NAG Fortran Library Chapter Introduction

F08 – Least-squares and Eigenvalue Problems (LAPACK)

Contents

1 Scope of the Chapter

This chapter provides routines for the solution of linear least-squares problems, eigenvalue problems and singular value problems, as well as associated computations. It provides routines for:

- solution of linear least-squares problems
- solution of symmetric eigenvalue problems
- solution of nonsymmetric eigenvalue problems
- solution of singular value problems
- solution of generalized symmetric-definite eigenvalue problems
- matrix factorizations associated with the above problems
- estimating condition numbers of eigenvalues and eigenvectors
- estimating the numerical rank of a matrix
- solution of the Sylvester matrix equation

Routines are provided for both *real* and *complex* data.

For a general introduction to the solution of linear least-squares problems, you should turn first to Chapter F04. The decision trees, at the end of Chapter F04, direct you to the most appropriate routines in Chapter F04 or Chapter F08. Chapter F04 contains *Black Box* routines which enable standard linear least-squares problems to be solved by a call to a single routine.

For a general introduction to eigenvalue and singular value problems, you should turn first to Chapter F02. The decision trees, at the end of Chapter F02, direct you to the most appropriate routines in Chapter F02. Chapter F02 contains *Black Box* routines which enable some standard types of problem to be solved by a call to a single routine. Often routines in Chapter F02 call Chapter F08 routines to perform the necessary computational tasks. However, divide and conquer algorithms for symmetric (Hermitian) eigenvalue problem are available only in this chapter and they can be considered as *Black Box* routines.

The routines in this chapter (F08) handle only *dense*, *band*, *tridiagonal* and *Hessenberg* matrices (not matrices with more specialized structures, or general sparse matrices). The decision trees in Section 4 direct you to the most appropriate routines in Chapter F08.

The routines in this chapter have all been derived from the LAPACK project (see Anderson *et al.* (1999)). They have been designed to be efficient on a wide range of high-performance computers, without compromising efficiency on conventional serial machines.

It is not expected that every user will need to read all of the following sections, but rather will pick out those sections relevant to their particular problem.

2 Background to the Problems

This section is only a brief introduction to the numerical solution of linear least-squares problems, eigenvalue and singular value problems. Consult a standard textbook for a more thorough discussion, for example Golub and van Loan (1996).

2.1 Linear Least-squares Problems

The *linear least-squares problem* is

$$\underset{x}{\text{minimize}} \|b - Ax\|_2, \tag{1}$$

where A is an m by n matrix, b is a given m element vector and x is the n element solution vector.

In the most usual case $m \geq n$ and $\text{rank}(A) = n$, so that A has *full rank* and in this case the solution to problem ;(1) of is unique; the problem is also referred to as finding a *least-squares solution* to an *overdetermined* system of linear equations.

When $m < n$ and $\text{rank}(A) = m$, there are an infinite number of solutions x which exactly satisfy $b - Ax = 0$. In this case it is often useful to find the unique solution x which minimizes $\|x\|_2$, and the

problem is referred to as finding a *minimum-norm solution* to an *underdetermined* system of linear equations.

In the general case when we may have $\text{rank}(A) < \min(m, n)$ – in other words, A may be *rank-deficient* – we seek the *minimum-norm least-squares* solution x which minimizes both $\|x\|_2$ and $\|b - Ax\|_2$.

This chapter (F08) contains computational routines that can be combined with routines in Chapter F07 to solve these linear least-squares problems. Chapter F04 contains Black Box routines to solve these linear least-squares problems in standard cases. The next two sections discuss the factorizations that can be used in the solution of linear least-squares problems.

2.2 Orthogonal Factorizations and Least-squares Problems

A number of routines are provided for factorizing a general rectangular m by n matrix A, as the product of an *orthogonal* matrix (*unitary* if complex) and a *triangular* (or possibly trapezoidal) matrix.

A real matrix Q is *orthogonal* if $Q^T Q = I$; a complex matrix Q is *unitary* if $Q^H Q = I$. Orthogonal or unitary matrices have the important property that they leave the two-norm of a vector invariant, so that

$$\|x\|_2 = \|Qx\|_2,$$

if Q is orthogonal or unitary. They usually help to maintain numerical stability because they do not amplify rounding errors.

Orthogonal factorizations are used in the solution of linear least-squares problems. They may also be used to perform preliminary steps in the solution of eigenvalue or singular value problems, and are useful tools in the solution of a number of other problems.

2.2.1 *QR* factorization

The most common, and best known, of the factorizations is the QR *factorization* given by

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}, \quad \text{if } m \geq n,$$

where R is an n by n upper triangular matrix and Q is an m by m orthogonal (or unitary) matrix. If A is of full rank n, then R is non-singular. It is sometimes convenient to write the factorization as

$$A = (Q_1 \; Q_2)\begin{pmatrix} R \\ 0 \end{pmatrix}$$

which reduces to

$$A = Q_1 R,$$

where Q_1 consists of the first n columns of Q, and Q_2 the remaining $m - n$ columns.

If $m < n$, R is trapezoidal, and the factorization can be written

$$A = Q(R_1 \; R_2), \quad \text{if } m < n,$$

where R_1 is upper triangular and R_2 is rectangular.

The QR factorization can be used to solve the linear least-squares problem ;(1) of when $m \geq n$ and A is of full rank, since

$$\|b - Ax\|_2 = \|Q^T b - Q^T Ax\|_2 = \left\| \begin{matrix} c_1 - Rx \\ c_2 \end{matrix} \right\|,$$

where

$$c \equiv \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} Q_1^T b \\ Q_2^T b \end{pmatrix} = Q^T b;$$

and c_1 is an n element vector. Then x is the solution of the upper triangular system

$$Rx = c_1.$$

The residual vector r is given by

$$r = b - Ax = Q\begin{pmatrix} 0 \\ c_2 \end{pmatrix}.$$

The residual sum of squares $\|r\|_2^2$ may be computed without forming r explicitly, since

$$\|r\|_2 = \|b - Ax\|_2 = \|c_2\|_2.$$

2.2.2 LQ factorization

The LQ *factorization* is given by

$$A = (L \quad 0)Q = (L \quad 0)\begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = LQ_1, \quad \text{if } m \leq n,$$

where L is m by m lower triangular, Q is n by n orthogonal (or unitary), Q_1 consists of the first m rows of Q, and Q_2 the remaining $n - m$ rows.

The LQ factorization of A is essentially the same as the QR factorization of A^T (A^H if A is complex), since

$$A = (L \quad 0)Q \Leftrightarrow A^T = Q^T\begin{pmatrix} L^T \\ 0 \end{pmatrix}.$$

The LQ factorization may be used to find a minimum norm solution of an underdetermined system of linear equations $Ax = b$ where A is m by n with $m < n$ and has rank m. The solution is given by

$$x = Q^T\begin{pmatrix} L^{-1}b \\ 0 \end{pmatrix}.$$

2.2.3 QR factorization with column pivoting

To solve a linear least-squares problem ;(1) of when A is not of full rank, or the rank of A is in doubt, we can perform either a QR factorization with column pivoting or a singular value decomposition.

The QR *factorization with column pivoting* is given by

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}P^T, \quad m \geq n,$$

where Q and R are as before and P is a (real) permutation matrix, chosen (in general) so that

$$|r_{11}| \geq |r_{22}| \geq \cdots \geq |r_{nn}|$$

and moreover, for each k,

$$|r_{kk}| \geq \|R_{k:j,j}\|_2, \quad j = k+1, \ldots, n.$$

If we put

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

where R_{11} is the leading k by k upper triangular submatrix of R then, in exact arithmetic, if $\text{rank}(A) = k$, the whole of the submatrix R_{22} in rows and columns $k+1$ to n would be zero. In numerical computation, the aim must be to determine an index k, such that the leading submatrix R_{11} is well-conditioned, and R_{22} is negligible, so that

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \simeq \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}.$$

Then k is the effective rank of A. See Golub and van Loan (1996) for a further discussion of numerical rank determination.

The so-called basic solution to the linear least-squares problem ;(1) of can be obtained from this factorization as

$$x = P\begin{pmatrix} R_{11}^{-1}\hat{c}_1 \\ 0 \end{pmatrix},$$

where $\hat{c}_1$ consists of just the first k elements of $c = Q^T b$.

2.3 The Singular Value Decomposition

The *singular value decomposition* (SVD) of an m by n matrix A is given by

$$A = U\Sigma V^T, \quad (A = U\Sigma V^H \text{ in the complex case})$$

where U and V are orthogonal (unitary) and Σ is an m by n diagonal matrix with real diagonal elements, σ_i, such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0.$$

The σ_i are the *singular values* of A and the first $\min(m,n)$ columns of U and V are the *left* and *right singular vectors* of A. The singular values and singular vectors satisfy

$$Av_i = \sigma_i u_i \quad \text{and} \quad A^T u_i = \sigma_i v_i \quad (\text{or } A^H u_i = \sigma_i v_i)$$

where u_i and v_i are the ith columns of U and V respectively.

The computation proceeds in the following stages.

1. The matrix A is reduced to bidiagonal form $A = U_1 B V_1^T$ if A is real ($A = U_1 B V_1^H$ if A is complex), where U_1 and V_1 are orthogonal (unitary if A is complex), and B is real and upper bidiagonal when $m \geq n$ and lower bidiagonal when $m < n$, so that B is nonzero only on the main diagonal and either on the first superdiagonal (if $m \geq n$) or the first subdiagonal (if $m < n$).
2. The SVD of the bidiagonal matrix B is computed as $B = U_2 \Sigma V_2^T$, where U_2 and V_2 are orthogonal and Σ is diagonal as described above. The singular vectors of A are then $U = U_1 U_2$ and $V = V_1 V_2$.

If $m \gg n$, it may be more efficient to first perform a QR factorization of A, and then compute the SVD of the n by n matrix R, since if $A = QR$ and $R = U\Sigma V^T$, then the SVD of A is given by $A = (QU)\Sigma V^T$.

Similarly, if $m \ll n$, it may be more efficient to first perform an LQ factorization of A.

2.4 The Singular Value Decomposition and Least-squares Problems

The SVD may be used to find a minimum norm solution to a (possibly) rank-deficient linear least-squares problem ;(1) of . The effective rank, k, of A can be determined as the number of singular values which exceed a suitable threshold. Let $\hat{\Sigma}$ be the leading k by k submatrix of Σ, and $\hat{V}$ be the matrix consisting of the first k columns of V. Then the solution is given by

$$x = \hat{V}\hat{\Sigma}^{-1}\hat{c}_1,$$

where $\hat{c}_1$ consists of the first k elements of $c = U^T b = U_2^T U_1^T b$.

2.5 Symmetric Eigenvalue Problems

The *symmetric eigenvalue problem* is to find the *eigenvalues*, λ, and corresponding *eigenvectors*, $z \neq 0$, such that

$$Az = \lambda z, \quad A = A^T, \quad \text{where } A \text{ is real.}$$

For the *Hermitian eigenvalue problem* we have

$$Az = \lambda z, \quad A = A^H, \quad \text{where } A \text{ is complex.}$$

For both problems the eigenvalues λ are real.

When all eigenvalues and eigenvectors have been computed, we write

$$A = Z\Lambda Z^T \quad (\text{or } A = Z\Lambda Z^H \text{ if complex}),$$

where Λ is a diagonal matrix whose diagonal elements are the eigenvalues, and Z is an orthogonal (or unitary) matrix whose columns are the eigenvectors. This is the classical *spectral factorization* of A.

The basic task of the symmetric eigenproblem routines is to compute values of λ and, optionally, corresponding vectors z for a given matrix A. This computation proceeds in the following stages.

1. The real symmetric or complex Hermitian matrix A is reduced to *real tridiagonal form* T. If A is real symmetric this decomposition is $A = QTQ^T$ with Q orthogonal and T symmetric tridiagonal. If A is complex Hermitian, the decomposition is $A = QTQ^H$ with Q unitary and T, as before, *real* symmetric tridiagonal.
2. Eigenvalues and eigenvectors of the real symmetric tridiagonal matrix T are computed. If all eigenvalues and eigenvectors are computed, this is equivalent to factorizing T as $T = S\Lambda S^T$, where S is orthogonal and Λ is diagonal. The diagonal entries of Λ are the eigenvalues of T, which are also the eigenvalues of A, and the columns of S are the eigenvectors of T; the eigenvectors of A are the columns of $Z = QS$, so that $A = Z\Lambda Z^T$ ($Z\Lambda Z^H$ when A is complex Hermitian).

This chapter now supports three primary algorithms for computing eigenvalues and eigenvectors of real symmetric matrices and complex Hermitian matrices. They are:

(i) the divide and conquer algorithm;

(ii) the QR algorithm;

(iii) bisection followed by inverse iteration.

The divide and conquer algorithm is generally more efficient than the traditional QR algorithm and is recommended for computing all eigenvalues and eigenvectors. Furthermore, eigenvalues and eigenvectors can be obtained by calling one single routine in the case of the divide and conquer algorithm. In general, more than one routine has to be called if the QR algorithm or bisection followed by inverse iteration is used.

2.6 Generalized Symmetric-Definite Eigenvalue Problems

This section is concerned with the solution of the generalized eigenvalue problems $Az = \lambda Bz$, $ABz = \lambda z$, and $BAz = \lambda z$, where A and B are real symmetric or complex Hermitian and B is positive-definite. Each of these problems can be reduced to a standard symmetric eigenvalue problem, using a Cholesky factorization of B as either $B = LL^T$ or $B = U^TU$ (LL^H or U^HU in the Hermitian case).

With $B = LL^T$, we have

$$Az = \lambda Bz \Rightarrow (L^{-1}AL^{-T})(L^Tz) = \lambda(L^Tz).$$

Hence the eigenvalues of $Az = \lambda Bz$ are those of $Cy = \lambda y$, where C is the symmetric matrix $C = L^{-1}AL^{-T}$ and $y = L^Tz$. In the complex case C is Hermitian with $C = L^{-1}AL^{-H}$ and $y = L^Hz$.

Table 1 summarizes how each of the three types of problem may be reduced to standard form $Cy = \lambda y$, and how the eigenvectors z of the original problem may be recovered from the eigenvectors y of the reduced problem. The table applies to real problems; for complex problems, transposed matrices must be replaced by conjugate-transposes.

	Type of problem	Factorization of B	Reduction	Recovery of eigenvectors
1.	$Az = \lambda Bz$	$B = LL^T$ $B = U^T U$	$C = L^{-1}AL^{-T}$ $C = U^{-T}AU^{-1}$	$z = L^{-T}y$ $z = U^{-1}y$
2.	$ABz = \lambda z$	$B = LL^T$ $B = U^T U$	$C = L^T AL$ $C = UAU^T$	$z = L^{-T}y$ $z = U^{-1}y$
3.	$BAz = \lambda z$	$B = LL^T$ $B = U^T U$	$C = L^T AL$ $C = UAU^T$	$z = Ly$ $z = U^T y$

Table1
Reduction of generalized symmetric-definite eigenproblems to standard problems

When the generalized symmetric-definite problem has been reduced to the corresponding standard problem $Cy = \lambda y$, this may then be solved using the routines described in the previous section. No special routines are needed to recover the eigenvectors z of the generalized problem from the eigenvectors y of the standard problem, because these computations are simple applications of Level 2 or Level 3 BLAS (see Chapter F06).

2.7 Packed Storage for Symmetric Matrices

Routines which handle symmetric matrices are usually designed so that they use either the upper or lower triangle of the matrix; it is not necessary to store the whole matrix. If either the upper or lower triangle is stored conventionally in the upper or lower triangle of a two-dimensional array, the remaining elements of the array can be used to store other useful data. However, that is not always convenient, and if it is important to economize on storage, the upper or lower triangle can be stored in a one-dimensional array of length $n(n+1)/2$; that is, the storage is almost halved.

This storage format is referred to as *packed storage*; it is described in Section 3.3.

Routines designed for packed storage are usually less efficient, especially on high-performance computers, so there is a trade-off between storage and efficiency.

2.8 Band Matrices

A *band* matrix is one whose elements are confined to a relatively small number of sub-diagonals or super-diagonals on either side of the main diagonal. Algorithms can take advantage of bandedness to reduce the amount of work and storage required. The storage scheme for band matrices is described in Section 3.3.

If the problem is generalized symmetric definite eigenvalue problem $Az = \lambda Bz$ and the matrices A and B are additionally banded, the matrix C as defined in Section 2.6 is, in general, full. We can reduce the problem to a banded standard problem by modifying the definition of C thus:

$$C = X^T AX, \quad \text{where} \quad X = U^{-1}Q \quad \text{or } L^{-T}Q,$$

where Q is an orthogonal matrix chosen to ensure that C has bandwidth no greater than that of A.

A further refinement is possible when A and B are banded, which halves the amount of work required to form C. Instead of the standard Cholesky factorization of B as $U^T U$ or LL^T, we use a *split Cholesky* factorization $B = S^T S$, where

$$S = \begin{pmatrix} U_{11} & \\ M_{21} & L_{22} \end{pmatrix}$$

with U_{11} upper triangular and L_{22} lower triangular of order approximately $n/2$; S has the same bandwidth as B.

2.9 Nonsymmetric Eigenvalue Problems

The *nonsymmetric eigenvalue problem* is to find the *eigenvalues*, λ, and corresponding *eigenvectors*, $v \neq 0$, such that

$$Av = \lambda v.$$

More precisely, a vector v as just defined is called a *right eigenvector* of A, and a vector $u \neq 0$ satisfying

$$u^T A = \lambda u^T \quad (u^H A = \lambda u^H \quad \text{when } u \text{ is complex})$$

is called a *left eigenvector* of A.

A real matrix A may have complex eigenvalues, occurring as complex conjugate pairs.

This problem can be solved via the *Schur factorization* of A, defined in the real case as

$$A = ZTZ^T,$$

where Z is an orthogonal matrix and T is an upper quasi-triangular matrix with 1 by 1 and 2 by 2 diagonal blocks, the 2 by 2 blocks corresponding to complex conjugate pairs of eigenvalues of A. In the complex case, the Schur factorization is

$$A = ZTZ^H,$$

where Z is unitary and T is a complex upper triangular matrix.

The columns of Z are called the *Schur vectors*. For each k ($1 \leq k \leq n$), the first k columns of Z form an orthonormal basis for the *invariant subspace* corresponding to the first k eigenvalues on the diagonal of T. Because this basis is orthonormal, it is preferable in many applications to compute Schur vectors rather than eigenvectors. It is possible to order the Schur factorization so that any desired set of k eigenvalues occupy the k leading positions on the diagonal of T.

The two basic tasks of the nonsymmetric eigenvalue routines are to compute, for a given matrix A, all n values of λ and, if desired, their associated right eigenvectors v and/or left eigenvectors u, and the Schur factorization.

These two basic tasks can be performed in the following stages.

1. A general matrix A is reduced to *upper Hessenberg form* H which is zero below the first subdiagonal. The reduction may be written $A = QHQ^T$ with Q orthogonal if A is real, or $A = QHQ^H$ with Q unitary if A is complex.
2. The upper Hessenberg matrix H is reduced to Schur form T, giving the Schur factorization $H = STS^T$ (for H real) or $H = STS^H$ (for H complex). The matrix S (the Schur vectors of H) may optionally be computed as well. Alternatively S may be postmultiplied into the matrix Q determined in stage 1, to give the matrix $Z = QS$, the Schur vectors of A. The eigenvalues are obtained from the diagonal elements or diagonal blocks of T.
3. Given the eigenvalues, the eigenvectors may be computed in two different ways. Inverse iteration can be performed on H to compute the eigenvectors of H, and then the eigenvectors can be multiplied by the matrix Q in order to transform them to eigenvectors of A. Alternatively the eigenvectors of T can be computed, and optionally transformed to those of H or A if the matrix S or Z is supplied.

The accuracy with which eigenvalues can be obtained can often be improved by *balancing* a matrix. This is discussed further in Section 2.12.6 below.

2.10 Generalized Nonsymmetric Eigenvalue Problem

The *generalized nonsymmetric eigenvalue problem* is to find the eigenvalues, λ, and corresponding *eigenvectors*, $v \neq 0$, such that

$$Av = \lambda Bv.$$

More precisely, a vector v as just defined is called a *right eigenvector* of the matrix pair (A, B), and a vector $u \neq 0$ satisfying

$$u^T A = \lambda u^T B \quad (u^H A = \lambda u^H B \text{ when } u \text{ is complex})$$

is called a *left eigenvector* of the matrix pair (A, B).

If B is singular then the problem has one or more *infinite eigenvalues* $\lambda = \infty$, corresponding to $Bv = 0$. Note that if A is non-singular, then the equivalent problem $\mu Av = Bv$ is perfectly well defined and an infinite eigenvalue corresponds to $\mu = 0$. To deal with both finite (including zero) and infinite eigenvalues, the routines in this chapter do not compute λ explicitly, but rather return a pair of numbers (α, β) such that if $\beta \neq 0$

$$\lambda = \alpha/\beta$$

and if $\alpha \neq 0$ and $\beta = 0$ then $\lambda = \infty$. β is always returned as real and non-negative. Of course, computationally an infinite eigenvalue may correspond to a small β rather than an exact zero.

For a given pair (A, B) the set of all the matrices of the form $(A - \lambda B)$ is called a matrix *pencil* and λ and v are said to be an eigenvalue and eigenvector of the pencil $(A - \lambda B)$. If A and B are both singular and share a common null-space then

$$det(A - \lambda B) \equiv 0$$

so that the pencil $(A - \lambda B)$ is *singular* for all λ. In other words any λ can be regarded as an eigenvalue. In exact arithmetic a singular pencil will have $\alpha = \beta = 0$ for some (α, β). Computationally if some pair (α, β) is small then the pencil is singular, or nearly singular, and no reliance can be placed on any of the computed eigenvalues. Singular pencils can also manifest themselves in other ways; see, in particular, Sections 2.3.5.2 and 4.11.1.4 of Anderson *et al.* (1999) for further details.

The generalized eigenvalue problem can be solved via the *generalized Schur factorization* of the pair (A, B) defined in the real case as

$$A = QSZ^T, \quad B = QTZ^T,$$

where Q and Z are orthogonal, T is upper triangular with non-negative diagonal elements and S is upper quasi-triangular with 1 by 1 and 2 by 2 diagonal blocks, the 2 by 2 blocks corresponding to complex conjugate pairs of eigenvalues. In the complex case, the generalized Schur factorization is

$$A = QSZ^H, \quad B = QTZ^H,$$

where Q and Z are unitary and S and T are upper triangular, with T having real non-negative diagonal elements. The columns of Q and Z are called respectively the *left* and *right generalized Schur vectors* and span pairs of *deflating subspaces* of A and B, which are a generalization of invariant subspaces.

The two basic tasks of the generalized nonsymmetric eigenvalue routines are to compute, for a given pair (A, B), all n values of λ and, if desired, their associated right eigenvectors v and/or left eigenvectors u, and the generalized Schur factorization.

These two basic tasks can be performed in the following stages.

1. The matrix pair (A, B) is reduced to *generalized upper* Hessenberg form (H, R), where H is upper Hessenberg (zero below the first subdiagonal) and R is upper triangular. The reduction may be written as $A = Q_1 H Z_1^T, B = Q_1 R Z_1^T$ in the real case with Q_1 and Z_1 orthogonal, and $A = Q_1 H Z_1^H, B = Q_1 R Z_1^H$ in the complex case with Q_1 and Z_1 unitary.
2. The generalized upper Hessenberg form (H, R) is reduced the the generalized Schur form (S, T) using the generalized Schur factorization $H = Q_2 S Z_2^T$, $R = Q_2 T Z_2^T$ in the real case with Q_2 and Z_2 orthogonal, and $H = Q_2 S Z_2^H, R = Q_2 T Z_2^H$ in the complex case. The generalized Schur vectors of (A, B) are given by $Q = Q_1 Q_2$, $Z = Z_1 Z_2$. The eigenvalues are obtained from the diagonal elements (or blocks) of the pair (S, T).
3. Given the eigenvalues, the eigenvectors of the pair (S, T) can be computed, and optionally transformed to those of (H, R) or (A, B).

The accuracy with which eigenvalues can be obtained can often be improved by *balancing* a matrix pair. This is discussed further in Section 2.12.8 below.

2.11 The Sylvester Equation

The Sylvester equation is a matrix equation of the form

$$AX + XB = C,$$

where A, B, and C are given matrices with A being m by m, B an n by n matrix and C, and the solution matrix X, m by n matrices. The solution of a special case of this equation occurs in the computation of the condition number for an invariant subspace, but a combination of routines in this chapter allows the solution of the general Sylvester equation.

2.12 Error and Perturbation Bounds and Condition Numbers

In this section we discuss the effects of rounding errors in the solution process and the effects of uncertainties in the data, on the solution to the problem. A number of the routines in this chapter return information, such as condition numbers, that allow these effects to be assessed. First we discuss some notation used in the error bounds of later sections.

The bounds usually contain the factor $p(n)$ (or $p(m,n)$), which grows as a function of the matrix dimension n (or matrix dimensions m and n). It measures how errors can grow as a function of the matrix dimension, and represents a potentially different function for each problem. In practice, it usually grows just linearly; $p(n) \leq 10n$ is often true, although generally only much weaker bounds can be actually proved. We normally describe $p(n)$ as a 'modestly growing' function of n. For detailed derivations of various $p(n)$, see Golub and van Loan (1996) and Wilkinson (1965).

For linear equation (see Chapter F07) and least-squares solvers, we consider bounds on the relative error $\|x - \hat{x}\|/\|x\|$ in the computed solution $\hat{x}$, where x is the true solution. For eigenvalue problems we consider bounds on the error $|\lambda_i - \hat{\lambda}_i|$ in the ith computed eigenvalue $\hat{\lambda}_i$, where λ_i is the true ith eigenvalue. For singular value problems we similarly consider bounds $|\sigma_i - \hat{\sigma}_i|$.

Bounding the error in computed eigenvectors and singular vectors $\hat{v}_i$ is more subtle because these vectors are not unique: even though we restrict $\|\hat{v}_i\|_2 = 1$ and $\|v_i\|_2 = 1$, we may still multiply them by arbitrary constants of absolute value 1. So to avoid ambiguity we bound the *angular difference* between $\hat{v}_i$ and the true vector v_i, so that

$$\begin{aligned} \theta(v_i, \hat{v}_i) &= \text{acute angle between } v_i \text{ and } \hat{v}_i \\ &= \arccos |v_i^H \hat{v}_i|. \end{aligned} \tag{2}$$

When $\theta(v_i, \hat{v}_i)$ is small, we can choose a constant α with absolute value 1 so that $\|\alpha v_i - \hat{v}_i\|_2 \approx \theta(v_i, \hat{v}_i)$.

In addition to bounds for individual eigenvectors, bounds can be obtained for the spaces spanned by collections of eigenvectors. These may be much more accurately determined than the individual eigenvectors which span them. These spaces are called *invariant subspaces* in the case of eigenvectors, because if v is any vector in the space, Av is also in the space, where A is the matrix. Again, we will use angle to measure the difference between a computed space $\hat{S}$ and the true space S: $some_s$

$$\begin{aligned} \theta(S, \hat{S}) &= \text{acute angle between } S \text{ and } \hat{S} \\ &= \max_{\substack{s \in S \\ s \neq 0}} \min_{\substack{\hat{s} \in \hat{S} \\ \hat{s} \neq 0}} \theta(s, \hat{s}) \quad \text{or} \quad \max_{\substack{\hat{s} \in \hat{S} \\ \hat{s} \neq 0}} \min_{\substack{s \in S \\ s \neq 0}} \theta(s, \hat{s}) \end{aligned} \tag{3}$$

$\theta(S, \hat{S})$ may be computed as follows. Let S be a matrix whose columns are orthonormal and span S. Similarly let $\hat{S}$ be an orthonormal matrix with columns spanning $\hat{S}$. Then

$$\theta(S, \hat{S}) = \arccos \sigma_{\min}(S^H \hat{S}).$$

Finally, we remark on the accuracy of the bounds when they are large. Relative errors like $\|\hat{x} - x\|/\|x\|$ and angular errors like $\theta(\hat{v}_i, v_i)$ are only of interest when they are much less than 1. Some stated bounds are not strictly true when they are close to 1, but rigorous bounds are much more complicated and supply little extra information in the interesting case of small errors. These bounds are indicated by using the symbol $\lesssim$, or 'approximately less than', instead of the usual $\leq$. Thus, when these bounds are close to 1 or greater, they indicate that the computed answer may have no significant digits at all, but do not otherwise bound the error.

2.12.1 Least-squares problems

The conventional error analysis of linear least-squares problems goes as follows. The problem is to find the x minimizing $\|Ax - b\|_2$. Let $\hat{x}$ be the solution computed using one of the methods described above. We discuss the most common case, where A is overdetermined (i.e., has more rows than columns) and has full rank.

Then the computed solution $\hat{x}$ has a small normwise backward error. In other words $\hat{x}$ minimizes $\|(A + E)\hat{x} - (b + f)\|_2$, where

$$\max\left(\frac{\|E\|_2}{\|A\|_2}, \frac{\|f\|_2}{\|b\|_2}\right) \leq p(n)\epsilon$$

and $p(n)$ is a modestly growing function of n and ϵ is the machine precision. Let $\kappa_2(A) = \sigma_{\max}(A)/\sigma_{\min}(A)$, $\rho = \|Ax - b\|_2$, and $\sin(\theta) = \rho/\|b\|_2$. Then if $p(n)\epsilon$ is small enough, the error $\hat{x} - x$ is bounded by

$$\frac{\|x - \hat{x}\|_2}{\|x\|_2} \lesssim p(n)\epsilon\left\{\frac{2\kappa_2(A)}{\cos(\theta)} + \tan(\theta)\kappa_2^2(A)\right\}.$$

If A is rank-deficient, the problem can be *regularized* by treating all singular values less than a user-specified threshold as exactly zero. See Golub and van Loan (1996) for error bounds in this case, as well as for the underdetermined case.

The solution of the overdetermined, full-rank problem may also be characterized as the solution of the linear system of equations

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}\begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}.$$

By solving this linear system (see Chapter F07) componentwise error bounds can also be obtained Arioli *et al.* (1989).

2.12.2 The singular value decomposition

The usual error analysis of the SVD algorithm is as follows (see Golub and van Loan (1996)).

The computed SVD, $\hat{U}\hat{\Sigma}\hat{V}^T$, is nearly the exact SVD of $A + E$, i.e., $A + E = (\hat{U} + \delta\hat{U})\hat{\Sigma}(\hat{V} + \delta\hat{V})$ is the true SVD, so that $\hat{U} + \delta\hat{U}$ and $\hat{V} + \delta\hat{V}$ are both orthogonal, where $\|E\|_2/\|A\|_2 \leq p(m,n)\epsilon$, $\|\delta\hat{U}\| \leq p(m,n)\epsilon$, and $\|\delta\hat{V}\| \leq p(m,n)\epsilon$. Here $p(m,n)$ is a modestly growing function of m and n and ϵ is the machine precision. Each computed singular value $\hat{\sigma}_i$ differs from the true σ_i by an amount satisfying the bound

$$|\hat{\sigma}_i - \sigma_i| \leq p(m,n)\epsilon\sigma_1.$$

Thus large singular values (those near σ_1) are computed to high relative accuracy and small ones may not be.

The angular difference between the computed left singular vector $\hat{u}_i$ and the true u_i satisfies the approximate bound

$$\theta(\hat{u}_i, u_i) \lesssim \frac{p(m,n)\epsilon\|A\|_2}{\text{gap}_i}$$

where

$$\text{gap}_i = \min_{j \neq i}|\sigma_i - \sigma_j|$$

is the *absolute gap* between σ_i and the nearest other singular value. Thus, if σ_i is close to other singular values, its corresponding singular vector u_i may be inaccurate. The same bound applies to the computed right singular vector $\hat{v}_i$ and the true vector v_i. The gaps may be easily obtained from the computed singular values.

Let $\hat{\boldsymbol{S}}$ be the space spanned by a collection of computed left singular vectors $\{\hat{u}_i, i \in \boldsymbol{I}\}$, where $\boldsymbol{I}$ is a subset of the integers from 1 to n. Let $\boldsymbol{S}$ be the corresponding true space. Then

$$\theta(\hat{\boldsymbol{S}}, \boldsymbol{S}) \lesssim \frac{p(m,n)\epsilon\|A\|_2}{\text{gap}_{\boldsymbol{I}}}.$$

where

$$\text{gap}_{\boldsymbol{I}} = \min\{|\sigma_i - \sigma_j| \quad \text{for } i \in \boldsymbol{I},\ j \notin \boldsymbol{I}\}$$

is the absolute gap between the singular values in $\boldsymbol{I}$ and the nearest other singular value. Thus, a cluster of close singular values which is far away from any other singular value may have a well determined space $\hat{\boldsymbol{S}}$ even if its individual singular vectors are ill-conditioned. The same bound applies to a set of right singular vectors $\{\hat{v}_i, i \in \boldsymbol{I}\}$.

In the special case of bidiagonal matrices, the singular values and singular vectors may be computed much more accurately (see Demmel and Kahan (1990)). A bidiagonal matrix B has nonzero entries only on the main diagonal and the diagonal immediately above it (or immediately below it). Reduction of a dense matrix to bidiagonal form B can introduce additional errors, so the following bounds for the bidiagonal case do not apply to the dense case.

Using the routines in this chapter, each computed singular value of a bidiagonal matrix is accurate to nearly full relative accuracy, no matter how tiny it is, so that

$$|\hat{\sigma}_i - \sigma_i| \leq p(m,n)\epsilon\sigma_i.$$

The computed left singular vector $\hat{u}_i$ has an angular error at most about

$$\theta(\hat{\boldsymbol{u}}_i, u_i) \lesssim \frac{p(m,n)\epsilon}{\text{relgap}_i}$$

where

$$\text{relgap}_i = \min_{j \neq i} |\sigma_i - \sigma_j|/(\sigma_i + \sigma_j)$$

is the *relative gap* between σ_i and the nearest other singular value. The same bound applies to the right singular vector $\hat{v}_i$ and v_i. Since the relative gap may be much larger than the absolute gap, this error bound may be much smaller than the previous one. The relative gaps may be easily obtained from the computed singular values.

2.12.3 The symmetric eigenproblem

The usual error analysis of the symmetric eigenproblem is as follows (see Parlett (1980)).

The computed eigendecomposition $\hat{Z}\hat{\Lambda}\hat{Z}^T$ is nearly the exact eigendecomposition of $A + E$, i.e., $A + E = (\hat{Z} + \delta\hat{Z})\hat{\Lambda}(\hat{Z} + \delta\hat{Z})^T$ is the true eigendecomposition so that $\hat{Z} + \delta\hat{Z}$ is orthogonal, where $\|E\|_2/\|A\|_2 \leq p(n)\epsilon$ and $\|\delta\hat{Z}\|_2 \leq p(n)\epsilon$ and $p(n)$ is a modestly growing function of n and ϵ is the machine precision. Each computed eigenvalue $\hat{\lambda}_i$ differs from the true λ_i by an amount satisfying the bound

$$|\hat{\lambda}_i - \lambda_i| \leq p(n)\epsilon\|A\|_2.$$

Thus large eigenvalues (those near $\max_i |\lambda_i| = \|A\|_2$) are computed to high relative accuracy and small ones may not be.

The angular difference between the computed unit eigenvector $\hat{z}_i$ and the true z_i satisfies the approximate bound

$$\theta(\hat{z}_i, z_i) \lesssim \frac{p(n)\epsilon\|A\|_2}{\text{gap}_i}$$

if $p(n)\epsilon$ is small enough, where

$$\text{gap}_i = \min_{j \neq i} |\lambda_i - \lambda_j|$$

is the *absolute gap* between λ_i and the nearest other eigenvalue. Thus, if λ_i is close to other eigenvalues,

its corresponding eigenvector z_i may be inaccurate. The gaps may be easily obtained from the computed eigenvalues.

Let $\hat{\boldsymbol{S}}$ be the invariant subspace spanned by a collection of eigenvectors $\{\hat{z}_i, i \in \boldsymbol{I}\}$, where $\boldsymbol{I}$ is a subset of the integers from 1 to n. Let $\boldsymbol{S}$ be the corresponding true subspace. Then

$$\theta(\hat{\boldsymbol{S}}, \boldsymbol{S}) \lesssim \frac{p(n)\epsilon\|A\|_2}{\text{gap}_{\boldsymbol{I}}}$$

where

$$\text{gap}_{\boldsymbol{I}} = \min\{|\lambda_i - \lambda_j| \quad \text{for } i \in \boldsymbol{I},\ j \notin \boldsymbol{I}\}$$

is the absolute gap between the eigenvalues in $\boldsymbol{I}$ and the nearest other eigenvalue. Thus, a cluster of close eigenvalues which is far away from any other eigenvalue may have a well determined invariant subspace $\hat{\boldsymbol{S}}$ even if its individual eigenvectors are ill-conditioned.

In the special case of a real symmetric tridiagonal matrix T, routines in this chapter can compute the eigenvalues and eigenvectors much more accurately. See Anderson *et al.* (1999) for further details.

2.12.4 The generalized symmetric-definite eigenproblem

The three types of problem to be considered are $A - \lambda B$, $AB - \lambda I$ and $BA - \lambda I$. In each case A and B are real symmetric (or complex Hermitian) and B is positive-definite. We consider each case in turn, assuming that routines in this chapter are used to transform the generalized problem to the standard symmetric problem, followed by the solution of the the symmetric problem. In all cases

$$\text{gap}_i = \min_{j \neq i} |\lambda_i - \lambda_j|$$

is the *absolute gap* between λ_i and the nearest other eigenvalue.

1. $A - \lambda B$. The computed eigenvalues $\hat{\lambda}_i$ can differ from the true eigenvalues λ_i by an amount

   $$|\hat{\lambda}_i - \lambda_i| \lesssim p(n)\epsilon\|B^{-1}\|_2\|A\|_2.$$

   The angular difference between the computed eigenvector $\hat{z}_i$ and the true eigenvector z_i is

   $$\theta(\hat{z}_i, z_i) \lesssim \frac{p(n)\epsilon\|B^{-1}\|_2\|A\|_2(\kappa_2(B))^{1/2}}{\text{gap}_i}.$$

2. $AB - \lambda I$ or $BA - \lambda I$. The computed eigenvalues $\hat{\lambda}_i$ can differ from the true eigenvalues λ_i by an amount

   $$|\hat{\lambda}_i - \lambda_i| \lesssim p(n)\epsilon\|B\|_2\|A\|_2.$$

   The angular difference between the computed eigenvector $\hat{z}_i$ and the true eigenvector z_i is

   $$\theta(\hat{z}_i, z_i) \lesssim \frac{q(n)\epsilon\|B\|_2\|A\|_2(\kappa_2(B))^{1/2}}{\text{gap}_i}.$$

These error bounds are large when B is ill-conditioned with respect to inversion ($\kappa_2(B)$ is large). It is often the case that the eigenvalues and eigenvectors are much better conditioned than indicated here. One way to get tighter bounds is effective when the diagonal entries of B differ widely in magnitude, as for example with a *graded matrix*.

1. $A - \lambda B$. Let $D = \text{diag}(b_{11}^{-1/2}, \ldots, b_{nn}^{-1/2})$ be a diagonal matrix. Then replace B by DBD and A by DAD in the above bounds.

2. $AB - \lambda I$ or $BA - \lambda I$. Let $D = \text{diag}(b_{11}^{-1/2}, \ldots, b_{nn}^{-1/2})$ be a diagonal matrix. Then replace B by DBD and A by $D^{-1}AD^{-1}$ in the above bounds.

Further details can be found in Anderson *et al.* (1999).

2.12.5 The nonsymmetric eigenproblem

The nonsymmetric eigenvalue problem is more complicated than the symmetric eigenvalue problem. In this section, we just summarize the bounds. Further details can be found in Anderson *et al.* (1999).

We let $\hat{\lambda}_i$ be the ith computed eigenvalue and λ_i the ith true eigenvalue. Let $\hat{v}_i$ be the corresponding computed right eigenvector, and v_i the true right eigenvector (so $Av_i = \lambda_i v_i$). If $\boldsymbol{I}$ is a subset of the integers from 1 to n, we let $\lambda_{\boldsymbol{I}}$ denote the average of the selected eigenvalues: $\lambda_{\boldsymbol{I}} = (\sum_{i\in\boldsymbol{I}} \lambda_i)/(\sum_{i\in\boldsymbol{I}} 1)$, and similarly for $\hat{\lambda}_{\boldsymbol{I}}$. We also let $\boldsymbol{S_I}$ denote the subspace spanned by $\{v_i, i \in \boldsymbol{I}\}$; it is called a right invariant subspace because if v is any vector in $\boldsymbol{S_I}$ then Av is also in $\boldsymbol{S_I}$. $\hat{\boldsymbol{S}}_{\boldsymbol{I}}$ is the corresponding computed subspace.

The algorithms for the nonsymmetric eigenproblem are normwise backward stable: they compute the exact eigenvalues, eigenvectors and invariant subspaces of slightly perturbed matrices $A + E$, where $\|E\| \leq p(n)\epsilon\|A\|$. Some of the bounds are stated in terms of $\|E\|_2$ and others in terms of $\|E\|_F$; one may use $p(n)\epsilon$ for either quantity.

Routines are provided so that, for each $(\hat{\lambda}_i, \hat{v}_i)$ pair the two values s_i and sep_i, or for a selected subset $\boldsymbol{I}$ of eigenvalues the values $s_{\boldsymbol{I}}$ and $sep_{\boldsymbol{I}}$ can be obtained, for which the error bounds in Table 2 are true for sufficiently small $\|E\|$, (which is why they are called asymptotic):

Simple eigenvalue	$\lvert\hat{\lambda}_i - \lambda_i\rvert \lesssim \|E\|_2/s_i$
Eigenvalue cluster	$\lvert\hat{\lambda}_{\boldsymbol{I}} - \lambda_{\boldsymbol{I}}\rvert \lesssim \|E\|_2/s_{\boldsymbol{I}}$
Eigenvector	$\theta(\hat{\vartheta}_i, \vartheta_i) \lesssim \|E\|_F/sep_i$
Invariant subspace	$\theta(\hat{\boldsymbol{S}}_{\boldsymbol{I}}, S_{\boldsymbol{I}}) \lesssim \|E\|_F/sep_{\boldsymbol{I}}$

Table 2
Asymptotic error bounds for the nonsymmetric eigenproblem

If the problem is ill-conditioned, the asymptotic bounds may only hold for extremely small $\|E\|$. The global error bounds of Table 3 are guaranteed to hold for all $\|E\|_F < s \times sep/4$:

Simple eigenvalue	$\lvert\hat{\lambda}_i - \lambda_i\rvert \leq n\|E\|_2/s_i$	Holds for all E
Eigenvalue cluster	$\lvert\hat{\lambda}_{\boldsymbol{I}} - \lambda_{\boldsymbol{I}}\rvert \leq 2\|E\|_2/s_{\boldsymbol{I}}$	Requires $\|E\|_F < s_{\boldsymbol{I}} \times sep_{\boldsymbol{I}}/4$
Eigenvector	$\theta(\hat{\vartheta}_i, \vartheta_i) \leq \arctan(2\|E\|_F/(sep_i - 4\|E\|_F/s_i))$	Requires $\|E\|_F < s_i \times sep_i/4$
Invariant subspace	$\theta(\hat{\boldsymbol{S}}_{\boldsymbol{I}}, S_{\boldsymbol{I}}) \leq \arctan(2\|E\|_F/(sep_{\boldsymbol{I}} - 4\|E\|_F/s_{\boldsymbol{I}}))$	Requires $\|E\|_F < s_{\boldsymbol{I}} \times sep_{\boldsymbol{I}}/4$

Table 3
Global error bounds for the nonsymmetric eigenproblem

2.12.6 Balancing and condition for the nonsymmetric eigenproblem

There are two preprocessing steps one may perform on a matrix A in order to make its eigenproblem easier. The first is *permutation*, or reordering the rows and columns to make A more nearly upper triangular (closer to Schur form): $A' = PAP^T$, where P is a permutation matrix. If A' is permutable to upper triangular form (or close to it), then no floating-point operations (or very few) are needed to reduce it to Schur form. The second is *scaling* by a diagonal matrix D to make the rows and columns of A' more nearly equal in norm: $A'' = DA'D^{-1}$. Scaling can make the matrix norm smaller with respect to the eigenvalues, and so possibly reduce the inaccuracy contributed by roundoff (see Chapter, II/11 of Wilkinson and Reinsch (1971)). We refer to these two operations as *balancing*.

Permuting has no effect on the condition numbers or their interpretation as described previously. Scaling, however, does change their interpretation and further details can be found in Anderson *et al.* (1999).

2.12.7 The generalized nonsymmetric eigenvalue problem

The algorithms for the generalized nonsymmetric eigenvalue problem are normwise backward stable: they compute the exact eigenvalues (as the pairs (α, β)), eigenvectors and deflating subspaces of slightly perturbed pairs $(A + E, B + F)$, where

$$||(E, F)||_F \leq p(n)\epsilon||(A, B)||_F.$$

Currently routines are not provided for computing bounds on the eigenvalues, eigenvectors and deflating subspaces; these will be provided at a future mark.

2.12.8 Balancing the generalized eigenvalue problem

As with the standard nonsymmetric eigenvalue problem, there are two preprocessing steps one may perform on a matrix pair (A, B) in order to make its eigenproblem easier; permutation and scaling, which together are referred to as balancing, as indicated in the following two steps.

1. The balancing routine first attempts to permute A and B to block upper triangular form by a similarity transformation:

$$PAP^T = F = \begin{pmatrix} F_{11} & F_{12} & F_{13} \\ & F_{22} & F_{23} \\ & & F_{33} \end{pmatrix},$$

$$PBP^T = G = \begin{pmatrix} G_{11} & G_{12} & G_{13} \\ & G_{22} & G_{23} \\ & & G_{33} \end{pmatrix},$$

   where P is a permutation matrix, F_{11}, F_{33}, G_{11} and G_{33} are upper triangular. Then the diagonal elements of the matrix (F_{11}, G_{11}) and (G_{33}, H_{33}) are generalized eigenvalues of (A, B). The rest of the generalized eigenvalues are given by the matrix pair (F_{22}, G_{22}). Subsequent operations to compute the eigenvalues of (A, B) need only be applied to the matrix (F_{22}, G_{22}); this can save a significant amount of work if (F_{22}, G_{22}) is smaller than the original matrix pair (A, B). If no suitable permutation exists (as is often the case), then there is no gain in efficiency or accuracy.

2. The balancing routine applies a diagonal similary transformation to (F, G), to make the rows and columns of (F_{22}, G_{22}) as close as in norm as possible:

$$DFD^{-1} = \begin{pmatrix} I & & \\ & D_{22} & \\ & & I \end{pmatrix} \begin{pmatrix} F_{11} & F_{12} & F_{13} \\ & F_{22} & F_{23} \\ & & F_{33} \end{pmatrix} \begin{pmatrix} I & & \\ & D_{22}^{-1} & \\ & & I \end{pmatrix}$$

$$DGD^{-1} = \begin{pmatrix} I & & \\ & D_{22} & \\ & & I \end{pmatrix} \begin{pmatrix} G_{11} & G_{12} & G_{13} \\ & G_{22} & G_{23} \\ & & G_{33} \end{pmatrix} \begin{pmatrix} I & & \\ & D_{22}^{-1} & \\ & & I \end{pmatrix}.$$

   This transformation usually improves the accuracy of computed generalized eigenvalues and eigenvectors. However, there are exceptional occasions when this transformation increases the norm of the pencil; in this case accuracy could be lower with diagonal balancing.

   See Anderson *et al.* (1999) for further details.

2.13 Block Algorithms

A number of the routines in this chapter use what is termed a *block algorithm*. This means that at each major step of the algorithm a *block* of rows or columns is updated, and much of the computation is performed by matrix-matrix operations on these blocks. The matrix-matrix operations are performed by calls to the Level 3 BLAS (see Chapter F06), which are the key to achieving high performance on many modern computers. In the case of the QR algorithm for reducing an upper Hessenberg matrix to Schur

form, a multishift strategy is used in order to improve performance. See Golub and van Loan (1996) or Anderson *et al.* (1999) for more about block algorithms and the multishift strategy.

The performance of a block algorithm varies to some extent with the ***block size*** – that is, the number of rows or columns per block. This is a machine-dependent parameter, which is set to a suitable value when the library is implemented on each range of machines. Users of the library do not normally need to be aware of what value is being used. Different block sizes may be used for different routines. Values in the range 16 to 64 are typical.

On more conventional machines there is often no advantage from using a block algorithm, and then the routines use an *unblocked* algorithm (effectively a block size of 1), relying solely on calls to the Level 2 BLAS (see Chapter F06 again).

The only situation in which a user needs some awareness of the block size is when it affects the amount of workspace to be supplied to a particular routine. This is discussed in Section 3.4.3.

3 Recommendations on Choice and Use of Available Routines

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

3.1 Available Routines

The tables in the following subsections show the routines which are provided for performing different computations on different types of matrices. Each entry in the table gives the NAG routine name, the LAPACK single precision name, and the LAPACK double precision name (see Section 3.2).

For many computations it is necessary to call two or more routines in sequence some commonly required sequences of routines are indicated below; an asterisk ($*$) against a routine name means that the sequence of calls is illustrated in the example program for that routine. (But remember that Black Box routines for the same computations may be provided in Chapter F02 or Chapter F04.)

3.1.1 Orthogonal factorizations

Routines are provided for QR factorization (with and without column pivoting), and for LQ factorization (without pivoting only), of a general real or complex rectangular matrix.

The factorization routines do not form the matrix Q explicitly, but represent it as a product of elementary reflectors (see Section 3.3.6). Additional routines are provided to generate all or part of Q explicitly if it is required, or to apply Q in its factored form to another matrix (specifically to compute one of the matrix products QC, Q^TC, CQ or CQ^T with Q^T replaced by Q^H if C and Q are complex).

	Factorize without pivoting	**Factorize with pivoting**	**Generate Matrix *Q***	**Apply matrix *Q***
QR factorization, real matrices	F08AEF SGEQRF DGEQRF	F08BEF SGEQPF DGEQPF	F08AFF SORGQR DORGQR	F08AGF SORMQR DORMQR
LQ factorization, real matrices	F08AHF SGELQF DGELQF		F08AJF SORGLQ DORGLQ	F08AKF SORMLQ DORMLQ
QR factorization, complex matrices	F08ASF CGEQRF ZGEQRF	F08BSF CGEQPF ZGEQPF	F08ATF CUNGQR ZUNMQR	F08AUF CUNMQR ZUNGQR
LQ factorization, complex matrices	F08AVF CGELQF ZGELQF		F08AWF CUNGLQ ZUNGLQ	F08AXF CUNMQL ZUNMLQ

To solve linear least-squares problems, as described in Section 2.2.1 or Section 2.2.3, routines based on the QR factorization can be used:

real data, full-rank problem	F08AEF*, F08AGF, F06YJF
complex data, full-rank problem	F08ASF*, F08AUF, F06ZJF
real data, rank-deficient problem	F08BEF*, F08AGF, F06YJF
complex data, rank-deficient problem	F08BSF*, F08AUF, F06ZJF

To find the minimum norm solution of under-determined systems of linear equations, as described in Section 2.2.2, routines based on the LQ factorization can be used:

real data, full-rank problem	F08AHF*, F06YJF, F08AKF
complex data, full-rank problem	F08AVF*, F06ZJF, F08AXF

3.1.2 Singular value problems

Routines are provided to reduce a general real or complex rectangular matrix A to real bidiagonal form B by an orthogonal transformation $A = QBP^T$ (or by a unitary transformation $A = QBP^H$ if A is complex). Different routines allow a full matrix A to be stored conventionally (see Section 3.3.1), or a band matrix to use band storage (see Section 3.3.3).

The routines for reducing full matrices do not form the matrix Q or P explicitly; additional routines are provided to generate all or part of them, or to apply them to another matrix, as with the routines for orthogonal factorizations. Explicit generation of Q or P is required before using the bidiagonal QR algorithm to compute left or right singular vectors of A.

The routines for reducing band matrices have options to generate Q or P if required.

Further routines are provided to compute all or part of the singular value decomposition of a real bidiagonal matrix; the same routines can be used to compute the singular value decomposition of a real or complex matrix that has been reduced to bidiagonal form.

	Reduce to bidiagonal form	**Generate matrix *Q* or P^T**	**Apply matrix *Q* or *P***	**Reduce band matrix to bidiagonal form**	**SVD of bidiagonal form (*QR* algorithm)**
real matrices	F08KEF SGEBRD DGEBRD	F08KFF SORGBR DORGBR	F08KGF SORMBR DORMBR	F08LEF SGBBRD DGBBRD	F08MEF SBDSQR DBDSQR
complex matrices	F08KSF CGEBRD ZGEBRD	F08KTF CUNGBR ZUNGBR	F08KUF CUNMBR ZUNMBR	F08LSF CGBBRD ZGBBRD	F08MSF CBDSQR ZBDSQR

To compute the singular values and vectors of a rectangular matrix, as described in Section 2.3, use the following sequence of calls:

Rectangular matrix (standard storage)

real matrix, singular values and vectors	F08KEF, F08KFF*, F08MEF
complex matrix, singular values and vectors	F08KSF, F08KTF*, F08MSF

Rectangular matrix (banded)

real matrix, singular values and vectors	F08LEF, F08MEF
complex matrix, singular values and vectors	F08LSF, F08MSF

To use the singular value decomposition to solve a linear least-squares problem, as described in Section 2.4, the following routines are required:

real data:	F08KEF, F08KGF, F08KFF, F08MEF, F06YAF
complex data:	F08KSF, F08KUF, F08KTF, F08MSF, F06ZAF

3.1.3 Symmetric eigenvalue problems

Routines are provided to reduce a real symmetric or complex Hermitian matrix A to real tridiagonal form T by an orthogonal similarity transformation $A = QTQ^T$ (or by a unitary transformation $A = QTQ^H$ if A is complex). Different routines allow a full matrix A to be stored conventionally (see Section 3.3.1) or in packed storage (see Section 3.3.2); or a band matrix to use band storage (see Section 3.3.3).

The routines for reducing full matrices do not form the matrix Q explicitly; additional routines are provided to generate Q, or to apply it to another matrix, as with the routines for orthogonal factorizations. Explicit generation of Q is required before using the QR algorithm to find all the eigenvectors of A; application of Q to another matrix is required after eigenvectors of T have been found by inverse iteration, in order to transform them to eigenvectors of A.

The routines for reducing band matrices have an option to generate Q if required.

	Reduce to tridiagonal form	**Generate matrix *Q***	**Apply matrix *Q***
real symmetric matrices	F08FEF SSYTRD DSYTRD	F08FFF SORGTR DORGTR	F08FGF SORMTR DORMTR
real symmetric matrices (packed storage)	F08GEF SSPTRD DSPTRD	F08GFF SOPGTR DOPGTR	F08GGF SOPMTR DOPMTR
real symmetric band matrices	F08HEF SSBTRD DSBTRD		
complex Hermitian matrices	F08FSF CHETRD ZHETRD	F08FTF CUNGTR ZUNGTR	F08FUF CUNMTR ZUNMTR
complex Hermitian matrices (packed storage)	F08GSF CHPTRD ZHPTRD	F08GTF CUPGTR ZUPGTR	F08GUF CUPMTR ZUPMTR
complex Hermitian band matrices	F08HSF CHBTRD ZHBTRD		

A variety of routines are provided to compute eigenvalues and eigenvectors of the real symmetric tridiagonal matrix T, some computing all eigenvalues and eigenvectors, some computing selected eigenvalues and eigenvectors. The same routines can be used to compute eigenvalues and eigenvectors of a real symmetric or complex Hermitian matrix which has been reduced to tridiagonal form.

Eigenvalues and eigenvectors of real symmetric tridiagonal matrices:

The original (non-reduced) matrix is Real or Complex Hermitian

all eigenvalues (root-free QR algorithm)	F08JFF
all eigenvalues (root-free QR algorithm called by divide and conquer)	F08JCF
selected eigenvalues (bisection)	F08JJF

The original (non-reduced) matrix is Real

all eigenvalues and eigenvectors (QR algorithm)	F08JEF
all eigenvalues and eigenvectors (divide and conquer)	F08JCF
all eigenvalues and eigenvectors (positive-definite case)	F08JGF
selected eigenvectors (inverse iteration)	F08JKF

The original (non-reduced) matrix is Complex Hermitian

all eigenvalues and eigenvectors (QR algorithm)	F08JSF
all eigenvalues and eigenvectors (positive-definite case)	F08JUF
selected eigenvectors (inverse iteration)	F08JXF

The following sequences of calls may be used to compute various combinations of eigenvalues and eigenvectors, as described in Section 2.5.

Sequences for computing eigenvalues and eigenvectors

Real Symmetric matrix (standard storage)

all eigenvalues and eigenvectors (using divide and conquer)	F08FCF
all eigenvalues and eigenvectors (using QR algorithm)	F08FEF, F08FFF*, F08JEF

selected eigenvalues and eigenvectors (bisection and inverse iteration)	F08FEF, F08JJF, F08JKF, F08FGF*

Real Symmetric matrix (packed storage)

all eigenvalues and eigenvectors (using divide and conquer)	F08GCF
all eigenvalues and eigenvectors (using QR algorithm)	F08GEF, F08GFF*, F08JEF
selected eigenvalues and eigenvectors (bisection and inverse iteration)	F08GEF, F08JJF, F08JKF, F08GGF*

Real Symmetric banded matrix

all eigenvalues and eigenvectors (using divide and conquer)	F08HCF
all eigenvalues and eigenvectors (using QR algorithm)	F08HEF*, F08JEF

Complex Hermitian matrix (standard storage)

all eigenvalues and eigenvectors (using divide and conquer)	F08FQF
all eigenvalues and eigenvectors (using QR algorithm)	F08FSF, F08FTF*, F08JSF
selected eigenvalues and eigenvectors (bisection and inverse iteration)	F08FSF, F08JJF, F08JXF, F08FUF*

Complex Hermitian matrix (packed storage)

all eigenvalues and eigenvectors (using divide and conquer)	F08GQF
all eigenvalues and eigenvectors (using QR algorithm)	F08GSF, F08GTF*, F08JSF
selected eigenvalues and eigenvectors (bisection and inverse iteration)	F08GSF, F08JJF, F08JXF, F08GUF*

Complex Hermitian banded matrix

all eigenvalues and eigenvectors (using divide and conquer)	F08HQF
all eigenvalues and eigenvectors (using QR algorithm)	F08HSF*, F08JSF

3.1.4 Generalized symmetric-definite eigenvalue problems

Routines are provided for reducing each of the problems $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$ to an equivalent standard eigenvalue problem $Cy = \lambda y$. Different routines allow the matrices to be stored either conventionally or in packed storage. The positive-definite matrix B must first be factorized using a routine from Chapter F07. There is also a routine which reduces the problem $Ax = \lambda Bx$ where A and B are banded, to an equivalent banded standard eigenvalue problem; this uses a split Cholesky factorization for which a routine in Chapter F08 is provided.

	Reduce to standard problem	**Reduce to standard problem (packed storage)**	**Reduce to standard problem (band matrices)**
real symmetric matrices	F08SEF SSYGST DSYGST	F08TEF SSPGST DSPGST	F08UEF SSBGST DSBGST
complex Hermitian matrices	F08SSF CHEGST ZHEGST	F08TSF CHPGST ZHPGST	F08USF CHBGST ZHBGST

The equivalent standard problem can then be solved using the routines discussed in Section 3.1.3. For example, to compute all the eigenvalues, the following routines must be called:

real symmetric-definite problem	F07FDF, F08SEF*, F08FEF, F08JFF
real symmetric-definite problem, packed storage	F07GDF, F08TEF*, F08GEF, F08JFF
real symmetric-definite banded problem	F08UFF*, F08UEF*, F08HEF, F08JFF
complex Hermitian-definite problem	F07FRF, F08SSF*, F08FSF, F08JFF
complex Hermitian-definite problem, packed storage	F07GRF, F08TSF*, F08GSF, F08JFF
complex Hermitian-definite banded problem	F08UTF*, F08USF*, F08HSF, F08JFF

If eigenvectors are computed, the eigenvectors of the equivalent standard problem must be transformed back to those of the original generalized problem, as indicated in Section 2.6; routines from Chapter F06 may be used for this.

3.1.5 Nonsymmetric eigenvalue problems

Routines are provided to reduce a general real or complex matrix A to upper Hessenberg form H by an orthogonal similarity transformation $A = QHQ^T$ (or by a unitary transformation $A = QHQ^H$ if A is complex).

These routines do not form the matrix Q explicitly; additional routines are provided to generate Q, or to apply it to another matrix, as with the routines for orthogonal factorizations. Explicit generation of Q is required before using the QR algorithm on H to compute the Schur vectors; application of Q to another matrix is needed after eigenvectors of H have been computed by inverse iteration, in order to transform them to eigenvectors of A.

Routines are also provided to balance the matrix before reducing it to Hessenberg form, as described in Section 2.12.6. Companion routines are required to transform Schur vectors or eigenvectors of the balanced matrix to those of the original matrix.

	Reduce to Hessenberg form	**Generate matrix *Q***	**Apply matrix *Q***	**Balance**	**Backtransform vectors after balancing**
real matrices	F08NEF SGEHRD DGEHRD	F08NFF SORGHR DORGHR	F08NGF SORMHR DORMHR	F08NHF SGEBAL DGEBAL	F08NJF SGEBAK DGEBAK
complex matrices	F08NSF CGEHRD ZGEHRD	F08NTF CUNGHR ZUNGHR	F08NUF CUNMHR ZUNMHR	F08NVF CGEBAL ZGEBAL	F08NWF CGEBAK ZGEBAK

Routines are provided to compute the eigenvalues and all or part of the Schur factorization of an upper Hessenberg matrix. Eigenvectors may be computed either from the upper Hessenberg form by inverse iteration, or from the Schur form by back-substitution; these approaches are equally satisfactory for computing individual eigenvectors, but the latter may provide a more accurate basis for a subspace spanned by several eigenvectors.

Additional routines estimate the sensitivities of computed eigenvalues and eigenvectors, as discussed in Section 2.12.5.

	Eigenvalues and Schur factorization (*QR* algorithm)	**Eigenvectors from Hessenberg form (inverse iteration)**	**Eigenvectors from Schur factorization**	**Sensitivities of eigenvalues and eigenvectors**
real matrices	F08PEF SHSEQR DHSEQR	F08PKF SHSEIN DHSEIN	F08QKF STREVC DTREVC	F08QLF STRSNA DTRSNA
complex matrices	F08PSF CHSEQR ZHSEQR	F08PXF CHSEIN ZHSEIN	F08QXF CTREVC ZTREVC	F08QYF CTRSNA ZTRSNA

Finally routines are provided for re-ordering the Schur factorization, so that eigenvalues appear in any desired order on the diagonal of the Schur form. The routines F08QFF and F08QTF simply swap two diagonal elements or blocks, and may need to be called repeatedly to achieve a desired order. The routines F08QGF and F08QUF perform the whole re-ordering process for the important special case where a specified cluster of eigenvalues is to appear at the top of the Schur form; if the Schur vectors are re-

ordered at the same time, they yield an orthonormal basis of the invariant subspace corresponding to the specified cluster of eigenvalues. These routines can also compute the sensitivities of the cluster of eigenvalues and the invariant subspace.

	Reorder Schur factorization	**Reorder Schur factorization, find basis of invariant subspace and estimate sensitivities**
real matrices	F08QFF STREXC DTREXC	F08QGF STRSEN DTRSEN
complex matrices	F08QTF CTREXC ZTREXC	F08QUF CTRSEN ZTRSEN

The following sequences of calls may be used to compute various combinations of eigenvalues, Schur vectors and eigenvectors, as described in Section 2.9:

real matrix, all eigenvalues and Schur factorization	F08NEF, F08NFF*, F08PEF
real matrix, all eigenvalues and selected eigenvectors	F08NEF, F08PEF, F08PKF, F08NGF*
real matrix, all eigenvalues and eigenvectors (with balancing)	F08NHF*, F08NEF, F08NFF, F08PEF, F08PKF, F08NJF
complex matrix, all eigenvalues and Schur factorization	F08NSF, F08NTF*, F08PSF
complex matrix, all eigenvalues and selected eigenvectors	F08NSF, F08PSF, F08PXF, F08NUF*
complex matrix, all eigenvalues and eigenvectors (with balancing)	F08NVF*, F08NSF, F08NTF, F08PSF, F08PXF, F08NWF

3.1.6 Generalized nonsymmetric eigenvalue problems

Routines are provided to reduce a real or complex matrix pair (A_1, R_1), where A_1 is general and R_1 is upper triangular, to generalized upper Hessenberg form by orthogonal transformations $A_1 = Q_1 H Z_1^T$, $R_1 = Q_1 R Z_1^T$, (or by unitary transformations $A_1 = Q_1 H Z_1^H$, $R = Q_1 R_1 Z_1^H$, in the complex case). These routines can optionally return Q_1 and/or Z_1. Note that to transform a general matrix pair (A, B) to the form (A_1, R_1) a QR factorization of B ($B = \tilde{Q} R_1$) should first be performed and the matrix A_1 obtained as $A_1 = \tilde{Q}^T A$ (see Section 3.1.1 above).

Routines are also provided to balance a general matrix pair before reducing it to generalized Hessenberg form, as described in Section 2.12.8. Companion routines are provided to transform vectors of the balanced pair to those of the original matrix pair.

	Reduce to generalized Hessenberg form	**Balance**	**Backtransform vectors after balancing**
real matrices	F08WEF SGGHRD DGGHRD	F08WHF SGGBAL DGGBAL	F08WJF SGGBAK DGGBAK
complex matrices	F08WSF CGGHRD GZZHRD	F08WVF CGGBAL ZGGBAL	F08WWF CGGBAK ZGGBAK

Routines are provided to compute the eigenvalues (as the pairs (α, β)) and all or part of the generalized Schur factorization of a generalized upper Hessenberg matrix pair. Eigenvectors may be computed from the generalized Schur form by back-substitution.

	Eigenvalues and generalized Schur factorization (*QZ* algorithm)	**Eigenvectors from generalized Schur factorization**
real matrices	F08XEF SHGEQZ DHGEQZ	F08YKF STGEVC DTGEVC
complex matrices	F08XSF (CHGEQZ/ZHGEQZ) CHGEQZ ZHGEQZ	F08YXF CTGEVC ZTGEVC

The following sequences of calls may be used to compute various combinations of eigenvalues, generalized Schur vectors and eigenvectors

real matrix pair, all eigenvalues (with balancing)	F08WHF, F08AEF, F08AGF, F08WEF, F08XEF*
real matrix pair, all eigenvalues and generalized Schur factorization	F08AEF, F08AGF, F08AFF, F08WEF, F08XEF
real matrix pair, all eigenvalues and eigenvectors (with balancing)	F08WHF, F08AEF, F08AGF, F06QHF, F06QFF, F08AFF, F08WEF, F08XEF, F08YKF*, F08WJF
complex matrix pair, all eigenvalues (with balancing)	F08WVF, F08ASF, F08AUF, F08WSF, F08XSF*
complex matrix pair, all eigenvalues and generalized Schur factorization	F08ASF, F08AUF, F08ATF, F08WSF, F08XSF
complex matrix pair, all eigenvalues and eigenvectors (with balancing)	F08WVF, F08ASF, F08AUF, F06THF, F06TFF, F08ATF, F08WSF, F08XSF, F08YXF*, F08WWF

3.1.7 Sylvester's equation

Routines are provided to solve the real or complex Sylvester equation $AX \pm XB = C$, where A and B are upper quasi-triangular if real, or upper triangular if complex. To solve the general form of Sylvester's equation in which A and B are general square matrices, A and B must be reduced to upper (quasi-) triangular form by the Schur factorization, using routines described in Section 3.1.5. For more details, see the documents for the routines listed below.

	solve Sylvester's equation
real matrices	F08QHF STRSYL DTRSYL
complex matrices	F08QVF CTRSYL ZTRSYL

3.2 NAG Names and LAPACK Names

As well as the NAG routine name (beginning F08-), the tables in Section 3.1 show the LAPACK routine names in both single and double precision.

The routines may be called either by their NAG names or by their LAPACK names. When using a single precision implementation of the NAG Library, the single precision form of the LAPACK name must be

used (beginning with S- or C-); when using a double precision implementation of the NAG Library, the double precision form of the LAPACK name must be used (beginning with D- or Z-).

References to F08 routines in the manual normally include the LAPACK single and double precision names, in that order – for example F08AEF (SGEQRF/DGEQRF) (SGEQRF/DGEQRF). The LAPACK routine names follow a simple scheme (which is similar to that used for the BLAS in Chapter F06). Each name has the structure **XYYZZZ**, where the components have the following meanings:

– the initial letter **X** indicates the data type (real or complex) and precision:

S – real, single precision (in Fortran 77, `REAL`)
D – real, double precision (in Fortran 77, `DOUBLE PRECISION`)
C – complex, single precision (in Fortran 77, `COMPLEX`)
Z – complex, double precision (in Fortran 77, `COMPLEX*16 or DOUBLE COMPLEX`)

– the 2nd and 3rd letters **YY** indicate the type of the matrix A or matrix pair (A, B) (and in some cases the storage scheme):

BD – bidiagonal
GB – general band
GE – general
GG – general pair (B may be triangular)
HG – generalized upper Hessenberg
HS – upper Hessenberg
OP – (real) orthogonal (packed storage)
UP – (complex) unitary (packed storage)
OR – (real) orthogonal
UN – (complex) unitary
PT – symmetric or Hermitian positive-definite tridiagonal
SB – (real) symmetric band
HB – (complex) Hermitian band
SP – symmetric (packed storage)
HP – Hermitian (packed storage)
ST – (real) symmetric tridiagonal
SY – symmetric
HE – Hermitian
TG – triangular pair (one may be quasi-triangular)
TR – triangular (or quasi-triangular)

– the last 3 letters **ZZZ** indicate the computation performed. For example, QRF is a QR factorization.

Thus the routine SGEQRF performs a QR factorization of a real general matrix in a single precision implementation of the Library; the corresponding routine in a double precision implementation is DGEQRF.

Some sections of the routine documents – Section 2 (Specification) and Section 9.1 (Example program) – print the LAPACK name in ***bold italics***, according to the NAG convention of using bold italics for precision-dependent terms – for example, ***sgeqrf***, which should be interpreted as either SGEQRF (in single precision) or DGEQRF (in double precision).

3.3 Matrix Storage Schemes

In this chapter the following storage schemes are used for matrices:

– conventional storage in a two-dimensional array;

– packed storage for symmetric or Hermitian matrices;

– packed storage for orthogonal or unitary matrices;

– band storage for general, symmetric or Hermitian band matrices;

– storage of bidiagonal, symmetric or Hermitian tridiagonal matrices in two one-dimensional arrays.

These storage schemes are compatible with those used in Chapter F06 and Chapter F07, but different schemes for packed, band and tridiagonal storage are used in a few older routines in Chapter F01, Chapter F02, Chapter F03 and Chapter F04.

In the examples below, $*$ indicates an array element which need not be set and is not referenced by the routines. The examples illustrate only the relevant leading rows and columns of the arrays; array arguments may of course have additional rows or columns, according to the usual rules for passing array arguments in Fortran 77.

3.3.1 Conventional storage

The default scheme for storing matrices is the obvious one: a matrix A is stored in a two-dimensional array A, with matrix element a_{ij} stored in array element $\mathrm{A}(i,j)$.

If a matrix is *triangular* (upper or lower, as specified by the argument UPLO when present), only the elements of the relevant triangle are stored; the remaining elements of the array need not be set. Such elements are indicated by $*$ in the examples below. For example, when $n = 4$:

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

Similarly, if the matrix is upper Hessenberg, or if the matrix is quasi-upper triangular, elements below the first subdiagonal need not be set.

Routines that handle *symmetric* or *Hermitian* matrices allow for either the upper or lower triangle of the matrix (as specified by UPLO) to be stored in the corresponding elements of the array; the remaining elements of the array need not be set. For example, when $n = 4$:

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

3.3.2 Packed storage

Symmetric and Hermitian matrices may be stored more compactly, if the relevant triangle (again as specified by UPLO) is packed *by columns* in a one-dimensional array. In Chapter F07 and Chapter F08, arrays that hold matrices in packed storage, have argument names ending in 'P'. So:

if UPLO = 'U', a_{ij} is stored in $\mathrm{AP}(i + j(j-1)/2)$ for $i \le j$;

if UPLO = 'L', a_{ij} is stored in $\mathrm{AP}(i + (2n-j)(j-1)/2)$ for $j \le i$.

For example:

UPLO	Triangle of matrix A	Packed storage in array AP
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$a_{11}\ \underbrace{a_{12}a_{22}}\ \underbrace{a_{13}a_{23}a_{33}}\ \underbrace{a_{14}a_{24}a_{34}a_{44}}$
'L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}a_{21}a_{31}a_{41}}\ \underbrace{a_{22}a_{32}a_{42}}\ \underbrace{a_{33}a_{43}}\ a_{44}$

Note that for symmetric matrices, packing the upper triangle by columns is equivalent to packing the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the upper triangle by rows. For Hermitian matrices, packing the upper triangle by columns is equivalent to packing the conjugate of the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the conjugate of the upper triangle by rows.

3.3.3 Band storage

A general m by n band matrix with k_l subdiagonals and k_u superdiagonals may be stored compactly in a two-dimensional array with $k_l + k_u + 1$ rows and n columns. Columns of the matrix are stored in corresponding columns of the array, and diagonals of the matrix are stored in rows of the array. This storage scheme should be used in practice only if $k_l, k_u \ll n$, although routines in Chapter F07 and Chapter F08 work correctly for all values of k_l and k_u. In Chapter F07 and Chapter F08, arrays that hold matrices in band storage have argument names ending in 'B'. So:

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

For example, when $m = 6$, $n = 5$, $k_l = 2$ and $k_u = 1$:

general band matrix A	Band storage in array AB
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \\ & & & a_{64} & a_{65} \end{pmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} \\ a_{31} & a_{42} & a_{53} & a_{64} & * \end{matrix}$

A symmetric or Hermitian band matrix with k subdiagonals and superdiagonals may be stored more compactly in a two-dimensional array with $k + 1$ rows and n columns. Only the upper or lower triangle (as specified by UPLO) need to be stored. So:

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

if UPLO = 'L', a_{ij} is stored in AB$(1 + i - j, j)$ for $j \le i \le \min(n, j + k)$.

For example, when $n = 5$ and $k = 2$:

UPLO	Hermitian band matrix A	Band storage in array AB
'U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} & \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} & a_{35} \\ & \bar{a}_{24} & \bar{a}_{34} & a_{44} & a_{45} \\ & & \bar{a}_{35} & \bar{a}_{45} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{matrix}$
'L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & & \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} & \bar{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \bar{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

3.3.4 Tridiagonal and bidiagonal matrices

A symmetric tridiagonal or bidiagonal matrix is stored in two one-dimensional arrays, one of length n containing the diagonal elements, and one of length $n-1$ containing the off-diagonal elements. (Older routines in Chapter F02 store the off-diagonal elements in elements $2:n$ of a vector of length n.)

3.3.5 Real diagonal elements of complex matrices

Complex Hermitian matrices have diagonal matrices that are by definition purely real. In addition, some complex triangular matrices computed by F08 routines are defined by the algorithm to have real diagonal elements – in QR factorization, for example.

If such matrices are supplied as input to F08 routines, the imaginary parts of the diagonal elements are not referenced, but are assumed to be zero. If such matrices are returned as output by F08 routines, the computed imaginary parts are explicitly set to zero.

3.3.6 Representation of orthogonal or unitary matrices

A real orthogonal or complex unitary matrix (usually denoted Q) is often represented in the NAG Library as a product of *elementary reflectors* – also referred to as *elementary Householder matrices* (usually denoted H_i). For example,

$$Q = H_1 H_2 \cdots H_k.$$

Most users need not be aware of the details, because routines are provided to work with this representation, either to generate all or part of Q explicitly, or to multiply a given matrix by Q or Q^T (Q^H in the complex case) without forming Q explicitly.

Nevertheless, the following further details may occasionally be useful.

An elementary reflector (or elementary Householder matrix) H of order n is a unitary matrix of the form

$$H = I - \tau v v^H \tag{4}$$

where τ is a scalar, and v is an n element vector, with $|\tau|^2 \|v\|_2^2 = 2 \times \mathrm{Re}(\tau)$; v is often referred to as the *Householder vector*. Often v has several leading or trailing zero elements, but for the purpose of this discussion assume that H has no such special structure.

There is some redundancy in the representation (4), which can be removed in various ways. The representation used in Chapter F08 and in LAPACK (which differs from those used in some of the routines in Chapter F01, Chapter F02, Chapter F04 and Chapter F06) sets $v_1 = 1$; hence v_1 need not be stored. In real arithmetic, $1 \le \tau \le 2$, except that $\tau = 0$ implies $H = I$.

In complex arithmetic, τ may be complex, and satisfies $1 \le \mathrm{Re}(\tau) \le 2$ and $|\tau - 1| \le 1$. Thus a complex H is not Hermitian (as it is in other representations), but it is unitary, which is the important property. The advantage of allowing τ to be complex is that, given an arbitrary complex vector x, H can be computed so that

$$H^H x = \beta(1, 0, \ldots, 0)^T$$

with *real* β. This is useful, for example, when reducing a complex Hermitian matrix to real symmetric tridiagonal form, or a complex rectangular matrix to real bidiagonal form.

3.4 Parameter Conventions

3.4.1 Option parameters

Most routines in this chapter have one or more option parameters, of type CHARACTER. The descriptions in Section 5 of the routine documents refer only to upper case values (for example 'U' or 'L'); however in every case, the corresponding lower case characters may be supplied (with the same meaning). Any other value is illegal.

A longer character string can be passed as the actual parameter, making the calling program more readable, but only the first character is significant. (This is a feature of Fortran 77.) For example:

```
CALL SSYTRD ('Upper',...)
```

3.4.2 Problem dimensions

It is permissible for the problem dimensions (for example, M or N) to be passed as zero, in which case the computation (or part of it) is skipped. Negative dimensions are regarded as an error.

3.4.3 Length of work arrays

A number of routines implementing block algorithms require workspace sufficient to hold one block of rows or columns of the matrix if they are to achieve optimum levels of performance – for example, workspace of size $n \times nb$, where nb is the optimum block size. In such cases, the actual declared length of the work array must be passed as a separate argument LWORK, which immediately follows WORK in the argument-list.

The routine will still perform correctly when less workspace is provided: it simply uses the largest block size allowed by the amount of workspace supplied, as long as this is likely to give better performance than the unblocked algorithm. On exit, WORK(1) contains the minimum value of LWORK which would allow the routine to use the optimum block size; this value of LWORK can be used for subsequent runs.

If LWORK indicates that there is insufficient workspace to perform the unblocked algorithm, this is regarded as an illegal value of LWORK, and is treated like any other illegal parameter value (see Section 3.4.4).

If you are in doubt how much workspace to supply and are concerned to achieve optimum performance, supply a generous amount (assume a block size of 64, say), and then examine the value of WORK(1) on exit.

3.4.4 Error-handling and the diagnostic parameter INFO

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

Whereas IFAIL is an *Input/Output* parameter and must be set before calling a routine, INFO is purely an *Output* parameter and need not be set before entry.

INFO indicates the success or failure of the computation, as follows:

INFO $= 0$: successful termination

INFO < 0: failure in the course of computation, control returned to the calling program

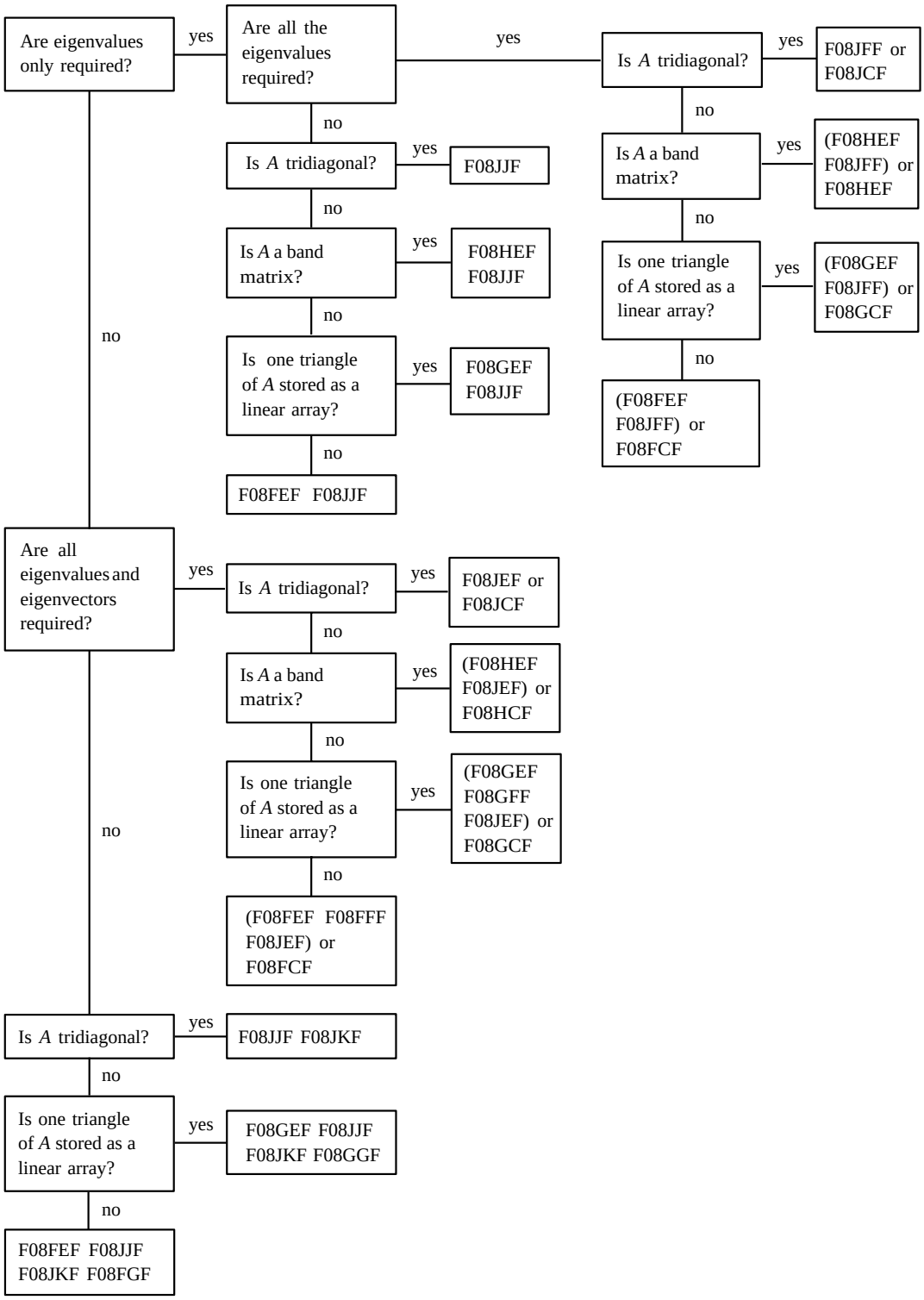
If the routine document specifies that the routine may terminate with INFO < 0, then it is **essential to test INFO on exit** from the routine. (This corresponds to a *soft failure* in terms of the usual NAG error-handling terminology.) No error message is output.

All routines check that input parameters such as N or LDA or option parameters of type CHARACTER have permitted values. If an illegal value of the ith parameter is detected, INFO is set to $-i$, a message is output, and execution of the program is terminated. (This corresponds to a *hard failure* in the usual NAG terminology.)

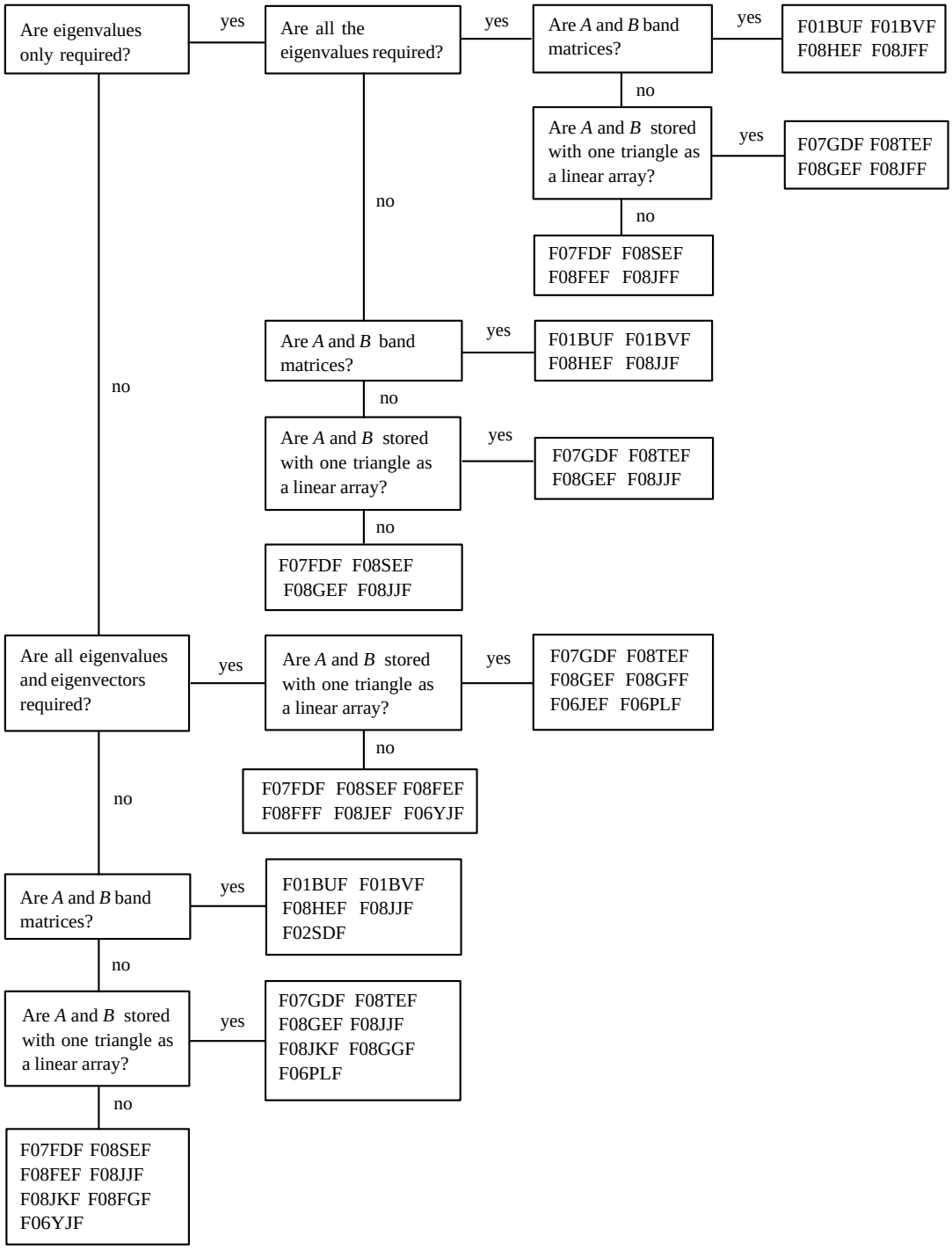
4 Decision Trees

4.1 General purpose routines (eigenvalues and eigenvectors)

Tree 1: Real Symmetric Matrices

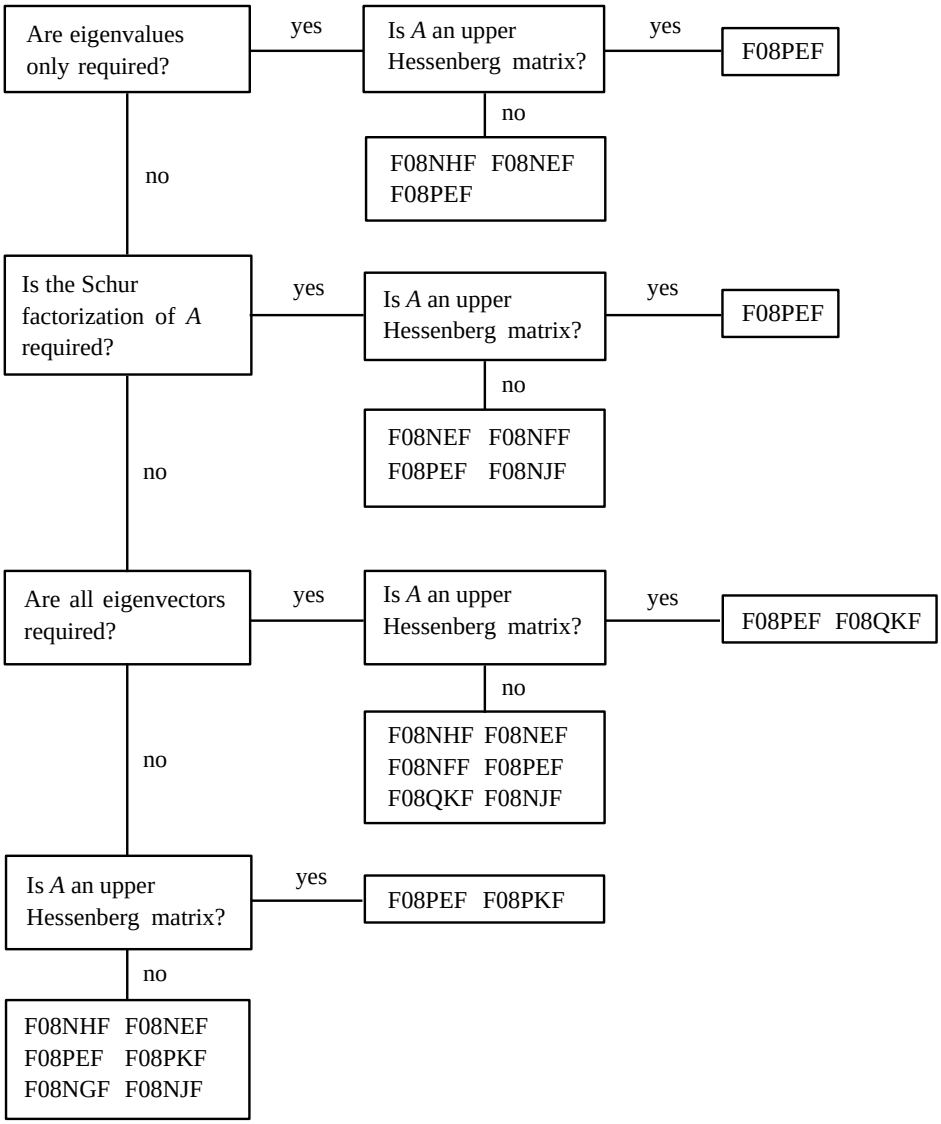


Tree 2: Real Generalized Symmetric-definite Eigenvalue Problems

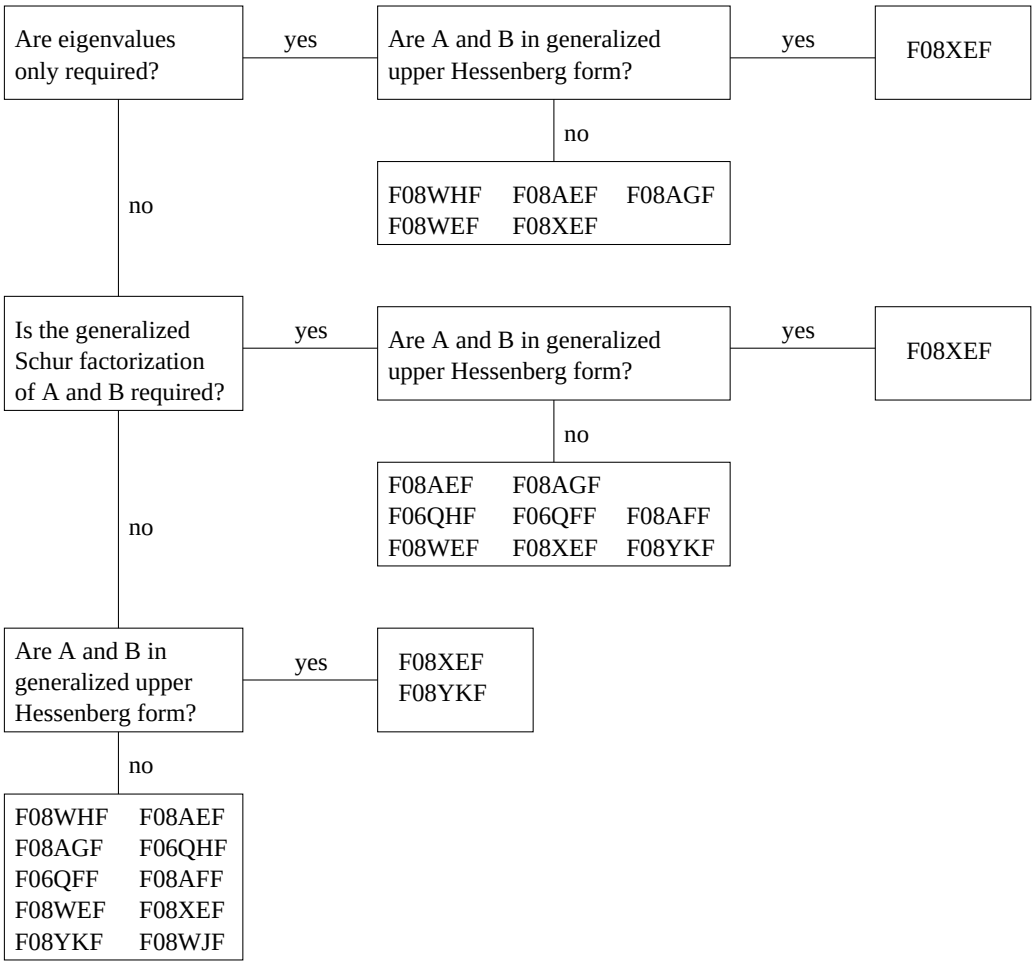


Note: the routines for band matrices only handle the problem $Ax = \lambda Bx$; the other routines handle all three types of problems ($Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$) except that, if the problem is $BAx = \lambda x$ and eigenvectors are required, F06PHF must be used instead of F06PLF, and F06YFF instead of F06YJF.

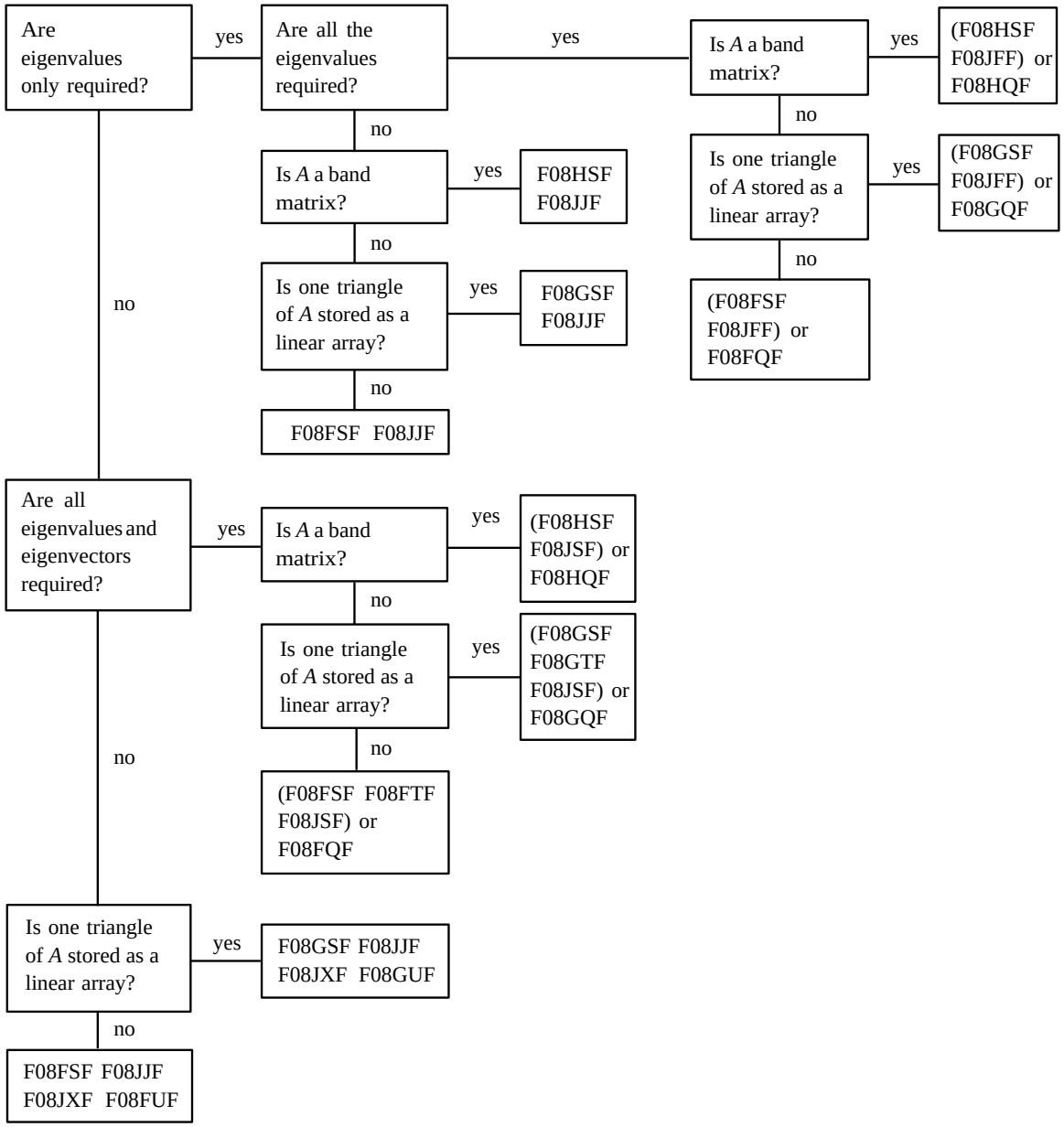
Tree 3: Real Nonsymmetric Matrices



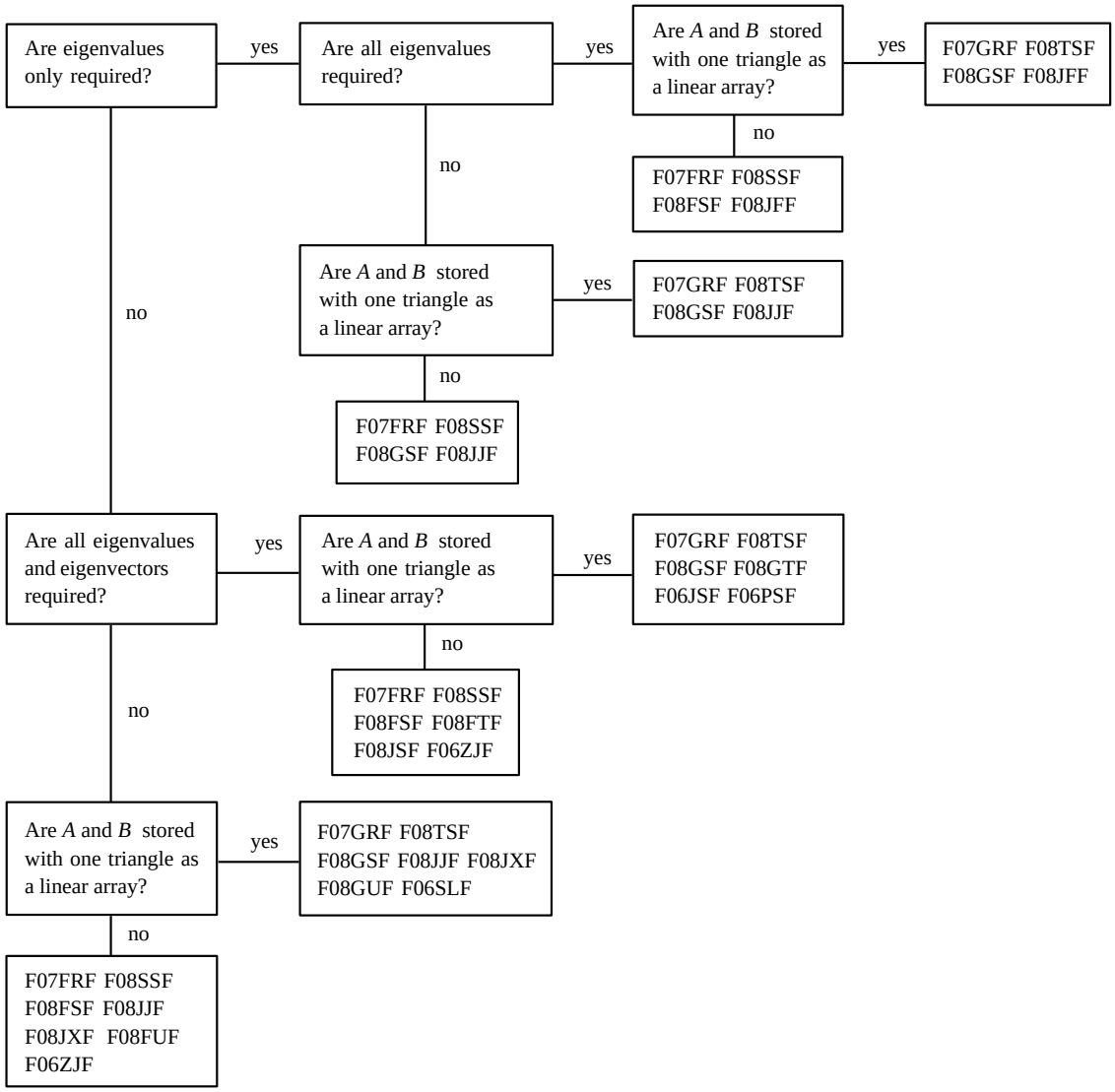
Tree 4: Real Generalized Nonsymmetric Eigenvalue Problems



Tree 5: Complex Hermitian Matrices

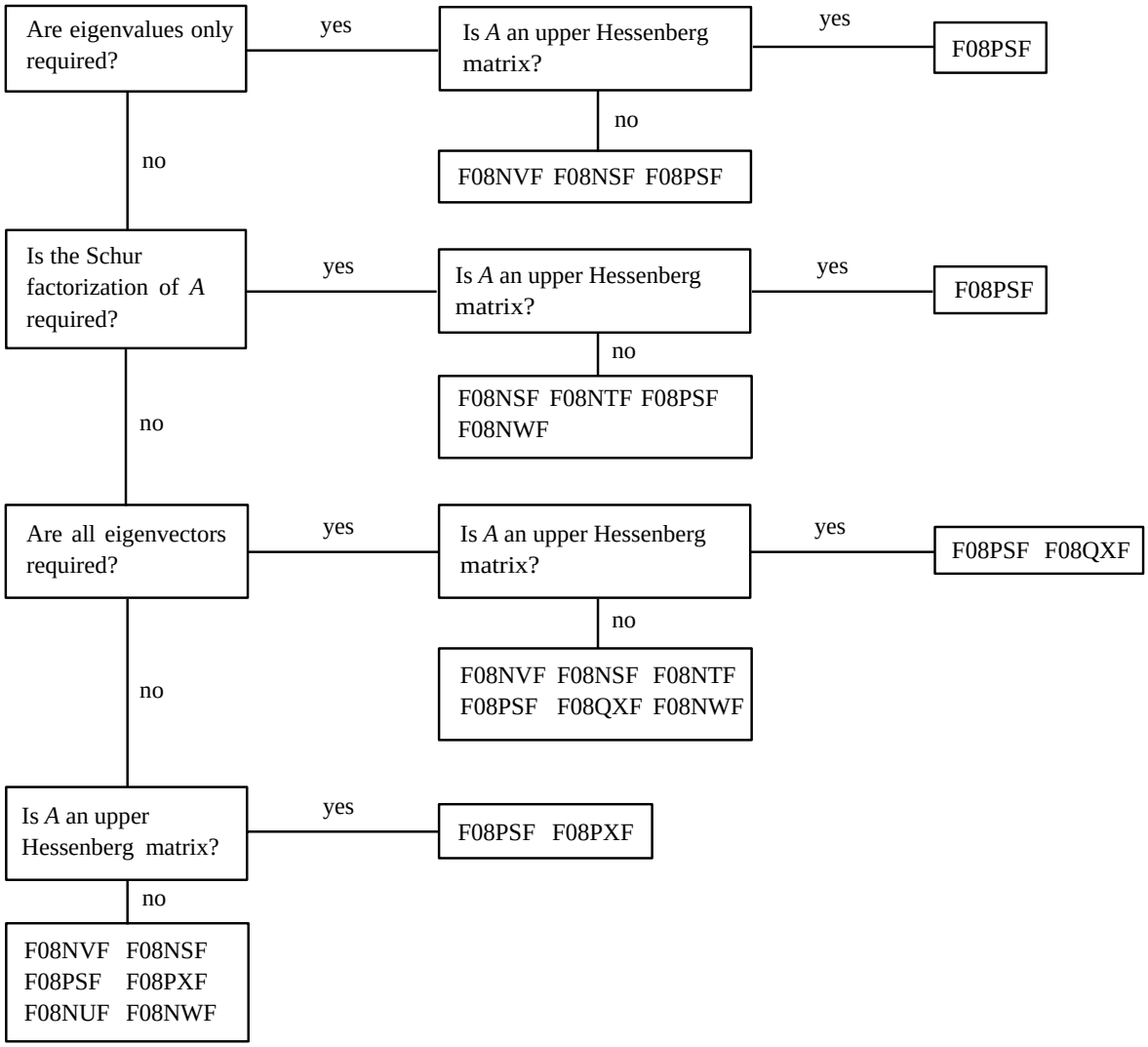


Tree 6: Complex Generalized Hermitian-definite Eigenvalue Problems

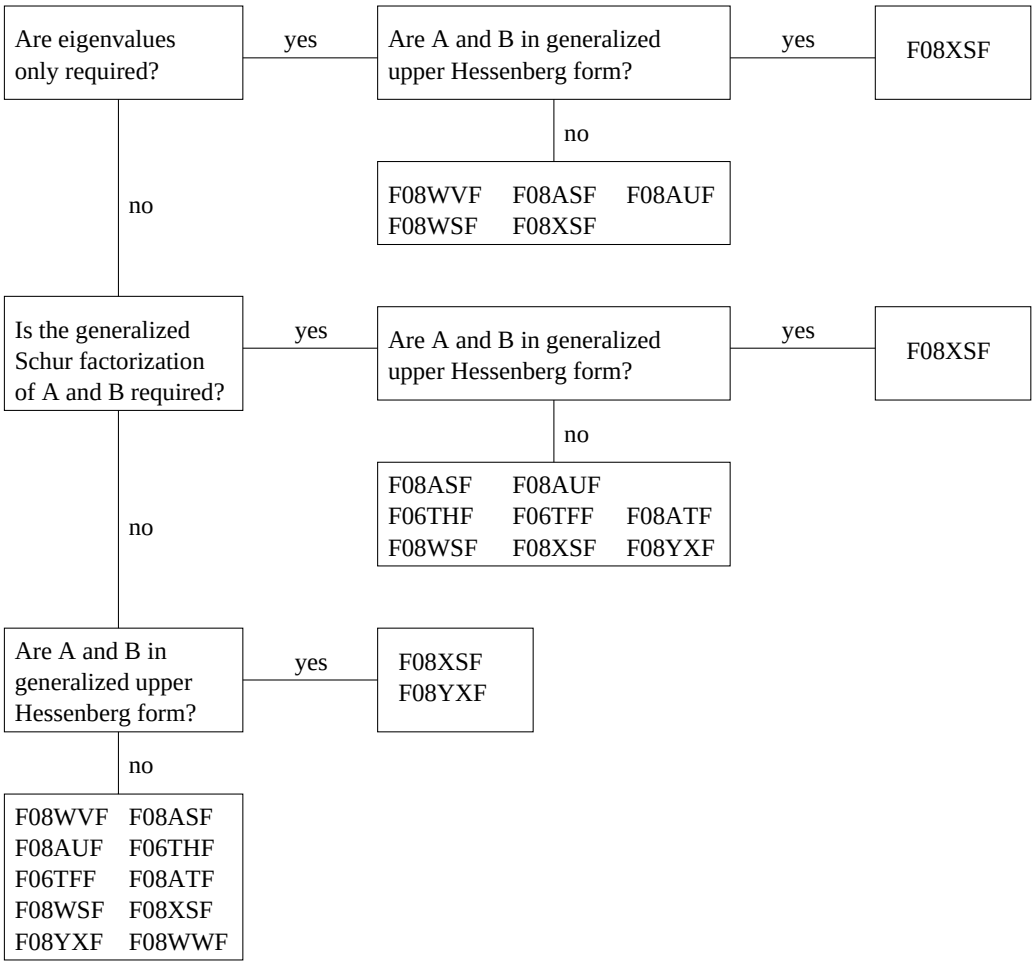


Note: the same routines are required for all three types of problem ($Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$) except that, if the problem is $BAx = \lambda x$ and eigenvectors are required, F06SHF must be used instead of F06SLF, and F06ZFF instead of F06ZJF.

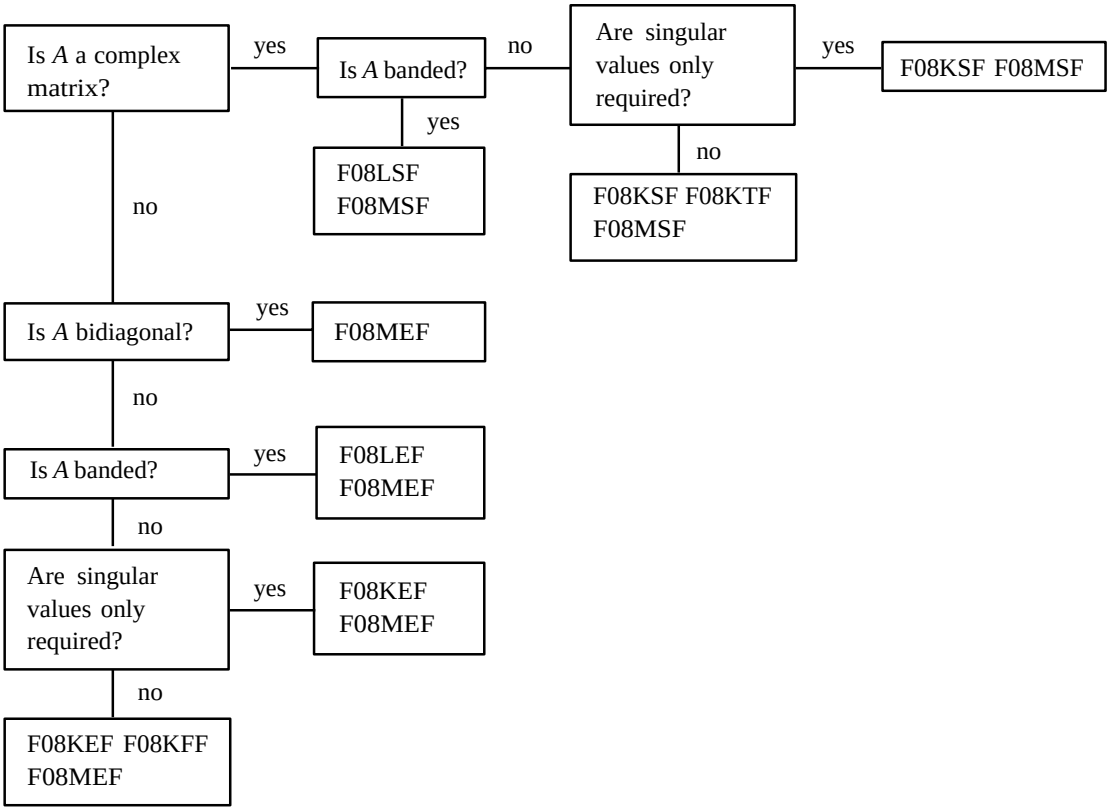
Tree 7: Complex Nonhermitian Matrices



Tree 8: Complex Generalized Nonhermitian Eigenvalue Problems



4.2 General purpose routines (singular value decomposition)



5 Indexes of LAPACK routines

Real Matrices			Complex Matrices		
LAPACK single precision	**LAPACK double precision**	**NAG**	**LAPACK single precision**	**LAPACK double precision**	**NAG**
SBDSQR	DBDSQR	F08MEF	CBDSQR	ZBDSQR	F08MSF
SGBBRD	DGBBRD	F08LEF	CGBBRD	ZGBBRD	F08LSF
SGEBAK	DGEBAK	F08NJF	CGEBAK	ZGEBAK	F08NWF
SGEBAL	DGEBAL	F08NHF	CGEBAL	ZGEBAL	F08NVF
SGEBRD	DGEBRD	F08KEF	CGEBRD	ZGEBRD	F08KSF
SGEHRD	DGEHRD	F08NEF	CGEHRD	ZGEHRD	F08NSF
SGELQF	DGELQF	F08AHF	CGELQF	ZGELQF	F08AVF
SGEQPF	DGEQPF	F08BEF	CGEQPF	ZGEQPF	F08BSF
SGEQRF	DGEQRF	F08AEF	CGEQRF	ZGEQRF	F08ASF
SGGBAK	DGGBAK	F08WJF	CGGBAK	ZGGBAK	F08WWF
SGGBAL	DGGBAL	F08WHF	CGGBAL	ZGGBAL	F08WVF
SGGHRD	DGGHRD	F08WEF	CGGHRD	ZGGHRD	F08WSF
SHGEQZ	DHGEQZ	F08XEF	CHBEVD	ZHBEVD	F08HQF
SHSEIN	DHSEIN	F08PKF	CHBGST	ZHBGST	F08USF
SHSEQR	DHSEQR	F08PEF	CHBTRD	ZHBTRD	F08HSF
SOPGTR	DOPGTR	F08GFF	CHEEVD	ZHEEVD	F08FQF
SOPMTR	DOPMTR	F08GGF	CHEGST	ZHEGST	F08SSF
SORGBR	DORGBR	F08KFF	CHETRD	ZHETRD	F08FSF
SORGHR	DORGHR	F08NFF	CHGEQZ	ZHGEQZ	F08XSF
SORGLQ	DORGLQ	F08AJF	CHPEVD	ZHPEVD	F08GQF
SORGQR	DORGQR	F08AFF	CHPGST	ZHPGST	F08TSF
SORGTR	DORGTR	F08FFF	CHPTRD	ZHPTRD	F08GSF
SORMBR	DORMBR	F08KGF	CHSEIN	ZHSEIN	F08PXF
SORMHR	DORMHR	F08NGF	CHSEQR	ZHSEQR	F08PSF
SORMLQ	DORMLQ	F08AKF	CPBSTF	ZPBSTF	F08UTF
SORMQR	DORMQR	F08AGF	CPTEQR	ZPTEQR	F08JUF
SORMTR	DORMTR	F08FGF	CSTEIN	ZSTEIN	F08JXF
SPBSTF	DPBSTF	F08UFF	CSTEQR	ZSTEQR	F08JSF
SPTEQR	DPTEQR	F08JGF	CTGEVC	ZTGEVC	F08YXF
SSBEVD	DSBEVD	F08HCF	CTREVC	ZTREVC	F08QXF
SSBGST	DSBGST	F08UEF	CTREXC	ZTREXC	F08QTF
SSBTRD	DSBTRD	F08HEF	CTRSEN	ZTRSEN	F08QUF
SSPEVD	DSPEVD	F08GCF	CTRSNA	ZTRSNA	F08QYF
SSPGST	DSPGST	F08TEF	CTRSYL	ZTRSYL	F08QVF
SSPTRD	DSPTRD	F08GEF	CUNGBR	ZUNGBR	F08KTF
SSTEBZ	DSTEBZ	F08JJF	CUNGHR	ZUNGHR	F08NTF
SSTEIN	DSTEIN	F08JKF	CUNGLQ	ZUNGLQ	F08AWF
SSTEQR	DSTEQR	F08JEF	CUNGQR	ZUNGQR	F08ATF
SSTERF	DSTERF	F08JFF	CUNGTR	ZUNGTR	F08FTF
SSTEVD	DSTEVD	F08JCF	CUNMBR	ZUNMBR	F08KUF
SSYEVD	DSYEVD	F08FCF	CUNMHR	ZUNMHR	F08NUF
SSYGST	DSYGST	F08SEF	CUNMLQ	ZUNMLQ	F08AXF
SSYTRD	DSYTRD	F08FEF	CUNMQR	ZUNMQR	F08AUF
STGEVC	DTGEVC	F08YKF	CUNMTR	ZUNMTR	F08FUF
STREVC	DTREVC	F08QKF	CUPGTR	ZUPGTR	F08GTF
STREXC	DTREXC	F08QFF	CUPMTR	ZUPMTR	F08GUF
STRSEN	DTRSEN	F08QGF			
STRSNA	DTRSNA	F08QLF			
STRSYL	DTRSYL	F08QHF			

Table 4

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

Arioli M, Duff I S and De Rijk P P M (1989) On the augmented system approach to sparse least-squares problems *Numer. Math.* **55** 667–684

Demmel J W and Kahan W (1990) Accurate singular values of bidiagonal matrices *SIAM J. Sci. Statist. Comput.* **11** 873–912

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

Moler C B and Stewart G W (1973) An algorithm for generalized matrix eigenproblems *SIAM J. Numer. Anal.* **10** 241–256

Parlett B N (1980) *The Symmetric Eigenvalue Problem* Prentice-Hall

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

Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation II, Linear Algebra* Springer-Verlag

Stewart G W and Sun J-G (1990) *Matrix Perturbation Theory* Academic Press, London

Ward R C (1981) Balancing the generalized eigenvalue problem *SIAM J. Sci. Stat. Comp.* **2** 141–152