NAG Library Chapter Introduction

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;

computing scaling factors to equilibrate a matrix.

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, in Section 4 in 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 (Chapter F07) handle only *dense* and *band* matrices (not matrices with more specialised 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 *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 $\hat{x}$
$\\|x\\|_\infty = \max_i \|x_i\|$	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 component-wise, 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)

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

where U is upper triangular and L is lower triangular.

Symmetric positive semidefinite matrices (pivoted Cholesky factorization)

$$A = PU^{\mathrm{T}}UP^{\mathrm{T}} \quad \text{or} \quad A = PLL^{\mathrm{T}}P^{\mathrm{T}}$$

where P is a permutation matrix, U is upper triangular and L is lower triangular. The permutation matrix P (which represents row-and-column interchanges) is needed to ensure numerical stability and to reveal the numerical rank of A.

Symmetric indefinite matrices (Bunch–Kaufman factorization)

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

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 $A = UDU^{\mathrm{T}}$ or $A = LDL^{\mathrm{T}}$ 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)

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

Symmetric positive definite matrices (Cholesky factorization)

$$\begin{array}{lcl} U^{\mathrm{T}}y=b & \text{or} & Ly=b \\ Ux=y & & L^{\mathrm{T}}x=y \end{array}$$

Symmetric indefinite matrices (Bunch–Kaufman factorization)

$$\begin{array}{lcl} PUDy=b & \text{or} & PLDy=b \\ U^{\mathrm{T}}P^{\mathrm{T}}x=y & & L^{\mathrm{T}}P^{\mathrm{T}}x=y \end{array}$$

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

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

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

$$\kappa(A)=\|A\|.\left\|A^{-1}\right\|. \tag{3}$$

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 (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 Scaling and Equilibration

The condition of a matrix and hence the accuracy of the computed solution, may be improved by scaling; thus if D_1 and D_2 are diagonal matrices with positive diagonal elements, then

$$B=D_1AD_2$$

is the scaled matrix. A general matrix is said to be *equilibrated* if it is scaled so that the lengths of its rows and columns have approximately equal magnitude. Similarly a general matrix is said to be *row-*

equilibrated (column-equilibrated) if it is scaled so that the lengths of its rows (columns) have approximately equal magnitude. Note that row scaling can affect the choice of pivot when partial pivoting is used in the factorization of A.

A symmetric or Hermitian positive definite matrix is said to be equilibrated if the diagonal elements are all approximately equal to unity.

For further information on scaling and equilibration see Section 3.5.2 of Golub and Van Loan (1996), Section 7.2, 7.3 and 9.8 of Higham (1988) and Section 5 of Chapter 4 of Wilkinson (1965).

Routines are provided to return the scaling factors that equilibrate a matrix for general, general band, symmetric and Hermitian positive definite and symmetric and Hermitian positive definite band matrices.

2.4.4 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 *component-wise 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:

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

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 $\kappa(A)$.

2.4.5 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.

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 Formats

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 $n(n+1)/2$ or a two-dimensional array with $n(n+1)/2$ elements; in other words, the storage is almost halved.

The one-dimensional array storage format is referred to as packed storage; it is described in Section 3.3.2. The two-dimensional array storage format is referred to as Rectangular Full Packed (RFP) format; it is described in Section 3.3.3. They may also be used for triangular matrices.

Routines designed for these packed storage formats perform the same number of arithmetic operations as routines which use conventional storage. Those using a packed one-dimensional array are usually less efficient, especially on high-performance computers, so there is then a trade-off between storage and efficiency. The RFP routines are as efficient as for conventional storage, although only a small subset of routines use this format.

2.7 Band and Tridiagonal Matrices

A *band* matrix is one whose nonzero elements are confined to a relatively small number of subdiagonals or superdiagonals on either side of the main diagonal. A *tridiagonal* matrix is a special case of a band matrix with just one subdiagonal and one superdiagonal. 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 3.3.4.

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 superdiagonals or subdiagonals as the original matrix. In the LU factorization, the row-interchanges modify the band structure: if A has k_l subdiagonals and k_u superdiagonals, then L is not a band matrix but still has at most k_l nonzero elements below the diagonal in each column; and U has at most $k_l + k_u$ superdiagonals.

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 ***block size*** – that is, the number of rows or columns per block. This is a machine-dependent argument, which is set to a suitable value when the library is implemented on each range of machines. You 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 block size of 1), relying solely on calls to the Level 2 BLAS (see Chapter F06 again).

The only situation in which you need 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.

2.9 Mixed Precision LAPACK Routines

Some LAPACK routines use mixed precision arithmetic in an effort to solve problems more efficiently on modern hardware. They work by converting a double precision problem into an equivalent single precision problem, solving it and then using iterative refinement in double precision to find a full precision solution to the original problem. The method may fail if the problem is too ill-conditioned to allow the initial single precision solution, in which case the routines fall back to solve the original problem entirely in double precision. The vast majority of problems are not so ill-conditioned, and in those cases the technique can lead to significant gains in speed without loss of accuracy. This is particularly true on machines where double precision arithmetic is significantly slower than single precision.

3 Recommendations on Choice and Use of Available Routines

3.1 Available Routines

Tables 1 to 8 in Section 3.5 show the routines which are provided for performing different computations on different types of matrices. Tables 1 to 4 show routines for real matrices; Tables 5 to 8 show 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
- general tridiagonal
- symmetric or Hermitian positive definite
- symmetric or Hermitian positive definite (packed storage)
- symmetric or Hermitian positive definite (RFP 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 (RFP 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 RFP matrices) solve a system of linear equations (driver routines);

(b) (except for RFP matrices) 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) (except for RFP matrices) 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) (except for RFP matrices) refine the solution and compute forward and backward error bounds (see Sections 2.4.4 and 2.4.5); 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);

(h) (except for tridiagonal, symmetric indefinite, triangular and RFP matrices) compute scale factors to equilibrate the matrix (see Section 2.4.3).

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 to 8 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 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). Most names have 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`)

– exceptionally, the mixed precision LAPACK routines described in Section 2.9 replace the initial first letter by a pair of letters, as:

  DS – double precision routine using single precision internally

  ZC – double complex routine using single precision complex internally

– the 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

  PF – symmetric or Hermitian positive definite (RFP storage)

  PP – symmetric or Hermitian positive definite (packed storage)

  PB – symmetric or Hermitian positive definite band

SY – symmetric indefinite

SF – symmetric indefinite (RFP storage)

SP – symmetric indefinite (packed storage)

HE – (complex) Hermitian indefinite

HF – (complex) Hermitian indefinite (RFP storage)

HP – (complex) Hermitian indefinite (packed storage)

GT – general tridiagonal

PT – symmetric or Hermitian positive definite tridiagonal

TR – triangular

TF – triangular (RFP storage)

TP – triangular (packed storage)

TB – triangular band

– the last two or three letters **ZZ** or **ZZZ** indicate the computation performed. Examples are:

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 DGETRF performs a triangular factorization of a real general matrix in double precision; the corresponding routine for a complex general matrix is ZGETRF.

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;

– rectangular full packed (RFP) 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 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 * or ␣ in the examples below.

For example, when $n = 4$:

UPLO	Triangular matrix A	Storage in array A
`U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \sqcup & a_{22} & a_{23} & a_{24} \\ \sqcup & \sqcup & a_{33} & a_{34} \\ \sqcup & \sqcup & \sqcup & a_{44} \end{matrix}$
`L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & \sqcup & \sqcup & \sqcup \\ a_{21} & a_{22} & \sqcup & \sqcup \\ a_{31} & a_{32} & a_{33} & \sqcup \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

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'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} \\ \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \sqcup & a_{22} & a_{23} & a_{24} \\ \sqcup & \sqcup & a_{33} & a_{34} \\ \sqcup & \sqcup & \sqcup & a_{44} \end{matrix}$
`L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & \sqcup & \sqcup & \sqcup \\ a_{21} & a_{22} & \sqcup & \sqcup \\ a_{31} & a_{32} & a_{33} & \sqcup \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

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 this chapter, as in Chapters F06 and F08, arrays which hold matrices in packed storage, have names ending in P. For a matrix of order n, the array must have at least $n(n+1)/2$ elements. 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:

	Triangle of matrix A	Packed storage in array AP
UPLO = 'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$a_{11}\ \underbrace{a_{12}a_{22}}\ \underbrace{a_{13}a_{23}a_{33}}\ \underbrace{a_{14}a_{24}a_{34}a_{44}}$
UPLO = '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 Rectangular Full Packed (RFP) Storage

The rectangular full packed (RFP) storage format offers the same savings in storage as the packed storage format (described in Section 3.3.2), but is likely to be much more efficient in general since the block structure of the matrix is maintained. This structure can be exploited using block partition algorithms (see Section 2.8) in a similar way to matrices that use conventional storage.

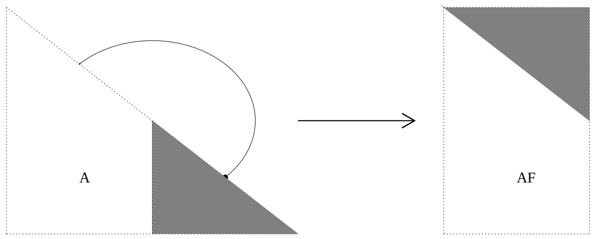

Figure 1 gives a graphical representation of the key idea of RFP for the particular case of a lower triangular matrix of even dimensions. In all cases the original triangular matrix of stored elements is separated into a trapezoidal part and a triangular part. The number of columns in these two parts is equal when the dimension of the matrix is even, $n = 2k$, while the trapezoidal part has $k + 1$ columns when $n = 2k + 1$. The smaller part is then transposed and fitted onto the trapezoidal part forming a rectangle. The rectangle has dimensions $2k + 1$ and q, where $q = k$ when n is even and $q = k + 1$ when n is odd.

For routines using RFP there is the option of storing the rectangle as described above (TRANSR = 'N') or its transpose (TRANSR = 'T', for real A) or its conjugate transpose (TRANSR = 'C', for complex A).

As an example, we first consider RFP for the case $n = 2k$ with $k = 3$.

If TRANSR = 'N', then AR holds A as follows:

For UPLO = 'U' the upper trapezoid $\mathrm{AR}(1:6, 1:3)$ consists of the last three columns of A upper. The lower triangle $\mathrm{AR}(5:7, 1:3)$ consists of the transpose of the first three columns of A upper.

For UPLO = 'L' the lower trapezoid $\mathrm{AR}(2:7, 1:3)$ consists of the first three columns of A lower. The upper triangle $\mathrm{AR}(1:3, 1:3)$ consists of the transpose of the last three columns of A lower.

If TRANSR = 'T', then AR in both UPLO cases is just the transpose of AR as defined when TRANSR = 'N'.

UPLO	**Triangle of matrix A**	**Rectangular Full Packed matrix AR**	
		TRANSR = 'N'	TRANSR = 'T'
`U'	$\begin{pmatrix} \mathbf{00} & \mathbf{01} & \mathbf{02} & 03 & 04 & 05 \\ & \mathbf{11} & \mathbf{12} & 13 & 14 & 15 \\ & & \mathbf{22} & 23 & 24 & 25 \\ & & & 33 & 34 & 35 \\ & & & & 44 & 45 \\ & & & & & 55 \end{pmatrix}$	$\begin{matrix} 03 & 04 & 05 \\ 13 & 14 & 15 \\ 23 & 24 & 25 \\ 33 & 34 & 35 \\ \mathbf{00} & 44 & 45 \\ \mathbf{01} & \mathbf{11} & 55 \\ \mathbf{02} & \mathbf{12} & \mathbf{22} \end{matrix}$	$\begin{matrix} 03 & 13 & 23 & 33 & \mathbf{00} & \mathbf{01} & \mathbf{02} \\ 04 & 14 & 24 & 34 & 44 & \mathbf{11} & \mathbf{12} \\ 05 & 15 & 25 & 35 & 45 & 55 & \mathbf{22} \end{matrix}$
`L'	$\begin{pmatrix} 00 & & & & & \\ 10 & 11 & & & & \\ 20 & 21 & 22 & & & \\ 30 & 31 & 32 & \mathbf{33} & & \\ 40 & 41 & 42 & \mathbf{43} & \mathbf{44} & \\ 50 & 51 & 52 & \mathbf{53} & \mathbf{54} & \mathbf{55} \end{pmatrix}$	$\begin{matrix} \mathbf{33} & \mathbf{43} & \mathbf{53} \\ 00 & \mathbf{44} & \mathbf{54} \\ 10 & 11 & \mathbf{55} \\ 20 & 21 & 22 \\ 30 & 31 & 32 \\ 40 & 41 & 42 \\ 50 & 51 & 52 \end{matrix}$	$\begin{matrix} \mathbf{33} & 00 & 10 & 20 & 30 & 40 & 50 \\ \mathbf{43} & \mathbf{44} & 11 & 21 & 31 & 41 & 51 \\ \mathbf{53} & \mathbf{54} & \mathbf{55} & 22 & 32 & 42 & 52 \end{matrix}$

Now we consider RFP for the case $n = 2k + 1$ and $k = 2$.

If TRANSR = 'N'. AR holds A as follows:

> if UPLO = 'U' the upper trapezoid $\mathrm{AR}(1:5, 1:3)$ consists of the last three columns of A upper. The lower triangle $\mathrm{AR}(4:5, 1:2)$ consists of the transpose of the first two columns of A upper;
>
> if UPLO = 'L' the lower trapezoid $\mathrm{AR}(1:5, 1:3)$ consists of the first three columns of A lower. The upper triangle $\mathrm{AR}(1:2, 2:3)$ consists of the transpose of the last two columns of A lower.

If TRANSR = 'T'. AR in both UPLO cases is just the transpose of AR as defined when TRANSR = 'N'.

UPLO	Triangle of matrix A	Rectangular Full Packed matrix AR	
		TRANSR = 'N'	TRANSR = 'T'
\`U'	$\begin{pmatrix} \mathbf{00} & \mathbf{01} & 02 & 03 & 04 \\ & \mathbf{11} & 12 & 13 & 14 \\ & & 22 & 23 & 24 \\ & & & 33 & 34 \\ & & & & 44 \end{pmatrix}$	$\begin{matrix} 02 & 03 & 04 \\ 12 & 13 & 14 \\ 22 & 23 & 24 \\ \mathbf{00} & 33 & 34 \\ \mathbf{01} & \mathbf{11} & 44 \end{matrix}$	$\begin{matrix} 02 & 12 & 22 & \mathbf{00} & \mathbf{01} \\ 03 & 13 & 23 & 33 & \mathbf{11} \\ 04 & 14 & 24 & 34 & 44 \end{matrix}$
\`L'	$\begin{pmatrix} 00 & & & & \\ 10 & 11 & & & \\ 20 & 21 & 22 & & \\ 30 & 31 & 32 & \mathbf{33} & \\ 40 & 41 & 42 & \mathbf{43} & \mathbf{44} \end{pmatrix}$	$\begin{matrix} 00 & \mathbf{33} & \mathbf{43} \\ 10 & 11 & \mathbf{44} \\ 20 & 21 & 22 \\ 30 & 31 & 32 \\ 40 & 41 & 42 \end{matrix}$	$\begin{matrix} 00 & 10 & 20 & 30 & 40 & 50 \\ \mathbf{33} & 11 & 21 & 31 & 41 & 51 \\ \mathbf{43} & \mathbf{44} & 22 & 32 & 42 & 52 \end{matrix}$

Explicitly, in the real matrix case, AR is a one-dimensional array of length $n(n+1)/2$ and contains the elements of A as follows:

for UPLO = 'U' and TRANSR = 'N',
- a_{ij} is stored in $\mathrm{AR}((2k+1)(i-1)+j+k+1)$, for $1 \leq j \leq k$ and $1 \leq i \leq j$, and
- a_{ij} is stored in $\mathrm{AR}((2k+1)(j-k-1)+i)$, for $k < j \leq n$ and $1 \leq i \leq j$;

for UPLO = 'U' and TRANSR = 'T',
- a_{ij} is stored in $\mathrm{AR}(q(j+k)+i)$, for $1 \leq j \leq k$ and $1 \leq i \leq j$, and
- a_{ij} is stored in $\mathrm{AR}(q(i-1)+j-k)$, for $k < j \leq n$ and $1 \leq i \leq j$;

for UPLO = 'L' and TRANSR = 'N',
- a_{ij} is stored in $\mathrm{AR}((2k+1)(j-1)+i+k-q+1)$, for $1 \leq j \leq q$ and $j \leq i \leq n$, and
- a_{ij} is stored in $\mathrm{AR}((2k+1)(i-k-1)+j-q)$, for $q < j \leq n$ and $j \leq i \leq n$;

for UPLO = 'L' and TRANSR = 'T',
- a_{ij} is stored in $\mathrm{AR}(q(i+k-q)+j)$, for $1 \leq j \leq q$ and $1 \leq i \leq n$, and
- a_{ij} is stored in $\mathrm{AR}(q(j-1-q)+i-k)$, for $q < j \leq n$ and $1 \leq i \leq n$.

In the case of complex matrices, the assumption is that the full matrix, if it existed, would be Hermitian. Thus, when TRANSR = 'N', the triangular portion of A that is, in the real case, transposed into the notional $(2k+1)$ by q RFP matrix is also conjugated. When TRANSR = 'C' the notional q by $(2k+1)$ RFP matrix is the conjugate transpose of the corresponding TRANSR = 'N' RFP matrix. Explicitly, for complex A, the array AR contains the elements (or conjugated elements) of A as follows:

for UPLO = 'U' and TRANSR = 'N',
- $\bar{a}_{ij}$ is stored in $\mathrm{AR}((2k+1)(i-1)+j+k+1)$, for $1 \leq j \leq k$ and $1 \leq i \leq j$, and
- a_{ij} is stored in $\mathrm{AR}((2k+1)(j-k-1)+i)$, for $k < j \leq n$ and $1 \leq i \leq j$;

for UPLO = 'U' and TRANSR = 'C',
- a_{ij} is stored in $\mathrm{AR}(q(j+k)+i)$, for $1 \leq j \leq k$ and $1 \leq i \leq j$, and
- $\bar{a}_{ij}$ is stored in $\mathrm{AR}(q(i-1)+j-k)$, for $k < j \leq n$ and $1 \leq i \leq j$;

for UPLO = 'L' and TRANSR = 'N',
- a_{ij} is stored in $\mathrm{AR}((2k+1)(j-1)+i+k-q+1)$, for $1 \leq j \leq q$ and $j \leq i \leq n$, and
- $\bar{a}_{ij}$ is stored in $\mathrm{AR}((2k+1)(i-k-1)+j-q)$, for $q < j \leq n$ and $j \leq i \leq n$;

for UPLO = 'L' and TRANSR = 'C',
- $\bar{a}_{ij}$ is stored in $\mathrm{AR}(q(i+k-q)+j)$, for $1 \leq j \leq q$ and $1 \leq i \leq n$, and
- a_{ij} is stored in $\mathrm{AR}(q(j-1-q)+i-k)$, for $q < j \leq n$ and $1 \leq i \leq n$.

3.3.4 Band storage

A 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 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} is stored in $\text{AB}(k_u + 1 + i - j, j)$ 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 superdiagonals, generated by fill-in as a result of row interchanges. This means that the matrix is stored according to the above scheme, but with $k_l + k_u$ superdiagonals.

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 subdiagonals or superdiagonals, only the upper or lower triangle (as specified by UPLO) need be stored:

if UPLO = 'U', a_{ij} is stored in $\text{AB}(k + 1 + i - j, j)$ for $\max(1, j - k) \leq i \leq j$;

if UPLO = 'L', a_{ij} is stored in $\text{AB}(1 + i - j, j)$ 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 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'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & & \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} & \bar{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \bar{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

Note that different storage schemes for band matrices are used by some routines in Chapters F01, F02, F03 and F04.

3.3.5 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.

3.3.6 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 Chapters F07 and F08, 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 arguments

Most routines in this chapter have one or more option arguments, 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 argument, making the calling program more readable, but only the first character is significant. (This is a feature of Fortran 77.) For example:

```
CALL DGETRS('Transpose',...)
```

3.4.2 Problem dimensions

It is permissible for the problem dimensions (for example, M in F07ADF (DGETRF), N or NRHS in F07AEF (DGETRS)) 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 argument LWORK, which immediately follows WORK in the argument-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 argument value (see Section 3.4.4), though WORK(1) will still be set as described above.

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 argument `INFO`

Routines in this chapter do not use the usual NAG Library error-handling mechanism, involving the argument IFAIL. Instead they have a diagnostic argument INFO. (Thus they preserve complete compatibility with the LAPACK specification.)

Whereas IFAIL is an *Input/Output* argument and must be set before calling a routine, INFO is purely an *Output* argument 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 arguments such as N or LDA or option arguments of type CHARACTER have permitted values. If an illegal value of the ith argument 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 Driver and Computational Routines

3.5.1 Real matrices

Each entry in the following tables, listing real matrices, gives:

the NAG routine name and

the double precision LAPACK routine name.

	Type of matrix and storage scheme		
Operation	**general**	**general band**	**general tridiagonal**
driver	F07AAF (DGESV)	F07BAF (DGBSV)	F07CAF (DGTSV)
expert driver	F07ABF (DGESVX)	F07BBF (DGBSVX)	F07CBF (DGTSVX)
mixed precision driver	F07ACF (DSGESV)		
factorize	F07ADF (DGETRF)	F07BDF (DGBTRF)	F07CDF (DGTTRF)
solve	F07AEF (DGETRS)	F07BEF (DGBTRS)	F07CEF (DGTTRS)
scaling factors	F07AFF (DGEEQU)	F07BFF (DGBEQU)	
condition number	F07AGF (DGECON)	F07BGF (DGBCON)	F07CGF (DGTCON)
error estimate	F07AHF (DGERFS)	F07BHF (DGBRFS)	F07CHF (DGTRFS)
invert	F07AJF (DGETRI)		

Table 1
Routines for real general matrices

	Type of matrix and storage scheme					
Operation	**symmetric positive definite**	**symmetric positive definite (packed storage)**	**symmetric positive definite (RFP storage)**	**symmetric positive definite band**	**symmetric positive definite tridiagonal**	**symmetric positive semidefinite**
driver	F07FAF (DPOSV)	F07GAF (DPPSV)		F07HAF (DPBSV)	F07JAF (DPTSV)	
expert driver	F07FBF (DPOSVX)	F07GBF (DPPSVX)		F07HBF (DPBSVX)	F07JBF (DPTSVX)	
mixed precision	F07FCF (DSPOSV)					
factorize	F07FDF (DPOTRF)	F07GDF (DPPTRF)	F07WDF (DPFTRF)	F07HDF (DPBTRF)	F07JDF (DPTTRF)	F07KDF (DPSTRF)
solve	F07FEF (DPOTRS)	F07GEF (DPPTRS)	F07WEF (DPFTRS)	F07HEF (DPBTRS)	F07JEF (DPTTRS)	
scaling factors	F07FFF (DPOEQU)	F07GFF (DPPEQU)		F07HFF (DPBEQU)		
condition number	F07FGF (DPOCON)	F07GGF (DPPCON)		F07HGF (DPBCON)	F07JGF (DPTCON)	
error estimate	F07FHF (DPORFS)	F07GHF (DPPRFS)		F07HHF (DPBRFS)	F07JHF (DPTRFS)	
invert	F07FJF (DPOTRI)	F07GJF (DPPTRI)	F07WJF (DPFTRI)			

Table 2
Routines for real symmetric positive definite and positive semidefinite matrices

	Type of matrix and storage scheme	
Operation	**symmetric indefinite**	**symmetric indefinite (packed storage)**
driver	F07MAF (DSYSV)	F07PAF (DSPSV)
expert driver	F07MBF (DSYSVX)	F07PBF (DSPSVX)
factorize	F07MDF (DSYTRF)	F07PDF (DSPTRF)
solve	F07MEF (DSYTRS)	F07PEF (DSPTRS)
condition number	F07MGF (DSYCON)	F07PGF (DSPCON)
error estimate	F07MHF (DSYRFS)	F07PHF (DSPRFS)
invert	F07MJF (DSYTRI)	F07PJF (DSPTRI)

Table 3
Routines for real symmetric indefinite matrices

	Type of matrix and storage scheme			
Operation	**triangular**	**triangular (packed storage)**	**triangular (RFP storage)**	**triangular band**
solve	F07TEF (DTRTRS)	F07UEF (DTPTRS)		F07VEF (DTBTRS)
condition number	F07TGF (DTRCON)	F07UGF (DTPCON)		F07VGF (DTBCON)
error estimate	F07THF (DTRRFS)	F07UHF (DTPRFS)		F07VHF (DTBRFS)
invert	F07TJF (DTRTRI)	F07UJF (DTPTRI)	F07WKF (DTFTRI)	

Table 4
Routines for real triangular matrices

3.5.2 Complex matrices

Each entry in the following tables, listing complex matrices, gives:

the NAG routine name and

the double precision LAPACK routine name.

	Type of matrix and storage scheme		
Operation	**general**	**general band**	**general tridiagonal**
driver	F07ANF (ZGESV)	F07BNF (ZGBSV)	F07CNF (ZGTSV)
expert driver	F07APF (ZGESVX)	F07BPF (ZGBSVX)	F07CPF (ZGTSVX)
mixed precision driver	F07AQF (ZCGESV)		
factorize	F07ARF (ZGETRF)	F07BRF (ZGBTRF)	F07CRF (ZGTTRF)
solve	F07ASF (ZGETRS)	F07BSF (ZGBTRS)	F07CSF (ZGTTRS)
scaling factors	F07ATF (ZGEEQU)	F07BTF (ZGBEQU)	
condition number	F07AUF (ZGECON)	F07BUF (ZGBCON)	F07CUF (ZGTCON)
error estimate	F07AVF (ZGERFS)	F07BVF (ZGBRFS)	F07CVF (ZGTRFS)
invert	F07AWF (ZGETRI)		

Table 5
Routines for complex general matrices

	Type of matrix and storage scheme					
Operation	**Hermitian positive definite**	**Hermitian positive definite (packed storage)**	**Hermitian positive definite (RFP storage)**	**Hermitian positive definite band**	**Hermitian positive definite tridiagonal**	**Hermitian positive semidefinite**
driver	F07FNF (ZPOSV)	F07GNF (ZPPSV)		F07HNF (ZPBSV)	F07JNF (ZPTSV)	
expert driver	F07FPF (ZPOSVX)	F07GPF (ZPPSVX)		F07HPF (ZPBSVX)	F07JPF (ZPTSVX)	
mixed precision driver	F07FQF (ZCPOSV)					
factorize	F07FRF (ZPOTRF)	F07GRF (ZPPTRF)	F07WRF (ZPFTRF)	F07HRF (ZPBTRF)	F07JRF (ZPTTRF)	F07KRF (ZPSTRF)
solve	F07FSF (ZPOTRS)	F07GSF (ZPPTRS)	F07WSF (ZPFTRS)	F07HSF (ZPBTRS)	F07JSF (ZPTTRS)	
scaling factors	F07FTF (ZPOE QU)	F07GTF (ZPPE QU)				
condition number	F07FUF (ZPO CON)	F07GUF (ZPPCON)		F07HUF (ZPBCON)	F07JUF (ZPTCON)	
error estimate	F07FVF (ZPORFS)	F07GVF (ZPPRFS)		F07HVF (ZPBRFS)	F07JVF (ZPTRFS)	
invert	F07FWF (ZPO TRI)	F07GWF (ZPPTRI)	F07WWF (ZPFTRI)			

Table 6
Routines for complex Hermitian positive definite and positive semidefinite matrices

	Type of matrix and storage scheme			
Operation	**Hermitian indefinite**	**symmetric indefinite (packed storage)**	**Hermitian indefinite band**	**symmetric indefinite tridiagonal**
driver	F07MNF (ZHESV)	F07NNF (ZSYSV)	F07PNF (ZHPSV)	F07QNF (ZSPSV)
expert driver	F07MPF (ZHESVX)	F07NPF (ZSYSVX)	F07PPF (ZHPSVX)	F07QPF (ZSPSVX)
factorize	F07MRF (ZHETRF)	F07NRF (ZSYTRF)	F07PRF (ZHPTRF)	F07QRF (ZSPTRF)
solve	F07MSF (ZHETRS)	F07NSF (ZSYTRS)	F07PSF (ZHPTRS)	F07QSF (ZSPTRS)
condition number	F07MUF (ZHECON)	F07NUF (ZSYCON)	F07PUF (ZHPCON)	F07QUF (ZSPCON)
error estimate	F07MVF (ZHERFS)	F07NVF (ZSYRFS)	F07PVF (ZHPRFS)	F07QVF (ZSPRFS)
invert	F07MWF (ZHETRI)	F07NWF (ZSYTRI)	F07PWF (ZHPTRI)	F07QWF (ZSPTRI)

Table 7
Routines for complex Hermitian and symmetric indefinite matrices

	Type of matrix and storage scheme			
Operation	**triangular**	**triangular (packed storage)**	**triangular (RFP storage)**	**triangular band**
solve	F07TSF (ZTRTRS)	F07USF (ZTPTRS)		F07VSF (ZTBTRS)
condition number	F07TUF (ZTRCON)	F07UUF (ZTPCON)		F07VUF (ZTBCON)
error estimate	F07TVF (ZTRRFS)	F07UVF (ZTPRFS)		F07VVF (ZTBRFS)
invert	F07TWF (ZTRTRI)	F07UWF (ZTPTRI)	F07WXF (ZTFTRI)	

Table 8
Routines for complex triangular matrices

4 Functionality Index

    - 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)
    - real symmetric positive definite tridiagonal matrix ... F07JGF (DPTCON)
    - real tridiagonal matrix ... F07CGF (DGTCON)
  - 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)

LDL^{T} factorization,
  - complex Hermitian positive definite tridiagonal matrix ... F07JRF (ZPTTRF)
  - real symmetric positive definite tridiagonal matrix ... F07JDF (DPTTRF)

LL^{T} or $U^{\mathrm{T}}U$ 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)
  - complex Hermitian positive definite matrix, RFP storage ... F07WRF (ZPFTRF)
  - complex Hermitian positive semidefinite matrix ... F07KRF (ZPSTRF)
  - real symmetric positive definite band matrix ... F07HDF (DPBTRF)
  - real symmetric positive definite matrix ... F07FDF (DPOTRF)
  - real symmetric positive definite matrix, packed storage ... F07GDF (DPPTRF)
  - real symmetric positive definite matrix, RFP storage ... F07WDF (DPFTRF)
  - real symmetric positive semidefinite matrix ... F07KDF (DPSTRF)

LU factorization,
  - complex band matrix ... F07BRF (ZGBTRF)
  - complex matrix ... F07ARF (ZGETRF)
  - complex tridiagonal matrix ... F07CRF (ZGTTRF)
  - real band matrix ... F07BDF (DGBTRF)
  - real matrix ... F07ADF (DGETRF)
  - real tridiagonal matrix ... F07CDF (DGTTRF)

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 Hermitian positive definite matrix, RFP storage ... F07WWF (ZPFTRI)
    - 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)
    - real symmetric positive definite matrix, RFP storage ... F07WJF (DPFTRI)
  - complex triangular matrix ... F07TWF (ZTRTRI)
  - complex triangular matrix, packed storage ... F07UWF (ZTPTRI)
  - complex triangular matrix, RFP storage,
    - expert driver ... F07WXF (ZTFTRI)
  - real triangular matrix ... F07TJF (DTRTRI)
  - real triangular matrix, packed storage ... F07UJF (DTPTRI)

5 Auxiliary Routines Associated with Library Routine Arguments

None.

6 Routines Withdrawn or Scheduled for Withdrawal

None.

7 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

Wilkinson J H (1965) *The Algebraic Eigenvalue Problem* Oxford University Press, Oxford