performs the Cochran Q test on cross-classified binary data
G08BAF	8	Mood's and David's tests on two samples of unequal size
G08CBF	14	Performs the one-sample Kolmogorov–Smirnov test for standard distributions
G08CCF	14	Performs the one-sample Kolmogorov–Smirnov test for a user-supplied distribution
G08CDF	14	Performs the two-sample Kolmogorov–Smirnov test
G08CGF	14	Performs the χ^2 goodness of fit test, for standard continuous distributions
G08DAF	8	Kendall's coefficient of concordance
G08EAF	14	Performs the runs up or runs down test for randomness
G08EBF	14	Performs the pairs (serial) test for randomness
G08ECF	14	Performs the triplets test for randomness
G08EDF	14	Performs the gaps test for randomness
G08RAF	12	Regression using ranks, uncensored data
G08RBF	12	Regression using ranks, right-censored data

Chapter G08

Nonparametric Statistics

Contents

1 Scope of the Chapter

The routines in this chapter perform nonparametric statistical tests which are based on distribution-free methods of analysis. For convenience, the chapter contents are divided into five types of test: tests of location, tests of dispersion, tests of distribution, tests of association and correlation, and tests of randomness. There are also routines to fit linear regression models using the ranks of the observations.

The emphasis in this chapter is on testing; users wishing to compute nonparametric correlations are referred to Chapter G02, which contains several routines for that purpose.

There are a large number of nonparametric tests available. A selection of some of the more commonly used tests are included in this chapter.

2 Background to the Problems

2.1 Parametric and Nonparametric Hypothesis Testing

Classical techniques of statistical inference often make numerous or stringent assumptions about the nature of the population or populations from which the observations have been drawn. For instance, a testing procedure might assume that the set of data was obtained from Normally distributed populations. It might be further assumed that the populations involved have equal variances, or that there is a known relationship between the variances. In the Normal case, the test statistic derived would usually be a function of the sample means and variances, since a Normal distribution is completely characterised by its mean and variance. Alternatively, it might be assumed that the set of data was obtained from other distributions of known form, such as the gamma or the exponential. Again, a testing procedure would be devised based upon the parameters characterising such a distribution.

The type of hypothesis testing just described is usually termed **parametric** inference. Distributional assumptions are made which imply that the parameters of the chosen distribution, as estimated from the data, are sufficient to characterise the difference in distribution between the populations.

However, problems arise with parametric methods of inference when these assumptions cannot be made, either because they are contrary to the known nature of the mechanism generating a population, or because the data obviously do not satisfy the assumptions. Some parametric procedures become unreliable under relatively minor departures from the hypothesised distributional form. In the Normal case for example, tests on variances are extremely sensitive to departures from Normality in the underlying distribution.

There are also common situations, particularly in the behavioural sciences, where much more basic assumptions than that of Normality cannot be made. Data values are not always measured on continuous or even numerical scales. They may be simply categorical in nature, relating to such quantities as voting intentions or food preferences.

Techniques of inference are therefore required which do not involve making detailed assumptions about the underlying mechanism generating the observations. The routines in this chapter perform such distribution-free tests, evaluating from a set of data the value of a test statistic, together with an estimate of its significance.

For a comparison of some distribution-based and distribution-free tests, the interested reader is referred to Chapter 31 of Kendall and Stuart [2]. For a briefer and less mathematical account, see Conover [1] or Siegel [3].

2.2 Types of Nonparametric Test

This introduction is concerned with explaining the basic concepts of hypothesis testing, and some familiarity with the subject is assumed. Chapter 22 of Kendall and Stuart [2] contains a detailed account, and the outline given in Conover [1] or Siegel [3] should be sufficient to understand this section.

Nonparametric tests may be grouped into five categories:

(1) Tests of location
(2) Tests of dispersion
(3) Distribution-free tests of fit
(4) Tests of association or correlation

(5) Tests of randomness

Tests can also be categorised by the design that they can be applied to:

(1) One sample
(2) Two related (paired) samples
(3) Two independent samples
(4) k (> 2) related (matched) samples
(5) k (> 2) independent samples

A third classification of a test relates to the type of data to which it may be applied. Variables are recorded on four scales of measurement: nominal (categorical), ordinal, interval, and ratio.

The nominal scale is used only to categorise data; for each category a name, perhaps numeric, is assigned so that two different categories will be identified by distinct names. The ordinal scale, as well as categorising the observations, orders the categories. Each is assigned a distinct identifying symbol, in such a way that the order of the symbols corresponds to the order of the categories. (The most common system for ordinal variables is to assign numerical identifiers to the categories, though if they have previously been assigned alphabetic characters, these may be transformed to a numerical system by any convenient method which preserves the ordering of the categories.) The interval scale not only categorises and orders the observations, but also quantifies the comparison between categories; this necessitates a common unit of measurement and an arbitrary zero-point. Finally, the ratio scale is similar to the interval scale, except that it has an absolute (as opposed to arbitrary) zero-point.

It is apparent that there are many possible combinations of these three characteristics of a problem, and many nonparametric tests have been derived to meet the different experimental situations. However, it is not usually a difficult matter to choose an appropriate test given the nature of the data and the type of test which one wishes to perform.

2.3 Principles of Nonparametric Tests

In this section, each type of test is considered in turn, and remarks are made on the design principles on which each is based.

2.3.1 Location tests

These tests are primarily concerned with inferences about differences in the location of the population distributions. In some cases however, the tests are only concerned with inferences about the population distributions unless added assumptions are made which allow the hypotheses to be stated in terms of the location parameters.

For most of these tests, data must be measured numerically on at least an ordinal scale, in order that a measure of location may be devised. Ordinal measurement implies that pairs of values may be compared and numerically ordered. A vector of n values may therefore be **ranked** from smallest to largest using the ordering operation. The resultant **ranks** contain all the information in the original data, but have the advantage that tests may be derived easily based on them, and no testing bias is introduced by the use of ordinal values as though they were measured on an interval scale. Note that the requirement of the measurement scale being ordinal does not imply that all tests of this type involve the actual ranking of the original data.

For the one-sample or matched pairs case, test statistics may be derived based on the number of observations (or differences) lying either side of zero (or some other fixed value), as in the sign test for example. Under the hypothesis that the median of the single population is zero or the difference in the medians of the paired populations is zero the number of positive and negative values should be similar. The Wilcoxon signed rank test goes further than the sign test by taking into account the magnitude of the single sample values or of the differences.

For the two-sample case, if median equality is hypothesised, the distribution of the ranks of each sample in the total pooled sample should be similar. Test statistics, such as the Mann–Whitney U statistic, which are based on the ranks of each sample and summarise the differences in rank sums for each sample, may be computed. These statistics are referred to their expected distributions under the null hypothesis. The above hypothesis can also be tested using the median test. Its test statistic is based on the number of values in each sample which are greater than or less than the pooled median of the two samples, rather than the ranks of each sample.

If median equality is hypothesised for several samples, the distribution ranks of the members of each sample in the total pooled sample should be 'homogeneous'. Test statistics can be derived which summarise the differences in rank sums for the various samples, and again referred to their expected distributions under the null hypothesis.

2.3.2 Dispersion tests

These provide a distribution-free alternative to such tests as the F-(variance-ratio) test for variance equality, which is very sensitive to non-Normality in the generating distribution.

The dispersions of two or more samples may be compared by pooling the samples and observing the distribution of ranks in the ranked pooled sample. Equal dispersions should be recognisable by there being a wide distribution of the extreme ranks between the members of different samples. Statistics are evaluated which quantify the dispersion of ranks between samples, and their significance may be found by evaluating their permutation distributions assuming that no dispersion difference exists.

2.3.3 Tests of fit

In the one-sample case, these are tests which investigate whether or not a sample of observations can be considered to follow a specified distribution. In the two-sample case, a test of fit investigates whether the two samples can be considered to have arisen from a common probability distribution.

For the one sample problem, the null hypothesis may specify only the distributional form, for example Normal(μ,σ^2), or it may incorporate actual parameter values, for example, Poisson with mean 10.

Some tests of this type proceed by forming the sample cumulative distribution function of the observations and computing a statistic whose value measures the departure of the sample cumulative distribution function from that of the null distribution. In the two-sample case, a statistic is computed which provides a measure of the difference between the sample cumulative distribution function of each sample. These tests are known as one- or two-sample Kolmogorov–Smirnov tests.

The significance for these test statistics can be computed directly for moderate sample sizes but for larger sample sizes asymptotic results are often used.

Another goodness of fit test is the χ^2 test. For this test, the data is first grouped into intervals and then the difference between the observed number of observations in each interval and the number expected, if the null hypothesis is true, is computed. A statistic based on these differences has asymptotically a χ^2-distribution.

2.3.4 Association and correlation tests

These are distribution-free analogues of tests based on such statistics as Pearson product-moment correlation coefficients.

Essentially they are based on rankings rather than the observed data values, and involve summing some function of the rank differences between the samples to obtain an overall measure of the concordance of ranks. This measure can be standardised by dividing by its theoretical maximum value for the given sample size and number of samples.

Significance levels may be calculated for quite small sample sizes by using an approximation to a χ^2-distribution.

2.3.5 Tests of randomness

These tests are designed to investigate sequences of observations and attempt to identify any deviations from randomness. There are clearly many ways in which a sequence may deviate from randomness. The tests provided here primarily detect some form of dependency between the observations in the sequence.

The most common application of this type of tests is in the area of random number generation. The tests are used as empirical tests on a sample of output from a generator to establish local randomness. Theoretical tests are necessary and useful for testing global randomness. Some of the more common empirical tests are discussed below.

A runs-up or runs-down test investigates whether runs of different lengths are occuring with greater or lesser frequency than would be expected under the null hypothesis of randomness. A run up is defined as a sequence of observations in which each observation is larger than the previous observation. The run up ends when an observation is smaller than the previous observation. A test statistic, modified to take into account the dependency between successive run lengths, is computed. The test statistic has an asymptotic χ^2-distribution.

The pairs test investigates the condition that, under the null hypothesis of randomness, the non-overlapping 2-tuples (pairs) of a sequence of observations from the interval [0,1] should be uniformly distributed over the unit square ($[0,1]^2$). The triplets test follows the same idea but considers 3-tuples and checks for uniformity over the unit cube ($[0,1]^3$). In each test, a test statistic, based on differences between the observed and expected distribution of the 2- or 3-tuples, is computed which has an asymptotic χ^2-distribution.

The gaps test considers the 'gaps' between successive occurences of observations in the sequence lying in a specified range. Under the null hypothesis of randomness, the gap length should follow a geometric distribution with a parameter based on the length of the specified range, relative to the overall length of the interval containing all possible observations. The expected number of 'gaps', of a certain length, under the null hypothesis may thus be computed together with a test statistic based on the differences between the observed and expected numbers of 'gaps' of different length. Again the test statistic has an asymptotic χ^2-distribution.

Other empirical tests such as the χ^2 goodness of fit test and the one-sample Kolmogorov–Smirnov test may be used to investigate a sequence for non-uniformity.

2.4 Regression using ranks

If the user wishes to fit a regression model but is unsure about what transformation to take for the observed response to obtain a linear model, then one strategy is to replace response observations by their ranks. Estimates for regression parameters can be found by maximizing a likelihood function based on the ranks and the proposed regression model. The present routines give approximate estimates which are adequate when the signal-to-noise ratio is small, which is often the case with data from the medical and social sciences. Approximate standard errors of estimated regression coefficients are found. Also χ^2 statistics can be used to test the null hypothesis of no regression effect.

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 are grouped into six categories. The fourth character of the routine name is used to denote this categorisation.

Sub-chapter	**Type of test**
G08A	Location
G08B	Dispersion
G08C	Fit
G08D	Correlation and Association
G08E	Randomness
G08R	Regression using Ranks

3.1 G08A – Location Tests

3.1.1 One sample or matched-pairs case

Note that a random sample of matched pairs, (x_i,y_i), may be reduced to a single sample by considering the differences, $d_i = x_i - y_i$ say, of each pair. The matched pair may be thought of as a single observation on a bivariate random variable.

G08AAF performs the sign test on two paired samples. Each pair is classified as a + or − depending on the sign of the difference between the two data values within the pair. Under the assumptions that the d_i are mutually independent and that the observations are measured on at least an ordinal scale, the sign test tests the hypothesis that for any pair sampled from the population distribution, Probabilty(+) = Probability(−). The hypothesis may be stated in terms of the equality of the location parameters but the test is no longer regarded as unbiased and consistent unless further assumptions are made. If the user wishes to test the hypothesis that the location parameters differ by a fixed amount then that amount must be added or subtracted from one of the samples as required before calling G08AAF.

G08AGF performs the one-sample Wilcoxon signed-rank test. The test may be used to test if the median of the population from which the random sample was taken is equal to some specified value (commonly used to test if the median is zero). In this test not only is the sign of the difference between the data values and the hypothesised median value important but also the magnitude of this difference. Thus, where the magnitude of the differences (or the data values themselves if the hypothesised median value is zero) is important this test is preferred to the sign test because it is more powerful. The test may easily be used to test whether the medians of two related populations are equal by taking the differences between the paired sample values and then testing the hypothesis that the median of the differences is zero, using the single sample of differences. The significance of the test statistic may be computed exactly for a moderate sample size but for a larger sample a Normal approximation is used. The exact method allows for ties in the differences.

3.1.2 Two independent samples

G08ACF performs the median test and G08AHF performs the Mann–Whitney U test.

For both tests the two samples are assumed to be random samples from their respective populations and mutually independent. The measurement scale must be at least ordinal.

Note that, although the median test may be generalised to more than two samples, G08ACF only deals with the two-sample case. For the median test, each observation is classified as being above or below the pooled median of the two samples. It may be used to test the hypothesis that the two population medians are equal; under the assumption that if the two population medians are equal then the probability of an observation exceeding the pooled median is the same for both populations.

The Mann–Whitney U test involves the ranking of the pooled sample. The Mann–Whitney test thus attaches importance to the position of each observation relative to the others and not just its position relative to the median of the pooled sample as in the median test. Thus when the magnitude of the differences between the observations is meaningful the Mann–Whitney U test is preferred as it is more powerful than the median test. The test tests whether the two population distributions are the same or not. If it is assumed that any difference between the the two population distributions is a difference in the location then the test is testing whether the population means are the same or not.

In G08AHF, the significance of the U test statistic is computed using a Normal approximation. If the exact significance is desired then either G08AJF or G08AKF must be used. G08AJF computes the exact significance of the U test statistic for the case where there are no ties in the pooled sample. It requires only the value of the statistic and the two sample sizes. G08AKF computes the exact significance of the U test statistic for the case where there are ties in the pooled sample. It requires the value of the statistic and the two sample sizes and the ranks of the observations of the two samples as provided by G08AHF. G08AHF returns an indicator to inform the user whether or not ties were found in the pooled sample.

3.1.3 More than two related samples

G08AEF performs the Friedman two-way analysis of variance. This test may in some ways be regarded as an extension of the sign test to the case of k, $(k > 2)$, related samples. The data is in the form of a number of multivariate observations which are assumed to be mutually independent. This test also assumes that the measurement within each observation across the k variates is at least ordinal so that the observation for each variate may be ranked according to some criteria.

For data which may be defined as either a zero or one, that is binary response data, G08ALF performs Cochran's Q-test to examine differences between the treatments within blocks.

3.1.4 More than two independent samples

G08AFF performs the Kruskal–Wallis one-way analysis of variance. The test assumes that each sample is a random sample from its respective distributions and in addition that there is both independence within the samples and mutual independence among the various samples. The test requires that the measurement scale is at least ordinal so that the pooled sample may be ranked.

3.2 G08B – Dispersion Tests

G08BAF performs either Mood's or David's test for dispersion differences, or both, for two independent samples of possibly unequal size.

For both tests the null hypothesis is that the two samples have equal dispersions, the routine returning a probability value which may be used to perform the test against a one-sided or two-sided alternative, in a way described in the routine document.

3.3 G08C – Tests of Fit

G08CBF and G08CCF both perform the one sample Kolmogorov–Smirnov distribution test. This test is used to test the null hypothesis that the random sample arises from a specified null distribution against one of three possible alternatives.

With G08CBF the user may choose a null distribution from one of the following: the uniform, Normal, gamma, beta, binomial, exponential, and Poisson. The parameter values may either be specified by the user or estimated from the data by the routine. With G08CCF the user must provide a function which will compute the value of the cumulative distribution function at any specified point for the null distribution. The alternative hypotheses available correspond to one- and two-sided tests. The distribution of the test statistic is computed using an exact method for a moderate sample size. For a larger sample size an asymptotic result is used.

G08CDF performs the two-sample Kolmogorov–Smirnov test which tests the null hypothesis that the two samples may be considered to have arisen from the same population distribution against one of three possible alternative hypotheses, again corresponding to one-sided and two-sided tests. The distribution of the test statistic is computed using an exact method for moderate sample sizes but for larger samples, approximations based on asymptotic results are used.

Note that G01EYF and G01EZF are available for computing the distributions of the one-sample and two-sample Kolmogorov–Smirnov statistics respectively.

G08CGF performs the χ^2 goodness of fit test on a single sample which again tests the null hypothesis that the sample arises from a specified null distribution. The user may choose a null distribution from one of the following: the Normal, uniform, exponential, χ^2, and gamma, or may define the distribution by specifying the probability that an observation lies in a certain interval for a range of intervals covering the support of the null distibution. The significance of this test is computed using the χ^2 distribution as an approximation to the distribution of the test statistic.

Tests of Normality may also be carried out using routines in Chapter G01.

3.4 G08D – Association and Correlation Tests

G08DAF computes Kendall's coefficient of concordance on k independent ranks of n objects. An example of its application would be to compare for consistency the results of a group of IQ tests performed on the same set of people. Allowance is made for tied rankings, and the approximate significance of the computed coefficient is found.

3.5 G08E – Tests of Randomness

G08EAF performs the runs-up test on a sequence of observations. The runs-down test may be performed by multiplying each observation by -1 before calling the routine. All runs whose length is greater than or equal to a certain chosen length will be treated as a single group.

G08EBF performs the pairs (serial) test on a sequence of observations from the interval [0,1]. The number of equal sub-intervals into which the interval [0,1] is to be divided must be specified.

G08ECF performs the triplets test on a sequence of observations from the interval [0,1]. The number of equal sub-intervals into which the interval [0,1] is to be divided must be specified.

G08EDF performs the gaps test on a sequence of observations. The total of the interval containing all possible values the observations could take must be specified together with the interval being used to define the 'gaps'. All 'gaps' whose length is greater than or equal to a certain chosen length will be treated as a single group.

3.6 G08R – Regression Using Ranks

G08RAF fits a multiple linear regression model in which the observations on the response variable are replaced by their ranks.

G08RBF performs the same function but takes into account observations which may be right-censored.

3.7 Related Routines

Tests of location and distribution may be based on scores which are estimates of the expected values of the order statistics. G01DHF may be used to compute Normal scores, an approximation to the Normal scores (Blom, Tukey or van der Waerden scores) or Savage (exponential) scores. For more accurate Normal scores G01DAF may be used. Other routines in subchapter G01D may be used to test for Normality.

4 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'.

G08ABF G08ADF G08CAF

5 References

[1] Conover W J (1980) *Practical Nonparametric Statistics* Wiley

[2] Kendall M G and Stuart A (1973) *The Advanced Theory of Statistics (Volume 2)* Griffin (3rd Edition)

[3] Siegel S (1956) *Non-parametric Statistics for the Behavioral Sciences* McGraw-Hill

Chapter G10 – Smoothing in Statistics

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G10ABF	16	Fit cubic smoothing spline, smoothing parameter given
G10ACF	16	Fit cubic smoothing spline, smoothing parameter estimated
G10BAF	16	Kernel density estimate using Gaussian kernel
G10CAF	16	Compute smoothed data sequence using running median smoothers
G10ZAF	16	Reorder data to give ordered distinct observations

Chapter G10

Smoothing in Statistics

Contents

1 Scope of the Chapter

This chapter is concerned with methods for smoothing data. Included are methods for density estimation, smoothing time series data, and statistical applications of splines. These methods may also be viewed as nonparametric modelling.

2 Background to the Problems

2.1 Smoothing Methods

Many of the methods used in statistics involve fitting a model, the form of which is determined by a small number of parameters, for example, a distribution model like the gamma distribution, a linear regression model or an autoregression model in time series. In these cases the fitting involves the estimation of the small number of parameters from the data. In modelling data with these models there are two important stages in addition to the estimation of the parameters; these are: the identification of a suitable model, for example, the selection of a gamma distribution rather than a Weibull distribution, and the checking to see if the fitted model adequately fits the data. While these parametric models can be fairly flexible, they will not adequately fit all data sets, especially if the number of parameters is to be kept small.

Alternative models based on smoothing can be used. These models will not be written explicitly in terms of parameters. They are sufficiently flexible for a much wider range of situations than parametric models. The main requirement for such a model to be suitable is that the underlying models would be expected to be smooth, so excluding those situations where, for example, a step function would be expected.

These smoothing methods can be used in a variety of ways, for example:

(1) producing smoothed plots to aid understanding;
(2) identifying of a suitable parametric model from the shape of the smoothed data;
(3) eliminating complex effects that are not of direct interest so that attention can be focused on the effects of interest.

Several smoothing techniques make use of a smoothing parameter which can be either chosen by the user or estimated from the data. The smoothing parameter balances the two criterion of smoothness of the fitted model and the closeness of the fit of the model to the data. Generally, the larger the smoothing parameter is, the smoother the fitted model will be, but for small values of the smoothing parameter the model will closely follow the data, and for large values the fit will be poorer.

The smoothing parameter can either be chosen using previous experience of a suitable value for such data, or estimated from the data. The estimation can be either formal, using a criterion such as the cross-validation, or informal by trying different values and examining the result by means of suitable graphs.

Smoothing methods can be used in three important areas of of statistics: regression modelling, distribution modelling and time series modelling.

2.2 Smoothing Splines and Regression Models

For a set of n observations (y_i,x_i), $i = 1,2,\ldots,n$, the spline provides a flexible smooth function for situations in which a simple polynomial or nonlinear regression model is not suitable.

Cubic smoothing splines arise as the function, f, with continuous first derivative which minimises

$$\sum_{i=1}^{n} w_i\{y_i - f(x_i)\}^2 + \rho \int_{-\infty}^{\infty} (f''(x))^2 dx,$$

where w_i is the (optional) weight for the ith observation and ρ is the smoothing parameter. This criterion consists of two parts: the first measures the fit of the curve and the second the smoothness of the curve. The value of the smoothing parameter, ρ, weights these two aspects: larger values of ρ give a smoother fitted curve but, in general, a poorer fit.

Splines are linear smoothers since the fitted values, $\hat{y} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n)^T$, can be written as a linear function of the observed values $y = (y_1, y_2, \ldots, y_n)^T$, that is,

$$\hat{y} = Hy$$

for a matrix H. The degrees of freedom for the spline is trace(H) giving residual degrees of freedom

$$\mathrm{trace}(I-H)=\sum_{i=1}^{n}(1-h_{ii}).$$

The diagonal elements of H, h_{ii}, are the leverages.

The parameter ρ can be estimated in a number of ways.

(1) The degrees of freedom for the spline can be specified, i.e., find ρ such that trace(H) $=\nu_0$ for given ν_0.

(2) Minimise the cross-validation (CV), i.e., find ρ such that the CV is mimimised, where

$$\mathrm{CV}=\frac{1}{n}\sum_{i=1}^{n}\left(\frac{r_i}{1-h_{ii}}\right)^2.$$

(3) Minimise generalised cross-validation (GCV), i.e., find ρ such that the GCV is minimised, where

$$\mathrm{GCV}=n\left(\frac{\sum_{i=1}^{n}r_i^2}{\left(\sum_{i=1}^{n}(1-h_{ii})\right)^2}\right).$$

2.3 Density Estimation

The object of density estimation is to produce from a set of observations a smooth nonparametric estimate of the unknown density function from which the observations were drawn. That is, given a sample of n observations, $x_1, x_2, \ldots, x_n$, from a distribution with unknown density function, $f(x)$, find an estimate of the density function, $\hat{f}(x)$. The simplest form of density estimator is the histogram; this may be defined by:

$$\hat{f}(x)=\frac{1}{nh}n_j;\quad a+(j-1)h<x<a+jh;\quad j=1,2,\ldots,n_s,$$

where n_j is the number of observations falling in the interval $a+(j-1)h$ to $a+jh$, a is the lower bound of the histogram and $b=n_sh$ is the upper bound. The value h is known as the window width. A simple development of this estimator would be the running histogram estimator

$$\hat{f}(x)=\frac{1}{2nh}n_x;\quad a\leq x\leq b,$$

where n_x is the number of observations falling in the interval $[x-h:x+h]$. This estimator can be written as

$$\hat{f}(x)=\frac{1}{nh}\sum_{i=1}^{n}w\left(\frac{x-x_i}{h}\right)$$

for a function w where

$$\begin{aligned}w(x)&=\tfrac{1}{2}\quad \text{if } -1<x<1\\&=0\quad \text{otherwise.}\end{aligned}$$

The function w can be considered as a kernel function. To produce a smoother density estimate, the kernel function, $K(t)$, which satisfies the following conditions can be used:

$$\int_{-\infty}^{\infty}K(t)dt=1 \text{ and } K(t)\geq 0.0.$$

The kernel density estimator is therefore defined as:

$$\hat{f}(x)=\frac{1}{nh}\sum_{i=1}^{n}K\left(\frac{x-x_i}{h}\right).$$

The choice of $K(\cdot)$ is usually not important but to ease computational burden, use can be made of Gausssian kernel defined as:

$$K(t)=\frac{1}{\sqrt{2\pi}}e^{-t^2/2}.$$

The smoothness of the estimator, $\hat{f}(x)$, depends on the window width, h. In general, the larger the value h is, the smoother the resulting density estimate is. There is, however, the problem of oversmoothing when the value of h is too large and essential features of the distribution function are removed. For example, if the distribution was bimodal, a large value of h may result in a unimodal estimate. The value of h has to be chosen such that the essential shape of the distribution is retained while effects due only to the observed sample are smoothed out. The choice of h can be aided by looking at plots of the density estimate for different values of h, or by using cross-validation methods; see Silverman [2].

Silverman [2] shows how the Gaussian kernel density estimator can be computed using a fast Fourier transform (FFT).

2.4 Smoothers for Time Series

If the data consists of a sequence of n observations recorded at equally spaced intervals, usually a time series, several robust smoothers are available. The fitted curve is intended to be robust to any outlying observations in the sequence, hence the techniques employed primarily make use of medians rather than means. These ideas come from the field of exploratory data analysis (EDA); see Tukey [3] and Velleman and Hoaglin [4]. The smoothers are based on the use of running medians to summarize overlapping segments; these provide a simple but flexible curve.

In EDA terminology, the fitted curve and the residuals are called the smooth and the rough respectively, so that

$$\text{Data} = \text{Smooth} + \text{Rough}.$$

Using the notation of Tukey, one of the smoothers commonly used is 4253H, twice. This consists of a running median of 4, then 2, then 5, then 3. This is then followed by what is known as Hanning. Hanning is a running weighted mean, the weights being 1/4, 1/2 and 1/4. The result of this smoothing is then 'reroughed'. This involves computing residuals from the computed fit, applying the same smoother to the residuals and adding the result to the smooth of the first pass.

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 following routines fit smoothing splines:

G10ABF computes a cubic smoothing spline for a given value of the smoothing parameter. The results returned include the values of leverages and the coefficients of the cubic spline. Options allow only parts of the computation to be performed when the routine is used to estimate the value of the smoothing parameter or as when it is part of an iterative procedure such as that used in fitting generalized additive models; see Hastie and Tibshrani [1].

G10ACF estimates the value of the smoothing parameter using one of three criteria and fits the cubic smoothing spline using that value.

G10ABF and G10ACF require the x_i to be strictly increasing. If two or more observations have the same x_i-value then they should be replaced by a single observation with y_i equal to the (weighted) mean of the y-values and weight, w_i, equal to the sum of the weights. This operation can be performed by G10ZAF.

The following routine produces an estimate of the density function:

G10BAF computes a density estimate using a Normal kernel.

The following routine produces a smoothed estimate for a time series:

G10CAF computes a smoothed series using running median smoothers.

The following service routine is also available:

G10ZAF orders and weights the (x, y) input data to produce a data set strictly monotonic in x.

4 References

[1] Hastie T J and Tibshirani R J (1990) *Generalized Additive Models* Chapman and Hall

[2] Silverman B W (1990) *Density Estimation* Chapman and Hall

[3] Tukey J W (1977) *Exploratory Data Analysis* Addison-Wesley

[4] Velleman P F and Hoaglin D C (1981) *Applications, Basics, and Computing of Exploratory Data Analysis* Duxbury Press, Boston, MA

Chapter G11 – Contingency Table Analysis

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G11AAF	16	χ^2 statistics for two-way contingency table
G11BAF	17	Computes multiway table from set of classification factors using selected statistic
G11BBF	17	Computes multiway table from set of classification factors using given percentile/quantile
G11BCF	17	Computes marginal tables for multiway table computed by G11BAF or G11BBF
G11CAF	19	Returns parameter estimates for the conditional analysis of stratified data
G11SAF	12	Contingency table, latent variable model for binary data
G11SBF	12	Frequency count for G11SAF

Chapter G11

Contingency Table Analysis

Contents

1 Scope of the Chapter

The routines in this chapter are for the analysis of discrete multivariate data. One suite of routines computes tables while other routines are for the analysis of two-way contingency tables, conditional logistic models and one-factor analysis of binary data.

Routines in Chapter G02 may be used to fit generalized linear models to discrete data including binary data and contingency tables.

2 Background to the Problems

2.1 Discrete Data

Discrete variables can be defined as variables which take a limited range of values. Discrete data can be usefully categorized into three types.

Binary data. The variables can take one of two values, for example, yes or no. The data may be grouped, for example, the number of yes responses in ten questions.

Categorical data. The variables can take one of two or more values or levels, but the values are not considered to have any ordering, for example, the values may be red, green, blue or brown.

Ordered categorical data. This is similar to categorical data but an ordering can be placed on the levels, for example: poor, average or good.

Data containing discrete variables can be analysed by computing summaries and measures of association and by fitting models.

2.2 Tabulation

The basic summary for multivariate discrete data is the multidimensional table in which each dimension is specified by a discrete variable. If the cells of the table are the number of observations with the corresponding values of the discrete variables then it is a contingency table. The discrete variables that can be used to classify a table are known as factors. For example, the factor sex would have the levels male and female. These can be coded as 1 and 2 respectively. Given several factors a multi-way table can be constructed such that each cell of the table represents one level from each factor. For example, a sample of 120 observations with the two factors sex and habitat, habitat having three levels: inner-city, suburban and rural, would give the 2×3 contingency table:

Sex	**Habitat**		
	Inner-city	Suburban	Rural
Male	32	27	15
Female	21	19	6

If the sample also contains continuous variables such as age, the average for the observations in each cell could be computed,

Sex	**Habitat**		
	Inner-city	Suburban	Rural
Male	25.5	30.3	35.6
Female	23.2	29.1	30.4

or other summary statistics.

Given a table, the totals or means for rows, columns etc. may be required. Thus the above contingency table with marginal totals is:

Sex	**Habitat**			
	Inner-city	Suburban	Rural	Total
Male	32	27	15	74
Female	21	19	6	46
Total	53	46	21	120

Note that the marginal totals for columns is itself a 2×1 table. Also, other summary statistics could be used to produce the marginal tables such as means or medians. Having computed the marginal tables, the cells of the original table may be expressed in terms of the margins, for example, in the above table the cells could be expressed as percentages of the column totals.

2.3 Discrete Response Variables and Logistic Regression

A second important categorization in addition to that given in Section 2.1 is whether one of the discrete variables can be considered as a response variable or whether it is just the association between the discrete variables that is being considered.

If the response variable is binary, for example, success or failure, then a logistic or probit regression model can be used. The logistic regression model relates the logarithm of the odds-ratio to a linear model. So if p_i is the probability of success, the model relates $\log(p_i/(1-p_i))$ to the explanatory variables. If the responses are independent then these models are special cases of the generalized linear model with binomial errors. However, there are cases when the binomial model is not suitable. For example, in a case-control study a number of cases (successes) and number of controls (failures) is chosen for a number of sets of case-controls. In this situation a conditional logistic analysis is required.

Handling a categorical or ordered categorical response variable is more complex, for a discussion on the appropriate models see McCullagh and Nelder [7]. These models generally use a Poisson distribution.

Note that if the response variable is a continuous variable and it is only the explanatory variables that are discrete then the regression models described in Chapter G02 should be used.

2.4 Contingency Tables

If there is no response variable then to investigate the association between discrete variables a contingency table can be computed and a suitable test performed on the table. The simplest case is the two-way table formed when considering two discrete variables. For a data set of n observations classified by the two variables with r and c levels respectively, a two-way table of frequencies or counts with r rows and c columns can be computed.

$$\begin{array}{cccc|c} n_{11} & n_{12} & \cdots & n_{1c} & n_{1.} \\ n_{21} & n_{22} & \cdots & n_{2c} & n_{2.} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ n_{r1} & n_{r2} & \cdots & n_{rc} & n_{r.} \\ \hline n_{.1} & n_{.2} & \cdots & n_{.c} & n \end{array}$$

If p_{ij} is the probability of an observation in cell ij then the the model which assumes no association between the two variables is the model

$$p_{ij} = p_{i.}p_{.j}$$

where $p_{i.}$ is the marginal probability for the row variable and $p_{.j}$ is the marginal probability for the column variable, the marginal probability being the probability of observing a particular value of the variable ignoring all other variables. The appropriateness of this model can be assessed by two commonly used statistics:

the Pearson χ^2 statistic

$$\sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(n_{ij}-f_{ij})^2}{f_{ij}},$$

and the likelihood ratio test statistic

$$2\sum_{i=1}^{r}\sum_{j=1}^{c} n_{ij} \times \log(n_{ij}/f_{ij}).$$

The f_{ij} are the fitted values from the model; these values are the expected cell frequencies and are given by

$$f_{ij} = n\hat{p}_{ij} = n\hat{p}_{i.}\hat{p}_{.j} = n(n_{i.}/n)(n_{.j}/n) = n_{i.}n_{.j}/n.$$

Under the hypothesis of no association between the two classification variables, both these statistics have, approximately, a χ^2 distribution with $(c-1)(r-1)$ degrees of freedom. This distribution is arrived at under the assumption that the expected cell frequencies, f_{ij}, are not too small.

In the case of the 2×2 table, i.e., $c = 2$ and $r = 2$, the χ^2 approximation can be improved by using Yates's continuity correction factor. This decreases the absolute value of $(n_{ij} - f_{ij})$ by 1/2. For 2×2 tables with a small values of n the exact probabilities can be computed; this is known as Fisher's exact test.

An alternative approach, which can easily be generalized to more than two variables, is to use log-linear models. A log-linear model for two variables can be written as

$$\log(p_{ij}) = \log(p_{i.}) + \log(p_{.j}).$$

A model like this can be fitted as a generalized linear model with Poisson error with the cell counts, n_{ij}, as the response variable.

2.5 Latent Variable Models

Latent variable models play an important role in the analysis of multivariate data. They have arisen in response to practical needs in many sciences, especially in psychology, educational testing and other social sciences.

Large-scale statistical enquiries, such as social surveys, generate much more information than can be easily absorbed without condensation. Elementary statistical methods help to summarize the data by looking at individual variables or the relationship between a small number of variables. However, with many variables it may still be difficult to see any pattern of inter-relationships. Our ability to visualize relationships is limited to two or three dimensions putting us under strong pressure to reduce the dimensionality of the data and yet preserve as much of the structure as possible. The question is thus one of how to replace the many variables with which we start by a much smaller number, with as little loss of information as possible.

One approach to the problem is to set up a model in which the dependence between the observed variables is accounted for by one or more latent variables. Such a model links the large number of observable variables with a much smaller number of latent variables.

Factor analysis, as described in Chapter G03, is based on a linear model of this kind when the observed variables are continuous. Here we consider the case where the observed variables are binary (e.g., coded 0/1 or true/false) and where there is one latent variable. In educational testing this is known as latent trait analysis, but, more generally, as factor analysis of binary data.

A variety of methods and models have been proposed for this problem. The models used here are derived from the general approach of Bartholomew [1] and [2]. The user is referred to [1] for further information on the models and to [3] for details of the method and application.

3 Recommendations on Choice and Use of Available Routines

3.1 Tabulation

The following routines can be used to perform the tabulation of discrete data.

G11BAF computes a multidimensional table from a set of discrete variables or classification factors. The cells of the table may be counts or a summary statistic {total, mean, variance, largest or smallest} computed for an associated continuous variable. Alternatively, G11BAF will update an existing table with further data.

G11BBF computes a multidimensional table from a set of discrete variables or classification factor where the cells are the percentile or quantile for an associated variable. For example, G11BBF can be used to produce a table of medians.

G11BCF computes a marginal table from a table computed by G11BAF or G11BBF using a summary statistic {total, mean, median variance, largest or smallest}.

3.2 Analysis of Contingency Tables

G11AAF computes the Pearson and likelihood ratio χ^2 statistics for a two-way contingency table. For 2×2 tables Yates's correction factor is used and for small samples, $n \leq 40$, Fisher's exact test is used.

In addition, G02GCF can be used to fit a log-linear model to a contingency table.

3.3 Binary data

The following routines can be used to analyse binary data.

G11SAF fits a latent variable model to binary data. The frequency distribution of score patterns is required as input data. If the user's data is in the form of individual score patterns, then the service routine G11SBF may be used to calculate the frequency distribution.

G11CAF estimates the parameters for a conditional logistic model.

In addition, G02GBF fits generalized linear models to binary data.

4 Routines Withdrawn or Scheduled for Withdrawal

None since Mark 13.

5 References

[1] Bartholomew D J (1980) Factor analysis for categorical data (with Discussion) *J. Roy. Statist. Soc. Ser. B* **42** 293–321

[2] Bartholomew D J (1984) The foundations of factor analysis *Biometrika* **71** 221–232

[3] Bartholomew D J (1987) *Latent Variable Models and Factor Analysis* Griffin

[4] Everitt B S (1977) *The Analysis of Contingency Tables* Chapman and Hall

[5] Kendall M G and Stuart A (1969) *The Advanced Theory of Statistics (Volume 1)* Griffin (3rd Edition)

[6] Kendall M G and Stuart A (1973) *The Advanced Theory of Statistics (Volume 2)* Griffin (3rd Edition)

[7] McCullagh P and Nelder J A (1983) *Generalized Linear Models* Chapman and Hall

Chapter G12 – Survival Analysis

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G12AAF	15	Computes Kaplan–Meier (product-limit) estimates of survival probabilities
G12BAF	17	Fits Cox's proportional hazard model
G12ZAF	19	Creates the risk sets associated with the Cox proportional hazards model for fixed covariates

Chapter G12

Survival Analysis

Contents

1 Scope of the Chapter

This chapter is concerned with statistical techniques used in the analysis of survival/reliability/failure time data.

Other chapters contain routines which are also used to analyse this type of data. Chapter G02 contains generalized linear models, Chapter G07 contains routines to fit distribution models, and Chapter G08 contains rank based methods.

2 Background to the Problems

2.1 Introduction to Terminology

This chapter is concerned with the analysis on the time, t, to a single event. This type of analysis occurs commonly in two areas. In medical research it is known as survival analysis and is often the time from the start of treatment to the occurrence of a particular condition or of death. In engineering it is concerned with reliability and the analysis of failure times, that is how long a component can be used until it fails. In this chapter the time t will be referred to as the **failure time**.

Let the probability density function of the failure time be $f(t)$, then the **survivor function**, $S(t)$, which is the probability of surviving to at least time t, is given by

$$S(t) = \int_t^\infty f(\tau)d\tau = 1 - F(t)$$

where $F(t)$ is the cumulative density function. The **hazard function**, $\lambda(t)$, is the probability that failure occurs at time t given that the individual survived up to time t, and is given by

$$\lambda(t) = f(t)/S(t).$$

The **cumulative hazard** rate is defined as

$$\Lambda(t) = \int_0^t \lambda(\tau)d\tau,$$

hence $S(t) = e^{-\Lambda(t)}$.

It is common in survival analysis for some of the data to be **right censored**. That is, the exact failure time is not known, only that failure occurred after a known time. This may be due to the experiment being terminated before all the individuals have failed, or an individual being removed from the experiment for a reason not connected with effects being tested in the experiment. The presence of censored data leads to complications in the analysis.

2.2 Estimating the Survivor Function and Hazard Plotting

The most common estimate of the survivor function for censored data is the **Kaplan–Meier** or **product-limit** estimate,

$$\hat{S}(t) = \prod_{j=1}^{i}\left(\frac{n_j - d_j}{n_j}\right), \quad t_i \leq t < t_{i+1}$$

where d_j is the number of failures occurring at time t_j out of n_j surviving to t_j. This is a step function with steps at each failure time but not at censored times.

As $S(t) = e^{-\Lambda(t)}$ the cumulative hazard rate can be estimated by

$$\hat{\Lambda}(t) = -\log(\hat{S}(t)).$$

A plot of $\hat{\Lambda}(t)$ or $\log(\hat{\Lambda}(t))$ against t or $\log t$ is often useful in identifying a suitable parametric model for the survivor times. The following relationships can be used in the identification.

(a) Exponential distribution: $\Lambda(t) = \lambda t$.
(b) Weibull distribution: $\log(\Lambda(t)) = \log\lambda + \gamma\log t$.
(c) Gompertz distribution: $\log(\lambda(t)) = \log\lambda + \gamma t$.
(d) Extreme value (smallest) distribution: $\log(\Lambda(t)) = \lambda(t - \gamma)$.

2.3 Proportional Hazard Models

Often in the analysis of survival data the relationship between the hazard function and the a number of explanatory variables or covariates is modelled. The covariates may be, for example, group or treatment indicators or measures of the state of the individual at the start of the observational period. There are two types of covariate time independent covariates such as those described above which do not change value during the observational period and time dependent covariates. The latter can be classified as either external covariates, in which case they are not directly involved with the failure mechanism, or as internal covariates which are time dependent measurements taken on the individual.

The most common function relating the covariates to the hazard function is the proportional hazard function

$$\lambda(t,z) = \lambda_0(t)\exp(\beta^T z)$$

where $\lambda_0(t)$ is a baseline hazard function, z is a vector of covariates and β is a vector of unknown parameters. The assumption is that the covariates have a multiplicative effect on the hazard.

The form of $\lambda_0(t)$ can be one of the distributions considered above or a non-parametric function. In the case of the exponential, Weibull and extreme value distributions the proportional hazard model can be fitted to censored data using the method described by Aitkin and Clayton [1] which uses a generalized linear model with Poisson errors. Other possible models are the gamma distribution and the lognormal distribution.

2.4 Cox's Proportional Hazard Model

Rather than using a specified form for the hazard function, Cox [2] considered the case when $\lambda_0(t)$ was an unspecified function of time. To fit such a model assuming fixed covariates a marginal likelihood is used. For each of the times at which a failure occurred, t_i, the set of those who were still in the study is considered, this includes any that were censored at t_i. This set is known as the risk set for time t_i and denoted by $R(t_{(i)})$. Given the risk set the probability that out of all possible sets of d_i subjects that could have failed the actual observed d_i cases failed can be written as

$$\frac{\exp(s_i^T\beta)}{\sum\exp(z_l^T\beta)} \tag{1}$$

where s_i is the sum of the covariates of the d_i individuals observed to fail at $t_{(i)}$ and the summation is over all distinct sets of n_i individuals drawn from $R(t_{(i)})$. This leads to a complex likelihood. If there are no ties in failure times the likelihood reduces to

$$L = \prod_{i=1}^{n_d} \frac{\exp(z_i^T\beta)}{[\sum_{l\in R(t_{(i)})}\exp(z_l^T\beta)]} \tag{2}$$

where n_d is the number of distinct failure times. For cases where there are ties the following approximation, due to Peto [2], can be used:

$$L = \prod_{i=1}^{n_d} \frac{\exp(s_i^T\beta)}{[\sum_{l\in R(t_{(i)})}\exp(z_l^T\beta)]^{d_i}}. \tag{3}$$

Having fitted the model an estimate of the base-line survivor function (derived from $\lambda_0(t)$ and the residuals) can be computed to examine the suitability of the model, in particular the proportional hazard assumption.

3 Recommendations on Choice and Use of Available Routines

The following routines are available.

G12AAF computes Kaplan–Meier estimates of the survivor function and their standard deviations.

G12BAF fits the Cox proportional hazards model for fixed covariates.

G12ZAF creates the risk sets associated with the Cox proportional hazards model for fixed covariates.

The following routines from other chapters may also be useful in the analysis of survival data.

G01MBF the reciprocal of Mills' Ratio, that is the hazard rate for the Normal distribution.

G02GCF fits generalized linear model with Poisson errors (see Aitkin and Clayton [1]).

G02GDF fits generalized linear model with gamma errors.

G07BBF fits Normal distribution to censored data.

G07BEF fits Weibull distribution to censored data.

G08RBF fits linear model using likelihood based on ranks to censored data (see Kabfleisch and Prentice [4]).

G11CAF fits a conditional logistic model. When applied to the risk sets generated by G12ZAF the Cox proportional hazards model is fitted by exact marginal likelihood in the presence of tied observations.

4 Routines Withdrawn or Scheduled for Withdrawal

None since Mark 13.

5 References

[1] Aitkin M and Clayton D (1980) The fitting of exponential, Weibull and extreme value distributions to complex censored survival data using GLIM *Appl. Statist.* **29** 156–163

[2] Cox D R (1972) Regression models in life tables (with discussion) *J. Roy. Statist. Soc. Ser. B* **34** 187–220

[3] Gross A J and Clark V A (1975) *Survival Distributions: Reliability Applications in the Biomedical Sciences* Wiley

[4] Kalbfleisch J D and Prentice R L (1980) *The Statistical Analysis of Failure Time Data* Wiley

Chapter G13 – Time Series Analysis

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
G13AAF	9	Univariate time series, seasonal and non-seasonal differencing
G13ABF	9	Univariate time series, sample autocorrelation function
G13ACF	9	Univariate time series, partial autocorrelations from autocorrelations
G13ADF	9	Univariate time series, preliminary estimation, seasonal ARIMA model
G13AEF	9	Univariate time series, estimation, seasonal ARIMA model (comprehensive)
G13AFF	9	Univariate time series, estimation, seasonal ARIMA model (easy-to-use)
G13AGF	9	Univariate time series, update state set for forecasting
G13AHF	9	Univariate time series, forecasting from state set
G13AJF	10	Univariate time series, state set and forecasts, from fully specified seasonal ARIMA model
G13ASF	13	Univariate time series, diagnostic checking of residuals, following G13AEF or G13AFF
G13AUF	14	Computes quantities needed for range-mean or standard deviation-mean plot
G13BAF	10	Multivariate time series, filtering (pre-whitening) by an ARIMA model
G13BBF	11	Multivariate time series, filtering by a transfer function model
G13BCF	10	Multivariate time series, cross-correlations
G13BDF	11	Multivariate time series, preliminary estimation of transfer function model
G13BEF	11	Multivariate time series, estimation of multi-input model
G13BGF	11	Multivariate time series, update state set for forecasting from multi-input model
G13BHF	11	Multivariate time series, forecasting from state set of multi-input model
G13BJF	11	Multivariate time series, state set and forecasts from fully specified multi-input model
G13CAF	10	Univariate time series, smoothed sample spectrum using rectangular, Bartlett, Tukey or Parzen lag window
G13CBF	10	Univariate time series, smoothed sample spectrum using spectral smoothing by the trapezium frequency (Daniell) window
G13CCF	10	Multivariate time series, smoothed sample cross spectrum using rectangular, Bartlett, Tukey or Parzen lag window
G13CDF	10	Multivariate time series, smoothed sample cross spectrum using spectral smoothing by the trapezium frequency (Daniell) window
G13CEF	10	Multivariate time series, cross amplitude spectrum, squared coherency, bounds, univariate and bivariate (cross) spectra
G13CFF	10	Multivariate time series, gain, phase, bounds, univariate and bivariate (cross) spectra
G13CGF	10	Multivariate time series, noise spectrum, bounds, impulse response function and its standard error
G13DBF	11	Multivariate time series, multiple squared partial autocorrelations
G13DCF	12	Multivariate time series, estimation of VARMA model
G13DJF	15	Multivariate time series, forecasts and their standard errors
G13DKF	15	Multivariate time series, updates forecasts and their standard errors
G13DLF	15	Multivariate time series, differences and/or transforms (for use before G13DCF)
G13DMF	15	Multivariate time series, sample cross-correlation or cross-covariance matrices
G13DNF	15	Multivariate time series, sample partial lag correlation matrices, χ^2 statistics and significance levels

G13DPF	16	Multivariate time series, partial autoregression matrices
G13DSF	13	Multivariate time series, diagnostic checking of residuals, following G13DCF
G13DXF	15	Calculates the zeros of a vector autoregressive (or moving average) operator
G13EAF	17	Combined measurement and time update, one iteration of Kalman filter, time-varying, square root covariance filter
G13EBF	17	Combined measurement and time update, one iteration of Kalman filter, time-invariant, square root covariance filter

Chapter G13

Time Series Analysis

Contents

1 Scope of the Chapter

This chapter provides facilities for investigating and modelling the statistical structure of series of observations collected at equally spaced points in time. The models may then be used to forecast the series.

The chapter is divided into methods for

(a) univariate analysis, where consideration is given to the structure within a single series, and
(b) multivariate analysis in which the interdependence of two or more series may be investigated. This includes models with a single output series depending on a number of input series as well as multivariate models with each series considered on an equal footing.

For both univariate and multivariate methods, there is a further division into

(i) time domain methods, which model relations between series values separated by various time lags, and
(ii) frequency domain or spectral methods which interpret time series in terms of component sine waves of various frequencies.

2 Background to the Problems

2.1 Univariate Time Domain Analysis

Let the given time series be $x_1, x_2, \ldots, x_n$ where n is its length. The structure which is intended to be investigated, and which may be most evident to the eye in a graph of the series, can be broadly described as

(a) trends – linear or possibly higher-order polynomial;
(b) seasonal patterns, associated with fixed integer seasonal periods. The presence of such seasonality and the period will normally be known *a priori*. The pattern may be fixed, or slowly varying from one season to another;
(c) cycles, or waves of stable amplitude and period p (from peak to peak). The period is not necessarily integer, the corresponding absolute frequency (cycles/time unit) being $f = 1/p$ and angular frequency $\omega = 2\pi f$. The cycle may be of pure sinusoidal form like $\sin(\omega t)$, or the presence of higher harmonic terms may be indicated, e.g. by asymmetry in the wave form;
(d) quasi-cycles, i.e., waves of fluctuating period and amplitude; and
(e) irregular statistical fluctuations and swings about the overall mean or trend.

Trends, seasonal patterns, and cycles might be regarded as **deterministic** components following fixed mathematical equations, and the quasi-cycles and other statistical fluctuations as **stochastic** and describable by short-term correlation structure. For a finite data set it is not always easy to discriminate between these two types, and a common description using the class of autoregressive integrated moving-average (ARIMA) models is now widely used. The form of these models is that of difference equations (or recurrence relations) relating present and past values of the series. The user is referred to Box and Jenkins [2] for a thorough account of these models and how to use them. We follow their notation and outline the recommended steps in ARIMA model building for which routines are available.

2.1.1 Transformations

If the variance of the observations in the series is not constant across the range of observations it may be useful to apply a variance-stabilizing transformation to the series. A common situation is for the variance to increase with the magnitude of the observations and in this case typical transformations used are the log or square root transformation. A range–mean or standard deviation–mean plot provides a quick and easy way of detecting non-constant variance and of choosing, if required, a suitable transformation. This is a plot of the range or standard deviation of successive groups of observations against their means.

2.1.2 Differencing operations

These may be used to simplify the structure of a time series.

First-order differencing, i.e., forming the new series

$$\nabla x_t = x_t - x_{t-1}$$

will remove a linear trend. First-order seasonal differencing

$$\nabla_s x_t = x_t - x_{t-s}$$

eliminates a fixed seasonal pattern.

These operations reflect the fact that it is often appropriate to model a time series in terms of changes from one value to another. Differencing is also therefore appropriate when the series has something of the nature of a random walk, which is by definition the accumulation of independent changes.

Differencing may be applied repeatedly to a series giving

$$w_t = \nabla^d \nabla_s^D x_t$$

where d and D are the orders of differencing. The derived series w_t will be shorter, of length $N = n - d - s{\times}D$ and extend for $t = 1 + d + s{\times}D, \ldots, n$.

2.1.3 Sample autocorrelations

Given that a series has (possibly as a result of simplifying by differencing operations) a homogeneous appearance throughout its length, fluctuating with approximately constant variance about an overall mean level, it is appropriate to assume that its statistical properties are **stationary**. For most purposes the correlations ρ_k between terms x_t, x_{t+k} or w_t, w_{t+k} separated by lag k give an adequate description of the statistical structure and are estimated by the sample autocorrelation function (acf) r_k, for $k = 1, 2, \ldots$.

As described by Box and Jenkins [2], these may be used to indicate which particular ARIMA model may be appropriate.

2.1.4 Partial autocorrelations

The information in the autocorrelations ρ_k may be presented in a different light by deriving from them the coefficients of the partial autocorrelation function (pacf) $\phi_{k,k}$, for $k = 1, 2, \ldots$. $\phi_{k,k}$ measures the correlation between x_t and x_{t+k} conditional upon the intermediate values $x_{t+1}, x_{t+2}, \ldots, x_{t+k-1}$. The corresponding sample values $\hat{\phi}_{k,k}$ give further assistance in the selection of ARIMA models.

Both acf and pacf may be rapidly computed, particularly in comparison with the time taken to estimate ARIMA models.

2.1.5 Finite lag predictor coefficients and error variances

The partial autocorrelation coefficient $\phi_{k,k}$ is determined as the final parameter in the minimum variance predictor of x_t in terms of $x_{t-1}, x_{t-2}, \ldots, x_{t-k}$,

$$x_t = \phi_{k,1} x_{t-1} + \phi_{k,2} x_{t-2} + \cdots + \phi_{k,k} x_{t-k} + e_{k,t}$$

where $e_{k,t}$ is the prediction error, and the first subscript k of $\phi_{k,i}$ and $e_{k,t}$ emphasises the fact that the parameters will alter as k increases. Moderately good estimates $\hat{\phi}_{k,i}$ of $\phi_{k,i}$ are obtained from the sample acf, and after calculating the pacf up to lag L, the successive values $v_1, v_2, \ldots, v_L$ of the prediction error variance estimates, $v_k = \mathrm{var}(e_{k,t})$, are available, together with the final values of the coefficients $\hat{\phi}_{k,1}, \hat{\phi}_{k,2}, \ldots, \hat{\phi}_{k,L}$. If x_t has non-zero mean, $\bar{x}$, it is adequate to use $x_t - \bar{x}$, in place of x_t in the prediction equation.

Although Box and Jenkins [2] do not place great emphasis on these prediction coefficients, their use is advocated for example by Akaike [1], who recommends selecting an optimal order of the predictor as the lag for which the final prediction error (FPE) criterion $(1 + k/n)(1 - k/n)^{-1} v_k$ is a minimum.

2.1.6 ARIMA models

The correlation structure in stationary time series may often be represented by a model with a small number of parameters belonging to the autoregressive moving-average (ARMA) class. If the stationary series w_t has been derived by differencing from the original series x_t, then x_t is said to follow an ARIMA model. Taking $w_t = \nabla^d x_t$, the (non-seasonal) ARIMA (p, d, q) model with p autoregressive parameters $\phi_1, \phi_2, \ldots, \phi_p$ and q moving-average parameters $\theta_1, \theta_2, \ldots, \theta_q$, represents the structure of w_t by the equation

$$w_t = \phi_1 w_{t-1} + \cdots + \phi_p w_{t-p} + a_t - \theta_1 a_{t-1} - \cdots - \theta_q a_{t-q}, \quad (1)$$

where a_t is an uncorrelated series (white noise) with mean 0 and constant variance σ_a^2. If w_t has a non-zero mean c, then this is allowed for by replacing $w_t, w_{t-1}, \ldots$ by $w_t - c, w_{t-1} - c, \ldots$ in the model. Although c is often estimated by the sample mean of w_t this is not always optimal.

A series generated by this model will only be stationary provided restrictions are placed on $\phi_1, \phi_2, \ldots, \phi_p$ to avoid unstable growth of w_t. These are called **stationarity** constraints. The series a_t may also be usefully interpreted as the linear **innovations** in x_t (and in w_t), i.e., the error if x_t were to be predicted using the information in all past values $x_{t-1}, x_{t-2}, \ldots$, provided also that $\theta_1, \theta_2, \ldots, \theta_q$ satisfy **invertibility** constraints. This allows the series a_t to be regenerated by rewriting the model equation as

$$a_t = w_t - \phi_1 w_{t-1} - \ldots - \phi_p w_{t-p} + \theta_1 a_{t-1} + \ldots + \theta_q a_{t-q}. \quad (2)$$

For a series with short-term correlation only, i.e., r_k is not significant beyond some low lag q (see Box and Jenkins [2] for the statistical test), then the pure moving-average model MA(q) is appropriate, with no autoregressive parameters, i.e., $p = 0$.

Autoregressive parameters are appropriate when the acf pattern decays geometrically, or with a damped sinusoidal pattern which is associated with quasi-periodic behaviour in the series. If the sample pacf $\hat{\phi}_{k,k}$ is significant only up to some low lag p, then a pure autoregressive model AR(p) is appropriate, with $q = 0$. Otherwise moving-average terms will need to be introduced, as well as autoregressive terms.

The seasonal ARIMA (p, d, q, P, D, Q, s) model allows for correlation at lags which are multiples of the seasonal period s. Taking $w_t = \nabla^d \nabla_s^D x_t$, the series is represented in a two-stage manner via an intermediate series e_t

$$w_t = \Phi_1 w_{t-s} + \cdots + \Phi_P w_{t-s\times P} + e_t - \Theta_1 e_{t-s} - \cdots - \Theta_Q e_{t-s\times Q} \quad (3)$$

$$e_t = \phi_1 e_{t-1} + \cdots + \phi_p e_{t-p} + a_t - \theta_1 a_{t-1} - \cdots - \theta_q a_{t-q} \quad (4)$$

where Φ_i, Θ_i are the seasonal parameters and P, Q are the corresponding orders. Again, w_t may be replaced by $w_t - c$.

2.1.7 ARIMA model estimation

In theory, the parameters of an ARIMA model are determined by a sufficient number of autocorrelations $\rho_1, \rho_2, \ldots$. Using the sample values $r_1, r_2, \ldots$ in their place it is usually (but not always) possible to solve for the corresponding ARIMA parameters.

These are rapidly computed but are not fully efficient estimates, particularly if moving-average parameters are present. They do provide useful **preliminary** values for an efficient but relatively slow iterative method of estimation. This is based on the least-squares principle by which parameters are chosen to minimize the sum of squares of the innovations a_t, which are regenerated from the data using (2), or the reverse of (3) and (4) in the case of seasonal models.

Lack of knowledge of terms on the right-hand side of (2), when $t = 1, 2, \ldots, \max(p, q)$, is overcome by introducing q unknown series values $w_0, w_1, \ldots, w_{1-q}$ which are estimated as nuisance parameters, and using correction for transient errors due to the autoregressive terms. If the data $w_1, w_2, \ldots, w_N = w$ is viewed as a single sample from a multivariate Normal density whose covariance matrix V is a function of the ARIMA model parameters, then the exact likelihood of the parameters is

$$-\tfrac{1}{2} \log |V| - \tfrac{1}{2} w^T V^{-1} w.$$

The least-squares criterion as outlined above is equivalent to using the quadratic form

$$QF = w^T V^{-1} w$$

as an objective function to be minimized. Neglecting the term $-\frac{1}{2} \log |V|$ yields estimates which differ very little from the exact likelihood except in small samples, or in seasonal models with a small number of whole seasons contained in the data. In these cases bias in moving-average parameters may cause them to stick at the boundary of their constraint region, resulting in failure of the estimation method.

Approximate standard errors of the parameter estimates and the correlations between them are available after estimation.

The model residuals, $\hat{a}_t$, are the innovations resulting from the estimation and are usually examined for the presence of autocorrelation as a check on the adequacy of the model.

2.1.8 ARIMA model forecasting

An ARIMA model is particularly suited to extrapolation of a time series. The model equations are simply used for $t = n+1, n+2, \ldots$ replacing the unknown future values of a_t by zero. This produces future values of w_t, and if differencing has been used this process is reversed (the so-called integration part of ARIMA models) to construct future values of x_t.

Forecast error limits are easily deduced.

This process requires knowledge only of the model orders and parameters together with a limited set of the terms $a_{t-i}, e_{t-i}, w_{t-i}, x_{t-i}$ which appear on the right-hand side of the models (3) and (4) (and the differencing equations) when $t = n$. It does not require knowledge of the whole series.

We call this the state set. It is conveniently constituted after model estimation. Moreover, if new observations $x_{n+1}, x_{n+2}, \ldots$ come to hand, then the model equations can easily be used to update the state set before constructing forecasts from the end of the new observations. This is particularly useful when forecasts are constructed on a regular basis. The new innovations $a_{n+1}, a_{n+2}, \ldots$ may be compared with the residual standard deviation, σ_a, of the model used for forecasting, as a check that the model is continuing to forecast adequately.

2.2 Univariate Spectral Analysis

In describing a time series using spectral analysis the fundamental components are taken to be sinusoidal waves of the form $R\cos(\omega t + \phi)$, which for a given angular frequency ω, $0 \leq \omega \leq \pi$, is specified by its amplitude $R > 0$ and phase ϕ, $0 \leq \phi < 2\pi$. Thus in a time series of n observations it is not possible to distinguish more than $n/2$ independent sinusoidal components. The frequency range $0 \leq \omega \leq \pi$ is limited to a shortest wavelength of two sampling units because any wave of higher frequency is indistinguishable upon sampling (or is aliased with) a wave within this range. Spectral analysis follows the idea that for a series made up of a **finite** number of sine waves the amplitude of any component at frequency ω is given to order $1/n$ by

$$R^2 = \left(\frac{1}{n^2}\right)\left|\sum_{t=1}^{n} x_t e^{i\omega t}\right|^2 .$$

2.2.1 The sample spectrum

For a series $x_1, x_2, \ldots, x_n$ this is defined as

$$f^*(\omega) = \left(\frac{1}{2n\pi}\right)\left|\sum_{t=1}^{n} x_t e^{i\omega t}\right|^2 ,$$

the scaling factor now being chosen in order that

$$2\int_0^{\pi} f^*(\omega)d\omega = \sigma_x^2,$$

i.e., the spectrum indicates how the sample variance (σ_x^2) of the series is distributed over components in the frequency range $0 \leq \omega \leq \pi$.

It may be demonstrated that $f^*(\omega)$ is equivalently defined in terms of the sample autocorrelation function (acf) r_k of the series as

$$f^*(\omega) = \left(\frac{1}{2\pi}\right)\left(c_0 + 2\sum_{k=1}^{n-1} c_k \cos k\omega\right) ,$$

where $c_k = \sigma_x^2 r_k$ are the sample autocovariance coefficients.

If the series x_t does contain a **deterministic** sinusoidal component of amplitude R, this will be revealed in the sample spectrum as a sharp peak of approximate width π/n and height $(n/2\pi)R^2$. This is called the discrete part of the spectrum, the variance R^2 associated with this component being in effect concentrated at a single frequency.

If the series x_t has no deterministic components, i.e., is purely **stochastic** being stationary with acf r_k, then with increasing sample size the expected value of $f^*(\omega)$ converges to the theoretical spectrum – the **continuous** part

$$f(\omega) = \left(\frac{1}{2\pi}\right)\left(\gamma_0 + 2\sum_{k=1}^{\infty}\gamma_k \cos(\omega k)\right),$$

where γ_k are the theoretical autocovariances.

The sample spectrum does **not** however converge to this value but at each frequency point fluctuates about the theoretical spectrum with an exponential distribution, being independent at frequencies separated by an interval of $2\pi/n$ or more. Various devices are therefore employed to smooth the sample spectrum and reduce its variability. Much of the strength of spectral analysis derives from the fact that the error limits are multiplicative so that features may still show up as significant in a part of the spectrum which has a generally low level, whereas they are completely masked by other components in the original series. The spectrum can help to distinguish deterministic cyclical components from the stochastic quasi-cycle components which produce a broader peak in the spectrum. (The deterministic components can be removed by regression and the remaining part represented by an ARIMA model).

A large discrete component in a spectrum can distort the continuous part over a large frequency range surrounding the corresponding peak. This may be alleviated at the cost of slightly broadening the peak by tapering a portion of the data at each end of the series with weights which decay smoothly to zero. It is usual to correct for the mean of the series and for any linear trend by simple regression, since they would similarly distort the spectrum.

2.2.2 Spectral smoothing by lag window

The estimate is calculated directly from the sample covariances c_k as

$$f(\omega) = \left(\frac{1}{2\pi}\right)\left(c_0 + 2\sum_{k=1}^{M-1} w_k c_k \cos k\omega\right),$$

the smoothing being induced by the lag window weights w_k which extend up to a **truncation lag** M which is generally much less than n. The smaller the value of M, the greater the degree of smoothing, the spectrum estimates being independent only at a wider frequency separation indicated by the **bandwidth** b which is proportional to $1/M$. It is wise, however, to calculate the spectrum at intervals appreciably less than this. Although greater smoothing narrows the error limits, it can also distort the spectrum, particularly by flattening peaks and filling in troughs.

2.2.3 Direct spectral smoothing

The unsmoothed sample spectrum is calculated for a fine division of frequencies, then averaged over intervals centred on each frequency point for which the smoothed spectrum is required. This is usually at a coarser frequency division. The bandwidth corresponds to the width of the averaging interval.

2.3 Linear Lagged Relationships Between Time Series

We now consider the context in which one time series, called the dependent or output series $y_1, y_2, \ldots, y_n$, is believed to depend on one or more explanatory or input series, e.g. $x_1, x_2, \ldots, x_n$. This dependency may follow a simple linear regression, e.g.

$$y_t = vx_t + n_t$$

or more generally may involve lagged values of the input

$$y_t = v_0 x_t + v_1 x_{t-1} + v_2 x_{t-2} + \cdots + n_t.$$

The sequence $v_0, v_1, v_2, \ldots$ is called the **impulse response function** (IRF) of the relationship. The term n_t represents that part of y_t which cannot be explained by the input, and it is assumed to follow a univariate ARIMA model. We call n_t the (output) noise component of y_t, and it includes any constant term in the relationship. It is assumed that the input series, x_t, and the noise component, n_t, are independent.

The part of y_t which is explained by the input is called the input component z_t:

$$z_t = v_0 x_t + v_1 x_{t-1} + v_2 x_{t-2} + \cdots$$

so $y_t = z_t + n_t$.

The eventual aim is to model both these components of y_t on the basis of observations of $y_1, y_2, \ldots, y_n$ and $x_1, x_2, \ldots, x_n$. In applications to forecasting or control both components are important. In general there may be more than one input series, e.g. $x_{1,t}$ and $x_{2,t}$, which are assumed to be independent and corresponding components $z_{1,t}$ and $z_{2,t}$, so

$$y_t = z_{1,t} + z_{2,t} + n_t.$$

2.3.1 Transfer function models

In a similar manner to that in which the structure of a univariate series may be represented by a finite-parameter ARIMA model, the structure of an input component may be represented by a **transfer function** (TF) model with delay time b, p autoregressive-like parameters $\delta_1, \delta_2, \ldots, \delta_p$ and $q+1$ moving-average-like parameters $\omega_0, \omega_1, \ldots, \omega_q$:

$$z_t = \delta_1 z_{t-1} + \delta_2 z_{t-2} + \cdots + \delta_p z_{t-p} + \omega_0 x_{t-b} - \omega_1 x_{t-b-1} - \cdots - \omega_q x_{t-b-q}.$$

If $p > 0$ this represents an IRF which is infinite in extent and decays with geometric and/or sinusoidal behaviour. The parameters $\delta_1, \delta_2, \ldots, \delta_p$ are constrained to satisfy a stability condition identical to the stationarity condition of autoregressive models. There is no constraint on $\omega_0, \omega_1, \ldots, \omega_q$.

2.3.2 Cross-correlations

An important tool for investigating how an input series x_t affects an output series y_t is the sample **cross-correlation function** (CCF) $r_{xy}(k)$, for $k = 0, 1, 2, \ldots$ between the series. If x_t and y_t are (jointly) stationary time series this is an estimator of the theoretical quantity

$$\rho_{xy}(k) = \mathrm{corr}(x_t, y_{t+k}).$$

The sequence $r_{yx}(k)$, for $k = 0, 1, 2, \ldots$ is distinct from $r_{xy}(k)$, though it is possible to interpret

$$r_{yx}(k) = r_{xy}(-k).$$

When the series y_t and x_t are believed to be related by a transfer function model, the CCF is determined by the IRF $v_0, v_1, v_2, \ldots$ and the autocorrelation function of the input x_t.

In the **particular** case when x_t is an uncorrelated series or **white noise** (and is uncorrelated with any other inputs)

$$\rho_{xy}(k) \propto v_k$$

and the sample CCF can provide an estimate of v_k:

$$\tilde{v}_k = (s_y/s_x) r_{xy}(k)$$

where s_y and s_x are the sample standard deviations of y_t and x_t, respectively.

In theory the IRF coefficients $v_b, \ldots, v_{b+p+q}$ determine the parameters in the TF model, and using $\tilde{v}_k$ to estimate $\tilde{v}_k$ it is possible to solve for **preliminary** estimates of $\delta_1, \delta_2, \ldots, \delta_p$, $\omega_0, \omega_1, \ldots, \omega_q$.

2.3.3 Prewhitening or filtering by an ARIMA model

In general an input series x_t is not white noise, but may be represented by an ARIMA model with innovations or residuals a_t which are white noise. If precisely the same operations by which a_t is generated from x_t are applied to the output y_t to produce a series b_t, then the transfer function relationship between y_t and x_t is preserved between b_t and a_t. It is then possible to estimate

$$\tilde{v}_k = (s_b/s_a) r_{ab}(k).$$

The procedure of generating a_t from x_t (and b_t from y_t) is called prewhitening or filtering by an ARIMA model. Although a_t is necesarily white noise, this is not generally true of b_t.

2.3.4 Multi-input model estimation

The term multi-input model is used for the situation when one output series y_t is related to one or more input series $x_{j,t}$, as described in Section 2.3. If for a given input the relationship is a simple linear regression, it is called a simple input; otherwise it is a transfer function input. The error or noise term follows an ARIMA model.

Given that the orders of all the transfer function models and the ARIMA model of a multi-input model have been specified, the various parameters in those models may be (simultaneously) estimated.

The procedure used is closely related to the least-squares principle applied to the innovations in the ARIMA noise model.

The innovations are derived for any proposed set of parameter values by calculating the response of each input to the transfer functions and then evaluating the noise n_t as the difference between this response (combined for all the inputs) and the output. The innovations are derived from the noise using the ARIMA model in the same manner as for a univariate series, and as described in Section 2.1.5.

In estimating the parameters, consideration has to be given to the lagged terms in the various model equations which are associated with times prior to the observation period, and are therefore unknown. The subroutine descriptions provide the necessary detail as to how this problem is treated.

Also, as described in Section 2.1.6 the sum of squares criterion

$$S = \sum a_t^2$$

is related to the quadratic form in the exact log-likelihood of the parameters:

$$-\tfrac{1}{2}\log|V| - \tfrac{1}{2}w^T V^{-1} w.$$

Here w is the vector of appropriately differenced noise terms, and

$$w^T V^{-1} w = S/\sigma_a^2,$$

where σ_a^2 is the innovation variance parameter.

The least-squares criterion is therefore identical to minimization of the quadratic form, but is not identical to exact likelihood. Because V may be expressed as $M\sigma_a^2$, where M is a function of the ARIMA model parameters, substitution of σ_a^2 by its maximum likelihood estimator yields a concentrated (or profile) likelihood which is a function of

$$|M|^{1/N} S.$$

N is the length of the differenced noise series w, and $|M| = \det M$.

Use of the above quantity, called the deviance, D, as an objective function is preferable to the use of S alone, on the grounds that it is equivalent to exact likelihood, and yields estimates with better properties. However, there is an appreciable computational penalty in calculating D, and in large samples it differs very little from S, except in the important case of seasonal ARIMA models where the number of whole seasons within the data length must also be large.

The user is given the option of taking the objective function to be either S or D, or a third possibility, the marginal likelihood. This is similar to exact likelihood but can counteract bias in the ARIMA model due to the fitting of a large number of simple inputs.

Approximate standard errors of the parameter estimates and the correlations between them are available after estimation.

The model residuals $\hat{a}_t$ are the innovations resulting from the estimation, and they are usually examined for the presence of either autocorrelation or cross-correlation with the inputs. Absence of such correlation provides some confirmation of the adequacy of the model.

2.3.5 Multi-input model forecasting

A multi-input model may be used to forecast the output series provided future values (possibly forecasts) of the input series are supplied.

Construction of the forecasts requires knowledge only of the model orders and parameters, together with a limited set of the most recent variables which appear in the model equations. This is called the state set. It is conveniently constituted after model estimation. Moreover, if new observations $y_{n+1}, y_{n+2}, \ldots$ of the output series and $x_{n+1}, x_{n+2}, \ldots$ of (all) the independent input series become available, then the model equations can easily be used to update the state set before constructing forecasts from the end of the new observations. The new innovations $a_{n+1}, a_{n+2}, \ldots$ generated in this updating may be used to monitor the continuing adequacy of the model.

2.3.6 Transfer function model filtering

In many time series applications it is desired to calculate the response (or output) of a transfer function model for a given input series.

Smoothing, detrending, and seasonal adjustment are typical applications. The user must specify the orders and parameters of a transfer function model for the purpose being considered. This may then be applied to the input series.

Again, problems may arise due to ignorance of the input series values prior to the observation period. The transient errors which can arise from this cause may be substantially reduced by using 'backforecasts' of these unknown observations.

2.4 Multivariate Time Series

Multi-input modelling represents one output time series in terms of one or more input series. Although there are circumstances in which it may be more appropriate to analyse a set of time series by modelling each one in turn as the output series with the remainder as inputs, there is a more symmetric approach in such a context. These models are known as vector autoregressive moving-average (VARMA) models.

2.4.1 Differencing and transforming a multivariate time series

As in the case of a univariate time series, it may be useful to simplify the series by differencing operations which may be used to remove linear or seasonal trend, thus ensuring that the resulting series to be used in the model estimation is stationary. It may also be neccessary to apply transformations to the individual components of the multivariate series in order to stabilize the variance. Commonly used transformations are the log and square root transformations.

2.4.2 Model identification for a multivariate time series

Multivariate analogues of the autocorrelation and partial autocorrelation functions are available for analysing a set of k time series, $x_{i,1}, x_{i,2}, \ldots, x_{i,n}$, for $i = 1, 2, \ldots, k$, thereby making it possible to obtain some understanding of a suitable VARMA model for the observed series.

It is assumed that the time series have been differenced if necessary, and that they are jointly stationary. The lagged correlations between all possible pairs of series, i.e.,

$$\rho_{ijl} = \mathrm{corr}(x_{i,t}, x_{j,t+l})$$

are then taken to provide an adequate description of the statistical relationships between the series. These quantities are estimated by sample auto- and cross-correlations r_{ijl}. For each l these may be viewed as elements of a (lagged) autocorrelation matrix.

Thus consider the **vector process** x_t (with elements x_{it}) and lagged autocovariance matrices Γ_l with elements of $\sigma_i\sigma_j\rho_{ijl}$ where $\sigma_i^2 = \mathrm{var}(x_{i,t})$. Correspondingly, Γ_l is estimated by the matrix C_l with elements $s_i s_j r_{ijl}$ where s_i^2 is the sample variance of x_{it}.

For a series with short-term cross-correlation only, i.e., r_{ijl} is not significant beyond some low lag q, then the pure vector MA(q) model, with no autoregressive parameters, i.e., $p = 0$, is appropriate.

The correlation matrices provide a description of the joint statistical properties of the series. It is also possible to calculate matrix quantities which are closely analogous to the partial autocorrelations of univariate series (see Section 2.1.3). Wei [6] discusses both the partial autoregression matrices proposed by Tiao and Box [5] and partial lag correlation matrices.

In the univariate case the partial autocorrelation function (pacf) between x_t and x_{t+l} is the correlation coefficient between the two after removing the linear dependence on each of the intervening variables $x_{t+1}, x_{t+2}, \ldots, x_{t+l-1}$. This partial autocorrelation may also be obtained as the last regression coefficient associated with x_t when regressing x_{t+l} on its l lagged variables $x_{t+l-1}, x_{t+l-2}, \ldots, x_t$. Tiao and Box [5] extended this method to the multivariate case to define the partial autoregression matrix. Heyse and Wei [4] also extended the univariate definition of the pacf to derive the correlation matrix between the vectors x_t and x_{t+l} after removing the linear dependence on each of the intervening vectors $x_{t+1}, x_{t+2}, \ldots, x_{t+l-1}$, the partial lag correlation matrix.

Note that the partial lag correlation matrix is a correlation coefficient matrix since each of its elements is a properly normalised correlation coefficient. This is not true of the partial autoregression matrices (except in the univariate case for which the two types of matrix are the same). The partial lag correlation matrix at lag 1 also reduces to the regular correlation matrix at lag 1; this is not true of the partial autoregression matrices (again except in the univariate case).

Both the above share the same cut-off property for autoregressive processes; that is for an autoregressive process of order p, the terms of the matrix at lags $p+1$ and greater are zero. Thus if the sample partial cross-correlations are significant only up to some low lag p then a pure vector AR(p) model is appropriate with $q = 0$. Otherwise moving-average terms will need to be introduced as well as autoregressive terms.

Under the hypothesis that x_t is an autoregressive process of order $l-1$, n times the sum of the squared elements of the partial lag correlation matrix at lag l is asymptotically distributed as a χ^2 variable with k^2 degrees of freedom where k is the dimension of the multivariate time series. This provides a diagnostic aid for determining the order of an autoregressive model.

The partial autoregression matrices may be found by solving a multivariate version of the Yule–Walker equations to find the autoregression matrices, using the final regression matrix coefficient as the partial autoregression matrix at that particular lag.

The basis of these calculations is a multivariate autoregressive model:

$$x_t = \phi_{l,1} x_{t-1} + \cdots + \phi_{l,l} x_{t-l} + e_{l,t}$$

where $\phi_{l,1}, \phi_{l,2}, \ldots, \phi_{l,l}$ are **matrix coefficients**, and $e_{l,t}$ is the vector of errors in the prediction. These coefficients may be rapidly computed using a recursive technique which requires, and simultaneously furnishes, a backward prediction equation:

$$x_{t-l-1} = \psi_{l,1} x_{t-l} + \psi_{l,2} x_{t-l+1} + \cdots + \psi_{l,l} x_{t-1} + f_{l,t}$$

(in the univariate case $\psi_{l,i} = \phi_{l,i}$).

The forward prediction equation coefficients, $\phi_{l,i}$, are of direct interest, together with the covariance matrix D_l of the prediction errors $e_{l,t}$. The calculation of these quantities for a particular maximum equation lag $l = L$ involves calculation of the same quantities for increasing values of $l = 1, 2, \ldots, L$.

The quantities $v_l = \det D_l / \det \Gamma_0$ may be viewed as generalized variance ratios, and provide a measure of the efficiency of prediction (the smaller the better). The reduction from v_{l-1} to v_l which occurs on extending the order of the predictor to l may be represented as

$$v_l = v_{l-1}(1 - \rho_l^2)$$

where ρ_l^2 is a multiple squared partial autocorrelation coefficient associated with k^2 degrees of freedom.

Sample estimates of all the above quantities may be derived by using the series covariance matrices C_l, for $l = 1, 2, \ldots, L$, in place of Γ_l. The best lag for prediction purposes may be chosen as that which yields the minimum final prediction error (FPE) criterion:

$$\mathrm{FPE}(l) = v_l \times \frac{(1 + lk^2/n)}{(1 - lk^2/n)}.$$

An alternative method of estimating the sample partial autoregression matrices is by using multivariate least-squares to fit a series of multivariate autoregressive models of increasing order.

2.4.3 VARMA model estimation

The cross-correlation structure of a stationary multivariate time series may often be represented by a model with a small number of parameters belonging to the vector autoregressive moving-average (VARMA) class. If the stationary series w_t has been derived by transforming and/or differencing the original series x_t, then w_t is said to follow the VARMA model:

$$w_t = \phi_1 w_{t-1} + \cdots + \phi_p w_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - \cdots - \theta_q \epsilon_{t-q},$$

where ϵ_t is a vector of uncorrelated residual series (white noise) with zero mean and constant covariance matrix Σ, $\phi_1, \phi_2, \ldots, \phi_p$ are the p autoregressive (AR) **parameter matrices** and $\theta_1, \theta_2, \ldots, \theta_q$ are the q moving-average (MA) **parameter matrices**. If w_t has a non-zero mean μ, then this can be allowed for by replacing $w_t, w_{t-1}, \ldots$ by $w_t - \mu, w_{t-1} - \mu, \ldots$ in the model.

A series generated by this model will only be stationary provided restrictions are placed on $\phi_1, \phi_2, \ldots, \phi_p$ to avoid unstable growth of w_t. These are **stationarity** constraints. The series ϵ_t may also be usefully interpreted as the linear **innovations** in w_t, i.e., the error if w_t were to be predicted using the information in all past values $w_{t-1}, w_{t-2}, \ldots$, provided also that $\theta_1, \theta_2, \ldots, \theta_q$ satisfy what are known as **invertibility** constraints. This allows the series ϵ_t to be generated by rewriting the model equation as

$$\epsilon_t = w_t - \phi_1 w_{t-1} - \cdots - \phi_p w_{t-p} + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}.$$

The method of maximum likelihood may be used to estimate the parameters of a specified VARMA model from the observed multivariate time series together with their standard errors and correlations.

The residuals from the model may be examined for the presence of autocorrelations as a check on the adequacy of the fitted model.

2.4.4 VARMA model forecasting

Forecasts of the series may be constructed using a multivariate version of the univariate method. Efficient methods are available for updating the forecasts each time new observations become available.

2.5 Cross-spectral Analysis

The relationship between two time series may be investigated in terms of their sinusoidal components at different frequencies. At frequency ω a component of y_t of the form

$$R_y(\omega) \cos \omega t - \phi_y(\omega)$$

has its amplitude $R_y(\omega)$ and phase lag $\phi_y(\omega)$ estimated by

$$R_y(\omega) e^{i\phi_y(\omega)} = \frac{1}{n} \sum_{t=1}^{n} y_t e^{i\omega t}$$

and similarly for x_t. In the univariate analysis only the amplitude was important – in the cross analysis the phase is important.

2.5.1 The sample cross-spectrum

This is defined by

$$f_{xy}^{*}(\omega) = \frac{1}{2\pi n} \left(\sum_{t=1}^{n} y_t e^{i\omega t} \right) \left(\sum_{t=1}^{n} x_t e^{-i\omega t} \right).$$

It may be demonstrated that this is equivalently defined in terms of the sample CCF, $r_{xy}(k)$, of the series as

$$f_{xy}^{*}(\omega) = \frac{1}{2\pi} \sum_{-(n-1)}^{(n-1)} c_{xy}(k) e^{i\omega k}$$

where $c_{xy}(k) = s_x s_y r_{xy}(k)$ is the cross-covariance function.

2.5.2 The amplitude and phase spectrum

The cross-spectrum is specified by its real part or cospectrum $cf^*(\omega)$ and imaginary part or quadrature spectrum $qf^*(\omega)$, but for the purpose of interpretation the cross-amplitude spectrum and phase spectrum are useful:

$$A^*(\omega) = |f^*_{xy}(\omega)|, \ \phi^*(\omega) = \arg(f^*_{xy}(\omega)).$$

If the series x_t and y_t contain deterministic sinusoidal components of amplitudes R_y, R_x and phases ϕ_y, ϕ_x at frequency ω, then $A^*(\omega)$ will have a peak of approximate width π/n and height $(n/2\pi)R_y R_x$ at that frequency, with corresponding phase $\phi^*(\omega) = \phi_y - \phi_x$. This supplies no information that cannot be obtained from the two series separately. The statistical relationship between the series is better revealed when the series are purely stochastic and jointly stationary, in which case the expected value of $f^*_{xy}(\omega)$ converges with increasing sample size to the theoretical cross-spectrum

$$f_{xy}(\omega) = \frac{1}{2\pi}\sum_{-\infty}^{\infty} \gamma_{xy}(k)e^{i\omega k}$$

where $\gamma_{xy}(k) = \text{cov}(x_t, y_{t+k})$. The sample spectrum, as in the univariate case, does not, however, converge to the theoretical spectrum without some form of smoothing which either implicitly (using a lag window) or explicitly (using a frequency window) averages the sample spectrum $f^*_{xy(\omega)}$ over wider bands of frequency to obtain a smoothed estimate $\hat{f}_{xy}(\omega)$.

2.5.3 The coherency spectrum

If there is no statistical relationship between the series at a given frequency, then $f_{xy}(\omega) = 0$, and the smoothed estimate $\hat{f}_{xy}(\omega)$, will be close to 0. This is assessed by the squared coherency between the series:

$$\hat{W}(\omega) = \frac{|\hat{f}_{xy}(\omega)|^2}{\hat{f}_{xx}(\omega)\hat{f}_{yy}(\omega)}$$

where $\hat{f}_{xx}(\omega)$ is the corresponding smoothed univariate spectrum estimate for x_t, and similarly for y_t. The coherency can be treated as a squared multiple correlation. It is similarly invariant in theory not only to simple scaling of x_t and y_t, but also to filtering of the two series, and provides a useful test statistic for the relationship between autocorrelated series. Note that without smoothing,

$$|f^*_{xy}(\omega)|^2 = f^*_{xx}(\omega)f^*_{yy}(\omega),$$

so the coherency is 1 at all frequencies, just as a correlation is 1 for a sample of size 1. Thus smoothing is essential for cross-spectrum analysis.

2.5.4 The gain and noise spectrum

If y_t is believed to be related to x_t by a linear lagged relationship as in Section 2.3, i.e.,

$$y_t = v_0 x_t + v_1 x_{t-1} + v_2 x_{t-2} + \cdots + n_t,$$

then the theoretical cross-spectrum is

$$f_{xy}(\omega) = V(\omega)f_{xx}(\omega)$$

where

$$V(\omega) = G(\omega)e^{i\phi(\omega)} = \sum_{k=0}^{\infty} v_k e^{ik\omega}$$

is called the frequency response of the relationship.

Thus if x_t were a sinusoidal wave at frequency ω (and n_t were absent), y_t would be similar but multiplied in amplitude by $G(\omega)$ and shifted in phase by $\phi(\omega)$. Furthermore, the theoretical univariate spectrum

$$f_{yy}(\omega) = G(\omega)^2 f_{xx}(\omega) + f_n(\omega)$$

where n_t, with spectrum $f_n(\omega)$, is assumed independent of the input x_t.

Cross-spectral analysis thus furnishes estimates of the gain

$$\hat{G}(\omega) = |\hat{f}_{xy}(\omega)|/\hat{f}_{xx}(\omega)$$

and the phase

$$\hat{\phi}(\omega) = \arg\left(\hat{f}_{xy}(\omega)\right)$$

From these representations of the estimated frequency response $\hat{V}(\omega)$, parametric TF models may be recognised and selected. The noise spectrum may also be estimated as

$$\hat{f}_{y|x}(\omega) = \hat{f}_{yy}(\omega)\left(1 - \hat{W}(\omega)\right)$$

– a formula which reflects the fact that in essence a regression is being performed of the sinusoidal components of y_t on those of x_t over each frequency band.

Interpretation of the frequency response may be aided by extracting from $\hat{V}(\omega)$ estimates of the IRF $\hat{v}_k$. It is assumed that there is no anticipatory response between y_t and x_t, i.e., no coefficients v_k with $k = -1, -2$ are needed (their presence might indicate feedback between the series).

2.5.5 Cross-spectrum smoothing by lag window

The estimate of the cross-spectrum is calculated from the sample cross-variances as

$$\hat{f}_{xy}(\omega) = \frac{1}{2\pi} \sum_{-M+S}^{M+S} w_{k-S} c_{xy}(k) e^{i\omega k}.$$

The lag window w_k extends up to a truncation lag M as in the univariate case, but its centre is shifted by an alignment lag S usually chosen to coincide with the peak cross-correlation. This is equivalent to an alignment of the series for peak cross-correlation at lag 0, and reduces bias in the phase estimation.

The selection of the truncation lag M, which fixes the bandwidth of the estimate, is based on the same criteria as for univariate series, and the same choice of M and window shape should be used as in univariate spectrum estimation to obtain valid estimates of the coherency, gain etc., and test statistics.

2.5.6 Direct smoothing of the cross-spectrum

The computations are exactly as for smoothing of the univariate spectrum except that allowance is made for an implicit alignment shift S between the series.

2.6 Kalman Filters

Kalman filtering provides a method for the analysis of multi-dimensional time series. The underlying model is:

$$X_{t+1} = A_t X_t + B_t W_t$$

$$Y_t = C_t X_t + V_t$$

where X_t is the unobserved state vector, Y_t is the observed measurement vector, W_t is the state noise, V_t is the measurement noise, A_t is the state transition matrix, B_t is the noise coefficient matrix and C_t is the measurement coefficient matrix at time t. The state noise and the measurement noise are assumed to be uncorrelated with zero mean and covariance matrices:

$$E\{W_t W_t^T\} = Q_t \text{ and } E\{V_t V_t^T\} = R_t$$

If the system matrices A_t, B_t, C_t and the covariance matrices Q_t,R_t are known then Kalman filtering can be used to compute the minimum variance estimate of the stochastic variable X_t.

The estimate of X_t given observations Y_1 to Y_{t-1} is denoted by $\hat{X}_{t|t-1}$ with state covariance matrix $E\{\hat{X}_{t|t-1}\hat{X}_{t|t-1}^T\} = P_{t|t-1}$ while the estimate of X_t given observations Y_1 to Y_t is denoted by $\hat{X}_{t|t}$ with covariance matrix $E\{\hat{X}_{t|t}\hat{X}_{t|t}^T\} = P_{t|t}$.

The update of the estimate, $\hat{X}_{t+1|t}$, from time t to time $t+1$, is computed in two stages.

First, the update equations are:

$$\hat{X}_{t|t} = \hat{X}_{t|t-1} + K_t r_t, \quad P_{t|t} = (I - K_t C_t) P_{t|t-1}$$

where the residual $r_t = Y_t - C_t X_{t|t-1}$ has an associated covariance matrix $H_t = C_t P_{t|t-1} C_t^T + R_t$, and K_t is the Kalman gain matrix with

$$K_t = P_{t|t-1} C_t^T H_t^{-1}.$$

The second stage is the one-step-ahead prediction equations given by:

$$\hat{X}_{t+1|t} = A_t \hat{X}_{t|t}, \quad P_{t+1|t} = A_t P_{t|t} A_t^T + B_t Q_t B_t^T.$$

These two stages can be combined to give the one-step-ahead update-prediction equations:

$$\hat{X}_{t+1|t} = A_t \hat{X}_{t|t-1} + A_t K_t r_t.$$

The above equations thus provide a method for recursively calculating the estimates of the state vectors $\hat{X}_{t|t}$ and $\hat{X}_{t+1|t}$ and their covariance matrices $P_{t|t}$ and $P_{t+1|t}$ from their previous values. This recursive procedure can be viewed in a Bayesian framework as being the updating of the prior by the data Y_t.

The initial values $\hat{X}_{1|0}$ and $P_{1|0}$ are required to start the recursion. For stationary systems, $P_{1|0}$ can be computed from the following equation:

$$P_{1|0} = A_1 P_{1|0} A_1^T + B_1 Q_1 B_1^T,$$

which can be solved by iterating on the equation. For $\hat{X}_{1|0}$ the value $E\{X\}$ can be used if it is available.

2.6.1 Computational methods

To improve the stability of the computations the square root algorithm is used One recursion of the square root covariance filter algorithm which can be summarized as follows:

$$\begin{pmatrix} R_t^{1/2} & C_t S_t & 0 \\ 0 & A_t S_t & B_t Q_t^{1/2} \end{pmatrix} U = \begin{pmatrix} H_t^{1/2} & 0 & 0 \\ G_t & S_{t+1} & 0 \end{pmatrix}$$

where U is an orthogonal transformation triangularizing the left-hand pre-array to produce the right-hand post-array, S_t is the lower triangular Cholesky factor of the state covariance matrix $P_{t+1|t}$, $Q_t^{1/2}$ and $R_t^{1/2}$ are the lower triangular Cholesky factor of the covariance matrices Q and R and $H^{1/2}$ is the lower triangular Cholesky factor of the covariance matrix of the residuals. The relationship between the Kalman gain matrix, K_t, and G_t is given by

$$A_t K_t = G_t \left(H_t^{1/2}\right)^{-1}.$$

To improve the efficiency of the computations when the matrices A_t, B_t and C_t do not vary with time the system can be transformed to give a simpler structure, the transformed state vector is $U^* X$ where U^* is the transformation that reduces the matrix pair (A, C) to lower observer Hessenberg form. That is, the matrix U^* is computed such that the compound matrix,

$$\begin{pmatrix} CU^{*T} \\ U^* A U^{*T} \end{pmatrix}$$

is a lower trapezoidal matrix. The transformations need only be computed once at the start of a series, and the covariance matrices Q_t and R_t can still be time-varying.

2.6.2 Model fitting and forecasting

If the state space model contains unknown parameters, θ, these can be estimated using maximum likelihood. Assuming that W_t and V_t are normal variates the log-likelihood for observations $Y_t, t = 1, 2, \ldots, n$ is given by

$$\text{constant} - \frac{1}{2} \sum_{t=1}^{n} \ln(\det(H_t)) - \frac{1}{2} \sum_{t=1}^{t} r_t^T H_t^{-1} r_t$$

Optimal estimates for the unknown model parameters θ can then be obtained by using a suitable optimizer routine to maximize the likelihood function.

Once the model has been fitted forecasting can be performed by using the one-step-ahead prediction equations. The one-step-ahead prediction equations can also be used to 'jump over' any missing values in the series.

2.6.3 Kalman filter and time series models

Many commonly used time series models can be written as state space models. A univariate ARMA(p,q) model can be cast into the following state space form

$$\begin{aligned} x_t &= Ax_{t-1} + B\epsilon_t \\ w_t &= Cx_t \end{aligned}$$

where $r = \max(p, q+1)$, the first element of the state vector x_t is w_t,

$$A = \begin{pmatrix} \phi_1 & 1 & & & \\ \phi_2 & & 1 & & \\ . & & & . & \\ . & & & & . \\ \phi_{r-1} & & & & 1 \\ \phi_r & 0 & 0 & . & . & 0 \end{pmatrix}, \; B = \begin{pmatrix} 1 \\ -\theta_1 \\ -\theta_2 \\ . \\ . \\ . \\ -\theta_{r-1} \end{pmatrix} \text{ and } C^T = \begin{pmatrix} 1 \\ 0 \\ 0 \\ . \\ . \\ . \\ 0 \end{pmatrix}.$$

The representation for a k-variate ARMA(p,q) series (VARMA) is very similar to that given above, except now the state vector is of length kr and the ϕ's and θ's are now $k \times k$ matrices and the 1's in A, B and C are now the identity matrix of order k. If $p < r$ or $q + 1 < r$ then the appropriate ϕ or θ matrices are set to zero, respectively.

Since the compound matrix

$$\begin{pmatrix} C \\ A \end{pmatrix}$$

is already in lower observer Hessenberg form (i.e., it is lower trapezoidal with zeros in the top right-hand triangle) the invariant Kalman filter algorithm can be used directly without the need to generate a transformation matrix U^*.

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 Time Domain Techniques – ARMA type models

This section is divided into routines for univariate, input-output and multivariate time-series modelling. The term input-output refers to time-series modelling of a single-output series dependent on one or more input series; this is also referred to as transfer function or multi-input modelling. These areas are discussed in relation to the process of model identification, estimation, checking and forecasting.

3.1.1 Univariate Series

(a) Model identification

The routine G13AUF may be used in obtaining either a range–mean or standard deviation–mean plot for a series of observations, which may be useful in detecting the need for a variance-stabilising transformation. G13AUF computes the range or standard deviation and the mean for successive groups of observations and G01AGF may then be used to produce a scatter plot of range against mean or of standard deviation against mean.

The routine G13AAF may be used to difference a time series. The $N = n - d - s \times D$ values of the differenced time series which extends for $t = 1 + d + s \times D, \ldots, n$ are stored in the first N elements of the output array.

The routine G13ABF may be used for direct computation of the autocorrelations. It requires the time series as input, after optional differencing by G13AAF.

An alternative is to use G13CAF, which uses the FFT to carry out the convolution for computing the autocovariances. Circumstances in which this is recommended are

(i) if the main aim is to calculate the smoothed sample spectrum,

(ii) if the series length and maximum lag for the autocorrelations are both very large, in which case appreciable computing time may be saved.

For more precise recommendations, see Gentleman and Sande [3]. In this case the autocorrelations r_k need to be obtained from the autocovariances c_k by $r_k = c_k/c_0$.

The routine G13ACF computes the partial autocorrelation function and prediction error variance estimates from an input autocorrelation function. Note that G13DNF, which is designed for multivariate time series, may also be used to compute the partial autocorrelation function together with χ^2 statistics and their significance levels.

Finite lag predictor coefficients are also computed by the routine G13ACF. It may have to be used twice, firstly with a large value for the maximum lag L in order to locate the optimum FPE lag, then again with L reset to this lag.

The routine G13DXF may be used to check that the autoregressive part of the model is stationary and that the moving-average part is invertible.

(b) Model estimation

The routine G13ADF is used to compute preliminary estimates of the ARIMA model parameters, the sample autocorrelations of the appropriately differenced series being input. The model orders are required.

The main routine for parameter estimation for ARIMA models is G13AEF, and an easy-to-use version is G13AFF. Both these routines use the least-squares criterion of estimation.

In some circumstances the use of G13BEF or G13DCF, which use maximum likelihood, is recommended.

The routines require the time series values to be input, together with the ARIMA orders. Any differencing implied by the model is carried out internally. They also require the maximum number of iterations to be specified, and return the state set for use in forecasting.

G13AEF should be preferred to G13AFF for:

(i) more information about the differenced series, its backforecasts and the intermediate series;

(ii) greater control over the output at successive iterations;

(iii) more detailed control over the search policy of the non-linear least-squares algorithm;

(iv) more information about the first and second derivatives of the objective function during and upon completion of the iterations.

G13BEF is primarily designed for estimating relationships between time series. It is, however, easily used in a univariate mode for ARIMA model estimation. The advantage is that it allows (optional) use of the exact likelihood estimation criterion, which is not available in G13AEF or G13AFF. This is particularly recommended for models which have seasonal parameters, because it reduces the tendency of parameter estimates to become stuck at points on the parameter space boundary. The model parameters estimated in this routine should be passed over to G13AJF for use in univariate forecasting.

The routine G13DCF is primarily designed for fitting vector ARMA models to multivariate time series but may also be used in a univariate mode. It allows the use of either the exact or conditional likelihood estimation criterion, and allows the user to fit non-multiplicative seasonal models which are not available in G13AEF, G13AFF or G13BEF.

(c) Model checking

G13ASF calculates the correlations in the residuals from a model fitted by either G13AEF or G13AFF. In addition the standard errors and correlations of the residual autocorrelations are computed along with a portmanteau test for model adequacy. G13ASF can be used after a univariate model has been fitted by G13BEF, but care must be taken in selecting the correct inputs to G13ASF. Note that if G13DCF has been used to fit a non-multiplicative seasonal model to a univariate series then G13DSF may be used to check the adequacy of the model.

(d) Forecasting using an ARIMA model

Given that the state set produced on estimation of the ARIMA model by either G13AEF or G13AFF has been retained, G13AHF can be used directly to construct forecasts for $x_{n+1}, x_{n+2}, \ldots$, together with probability limits. If some further observations $x_{n+1}, x_{n+2}, \ldots$ have come to hand since model estimation (and there is no desire to re-estimate the model using the extended series), then G13AGF can be used to update the state set using the new observations, prior to forecasting from the end of the extended series. The original series is not required.

The routine G13AJF is provided for forecasting when the ARIMA model is known but the state set is unknown. For example, the model may have been estimated by a procedure other than the use of G13AEF or G13AFF, such as G13BEF. G13AJF constructs the state set and optionally constructs forecasts with probability limits. It is equivalent to a call to G13AEF with zero iterations requested, followed by an optional call to G13AHF, but it is much more efficient.

3.1.2 Input-output/transfer function modelling

(a) Model identification

Normally use G13BCF for direct computation of cross-correlations, from which cross-covariances may be obtained by multiplying by $s_y s_x$, and impulse response estimates (after prewhitening) by multiplying by s_y/s_x, where s_y, s_x are the sample standard deviations of the series.

An alternative is to use G13CCF, which exploits the FFT to carry out the convolution for computing cross-covariances. The criteria for this are the same as given in Section 3.1.1 for calculation of autocorrelations. The impulse response function may also be computed by spectral methods without prewhitening using G13CGF.

G13BAF may be used to prewhiten or filter a series by an ARIMA model.

G13BBF may be used to filter a time series using a transfer function model.

(b) Estimation of input-output model parameters

The routine G13BDF is used to obtain preliminary estimates of transfer function model parameters. The model orders and an estimate of the impulse response function (see Section 3.2.1) are required.

The simultaneous estimation of the transfer function model parameters for the inputs, and ARIMA model parameters for the output, is carried out by G13BEF.

This routine requires values of the output and input series, and the orders of all the models. Any differencing implied by the model is carried out internally.

The routine also requires the maximum number of iterations to be specified, and returns the state set for use in forecasting.

(c) Input-output model checking

The routine G13ASF, primarily designed for univariate time series, can be used to test the residuals from an input-output model.

(d) Forecasting using an input-output model

Given that the state set produced on estimation of the model by G13BEF has been retained, the routine G13BHF can be used directly to construct forecasts of the output series. Future values of the input series (possibly forecasts previously obtained using G13AHF) are required.

If further observations of the output and input series have become available since model estimation (and there is no desire to re-estimate the model using the extended series) then G13BGF can be used to update the state set using the new observations prior to forecasting from the end of the extended series. The original series are not required.

The routine G13BJF is provided for forecasting when the multi-input model is known, but the state set is unknown. The set of output and input series must be supplied to the routine which then constructs the state set (for future use with G13BGF and/or G13BHF) and also optionally constructs forecasts of the output series in a similar manner to G13BHF.

In constructing probability limits for the forecasts, it is possible to allow for the fact that future input series values may themselves have been calculated as forecasts using ARIMA models. Use of this option requires that these ARIMA models be supplied to the routine.

(e) Filtering a time series using a transfer function model

The routine for this purpose is G13BBF.

3.1.3 Multivariate series

(a) Model identification

The routine G13DLF may be used to difference the series. The user must supply the differencing parameters for each component of the multivariate series. The order of differencing for each individual component does not have to be the same. The routine may also be used to apply a log or square root transformation to the components of the series.

The routine G13DMF may be used to calculate the sample cross-correlation or cross-covariance matrices. It requires a set of time series as input. The user may request either the cross-covariances or cross-correlations.

The routine G13DNF computes the partial lag correlation matrices from the sample cross-correlation matrices computed by G13DMF, and the routine G13DPF computes the least-squares estimates of the partial autoregression matrices and their standard errors. Both routines compute a series of χ^2 statistic that aid the determination of the order of a suitable autoregressive model. G13DBF may also be used in the identification of the order of an autoregressive model. The routine computes multiple squared partial autocorrelations and predicitve error variance ratios from the sample cross-correlations or cross-covariances computed by G13DMF.

The routine G13DXF may be used to check that the autoregressive part of the model is stationary and that the moving-average part is invertible.

(b) Estimation of VARMA model parameters

The routine for this purpose is G13DCF. This routine requires a set of time series to be input, together with values for p and q. The user must also specify the maximum number of likelihood evaluations to be permitted and which parameters (if any) are to be held at their initial (user-supplied) values. The fitting criterion is either **exact** maximum likelihood or **conditional** maximum likelihood.

G13DCF is primarily designed for estimating relationships between time series. It may, however, easily be used in univariate mode for non-seasonal and non-multiplicative seasonal ARIMA model estimation. The advantage is that it allows (optional) use of the exact maximum likelihood estimation criterion, which is not available in either G13AEF or G13AFF. The conditional likelihood option is recommended for those models in which the parameter estimates display a tendency to become stuck at points on the boundary of the parameter space. When one of the series is known to be influenced by all the others, but the others in turn are mutually independent and do not influence the output series, then G13BEF (the transfer function model fitting routine) may be more appropriate to use.

(c) VARMA model checking

G13DSF calculates the cross-correlation matrices of residuals for a model fitted by G13DCF. In addition the standard errors and correlations of the residual correlation matrices are computed along with a portmanteau test for model adequacy.

(d) Forecasting using a VARMA model

The routine G13DJF may be used to construct a chosen number of forecasts using the model estimated by G13DCF. The standard errors of the forecasts are also computed. A reference vector is set up by G13DJF so that should any further observations become available the existing forecasts can be efficiently updated using G13DKF. On a call to G13DKF the reference vector itself is also updated so that G13DKF may be called again each time new observations are available.

3.2 Frequency Domain Techniques

3.2.1 Univariate spectral estimation

Two routines are available, G13CAF carrying out smoothing using a lag window and G13CBF carrying out direct frequency domain smoothing. Both can take as input the original series, but G13CAF alone can use the sample autocovariances as alternative input. This has some computational advantage if a variety of spectral estimates needs to be examined for the same series using different amounts of smoothing.

However, the real choice in most cases will be which of the four shapes of lag window in G13CAF to use, or whether to use the trapezium frequency window of G13CBF. The references may be consulted for advice on this, but the two most recommended lag windows are the Tukey and Parzen. The Tukey window has a very small risk of supplying negative spectrum estimates; otherwise, for the same bandwidth, both give very similar results, though the Parzen window requires a higher truncation lag (more acf values).

The frequency window smoothing procedure of G13CBF with a trapezium shape parameter $p \simeq \frac{1}{2}$ generally gives similar results for the same bandwidth as lag window methods with a slight advantage of somewhat less distortion around sharp peaks, but suffering a rather less smooth appearance in fine detail.

3.2.2 Cross-spectrum estimation

Two routines are available for the main step in cross-spectral analysis. To compute the cospectrum and quadrature spectrum estimates using smoothing by a lag window, G13CCF should be used. It takes as input either the original series or cross-covariances which may be computed in a previous call of the same routine or possibly using results from G13BCF. As in the univariate case, this gives some advantage if estimates for the same series are to be computed with different amounts of smoothing.

The choice of window shape will be determined as the same as that which has already been used in univariate spectrum estimation for the series.

For direct frequency domain smoothing, G13CDF should be used, with similar consideration for the univariate estimation in choice of degree of smoothing.

The cross-amplitude and squared coherency spectrum estimates are calculated, together with upper and lower confidence bounds, using G13CEF. For input the cross-spectral estimates from either G13CCF or G13CDF and corresponding univariate spectra from either G13CAF or G13CBF are required.

The gain and phase spectrum estimates are calculated together with upper and lower confidence bounds using G13CFF. The required input is as for G13CEF above.

The noise spectrum estimates and impulse response function estimates are calculated together with multiplying factors for confidence limits on the former, and the standard error for the latter, using G13CGF. The required input is again the same as for G13CEF above.

3.3 Kalman filtering

Two routines are available for Kalman filtering: G13EAF for time varying systems and G13ABF for time invariant systems. The latter will optionally compute the required transformation to lower observer Hessenberg form. Both these routines return the Cholesky factor of the residual covariance matrix, H_t, with the Cholesky factor of the state covariance matrix S_{t+1} and the Kalman gain matrix, K_t pre-multiplied by A_t, in the case of G13EBF these may be for the transformed system. To compute the updated state vector and the residual vector the required matrix-vector multiplications can be performed by F06PAF (SGEMV/DGEMV).

3.4 Time Series Simulation

There are routines available in the G05 chapter for generating a realisation of a time series from a specified model: G05EGF and G05EWF for univariate time series and G05HDF for multivariate time series.

4 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'.

G13DAF

5 References

[1] Akaike H (1971) Autoregressive model fitting for control *Ann. Inst. Statist. Math.* **23** 163–180

[2] Box G E P and Jenkins G M (1976) *Time Series Analysis: Forecasting and Control* Holden-Day (Revised Edition)

[3] Gentleman W S and Sande G (1966) Fast Fourier transforms for fun and profit *Proc. Joint Computer Conference, AFIPS* **29** 563–578

[4] Heyse J F and Wei W W S (1985) The partial lag autocorrelation function *Technical Report No. 32* Department of Statistics, Temple University, Philadelphia

[5] Tiao G C and Box G E P (1981) Modelling multiple time series with applications *J. Am. Stat. Assoc.* **76** 802–816

[6] Wei W W S (1990) *Time Series Analysis: Univariate and Multivariate Methods* Addison-Wesley

Chapter H – Operations Research

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
H02BBF	14	Integer LP problem (dense)
H02BFF	16	Interpret MPSX data file defining IP or LP problem, optimize and print solution
H02BUF	16	Convert MPSX data file defining IP or LP problem to format required by H02BBF or E04MFF
H02BVF	16	Print IP or LP solutions with user specified names for rows and columns
H02BZF	15	Integer programming solution, supplies further information on solution obtained by H02BBF
H02CBF	19	Integer QP problem (dense)
H02CCF	19	Read optional parameter values for H02CBF from external file
H02CDF	19	Supply optional parameter values to H02CBF
H02CEF	19	Integer LP or QP problem (sparse)
H02CFF	19	Read optional parameter values for H02CEF from external file
H02CGF	19	Supply optional parameter values to H02CEF
H03ABF	4	Transportation problem, modified 'stepping stone' method
H03ADF	18	Shortest path problem, Dijkstra's algorithm

Chapter H

Operations Research

Contents

1 Scope of the Chapter

This chapter provides routines to solve certain integer programming, transportation and shortest path problems.

2 Background to the Problems

General **linear programming** (LP) problems (see Dantzig [2]) are of the form:

find $x = (x_1, x_2, \ldots, x_n)^T$ to maximize $F(x) = \sum_{j=1}^{n} c_j x_j$

subject to linear constraints which may have the forms:

$$\begin{array}{lll}
\sum_{j=1}^{n} a_{ij} x_j = b_i, & i = 1, 2, \ldots, m_1 & \text{(equality)} \\
\sum_{j=1}^{n} a_{ij} x_j \leq b_i, & i = m_1 + 1, \ldots, m_2 & \text{(inequality)} \\
\sum_{j=1}^{n} a_{ij} x_j \geq b_i, & i = m_2 + 1, \ldots, m & \text{(inequality)} \\
x_j \geq l_j, & j = 1, 2, \ldots, n & \text{(simple bound)} \\
x_j \leq u_j, & j = 1, 2, \ldots, n & \text{(simple bound)}
\end{array}$$

This chapter deals with **integer programming** (IP) problems in which some or all the elements of the solution vector x are further constrained to be **integers**. For general LP problems where x takes only real (i.e., non-integer) values, refer to Chapter E04.

IP problems may or may not have a solution, which may or may not be unique.

Consider for example the following problem:

$$\begin{array}{lr}
\text{minimize} & 3x_1 + 2x_2 \\
\text{subject to} & 4x_1 + 2x_2 \geq 5 \\
 & 2x_2 \leq 5 \\
 & x_1 - x_2 \leq 2 \\
\text{and} & x_1 \geq 0,\ x_2 \geq 0.
\end{array}$$

The hatched area in Figure 1 is the **feasible region**, the region where all the constraints are satisfied, and the points within it which have integer co-ordinates are circled. The lines of hatching are in fact contours of decreasing values of the objective function $3x_1 + 2x_2$, and it is clear from Figure 1 that the optimum IP solution is at the point (1,1). For this problem the solution is unique.

However, there are other possible situations:

(a) there may be more than one solution; e.g., if the objective function in the above problem were changed to $x_1 + x_2$, both (1,1) and (2,0) would be IP solutions.

(b) the feasible region may contain no points with integer co-ordinates, e.g., if an additional constraint

$$3x_1 \leq 2$$

were added to the above problem.

(c) there may be no feasible region, e.g., if an additional constraint

$$x_1 + x_2 \leq 1$$

were added to the above problem.

(d) the objective function may have no finite minimum within the feasible region; this means that the feasible region is unbounded in the direction of decreasing values of the objective function, e.g., if the constraints

$$4x_1 + 2x_2 \geq 5, \quad x_1 \geq 0, \quad x_2 \geq 0,$$

were deleted from the above problem.

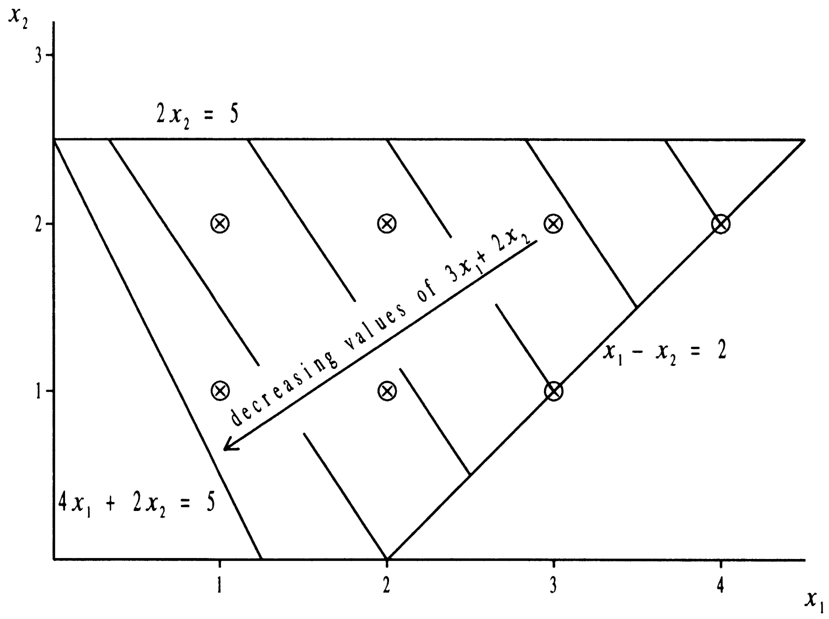


Figure 1

Algorithms for IP problems are usually based on algorithms for general LP problems, together with some procedure for constructing additional constraints which exclude non-integer solutions (see Beale [1]).

The Branch and Bound (B&B) method is a well-known and widely used technique for solving IP problems (see Beale [1] or Mitra [3]). It involves subdividing the optimum solution to the original LP problem into two mutually exclusive sub-problems by branching an integer variable that currently has a fractional optimal value. Each sub-problem can now be solved as an LP problem, using the objective function of the original problem. The process of branching continues until a solution for one of the sub-problems is feasible with respect to the integer problem. In order to prove the optimality of this solution, the rest of the sub-problems in the B&B tree must also be solved. Naturally, if a better integer feasible solution is found for any sub-problem, it should replace the one at hand.

A common method for specifying IP and LP problems in general is the use of the MPSX file format (see [4]). A full description of this file format is provided in the routine documents for H02BUF and E04MZF.

The efficiency in computations is enhanced by discarding inferior sub-problems. These are problems in the B&B search tree whose LP solutions are lower than (in the case of maximization) the best integer solution at hand.

The B&B method may also be applied to convex quadratic programming (QP) problems. Routines have been introduced into this chapter to formally apply the technique to dense general QP problems and to sparse LP or QP problems.

A special type of linear programming problem is the **transportation** problem in which there are $p \times q$ variables y_{kl} which represent quantities of goods to be transported from each of p sources to each of q destinations.

The problem is to minimize

$$\sum_{k=1}^{p}\sum_{l=1}^{q} c_{kl} y_{kl}$$

where c_{kl} is the unit cost of transporting from source k to destination l. The constraints are:

$$\sum_{l=1}^{q} y_{kl} = A_k \quad \text{(availabilities)}$$

$$\sum_{k=1}^{p} y_{kl} = B_l \quad \text{(requirements)}$$

$$y_{kl} \geq 0.$$

Note that the availabilities must equal the requirements:

$$\sum_{k=1}^{p} A_k = \sum_{l=1}^{q} B_l = \sum_{k=1}^{p}\sum_{l=1}^{q} y_{kl}$$

and if all the A_k and B_l are integers, then so are the optimal y_{kl}.

The **shortest path** problem is that of finding a path of minimum length between two distinct vertices n_s and n_e through a network. Suppose the vertices in the network are labelled by the integers $1, 2, \ldots, n$. Let (i,j) denote an ordered pair of vertices in the network (where i is the origin vertex and j the destination vertex of the arc), x_{ij} the amount of flow in arc (i,j) and d_{ij} the length of the arc (i,j). The LP formulation of the problem is thus given as

$$\text{minimize } \sum\sum d_{ij} x_{ij} \text{ subject to } Ax = b,\ \ 0 \le x \le 1, \tag{1}$$

where

$$a_{ij} = \begin{cases} +1 & \text{if arc } j \text{ is directed away from vertex } i, \\ -1 & \text{if arc } j \text{ is directed towards vertex } i, \\ 0 & \text{otherwise} \end{cases}$$

and

$$b_i = \begin{cases} +1 & \text{for } i = n_s, \\ -1 & \text{for } i = n_e, \\ 0 & \text{otherwise.} \end{cases}$$

The above formulation only yields a meaningful solution if $x_{ij} = 0$ or 1; that is, arc (i,j) forms part of the shortest route only if $x_{ij} = 1$. In fact since the optimal LP solution will (in theory) always yield $x_{ij} = 0$ or 1, (1) can also be solved as an IP problem. Note that the problem may also be solved directly (and more efficiently) using a variant of Dijkstra's algorithm (see [6]).

The **travelling salesman** problem is that of finding a minimum distance route round a given set of cities. The salesperson must visit each city only once before returning to his or her city of origin. It can be formulated as an IP problem in a number of ways. One such formulation is described in Williams [5]. There are currently no routines in the Library for solving such problems.

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.

H02BBF solves dense integer programming problems using a branch and bound method.

H02BFF solves dense integer or linear programming problems defined by a MPSX data file.

H02BUF converts an MPSX data file defining an integer or a linear programming problem to the form required by H02BBF or E04MFF.

H02BVF prints the solution to an integer or a linear programming problem using specified names for rows and columns.

H02BZF supplies further information on the optimum solution obtained by H02BBF.

H02CBF solves dense integer general quadratic programming problems.

H02CCF reads optional parameter values for H02CBF from external file.

H02CDF supplies optional parameter values to H02CBF.

H02CEF solves sparse integer linear programming or quadratic programming problems.

H02CFF reads optional parameter values for H02CEF from external file.

H02CGF supplies optional parameter values to H02CEF.

H03ABF solves transportation problems. It uses integer arithmetic throughout and so produces exact results. On a few machines, however, there is a risk of integer overflow without warning, so the integer values in the data should be kept as small as possible by dividing out any common factors from the coefficients of the constraint or objective functions.

H03ADF solves shortest path problems using Dijkstra's algorithm.

H02BBF, H02BFF and H03ABF treat all matrices as dense and hence are not intended for large sparse problems. For solving large sparse LP problems, use E04NKF or E04UGF.

4 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'.

H02BAF

5 References

[1] Beale E M (1977) Integer Programming *The State of the Art in Numerical Analysis* (ed D A H Jacobs) Academic Press

[2] Dantzig G B (1963) *Linear Programming and Extensions* Princeton University Press

[3] Mitra G (1973) Investigation of some branch and bound strategies for the solution of mixed integer linear programs *Math. Programming* **4** 155–170

[4] (1971) MPSX – Mathematical programming system *Program Number 5734 XM4* IBM Trade Corporation, New York

[5] Williams H P (1990) *Model Building in Mathematical Programming* (3rd Edition) Wiley

[6] Ahuja R K, Magnanti T L and Orlin J B (1993) *Network Flows: Theory, Algorithms, and Applications* Prentice Hall

---

Chapter M01 – Sorting

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
M01CAF	12	Sort a vector, real numbers
M01CBF	12	Sort a vector, integer numbers
M01CCF	12	Sort a vector, character data
M01DAF	12	Rank a vector, real numbers
M01DBF	12	Rank a vector, integer numbers
M01DCF	12	Rank a vector, character data
M01DEF	12	Rank rows of a matrix, real numbers
M01DFF	12	Rank rows of a matrix, integer numbers
M01DJF	12	Rank columns of a matrix, real numbers
M01DKF	12	Rank columns of a matrix, integer numbers
M01DZF	12	Rank arbitrary data
M01EAF	12	Rearrange a vector according to given ranks, real numbers
M01EBF	12	Rearrange a vector according to given ranks, integer numbers
M01ECF	12	Rearrange a vector according to given ranks, character data
M01EDF	19	Rearrange a vector according to given ranks, complex numbers
M01ZAF	12	Invert a permutation
M01ZBF	12	Check validity of a permutation
M01ZCF	12	Decompose a permutation into cycles

Chapter M01

Sorting

Contents

1 Scope of the Chapter

This chapter is concerned with sorting numeric or character data. It handles only the simplest types of data structure and it is concerned only with **internal** sorting – that is, sorting a set of data which can all be stored within the program.

Users with large files of data or complicated data structures to be sorted should use a comprehensive sorting program or package.

2 Background to the Problems

The usefulness of sorting is obvious (perhaps a little too obvious, since sorting can be expensive and is sometimes done when not strictly necessary). Sorting may traditionally be associated with data processing and non-numerical programming, but it has many uses within the realm of numerical analysis. For example, within the NAG Fortran Library, sorting is used to arrange eigenvalues in ascending order of absolute value; in the manipulation of sparse matrices and in the ranking of observations for nonparametric statistics.

The general problem may be defined as follows. We are given N items of data

$$R_1, R_2, \ldots, R_N.$$

Each item R_i contains a key K_i which can be ordered relative to any other key according to some specified criterion (for example, ascending numeric value). The problem is to determine a permutation

$$p(1), p(2), \ldots, p(N)$$

which puts the keys in order:

$$K_{p(1)} \leq K_{p(2)} \leq \cdots \leq K_{p(N)}$$

Sometimes we may wish actually to **rearrange** the items so that their keys are in order; for other purposes we may simply require a table of **indices** so that the items can be referred to in sorted order; or yet again we may require a table of **ranks**, that is, the positions of each item in the sorted order.

For example, given the single-character items, to be sorted into alphabetic order:

E B A D C

the indices of the items in sorted order are

3 2 5 4 1

and the ranks of the items are

5 2 1 4 3.

Indices may be converted to ranks, and vice versa, by simply computing the inverse permutation.

The items may consist solely of the key (each item may simply be a number). On the other hand, the items may contain additional information (for example, each item may be an eigenvalue of a matrix and its associated eigenvector, the eigenvalue being the key). In the latter case there may be many distinct items with equal keys, and it may be important to preserve the original order among them (if this is achieved, the sorting is called '**stable**').

There are a number of ingenious algorithms for sorting. For a fascinating discussion of them, and of the whole subject, see Knuth [1].

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.

Four categories of routines are provided:

– routines which rearrange the data into sorted order (M01C-);

– routines which determine the ranks of the data, leaving the data unchanged (M01D-);

– routines which rearrange the data according to pre-determined ranks (M01E-);

– service routines (M01Z-).

In the first two categories, routines are provided for ***real*** and integer numeric data, and for character data. In the third category there are routines for rearranging ***real***, ***complex***, integer and character data. Utilities for the manipulation of sparse matrices can be found in Chapter F11.

If the task is simply to rearrange a one-dimensional array of data into sorted order, then an M01C- routine should be used, since this requires no extra workspace and is faster than any other method. There are no M01C- routines for more complicated data structures, because the cost of rearranging the data is likely to outstrip the cost of comparisons. Instead, a combination of M01D- and M01E- routines, or some other approach, must be used as described below.

For many applications it is in fact preferable to separate the task of determining the sorted order (ranking) from the task of rearranging data into a pre-determined order; the latter task may not need to be performed at all. Frequently it may be sufficient to refer to the data in sorted order via an index vector, without rearranging it. Frequently also one set of data (e.g. a column of a matrix) is used for determining a set of ranks, which are then applied to other data (e.g. the remaining columns of the matrix).

To determine the ranks of a set of data, use an M01D- routine. Routines are provided for ranking one-dimensional arrays, and for ranking rows or columns of two-dimensional arrays. For ranking an arbitrary data structure, use M01DZF, which is, however, much less efficient than the other M01D- routines.

To create an index vector so that data can be referred to in sorted order, first call an M01D- routine to determine the ranks, and then call M01ZAF to convert the vector of ranks into an index vector.

To rearrange data according to pre-determined ranks: use an M01E- routine if the data is stored in a one-dimensional array; or if the data is stored in a more complicated structure

either use an index vector to generate a new copy of the data in the desired order

or rearrange the data without using extra storage by first calling M01ZCF and then using the simple code-framework given in the document for M01ZCF (assuming that the elements of data all occupy equal storage).

Examples of these operations can be found in the routine documents of the relevant routines.

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'.

M01AAF	M01ABF	M01ACF	M01ADF	M01AEF	M01AFF
M01AGF	M01AHF	M01AJF	M01AKF	M01ALF	M01AMF
M01ANF	M01APF	M01AQF	M01ARF	M01BAF	M01BBF
M01BCF	M01BDF				

6 References

[1] Knuth D E (1973) *The Art of Computer Programming (Volume 3)* Addison–Wesley (2nd Edition)

Chapter P01 – Error Trapping

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
P01ABF	12	Return value of error indicator/terminate with error message

Chapter P01

Error Trapping

Contents

1 Scope of the Chapter

This chapter is concerned with the trapping of error, failure or warning conditions by NAG Library routines. This introduction document describes the commonly occurring parameter IFAIL.

2 Background to the Problems

2.1 Errors, Failure and Warning Conditions

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

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

(i) On entry to the routine the value of a parameter is out of range. This means that it is not useful, or perhaps even meaningful, to begin computation.
(ii) During computation the routine decides that it cannot yield the desired results, and indicates a failure condition. For example, a matrix inversion routine will indicate a failure condition if it considers that the matrix is singular and so cannot be inverted.
(iii) Although the routine completes the computation and returns results, it cannot guarantee that the results are completely reliable; it therefore returns a warning. For example, an optimization routine may return a warning if it cannot guarantee that it has found a local minimum.

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

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

2.2 The IFAIL Parameter

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

IFAIL has **two** purposes:

(i) to allow the user to specify what action the Library routine should take if an error is detected;
(ii) to inform the user of the outcome of the call of the routine.

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

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

2.3 Hard Fail Option

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

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: IFAIL = n

** NAG hard failure - execution terminated
```

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

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

In some implementations of the NAG Library, when the hard fail option is invoked, the error message may be accompanied by dump or tracing information. The output channel used for the output of the error message is determined by X04AAF.

The hard fail option should be selected if the user is in any doubt about continuing the execution of the program after an unsuccessful call to a NAG Library routine.

2.4 Soft Fail Option

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

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

If the user sets IFAIL to 1, then no error message is output (**silent exit**).

If the user sets IFAIL to −1 (**noisy exit**), then before control is returned to the calling program, the following error message is output:

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: IFAIL = n

** NAG soft failure - control returned
```

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

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

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

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

(ii) exceptionally, certain routine documents may advise further calls with IFAIL left with its value on exit after the first call of the routine. In such cases the user should not reset the IFAIL-exit value between calls;

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

2.5 Historical Note

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

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

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

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

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

To implement the error mechanism described in Section 2, NAG Library routines call P01ABF.

This routine is therefore primarily of interest only to writers of NAG Fortran Library software. It is included in the general user manual for completeness. Users need not know how to call P01ABF directly though they may be aware of its existence.

Chapter S – Approximations of Special Functions

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
S01BAF	14	$\ln(1+x)$
S01EAF	14	Complex exponential, e^z
S07AAF	1	$\tan x$
S09AAF	1	$\arcsin x$
S09ABF	3	$\arccos x$
S10AAF	3	$\tanh x$
S10ABF	4	$\sinh x$
S10ACF	4	$\cosh x$
S11AAF	4	$\mathrm{arctanh} x$
S11ABF	4	$\mathrm{arcsinh} x$
S11ACF	4	$\mathrm{arccosh} x$
S13AAF	1	Exponential integral $E_1(x)$
S13ACF	2	Cosine integral $\mathrm{Ci}(x)$
S13ADF	5	Sine integral $\mathrm{Si}(x)$
S14AAF	1	Gamma function
S14ABF	8	Log Gamma function
S14ACF	14	$\psi(x) - \ln x$
S14ADF	14	Scaled derivatives of $\psi(x)$
S14BAF	14	Incomplete Gamma functions $P(a,x)$ and $Q(a,x)$
S15ABF	3	Cumulative normal distribution function $P(x)$
S15ACF	4	Complement of cumulative normal distribution function $Q(x)$
S15ADF	4	Complement of error function $\mathrm{erfc}(x)$
S15AEF	4	Error function $\mathrm{erf}(x)$
S15AFF	7	Dawson's integral
S15DDF	14	Scaled complex complement of error function, $\exp(-z^2)\mathrm{erfc}(-iz)$
S17ACF	1	Bessel function $Y_0(x)$
S17ADF	1	Bessel function $Y_1(x)$
S17AEF	5	Bessel function $J_0(x)$
S17AFF	5	Bessel function $J_1(x)$
S17AGF	8	Airy function $\mathrm{Ai}(x)$
S17AHF	8	Airy function $\mathrm{Bi}(x)$
S17AJF	8	Airy function $\mathrm{Ai}'(x)$
S17AKF	8	Airy function $\mathrm{Bi}'(x)$
S17DCF	13	Bessel functions $Y_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0,1,2,\ldots$
S17DEF	13	Bessel functions $J_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0,1,2,\ldots$
S17DGF	13	Airy functions $\mathrm{Ai}(z)$ and $\mathrm{Ai}'(z)$, complex z
S17DHF	13	Airy functions $\mathrm{Bi}(z)$ and $\mathrm{Bi}'(z)$, complex z
S17DLF	13	Hankel functions $H^{(j)}_{\nu+a}(z)$, $j = 1,2$, real $a \geq 0$, complex z, $\nu = 0,1,2,\ldots$
S18ACF	1	Modified Bessel function $K_0(x)$
S18ADF	1	Modified Bessel function $K_1(x)$
S18AEF	5	Modified Bessel function $I_0(x)$
S18AFF	5	Modified Bessel function $I_1(x)$
S18CCF	10	Modified Bessel function $e^x K_0(x)$
S18CDF	10	Modified Bessel function $e^x K_1(x)$
S18CEF	10	Modified Bessel function $e^{-\lvert x\rvert} I_0(x)$
S18CFF	10	Modified Bessel function $e^{-\lvert x\rvert} I_1(x)$
S18DCF	13	Modified Bessel functions $K_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0,1,2,\ldots$
S18DEF	13	Modified Bessel functions $I_{\nu+a}(z)$, real $a \geq 0$, complex z, $\nu = 0,1,2,\ldots$
S19AAF	11	Kelvin function $\mathrm{ber}\ x$

S19ABF	11	Kelvin function bei x
S19ACF	11	Kelvin function ker x
S19ADF	11	Kelvin function kei x
S20ACF	5	Fresnel integral $S(x)$
S20ADF	5	Fresnel integral $C(x)$
S21BAF	8	Degenerate symmetrised elliptic integral of 1st kind $R_C(x,y)$
S21BBF	8	Symmetrised elliptic integral of 1st kind $R_F(x,y,z)$
S21BCF	8	Symmetrised elliptic integral of 2nd kind $R_D(x,y,z)$
S21BDF	8	Symmetrised elliptic integral of 3rd kind $R_J(x,y,z,r)$
S21CAF	15	Jacobian elliptic functions sn, cn and dn

Chapter S

Approximations of Special Functions

Contents

1 Scope of the Chapter

This chapter is concerned with the provision of some commonly occurring physical and mathematical functions.

2 Background to the Problems

The majority of the routines in this chapter approximate real-valued functions of a single real argument, and the techniques involved are described in Section 2.1. In addition the chapter contains routines for elliptic integrals (see Section 2.2), Bessel and Airy functions of a complex argument (see Section 2.3), exponential of a complex argument, and complementary error function of a complex argument.

2.1 Functions of a Single Real Argument

Most of the routines for functions of a single real argument have been based on truncated Chebyshev expansions. This method of approximation was adopted as a compromise between the conflicting requirements of efficiency and ease of implementation on many different machine ranges. For details of the reasons behind this choice and the production and testing procedures followed in constructing this chapter see Schonfelder [7].

Basically, if the function to be approximated is $f(x)$, then for $x \in [a,b]$ an approximation of the form

$$f(x) = g(x) \sum_{r=0}{}' C_r T_r(t)$$

is used ($\sum'$ denotes, according to the usual convention, a summation in which the first term is halved), where $g(x)$ is some suitable auxiliary function which extracts any singularities, asymptotes and, if possible, zeros of the function in the range in question and $t = t(x)$ is a mapping of the general range $[a,b]$ to the specific range $[-1,+1]$ required by the Chebyshev polynomials, $T_r(t)$. For a detailed description of the properties of the Chebyshev polynomials see Clenshaw [5] and Fox and Parker [6].

The essential property of these polynomials for the purposes of function approximation is that $T_n(t)$ oscillates between ± 1 and it takes its extreme values $n+1$ times in the interval $[-1,+1]$. Therefore, provided the coefficients C_r decrease in magnitude sufficiently rapidly the error made by truncating the Chebyshev expansion after n terms is approximately given by

$$E(t) \simeq C_n T_n(t).$$

That is, the error oscillates between $\pm C_n$ and takes its extreme value $n+1$ times in the interval in question. Now this is just the condition that the approximation be a mini-max representation, one which minimizes the maximum error. By suitable choice of the interval, $[a,b]$, the auxiliary function, $g(x)$, and the mapping of the independent variable, $t(x)$, it is almost always possible to obtain a Chebyshev expansion with rapid convergence and hence truncations that provide near mini-max polynomial approximations to the required function. The difference between the true mini-max polynomial and the truncated Chebyshev expansion is seldom sufficiently great enough to be of significance.

The evaluation of the Chebyshev expansions follows one of two methods. The first and most efficient, and hence the most commonly used, works with the equivalent simple polynomial. The second method, which is used on the few occasions when the first method proves to be unstable, is based directly on the truncated Chebyshev series, and uses backward recursion to evaluate the sum. For the first method, a suitably truncated Chebyshev expansion (truncation is chosen so that the error is less than the ***machine precision***) is converted to the equivalent simple polynomial. That is, we evaluate the set of coefficients b_r such that

$$y(t) = \sum_{r=0}^{n-1} b_r t^r = \sum_{r=0}^{n-1}{}' C_r T_r(t).$$

The polynomial can then be evaluated by the efficient Horner's method of nested multiplications,

$$y(t) = (b_0 + t(b_1 + t(b_2 + \ldots t(b_{n-2} + tb_{n-1}))) \ldots).$$

This method of evaluation results in efficient routines but for some expansions there is considerable loss of accuracy due to cancellation effects. In these cases the second method is used. It is well known that if

$$\begin{aligned} b_{n-1} &= C_{n-1} \\ b_{n-2} &= 2tb_{n-1} + C_{n-2} \\ b_j &= 2tb_{j+1} - b_{j+2} + C_j, \quad j = n-3, n-4, \ldots, 0 \end{aligned}$$

then

$$\sum_{r=0}{}' C_r T_r(t) = \tfrac{1}{2}(b_0 - b_2)$$

and this is always stable. This method is most efficiently implemented by using three variables cyclically and explicitly constructing the recursion.

That is,

$$\begin{aligned} \alpha &= C_{n-1} \\ \beta &= 2t\alpha + C_{n-2} \\ \gamma &= 2t\beta - \alpha + C_{n-3} \\ \alpha &= 2t\gamma - \beta + C_{n-4} \\ \beta &= 2t\alpha - \gamma + C_{n-5} \\ &\ldots \\ &\ldots \\ \text{say } \alpha &= 2t\gamma - \beta + C_2 \\ \beta &= 2t\alpha - \gamma + C_1 \\ y(t) &= t\beta - \alpha + \tfrac{1}{2}C_0 \end{aligned}$$

The auxiliary functions used are normally functions compounded of simple polynomial (usually linear) factors extracting zeros, and the primary compiler-provided functions, sin, cos, ln, exp, sqrt, which extract singularities and/or asymptotes or in some cases basic oscillatory behaviour, leaving a smooth well-behaved function to be approximated by the Chebyshev expansion which can therefore be rapidly convergent.

The mappings of $[a, b]$ to $[-1, +1]$ used range from simple linear mappings to the case when b is infinite, and considerable improvement in convergence can be obtained by use of a bilinear form of mapping. Another common form of mapping is used when the function is even; that is, it involves only even powers in its expansion. In this case an approximation over the whole interval $[-a, a]$ can be provided using a mapping $t = 2\,(x/a)^2 - 1$. This embodies the evenness property but the expansion in t involves all powers and hence removes the necessity of working with an expansion with half its coefficients zero.

For many of the routines an analysis of the error in principle is given, namely, if E and ∇ are the absolute errors in function and argument and ϵ and δ are the corresponding relative errors, then

$$\begin{aligned} E &\simeq |f'(x)|\nabla \\ E &\simeq |xf'(x)|\delta \\ \epsilon &\simeq \left|\frac{xf'(x)}{f(x)}\right|\delta. \end{aligned}$$

If we ignore errors that arise in the argument of the function by propagation of data errors etc., and consider only those errors that result from the fact that a real number is being represented in the computer in floating-point form with finite precision, then δ is bounded and this bound is independent of the magnitude of x. For example, on an 11-digit machine

$$|\delta| \leq 10^{-11}.$$

(This of course implies that the absolute error $\nabla = x\delta$ is also bounded but the bound is now dependent on x.) However, because of this the last two relations above are probably of more interest. If possible the relative error propagation is discussed; that is, the behaviour of the error amplification factor $|xf'(x)/f(x)|$ is described, but in some cases, such as near zeros of the function which cannot be extracted explicitly, absolute error in the result is the quantity of significance and here the factor $|xf'(x)|$ is described. In general, testing of the functions has shown that their error behaviour follows fairly well these theoretical error behaviours. In regions where the error amplification factors are less than or of the order of one, the errors are slightly larger than the above predictions. The errors are here limited largely by the finite precision of arithmetic in the machine, but ϵ is normally no more than a few times greater than the bound on δ. In regions where the amplification factors are large, of order ten or greater, the theoretical analysis gives a good measure of the accuracy obtainable.

It should be noted that the definitions and notations used for the functions in this chapter are all taken from Abramowitz and Stegun [1]. Users are strongly recommended to consult this book for details before using the routines in this chapter.

2.2 Approximations to Elliptic Integrals

The functions provided here are symmetrised variants of the classic elliptic integrals. These alternative definitions have been suggested by Carlson (see [2], [3] and [4]) and he also developed the basic algorithms used in this chapter.

The standard integral of the first kind is represented by

$$R_F(x,y,z) = \frac{1}{2}\int_0^\infty \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}},$$

where $x, y, z \geq 0$ and at most one may be equal to zero.

The normalisation factor, $\frac{1}{2}$, is chosen so as to make

$$R_F(x,x,x) = 1/\sqrt{x}.$$

If any two of the variables are equal, R_F degenerates into the second function

$$R_C(x,y) = R_F(x,y,y) = \frac{1}{2}\int_0^\infty \frac{dt}{\sqrt{t+x}(t+y)},$$

where the argument restrictions are now $x \geq 0$ and $y \neq 0$.

This function is related to the logarithm or inverse hyperbolic functions if $0 < y < x$, and to the inverse circular functions if $0 \leq x \leq y$.

The integrals of the second kind are defined by

$$R_D(x,y,z) = \frac{3}{2}\int_0^\infty \frac{dt}{\sqrt{(t+x)(t+y)(t+z)^3}}$$

with $z > 0$, $x \geq 0$ and $y \geq 0$, but only one of x or y may be zero.

The function is a degenerate special case of the integral of the third kind

$$R_J(x,y,z,\rho) = \frac{3}{2}\int_0^\infty \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}(t+\rho)}$$

with $\rho \neq 0$ and $x, y, z \geq 0$ with at most one equality holding. Thus $R_D(x,y,z) = R_J(x,y,z,z)$. The normalisation of both these functions is chosen so that

$$R_D(x,x,x) = R_J(x,x,x,x) = 1/(x\sqrt{x}).$$

The algorithms used for all these functions are based on duplication theorems. These allow a recursion system to be established which constructs a new set of arguments from the old using a combination of arithmetic and geometric means. The value of the function at the original arguments can then be simply related to the value at the new arguments. These recursive reductions are used until the arguments differ from the mean by an amount small enough for a Taylor series about the mean to give sufficient accuracy when retaining terms of order less than six. Each step of the recurrences reduces the difference from the mean by a factor of four, and as the truncation error is of order six, the truncation error goes like $(4096)^{-n}$, where n is the number of iterations.

The above forms can be related to the more traditional canonical forms (see Abramowitz and Stegun [1], 17.2).

If we write $q = \cos^2\phi$, $r = 1 - m.\sin^2\phi$, $s = 1 + n.\sin^2\phi$, where $0 < \phi \leq \frac{1}{2}\pi$, we have:

the elliptic integral of the first kind:

$$F(\phi|m) = \int_0^{\sin\phi} (1-t^2)^{-1/2}(1-mt^2)^{-1/2}dt = \sin\phi.R_F(q,r,1);$$

the elliptic integral of the second kind:

$$\begin{aligned} E(\phi|m) &= \int_0^{\sin\phi} (1-t^2)^{-1/2}(1-mt^2)^{1/2}dt \\ &= \sin\phi.R_F(q,r,1) - \tfrac{1}{3}m.\sin^3\phi.R_D(q,r,1) \end{aligned}$$

the elliptic integral of the third kind:

$$\begin{aligned} \Pi(n;\phi|m) &= \int_0^{\sin\phi} (1-t^2)^{-1/2}(1-mt^2)^{-1/2}(1+nt^2)^{-1}dt \\ &= \sin\phi.R_F(q,r,1) - \tfrac{1}{3}n.\sin^3\phi.R_J(q,r,1,s). \end{aligned}$$

Also the complete elliptic integral of the first kind:

$$K(m) = \int_0^{\pi/2} (1 - m.\sin^2\theta)^{-1/2}d\theta = R_F(0, 1-m, 1);$$

the complete elliptic integral of the second kind:

$$E(m) = \int_0^{\pi/2} (1 - m.\sin^2\theta)^{1/2}d\theta = R_F(0, 1-m, 1) - \tfrac{1}{3}m.R_D(0, 1-m, 1).$$

2.3 Bessel and Airy Functions of a Complex Argument

The routines for Bessel and Airy functions of a real argument are based on Chebyshev expansions, as described in Section 2.1. The routines for functions of a complex argument, however, use different methods. These routines relate all functions to the modified Bessel functions $I_\nu(z)$ and $K_\nu(z)$ computed in the right-half complex plane, including their analytic continuations. I_ν and K_ν are computed by different methods according to the values of z and ν. The methods include power series, asymptotic expansions and Wronskian evaluations. The relations between functions are based on well known formulae (see Abramowitz and Stegun [1]).

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 Elliptic Integrals

IMPORTANT ADVICE: users who encounter elliptic integrals in the course of their work are strongly recommended to look at transforming their analysis directly to one of the Carlson forms, rather than to the traditional canonical Legendre forms. In general, the extra symmetry of the Carlson forms is likely to simplify the analysis, and these symmetric forms are much more stable to calculate.

The routine S21BAF for R_C is largely included as an auxiliary to the other routines for elliptic integrals. This integral essentially calculates elementary functions, e.g.

$$\begin{aligned} \ln x &= (x-1).R_C\left(\left(\frac{1+x}{2}\right)^2, x\right), \quad x>0; \\ \arcsin x &= x.R_C(1-x^2,1), \ |x| \le 1; \\ \operatorname{arcsinh} x &= x.R_C(1+x^2,1), \text{ etc.} \end{aligned}$$

In general this method of calculating these elementary functions is not recommended as there are usually much more efficient specific routines available in the Library. However, S21BAF may be used, for example, to compute $\ln x/(x-1)$ when x is close to 1, without the loss of significant figures that occurs when $\ln x$ and $x-1$ are computed separately.

3.2 Bessel and Airy Functions

For computing the Bessel functions $J_\nu(x)$, $Y_\nu(x)$, $I_\nu(x)$ and $K_\nu(x)$ where x is real and $\nu = 0$ or 1, special routines are provided, which are much faster than the more general routines that allow a complex argument and arbitrary real $\nu \geq 0$. Similarly, special routines are provided for computing the Airy functions and their derivatives $\mathrm{Ai}(x)$, $\mathrm{Bi}(x)$, $\mathrm{Ai}'(x)$, $\mathrm{Bi}'(x)$ for a real argument which are much faster than the routines for complex arguments.

4 Index

Error function, real argument	**S15AEF**
Exponential, complex	**S01EAF**
Exponential Integral	**S13AAF**
Fresnel Integral, C	**S20ADF**
Fresnel Integral, S	**S20ACF**
Gamma function	**S14AAF**
Gamma function, incomplete	**S14BAF**
Generalized Factorial function	**S14AAF**
Hankel function $H_\nu^{(1)}$ or $H_\nu^{(2)}$, complex argument, optionally scaled	**S17DLF**
Incomplete Gamma function	**S14BAF**
Jacobian elliptic functions, sn, cn, dn	**S21CAF**
Kelvin function, $\mathrm{bei}\, x$	**S19ABF**
Kelvin function, $\mathrm{ber}\, x$	**S19AAF**
Kelvin function, $\mathrm{kei}\, x$	**S19ADF**
Kelvin function, $\mathrm{ker}\, x$	**S19ACF**
Logarithm of Gamma function	**S14ABF**
Logarithm of $1+x$	**S01BAF**
Modified Bessel function, I_0, real argument	**S18AEF**
Modified Bessel function, I_1, real argument	**S18AFF**
Modified Bessel function, I_ν, complex argument, optionally scaled	**S18DEF**
Modified Bessel function, K_0, real argument	**S18ACF**
Modified Bessel function, K_1, real argument	**S18ADF**
Modified Bessel function, K_ν, complex argument, optionally scaled	**S18DCF**
Psi function	**S14ACF**
Psi function and derivatives, scaled	**S14ADF**
Scaled modified Bessel function, $e^{-\lvert x\rvert} I_0(x)$, real argument	**S18CEF**
Scaled modified Bessel function, $e^{-\lvert x\rvert} I_1(x)$, real argument	**S18CFF**
Scaled modified Bessel function, $e^{x} K_0(x)$, real argument	**S18CCF**
Scaled modified Bessel function, $e^{x} K_1(x)$, real argument	**S18CDF**
Sine, hyperbolic	**S10ABF**
Sine integral	**S13ADF**
Tangent, circular	**S07AAF**
Tangent, hyperbolic	**S10AAF**
Trigamma function, scaled	**S14ADF**

5 References

[1] Abramowitz M and Stegun I A (1972) *Handbook of Mathematical Functions* Dover Publications (3rd Edition)

[2] Carlson B C (1965) On computing elliptic integrals and functions *J. Math. Phys.* **44** 36–51

[3] Carlson B C (1977) *Special Functions of Applied Mathematics* Academic Press

[4] Carlson B C (1977) Elliptic integrals of the first kind *SIAM J. Math. Anal.* **8** 231–242

[5] Clenshaw C W (1962) Mathematical tables *Chebyshev Series for Mathematical Functions* HMSO

[6] Fox L and Parker I B (1968) *Chebyshev Polynomials in Numerical Analysis* Oxford University Press

[7] Schonfelder J L (1976) The production of special function routines for a multi-machine library *Softw. Pract. Exper.* **6 (1)**

Chapter X01 – Mathematical Constants

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
X01AAF	5	Provides the mathematical constant π
X01ABF	5	Provides the mathematical constant γ (Euler's Constant)

Chapter X01

Mathematical Constants

Contents

1 Scope of the Chapter

This chapter is concerned with the provision of mathematical constants required by other routines within the Library.

It should be noted that because of the trivial nature of the routines individual routine documents are not provided.

2 Background to the Problems

Some Library routines require mathematical constants to maximum ***machine precision***. These routines call Chapter X01 and thus lessen the number of changes that have to be made between different implementations of the Library.

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.

Although these routines are primarily intended for use by other routines they may be accessed directly by the user:

Constant	Fortran Specification
π	***real*** `FUNCTION X01AAF(X)` ***real*** `X`
γ (Euler's constant)	***real*** `FUNCTION X01ABF(X)` ***real*** `X`

The parameter X of these routines is a dummy parameter.

Chapter X02 – Machine Constants

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
X02AHF	9	The largest permissible argument for sin and cos
X02AJF	12	The machine precision
X02AKF	12	The smallest positive model number
X02ALF	12	The largest positive model number
X02AMF	12	The safe range parameter
X02ANF	15	The safe range parameter for complex floating-point arithmetic
X02BBF	5	The largest representable integer
X02BEF	5	The maximum number of decimal digits that can be represented
X02BHF	12	The floating-point model parameter, b
X02BJF	12	The floating-point model parameter, p
X02BKF	12	The floating-point model parameter e_{min}
X02BLF	12	The floating-point model parameter e_{max}
X02DAF	8	Switch for taking precautions to avoid underflow
X02DJF	12	The floating-point model parameter ROUNDS

Chapter X02

Machine Constants

Contents

1 Scope of the Chapter

This chapter is concerned with **parameters** which characterise certain aspects of the **computing environment** in which the NAG Fortran Library is implemented. They relate primarily to floating-point arithmetic, but also to integer arithmetic, the elementary functions and exception handling. The values of the parameters vary from one implementation of the Library to another, but within the context of a single implementation they are constants.

The parameters are intended for use primarily by other routines in the Library, but users of the Library may sometimes need to refer to them directly.

Each parameter-value is returned by a separate Fortran function. Because of the trivial nature of the functions, individual routine documents are not provided; the necessary details are given in Section 3 of this Introduction.

2 Background to the Problems

2.1 Floating-Point Arithmetic

2.1.1 A model of floating-point arithmetic

In order to characterise the important properties of floating-point arithmetic by means of a small number of parameters, NAG uses a simplified **model** of floating-point arithmetic. The parameters of the model can be chosen to provide a sufficiently close description of the behaviour of actual implementations of floating-point arithmetic, but not, in general, an exact description; actual implementations vary too much in the details of how numbers are represented or arithmetic operations are performed.

The model is based on that developed by Brown [1], but differs in some respects. The essential features are summarised here.

The model is characterised by four integer parameters and one logical parameter. The four integer parameters are:

b: the base

p: the precision (i.e., the number of significant base-b digits)

e_{min}: the minimum exponent

e_{max}: the maximum exponent

These parameters define a set of numerical values of the form:

$$f \times b^e$$

where the exponent e must lie in the range $[e_{\mathrm{min}}, e_{\mathrm{max}}]$, and the fraction f (also called the mantissa or significand) lies in the range $[1/b, 1)$, and may be written:

$$f = 0.f_1 f_2 ... f_p$$

Thus f is a p-digit fraction to the base b; the f_i are the base-b digits of the fraction: they are integers in the range 0 to $b-1$, and the leading digit f_1 must not be zero.

The set of values so defined (together with zero) are called **model numbers.** For example, if $b = 10$, $p = 5$, $e_{\mathrm{min}} = -99$ and $e_{\mathrm{max}} = +99$, then a typical model number is 0.12345×10^{67}.

The model numbers must obey certain rules for the computed results of the following basic arithmetic operations: addition, subtraction, multiplication, negation, absolute value, and comparisons. The rules depend on the value of the logical parameter ROUNDS.

If ROUNDS is **true**, then the computed result must be the nearest model number to the exact result (assuming that overflow or underflow does not occur); if the exact result is midway between two model numbers, then it may be rounded either way.

If ROUNDS is **false**, then: if the exact result is a model number, the computed result must be equal to the exact result; otherwise, the computed result may be either of the adjacent model numbers on either side of the exact result.

For division and square root, this latter rule is further relaxed (regardless of the value of ROUNDS): the computed result may also be one of the next adjacent model numbers on either side of the permitted values just stated.

On some machines, the full set of representable floating-point numbers conforms to the rules of the model with appropriate values of b, p, $e_{\min}$, $e_{\max}$ and ROUNDS. For example, for DEC VAX machines in single precision:

$$\begin{aligned} b &= 2 \\ p &= 24 \\ e_{\min} &= -127 \\ e_{\max} &= 127 \quad \text{and ROUNDS is } \textbf{true}. \end{aligned}$$

For machines supporting IEEE binary double precision arithmetic:

$$\begin{aligned} b &= 2 \\ p &= 53 \\ e_{\min} &= -1021 \\ e_{\max} &= 1024 \quad \text{and ROUNDS is } \textbf{true}. \end{aligned}$$

For other machines, values of the model parameters must be chosen which define a large subset of the representable numbers; typically it may be necessary to decrease p by 1 (in which case ROUNDS is always set to **false**), or to increase $e_{\min}$ or decrease $e_{\max}$ by a little bit. There are additional rules to ensure that arithmetic operations on those representable numbers that are not model numbers are consistent with arithmetic on model numbers.

(**Note.** The model used here differs from that described in Brown [1] in the following respects: square-root is treated, like division, as a weakly supported operator; and the logical parameter ROUNDS has been introduced to take account of machines with good rounding.)

2.1.2 Derived parameters of floating-point arithmetic

Most numerical algorithms require access, not to the basic parameters of the model, but to certain derived values, of which the most important are:

the ***machine precision*** ϵ: $= \left(\frac{1}{2}\right) \times b^{1-p}$ if ROUNDS is **true**,
$= b^{1-p}$ otherwise (but see Note below).

the smallest positive model number: $= b^{e_{\min}-1}$

the largest positive model number: $= (1-b^{-p}) \times b^{e_{\max}}$

Note. This value is increased very slightly in some implementations to ensure that the computed result of $1+\epsilon$ or $1-\epsilon$ differs from 1. For example in IEEE binary single precision arithmetic the value is set to $2^{-24}+2^{-47}$.

Two additional derived values are used in the NAG Fortran Library. Their definitions depend not only on the properties of the basic arithmetic operations just considered, but also on properties of some of the elementary functions. We define the **safe range** parameter to be the smallest positive model number z such that for any x in the range $[z, 1/z]$ the following can be computed without undue loss of accuracy, overflow, underflow or other error:

$-x$

$1/x$

$-1/x$

SQRT(x)

LOG(x)

EXP(LOG(x))

y**(LOG(x)/LOG(y)) for any y

In a similar fashion we define the safe range parameter for complex arithmetic as the smallest positive model number z such that for any x in the range $[z, 1/z]$ the following can be computed without any undue loss of accuracy, overflow, underflow or other error:

$-w$

$1/w$

$-1/w$

SQRT(w)

LOG(w)

EXP(LOG(w))

y**(LOG(w)/LOG(y)) for any y

ABS(w)

where w is any of x, ix, $x + ix$, $1/x$, i/x, $1/x + i/x$, and i is the square root of -1.

This parameter was introduced to take account of the quality of complex arithmetic on the machine. On machines with well implemented complex arithmetic, its value will differ from that of the real safe range parameter by a small multiplying factor less than 10. For poorly implemented complex arithmetic this factor may be larger by many orders of magnitude.

2.2 Other Aspects of the Computing Environment

No attempt has been made to characterise comprehensively any other aspects of the computing environment. The other functions in this chapter provide specific information that is occasionally required by routines in the Library.

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 Historical Note

At Mark 12 a new set of routines was introduced to return parameters of floating-point arithmetic. The new set of routines is more carefully defined, and they do not require a dummy parameter. They are listed in Section 3.2. The older routines have since been withdrawn (see Section 4).

3.2 Parameters of Floating-point Arithmetic

real FUNCTION X02AJF()	returns the **machine precision**, i.e., $(\frac{1}{2}) \times b^{1-p}$ if ROUNDS is **true** or b^{1-p} otherwise (or a value very slightly larger than this, see Section 2.1.2)
real FUNCTION X02AKF()	returns the smallest positive model number, i.e., $b^{e_{\min}-1}$
real FUNCTION X02ALF()	returns the largest positive model number, i.e., $(1 - b^{-p}) \times b^{e_{\max}}$
real FUNCTION X02AMF()	returns the **safe range** parameter as defined in Section 2.1.2
real FUNCTION X02ANF()	returns the **safe range** parameter for complex arithmetic as defined in Section 2.1.2
INTEGER FUNCTION X02BHF()	returns the model parameter b
INTEGER FUNCTION X02BJF()	returns the model parameter p
INTEGER FUNCTION X02BKF()	returns the model parameter $e_{\min}$
INTEGER FUNCTION X02BLF()	returns the model parameter $e_{\max}$
LOGICAL FUNCTION X02DJF()	returns the model parameter ROUNDS

3.3 Parameters of Other Aspects of the Computing Environment

real FUNCTION X02AHF(X) ***real*** X	returns the largest positive **real** argument for which the intrinsic functions SIN and COS return a result with some meaningful accuracy
INTEGER FUNCTION X02BBF(X) ***real*** X	returns the largest positive integer value
INTEGER FUNCTION X02BEF(X) ***real*** X	returns the maximum number of decimal digits which can be accurately represented over the whole range of floating-point numbers
LOGICAL FUNCTION X02DAF(X) ***real*** X	returns **false** if the system sets underflowing quantities to zero, without any error indication or undesirable warning or system overhead

The parameter X in these routines is a dummy parameter.

4 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'.

X02AAF	X02ABF	X02ACF	X02ADF	X02AEF	X02AFF
X02AGF	X02BAF	X02BCF	X02BDF	X02CAF	

5 References

[1] Brown W S (1981) A simple but realistic model of floating-point computation *ACM Trans. Math. Software* 7 445–480

6 Example Program

The example program listed below simply prints the values of all the functions in Chapter X02. Obviously the results will vary from one implementation of the Library to another. The results listed in Section 6.3 are those from a double precision implementation on a Silicon Graphics workstation.

6.1 Example Text

```
*     X02AJF Example Program Text
*     Mark 17 Revised.  NAG Copyright 1995.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Functions ..
      real             X02AHF, X02AJF, X02AKF, X02ALF, X02AMF, X02ANF
      INTEGER          X02BBF, X02BEF, X02BHF, X02BJF, X02BKF, X02BLF
      LOGICAL          X02DAF, X02DJF
      EXTERNAL         X02AHF, X02AJF, X02AKF, X02ALF, X02AMF, X02ANF,
     +                 X02BBF, X02BEF, X02BHF, X02BJF, X02BKF, X02BLF,
     +                 X02DAF, X02DJF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'X02AJF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '(results are machine-dependent)'
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The basic parameters of the model'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) ' X02BHF = ', X02BHF(),
```

```
     +   '  (the model parameter B)'
      WRITE (NOUT,99999) ' X02BJF = ', X02BJF(),
     +   '  (the model parameter P)'
      WRITE (NOUT,99999) ' X02BKF = ', X02BKF(),
     +   '  (the model parameter EMIN)'
      WRITE (NOUT,99999) ' X02BLF = ', X02BLF(),
     +   '  (the model parameter EMAX)'
      WRITE (NOUT,99998) ' X02DJF = ', X02DJF(),
     +   '  (the model parameter ROUNDS)'
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Derived parameters of floating-point arithmetic'
      WRITE (NOUT,*)
      WRITE (NOUT,*) ' X02AJF = ', X02AJF(), '  (the machine precision)'
      WRITE (NOUT,*) ' X02AKF = ', X02AKF(),
     +   '  (the smallest positive model number)'
      WRITE (NOUT,*) ' X02ALF = ', X02ALF(),
     +   '  (the largest positive model number)'
      WRITE (NOUT,*) ' X02AMF = ', X02AMF(),
     +   '  (the real safe range parameter)'
      WRITE (NOUT,*) ' X02ANF = ', X02ANF(),
     +   '  (the complex safe range parameter)'
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +  'Parameters of other aspects of the computing environment'
      WRITE (NOUT,*)
      WRITE (NOUT,*) ' X02AHF = ', X02AHF(0.0e0),
     +   '  (largest argument for SIN and COS)'
      WRITE (NOUT,99997) ' X02BBF = ', X02BBF(0.0e0),
     +   '  (largest positive integer)'
      WRITE (NOUT,99997) ' X02BEF = ', X02BEF(0.0e0),
     +   '  (precision in decimal digits)'
      WRITE (NOUT,99996) ' X02DAF = ', X02DAF(0.0e0),
     +   '  (indicates how underflow is handled)'
      STOP
*
99999 FORMAT (1X,A,I7,A)
99998 FORMAT (1X,A,L7,A)
99997 FORMAT (1X,A,I20,A)
99996 FORMAT (1X,A,L20,A)
      END
```

6.2 Example Data

None.

6.3 Example Results

```
X02AJF Example Program Results

(results are machine-dependent)

The basic parameters of the model

 X02BHF =       2  (the model parameter B)
 X02BJF =      53  (the model parameter P)
 X02BKF =   -1021  (the model parameter EMIN)
 X02BLF =    1024  (the model parameter EMAX)
 X02DJF =       T  (the model parameter ROUNDS)
```

```
Derived parameters of floating-point arithmetic

 X02AJF =   1.1102230246251600E-16  (the machine precision)
 X02AKF =   2.2250738585072107-308  (the smallest positive model number)
 X02ALF =   1.7976931348623093+308  (the largest positive model number)
 X02AMF =   2.2250738585072107-308  (the real safe range parameter)
 X02ANF =   2.2250738585072107-308  (the complex safe range parameter)

Parameters of other aspects of the computing environment

 X02AHF =   1.8014398509481900E+16  (largest argument for SIN and COS)
 X02BBF =           2147483647  (largest positive integer)
 X02BEF =                   15  (precision in decimal digits)
 X02DAF =                    F  (indicates how underflow is handled)
```

Chapter X03 – Inner Products

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
X03AAF	5	Real inner product added to initial value, basic/additional precision
X03ABF	5	Complex inner product added to initial value, basic/additional precision

Chapter X03

Inner Products

Contents

1 Scope of the Chapter

This chapter is concerned with the calculation of innerproducts required by other routines within the Library.

2 Background to the Problems

Some Library routines require to calculate the innerproduct

$$c+\sum_i x_i y_i,$$

preferably in additional precision, but, if this is unavailable or prohibitively expensive, then in basic precision. These routines call Chapter X03 so that machine dependencies of this type can be isolated to 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.

Although these routines are primarily intended for use by other Library routines they may be accessed directly by the user:

X03AAF Calculates the innerproduct for real values c, x_i and y_i,

X03ABF Calculates the innerproduct for complex values c, x_i and y_i,

---

Chapter X04 – Input/Output Utilities

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
X04AAF	7	Return or set unit number for error messages
X04ABF	7	Return or set unit number for advisory messages
X04ACF	19	Open unit number for reading, writing or appending, and associate unit with named file
X04ADF	19	Close file associated with given unit number
X04BAF	12	Write formatted record to external file
X04BBF	12	Read formatted record from external file
X04CAF	14	Print real general matrix (easy-to-use)
X04CBF	14	Print real general matrix (comprehensive)
X04CCF	14	Print real packed triangular matrix (easy-to-use)
X04CDF	14	Print real packed triangular matrix (comprehensive)
X04CEF	14	Print real packed banded matrix (easy-to-use)
X04CFF	14	Print real packed banded matrix (comprehensive)
X04DAF	14	Print complex general matrix (easy-to-use)
X04DBF	14	Print complex general matrix (comprehensive)
X04DCF	14	Print complex packed triangular matrix (easy-to-use)
X04DDF	14	Print complex packed triangular matrix (comprehensive)
X04DEF	14	Print complex packed banded matrix (easy-to-use)
X04DFF	14	Print complex packed banded matrix (comprehensive)
X04EAF	14	Print integer matrix (easy-to-use)
X04EBF	14	Print integer matrix (comprehensive)

Chapter X04

Input/Output Utilities

Contents

1 Scope of the Chapter

This chapter contains utility routines concerned with input and output to or from an external file.

2 Background to the Problems

2.1 Output from NAG Library Routines

Output from NAG library routines to an external file falls into two categories.

(a) **Error messages**
which are always associated with an error exit from a routine, that is, with a non-zero value of IFAIL as specified in Section 6 of the routine document.

(b) **Advisory messages**
which include output of final results, output of intermediate results to monitor the course of a computation, and various warning or informative messages.

Each category of output is written to its own Fortran output unit – the **error message unit** or the **advisory message unit**. In practice these may be the same unit number. Default unit numbers are provided for each implementation of the Library (see the Users' Note for your implementation); they may be changed by users. Output of error messages may be controlled by the setting of IFAIL (see the Essential Introduction or Chapter P01). Output of advisory messages may usually be controlled by the setting of some other parameter (e.g. MSGLVL) (or in some routines also by IFAIL). An alternative mechanism for completely suppressing output is to set the relevant unit number < 0.

At present only formatted records are output from the Library. All formatted output to an external file from within the Library is performed by X04BAF. Similarly, all formatted input from an external file is performed by X04BBF.

For further information about error and advisory messages, see Chapter P01.

When the library is being called from another language, such as C or Visual Basic, the routines X04ACF and X04ADF may be especially useful. X04ACF connects a file to a FORTRAN unit and X04ADF disconnects a file from a FORTRAN unit.

2.2 Matrix Printing Routines

Routines are provided to allow formatted output of

(a) general matrices stored in a two-dimensional array (real, complex and integer data types);

(b) triangular matrices stored in a packed one-dimensional array (real and complex data types);

(c) band matrices stored in a packed two-dimensional array (real and complex data types).

Routines in (b) and (c) allow printing of matrices stored in formats used in particular by Chapter F06 and Chapter F07 of the Library.

By appropriate choice of arguments the user can specify titles, labels, maximum output record length, and the format of individual matrix elements. All output is directed to the unit number for output of advisory messages, which may be altered by a call to X04ABF.

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.

Apart from the obvious utility of the matrix printing routines, users of the Library may need to call routines in Chapter X04 for the following purposes.

If the default unit number for error messages (given in the Users' Note for your implementation) is not satisfactory, it may be changed to a new value NERR by the statement

```
CALL X04AAF(1,NERR)
```

Similarly the unit number for advisory messages may be changed to a new value NADV by the statement

```
CALL X04ABF(1,NADV)
```

4 Index

Chapter X05 – Date and Time Utilities

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
X05AAF	14	Return date and time as an array of integers
X05ABF	14	Convert array of integers representing date and time to character string
X05ACF	14	Compare two character strings representing date and time
X05BAF	14	Return the CPU time

Chapter X05

Date and Time Utilities

Contents

1 Scope of the Chapter

This chapter provides routines to obtain the current real time, and the amount of processor time used.

2 Background to the Problems

2.1 Real Time

Routines are provided to obtain the current time in two different formats, and to compare two such times.

2.2 Processor Time

A routine is provided to return the current amount of processor time used. This allows the timing of a particular routine or section of code.

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.

X05AAF returns the current date/time in integer format.

X05ABF converts from integer to character string date/time.

X05ACF compares two date/time character strings.

X05BAF returns the amount of processor time used.