

NAG Fortran Library Chapter Introduction

F07 – Linear Equations (LAPACK)

Contents

1	Scope of the Chapter	2
2	Background to the Problems	2
2.1	Notation	2
2.2	Matrix Factorizations	3
2.3	Solution of Systems of Equations	3
2.4	Sensitivity and Error Analysis	4
2.4.1	Normwise error bounds	4
2.4.2	Estimating condition numbers	4
2.4.3	Componentwise error bounds	4
2.4.4	Iterative refinement of the solution	5
2.5	Matrix Inversion	5
2.6	Packed Storage	5
2.7	Band and Tridiagonal Matrices	5
2.8	Block Partitioned Algorithms	6
3	Recommendations on Choice and Use of Available Routines	6
3.1	Available Routines	6
3.2	NAG Names and LAPACK Names	7
3.3	Matrix Storage Schemes	8
3.3.1	Conventional storage	8
3.3.2	Packed storage	9
3.3.3	Band storage	10
3.3.4	Unit triangular matrices	11
3.3.5	Real diagonal elements of complex matrices	11
3.4	Parameter Conventions	11
3.4.1	Option parameters	11
3.4.2	Problem dimensions	11
3.4.3	Length of work arrays	11
3.4.4	Error-handling and the diagnostic parameter INFO	11
3.5	Tables of Available Computational Routines	12
4	Index	13
5	Routines Withdrawn or Scheduled for Withdrawal	16
6	References	16

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 Chapters F04 or F07 for solving your particular problem. In particular, Chapters F04 and F07 contain *Black Box* (or *driver*) 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 Chapter F07 routines to perform the necessary computational tasks.

There are two types of driver routines in this chapter: *simple drivers* which just return the solution to the linear equations; and *expert drivers* which also return condition and error estimates and, in many cases, also allow equilibration. The simple drivers for real matrices have names of the form F07_AF (D__SV) and for complex matrices have names of the form F07_NF (Z__SV). The expert drivers for real matrices have names of the form F07_BF (D__SVX) and for complex matrices have names of the form F07_PF (Z__SVX).

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.* (1999)). 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 example Golub and Van Loan (1996) for a more thorough discussion.

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 <i>computed</i> solution to $Ax = b$, (which usually differs from the exact solution x because of round-off error)
$r = b - A\hat{x}$	the <i>residual</i> corresponding to the computed solution $\hat{x}$
$\ x\ _{\infty} = \max_i x_i $	the ∞ -norm of the vector x
$\ x\ _1 = \sum_{j=1}^n x_j $	the 1-norm of the vector x

$\|A\|_\infty = \max_i \sum_j |a_{ij}|$ the ∞ -norm of the matrix A

$\|A\|_1 = \max_j \sum_{i=1}^n |a_{ij}|$ the 1-norm of the matrix A

$|x|$ the vector with elements $|x_i|$
 $|A|$ the matrix with elements $|a_{ij}|$

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)

$$A = U^T U \quad \text{or} \quad A = LL^T$$

where U is upper triangular and L is lower triangular.

Symmetric indefinite matrices (Bunch–Kaufman factorization)

$$A = PUDU^T P^T \quad \text{or} \quad A = PLDL^T P^T$$

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)

$$\begin{aligned} Ly &= P^T b \\ Ux &= y \end{aligned}$$

Symmetric positive-definite matrices (Cholesky factorization)

$$\begin{aligned} U^T y &= b \\ Ux &= y \end{aligned} \quad \text{or} \quad \begin{aligned} Ly &= b \\ L^T x &= y \end{aligned}$$

Symmetric indefinite matrices (Bunch–Kaufman factorization)

$$\begin{aligned} PUDy &= b \\ U^T P^T x &= y \end{aligned} \quad \text{or} \quad \begin{aligned} PL Dy &= b \\ L^T P^T x &= y \end{aligned}$$

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

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right) + \cdots \text{(2nd - order terms)}$$

where $\kappa(A)$ is the *condition number* of A defined by

$$\kappa(A) = \|A\| \|A^{-1}\|. \quad (3)$$

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 (see Higham (1988)). 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 :

$$\max_{ijk} \left(\frac{|\delta a_{ij}|}{|a_{ij}|}, \frac{|\delta b_k|}{|b_k|} \right) \leq \omega$$

where the *componentwise backward error bound* ω is given by

$$\omega = \max_i \frac{|r_i|}{(|A| \cdot |\hat{x}| + |b|)_i}.$$

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:

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq \frac{\| |A^{-1}| \cdot |r| \|_\infty}{\|x\|_\infty}.$$

Care is taken when computing this bound to allow for rounding errors in computing r . The norm $\| |A^{-1}| \cdot |r| \|_\infty$ 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 $i = 0, 1, \dots$, 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 $|A| \cdot |x|$ 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 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 $n(n+1)/2$; 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 and Tridiagonal 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. A *tridiagonal* matrix is a special case of a band matrix with just one sub-diagonal and one super-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 and Hermitian 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 Partitioned Algorithms

Many of the routines in this chapter use what is termed a *block partitioned 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 (1996) or Anderson *et al.* (1999) for more about block partitioned algorithms.

The performance of a block partitioned 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 some machines there may be no advantage from using a block partitioned 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: please 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 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 positive-definite tridiagonal
- symmetric or Hermitian indefinite
- symmetric or Hermitian indefinite (packed storage)

triangular
 triangular (packed storage)
 triangular band
 tridiagonal

For each of the above types of matrix (except where indicated), routines are provided to perform the following computations:

- (a) solve a system of linear equations (driver routines);
- (b) solve a system of linear equations with condition and error estimation (expert drivers);
- (c) (except for triangular matrices) factorize the matrix (see Section 2.2);
- (d) solve a system of linear equations, using the factorization (see Section 2.3);
- (e) 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;
- (f) refine the solution and compute forward and backward error bounds (see Sections 2.4.3 and 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);
- (g) (except for band and tridiagonal matrices) invert the matrix, using the factorization (see Section 2.5).

Thus, to solve a particular problem, it is usually only necessary to call a single driver routine, but alternatively two or more routines may be called 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 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 Chapter F07 routines in the manual normally include the LAPACK double precision names, for example, F07ADF (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 **XXYYZZZ**, 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)
- GT – general tridiagonal
- PT – symmetric or Hermitian positive-definite tridiagonal
- 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, ***dgetrf***, 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 two-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 part 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 parameter 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 one-dimensional array. In Chapters F07 and F08, arrays which hold matrices in packed storage have names ending in P . The storage of matrix elements a_{ij} in the packed array AP is as follows:

- 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 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 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 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, elements of matrix elements a_{ij} are stored as follows:

$$a_{ij} \text{ is stored in } AB(k_u + 1 + i - j, j) \text{ for } \max(1, j - k_u) \leq i \leq \min(n, j + k_l).$$

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 UPLO = '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 partitioned 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 may 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 *i*th 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 Computational Routines

It should be noted that all the LAPACK computational routines from Release 3 are included in the Library and can be called by their LAPACK name*, although not all of these routines are currently documented in Chapters F07 and F08.

Each entry gives:

the NAG routine name

the LAPACK routine name in a double precision implementation (substitute the initial letter ‘D’ for the letter ‘S’ for the LAPACK name in a single precision implementation)

Type of matrix and storage scheme	factorize	solve	condition number	error estimate	invert
general	F07ADF (DGETRF)	F07AEF (DGETRS)	F07AGF (DGECON)	F07AHF (DGERFS)	F07AJF (DGETRI)
general band	F07BDF (DGBTRF)	F07BEF (DGBTRS)	F07BGF (DGBCON)	F07BHF (DGBRFS)	
symmetric positive-definite	F07FDF (DPOTRF)	F07FEF (DPOTRS)	F07FGF (DPOCON)	F07FHF (DPORFS)	F07FJF (DPOTRI)
symmetric positive-definite (packed storage)	F07GDF (DPPTRF)	F07GEF (DPPTRS)	F07GGF (DPPCON)	F07GHF (DPPRFS)	F07GJF (DPPTRI)
symmetric positive-definite band	F07HDF (DPBTRF)	F07HEF (DPBTRS)	F07HGF (DPBCON)	F07HHF (DPBRFS)	
symmetric indefinite	F07MDF (DSYTRF)	F07MEF (DSYTRS)	F07MGF (DSYCON)	F07MHF (DSYRFS)	F07MJF (DSYTRI)
symmetric indefinite (packed storage)	F07PDF (DSPTRF)	F07PEF (DSPTRS)	F07PGF (DSPCON)	F07PHF (DSPRFS)	F07PJF (DSPTRI)
triangular		F07TEF (DTRTRS)	F07TGF (DTRCON)	F07THF (DTRRFS)	F07TJF (DTRTRI)
triangular (packed storage)		F07UEF (DTPTRS)	F07UGF (DTPCON)	F07UHF (DTPRFS)	F07UJF (DTPTRI)
triangular band		F07VEF (DTBTRS)	F07VGF (DTBCON)	F07VHF (DTBRFS)	

Each entry gives:

the NAG routine name

the LAPACK routine name in a double precision implementation (substitute the initial letter ‘Z’ for the letter ‘C’ for the LAPACK name in a single precision implementation)

Type of matrix and storage scheme	factorize	solve	condition number	error estimate	invert
general	F07ARF (ZGETRF)	F07ASF (ZGETRS)	F07AUF (ZGECON)	F07AVF (ZGERFS)	F07AWF (ZGETRI)
general band	F07BRF (ZGBTRF)	F07BSF (ZGBTRS)	F07BUF (ZGBCON)	F07BVF (ZGBRFS)	
Hermitian positive-definite	F07FRF (ZPOTRF)	F07FSF (ZPOTRS)	F07FUF (ZPOCON)	F07FVF (ZPORFS)	F07FWF (ZPOTRI)
Hermitian positive-definite (packed storage)	F07GRF (ZPPTRF)	F07GSF (ZPPTRS)	F07GUF (ZPPCON)	F07GVF (ZPPRFS)	F07GWF (ZPPTRI)
Hermitian positive-definite band	F07HRF (ZPBTRF)	F07HSF (ZPBTRS)	F07HUF (ZPBCON)	F07HVF (ZPBRFS)	
Hermitian indefinite	F07MRF (ZHETRF)	F07MSF (ZHETRS)	F07MUF (ZHECON)	F07MVF (ZHERFS)	F07MWF (ZHETRI)
symmetric indefinite	F07NRF (ZSYTRF)	F07NSF (ZSYTRS)	F07NUF (ZSYCON)	F07NVF (ZSYRFS)	F07NWF (ZSYTRI)
Hermitian indefinite (packed storage)	F07PRF (ZHPTRF)	F07PSF (ZHPTRS)	F07PUF (ZHPCON)	F07PVF (ZHPRFS)	F07PWF (ZHPTRI)
symmetric indefinite (packed storage)	F07QRF (ZSPTRF)	F07QSF (ZSPTRS)	F07QUF (ZSPCON)	F07QVF (ZSPRFS)	F07QWF (ZSPTRI)
triangular		F07TSF (ZTRTRS)	F07TUF (ZTRCON)	F07TVF (ZTRRFS)	F07TWF (ZTRTRI)
triangular (packed storage)		F07USF (ZTPTRS)	F07UUF (ZTPCON)	F07UVF (ZTPRFS)	F07UWF (ZTPTRI)
triangular band		F07VSF (ZTBTRS)	F07VUF (ZTBCON)	F07VVF (ZTBRFS)	

4 Index

Apply iterative refinement to the solution and compute error estimates:

after factorizing the matrix of coefficients:

complex band matrix	F07BVF (ZGBRFS)
complex Hermitian indefinite matrix	F07MVF (ZHERFS)
complex Hermitian indefinite matrix, packed storage	F07PVF (ZHPRFS)
complex Hermitian positive-definite band matrix	F07HVF (ZPBRFS)
complex Hermitian positive-definite matrix	F07FVF (ZPORFS)
complex Hermitian positive-definite matrix, packed storage	F07GVF (ZPPRFS)

complex matrix	F07AVF (ZGERFS)
complex symmetric indefinite matrix	F07NVF (ZSYRFS)
complex symmetric indefinite matrix, packed storage	F07QVF (ZSPRFS)
real band matrix	F07BHF (DGBRFS)
real matrix	F07AHF (DGERFS)
real symmetric indefinite matrix	F07MHF (DSYRFS)
real symmetric indefinite matrix, packed storage	F07PHF (DSPRFS)
real symmetric positive-definite band matrix	F07HHF (DPBRFS)
real symmetric positive-definite matrix	F07FHF (DPORFS)
real symmetric positive-definite matrix, packed storage	F07GHF (DPPRFS)
Compute error estimates:	
complex triangular band matrix	F07VVF (ZTBRFS)
complex triangular matrix	F07TVF (ZTRRFS)
complex triangular matrix, packed storage	F07UVF (ZTPRFS)
real triangular band matrix	F07VHF (DTBRFS)
real triangular matrix	F07THF (DTRRFS)
real triangular matrix, packed storage	F07UHF (DTPRFS)
Condition number estimation:	
after factorizing the matrix of coefficients:	
complex band matrix	F07BUF (ZGBCON)
complex Hermitian indefinite matrix	F07MUF (ZHECON)
complex Hermitian indefinite matrix, packed storage	F07PUF (ZHPCON)
complex Hermitian positive-definite band matrix	F07HUF (ZPBCON)
complex Hermitian positive-definite matrix	F07FUF (ZPOCON)
complex Hermitian positive-definite matrix, packed storage	F07GUF (ZPPCON)
complex matrix	F07AUF (ZGECON)
complex symmetric indefinite matrix	F07NUF (ZSYCON)
complex symmetric indefinite matrix, packed storage	F07QUF (ZSPCON)
real band matrix	F07BGF (DGBCON)
real matrix	F07AGF (DGECON)
real symmetric indefinite matrix	F07MGF (DSYCON)
real symmetric indefinite matrix, packed storage	F07PGF (DSPCON)
real symmetric positive-definite band matrix	F07HGF (DPBCON)
real symmetric positive-definite matrix	F07FGF (DPOCON)
real symmetric positive-definite matrix, packed storage	F07GGF (DPPCON)
complex triangular band matrix	F07VUF (ZTBCON)
complex triangular matrix	F07TUF (ZTRCON)
complex triangular matrix, packed storage	F07UUF (ZTPCON)
real triangular band matrix	F07VGF (DTBCON)
real triangular matrix	F07TGF (DTRCON)
real triangular matrix, packed storage	F07UGF (DTPCON)
LL^T or $U^T U$ factorization:	
complex Hermitian positive-definite band matrix	F07HRF (ZPBTRF)
complex Hermitian positive-definite matrix	F07FRF (ZPOTRF)
complex Hermitian positive-definite matrix, packed storage	F07GRF (ZPPTRF)
real symmetric positive-definite band matrix	F07HDF (DPBTRF)
real symmetric positive-definite matrix	F07FDF (DPOTRF)
real symmetric positive-definite matrix, packed storage	F07GDF (DPPTRF)
LU factorization:	
complex band matrix	F07BRF (ZGBTRF)
complex matrix	F07ARF (ZGETRF)
real band matrix	F07BDF (DGBTRF)
real matrix	F07ADF (DGETRF)
Matrix inversion:	
after factorizing the matrix of coefficients:	
complex Hermitian indefinite matrix	F07MWF (ZHETRI)
complex Hermitian indefinite matrix, packed storage	F07PWF (ZHPTRI)
complex Hermitian positive-definite matrix	F07FWF (ZPOTRI)

complex Hermitian positive-definite matrix, packed storage	F07GWF (ZPPTRI)
complex matrix	F07AWF (ZGETRI)
complex symmetric indefinite matrix	F07NWF (ZSYTRI)
complex symmetric indefinite matrix, packed storage	F07QWF (ZSPTRI)
real matrix	F07AJF (DGETRI)
real symmetric indefinite matrix	F07MJF (DSYTRI)
real symmetric indefinite matrix, packed storage	F07PJF (DSPTRI)
real symmetric positive-definite matrix	F07FJF (DPOTRI)
real symmetric positive-definite matrix, packed storage	F07GJF (DPPTRI)
complex triangular matrix	F07TWF (ZTRTRI)
complex triangular matrix, packed storage	F07UWF (ZTPTRI)
real triangular matrix	F07TJF (DTRTRI)
real triangular matrix, packed storage	F07UJF (DTPTRI)
<i>PLDL^TP^T</i> or <i>PUDU^TP^T</i> factorization:	
complex Hermitian indefinite matrix	F07MRF (ZHETRF)
complex Hermitian indefinite matrix, packed storage	F07PRF (ZHPTRF)
complex symmetric indefinite matrix	F07NRF (ZSYTRF)
complex symmetric indefinite matrix, packed storage	F07QRF (ZSPTRF)
real symmetric indefinite matrix	F07MDF (DSYTRF)
real symmetric indefinite matrix, packed storage	F07PDF (DSPTRF)
Solution of simultaneous linear equations:	
after factorizing the matrix of coefficients:	
complex band matrix	F07BSF (ZGBTRS)
complex Hermitian indefinite matrix	F07MSF (ZHETRS)
complex Hermitian indefinite matrix, packed storage	F07PSF (ZHPTRS)
complex Hermitian positive-definite band matrix	F07HSF (ZPBTRS)
complex Hermitian positive-definite matrix	F07FSF (ZPOTRS)
complex Hermitian positive-definite matrix, packed storage	F07GSF (ZPPTRS)
complex matrix	F07ASF (ZGETRS)
complex symmetric indefinite matrix	F07NSF (ZSYTRS)
complex symmetric indefinite matrix, packed storage	F07QSF (ZSPTRS)
real band matrix	F07BEF (DGBTRS)
real matrix	F07AEF (DGETRS)
real symmetric indefinite matrix	F07MEF (DSYTRS)
real symmetric indefinite matrix, packed storage	F07PEF (DSPTRS)
real symmetric positive-definite band matrix	F07HEF (DPBTRS)
real symmetric positive-definite matrix	F07FEF (DPOTRS)
real symmetric positive-definite matrix, packed storage	F07GEF (DPPTRS)
complex band matrices	F07BNF (ZGBSV)
complex Hermitian indefinite matrix	F07MNF (ZHESV)
complex Hermitian indefinite matrix, packed storage	F07PNF (ZHPSV)
complex Hermitian positive-definite band matrix	F07HNF (ZPBSV)
complex Hermitian positive-definite matrix	F07FNF (ZPOSV)
complex Hermitian positive-definite matrix, packed storage	F07GNF (ZPPSV)
complex Hermitian positive-definite tridiagonal matrix	F07JNF (ZPTSV)
complex matrix	F07ANF (ZGESV)
complex symmetric indefinite matrix	F07NNF (ZSYSV)
complex symmetric indefinite matrix, packed storage	F07QNF (ZSPSV)
complex triangular band matrix	F07VSF (ZTBTRS)
complex triangular matrix	F07TSF (ZTRTRS)
complex triangular matrix, packed storage	F07USF (ZTPTRS)
complex tridiagonal matrix	F07CNF (ZGTSV)
real band matrix	F07BAF (DGBSV)
real matrix	F07AAF (DGESV)
real symmetric indefinite matrix	F07MAF (DSYSV)
real symmetric indefinite matrix, packed storage	F07PAF (DSPSV)
real symmetric positive-definite band matrix	F07HAF (DPBSV)
real symmetric positive-definite matrix	F07FAF (DPOSV)

real symmetric positive-definite matrix, packed storage	F07GAF (DPPSV)
real symmetric positive-definite tridiagonal matrix	F07JAF (DPTSV)
real triangular band matrix	F07VEF (DTBTRS)
real triangular matrix	F07TEF (DTRTRS)
real triangular matrix, packed storage	F07UEF (DTPTRS)
real tridiagonal matrix	F07CAF (DGTSV)
with condition and error estimation:	
complex band matrix	F07BPF (ZGBSVX)
complex Hermitian indefinite matrix	F07MPF (ZHESVX)
complex Hermitian indefinite matrix, packed storage	F07PPF (ZHPSVX)
complex Hermitian positive-definite band matrix	F07HPF (ZPBSVX)
complex Hermitian positive-definite matrix	F07FPF (ZPOSVX)
complex Hermitian positive-definite matrix, packed storage	F07GPF (ZPPSVX)
complex Hermitian positive-definite tridiagonal matrix	F07JPF (ZPTSVX)
complex matrix	F07APF (ZGESVX)
complex symmetric indefinite matrix	F07NPF (ZSYSVX)
complex symmetric indefinite matrix, packed storage	F07QPF (ZSPSVX)
complex tridiagonal matrix	F07CPF (ZGTSVX)
real band matrix	F07BBF (DGBSVX)
real matrix	F07ABF (DGESVX)
real symmetric indefinite matrix	F07MBF (DSYSVX)
real symmetric indefinite matrix, packed storage	F07PBF (DSPSVX)
real symmetric positive-definite band matrix	F07HBF (DPBSVX)
real symmetric positive-definite matrix	F07FBF (DPOSVX)
real symmetric positive-definite matrix, packed storage	F07GBF (DPPSVX)
real symmetric positive-definite tridiagonal matrix	F07JBF (DPTSVX)
real tridiagonal matrix	F07CBF (DGTSVX)

5 Routines Withdrawn or Scheduled for Withdrawal

None.

6 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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