NAG C Library Chapter Introduction

f08 – Least-squares and Eigenvalue Problems (LAPACK)

Contents

1 Scope of the Chapter

This chapter provides functions for the solution of linear least-squares problems, eigenvalue problems and singular value problems, as well as associated computations. It provides functions 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 eigenvalue and eigenvector problems
- estimating the numerical rank of a matrix
- solution of the Sylvester matrix equation

Functions 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 functions in Chapter f04 or Chapter f08. Chapter f04 contains *Black Box* functions which enable standard linear least-squares problems to be solved by a call to a single function.

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 functions in Chapter f02. Chapter f02 contains *Black Box* functions which enable some standard types of problem to be solved by a call to a single function. Often functions in Chapter f02 call Chapter f08 functions 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* functions.

The functions 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 functions in Chapter f08.

The functions 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 an n element solution vector.

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

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

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

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

This chapter (f08) contains computational functions that can be combined with functions in Chapter f07 to solve these linear least-squares problems. 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 functions 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) when $m \geq n$ and A is of full rank, since

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

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) 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) 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). 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 functions 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 function in the case of the divide and conquer algorithm. In general, more than one function 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.	$BAz = \lambda z$	$B = LL^T$, $B = U^T U$	$C = L^T AL$, $C = UAU^T$	$z = Ly$, $z = U^T y$

Table 1
Reduction of generalized symmetric-definite eigenproblems to standard problems

When the generalized symmetric-definite problem has been reduced to the corresponding standard problem $Cy = \lambda y$, this may then be solved using the functions described in the previous section. No special functions 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 f16).

2.7 Packed Storage for Symmetric Matrices

Functions 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.

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

2.8 Band Matrices

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

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

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

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 functions 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 functions 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 functions 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 to 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 functions 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 functions 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

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

Here $\arccos(\theta)$ is in the standard range: $0 \leq \arccos(\theta) < \pi$.

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

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

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

$\theta(S, \hat{S})$ 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 functions 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 (1998)).

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, functions 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 functions 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.

Functions 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	$\theta(\hat{S}_{\boldsymbol{I}}, S_{\boldsymbol{I}}) \lesssim \|E\|_F/sep_{\boldsymbol{I}}$

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	$\theta(\hat{S}_{\boldsymbol{I}}, S_{\boldsymbol{I}}) \leq \arctan(2\|E\|_F/(sep_{\boldsymbol{I}} - 4\|E\|_F/s_{\boldsymbol{I}}))$	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 functions 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 function 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 function 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 functions 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 f16), 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 functions. 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 functions use an *unblocked* algorithm (effectively a block size of 1), relying solely on calls to the Level 2 BLAS (see Chapter f16 again).

3 Recommendations on Choice and Use of Available Functions

3.1 Available Functions

The tables in the following sub-sections show the functions which are provided for performing different computations on different types of matrices. Each entry in the table gives the NAG function short name and the LAPACK function name from which the NAG function long name is derived by prepending nag_ (see Section 3.2).

For many computations it is necessary to call two or more functions in sequence. Some commonly required sequences of functions are indicated below; an asterisk ($*$) against a function name means that the sequence of calls is illustrated in the example program for that function. (But remember that Black Box functions for the same computations may be provided in Chapter f02 or Chapter f04.)

3.1.1 Orthogonal factorizations

Functions 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 functions do not form the matrix Q explicitly, but represent it as a product of elementary reflectors (see Section 3.3.6). Additional functions 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	f08aec DGEQRF	f08bec DGEQPF	f08afc DORGQR	f08agc DORMQR
LQ factorization, real matrices	f08ahc DGELQF		f08ajc DORGLQ	f08akc DORMLQ
QR factorization, complex matrices	f08asc ZGEQRF	f08bsc ZGEQPF	f08atc ZUNGQR	f08auc ZUNMQR
LQ factorization, complex matrices	f08avc ZGELQF		f08awc ZUNGLQ	f08axc ZUNMLQ

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

real data, full-rank problem	f08aec*, f08agc, f16yjc
complex data, full-rank problem	f08asc*, f08auc, f16zjc
real data, rank-deficient problem	f08bec*, f08agc, f16yjc
complex data, rank-deficient problem	f08bsc*, f08auc, f16zjc

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

real data, full-rank problem	`f08ahc*, f16yjc, f08akc`
complex data, full-rank problem	`f08avc*, f16zjc, f08axc`

3.1.2 Singular value problems

Functions 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 functions 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 functions for reducing full matrices do not form the matrix Q or P explicitly; additional functions are provided to generate all or part of them, or to apply them to another matrix, as with the functions 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 functions for reducing band matrices have options to generate Q or P if required.

Further functions are provided to compute all or part of the singular value decomposition of a real bidiagonal matrix; the same functions 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	`f08kec` `DGEBRD`	`f08kfc` `DORGBR`	`f08kgc` `DORMBR`	`f08lec` `DGBBRD`	`f08mec` `DBDSQR`
complex matrices	`f08ksc` `ZGEBRD`	`f08ktc` `ZUNGBR`	`f08kuc` `ZUNMBR`	`f08lsc` `ZGBBRD`	`f08msc` `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	`f08kec, f08kfc*, f08mec`
complex matrix, singular values and vectors	`f08ksc, f08ktc*, f08msc`

Rectangular matrix (banded)

real matrix, singular values and vectors	`f08lec`
complex matrix, singular values and vectors	`f08lsc`

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

real data:	`f08kec, f08kgc, f08kfc, f08mec, f06yac`
complex data:	`f08ksc, f08kuc, f08ktc, f08msc, f06zac`

3.1.3 Symmetric eigenvalue problems

Functions 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 functions 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 functions for reducing full matrices do not form the matrix Q explicitly; additional functions are provided to generate Q, or to apply it to another matrix, as with the functions 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 functions 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	f08fec DSYTRD	f08ffc DORGTR	f08fgc DORMTR
real symmetric matrices (packed storage)	f08gec DSPTRD	f08gfc DOPGTR	f08ggc DOPMTR
real symmetric band matrices	f08hec DSBTRD		
complex Hermitian matrices	f08fsc ZHETRD	f08ftc ZUNGTR	f08fuc ZUNMTR
complex Hermitian matrices (packed storage)	f08gsc ZHPTRD	f08gtc ZUPGTR	f08guc ZUPMTR
complex Hermitian band matrices	f08hsc ZHBTRD		

A variety of functions 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 functions 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)	f08jfc
all eigenvalues (root-free QR algorithm called by divide and conquer)	f08jcc
selected eigenvalues (bisection)	f08jjc

The original (non-reduced) matrix is Real

all eigenvalues and eigenvectors (QR algorithm)	f08jec
all eigenvalues and eigenvectors (divide and conquer)	f08jcc
all eigenvalues and eigenvectors (positive-definite case)	f08jgc
selected eigenvectors (inverse iteration)	f08jkc

The original (non-reduced) matrix is Complex Hermitian

all eigenvalues and eigenvectors (QR algorithm)	f08jsc
all eigenvalues and eigenvectors (positive-definite case)	f08juc
selected eigenvectors (inverse iteration)	f08jxc

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)	f08fcc
all eigenvalues and eigenvectors (using QR algorithm)	f08fec, f08ffc*, f08jec
selected eigenvalues and eigenvectors (bisection and inverse iteration)	f08fec, f08jjc, f08jkc, f08fgc*

Real Symmetric matrix (packed storage)

all eigenvalues and eigenvectors (using divide and conquer)	`f08gcc`
all eigenvalues and eigenvectors (using QR algorithm)	`f08gec, f08gfc*, f08jec`
selected eigenvalues and eigenvectors (bisection and inverse iteration)	`f08gec, f08jjc, f08jkc, f08ggc*`

Real Symmetric banded matrix

all eigenvalues and eigenvectors (using divide and conquer)	`f08hcc`
all eigenvalues and eigenvectors (using QR algorithm)	`f08hec*, f08jec`

Complex Hermitian matrix (standard storage)

all eigenvalues and eigenvectors (using divide and conquer)	`f08fqc`
all eigenvalues and eigenvectors (using QR algorithm)	`f08fsc, f08ftc*, f08jsc`
selected eigenvalues and eigenvectors (bisection and inverse iteration)	`f08fsc, f08jjc, f08jxc, f08fuc*`

Complex Hermitian matrix (packed storage)

all eigenvalues and eigenvectors (using divide and conquer)	`f08gqc`
all eigenvalues and eigenvectors (using QR algorithm)	`f08gsc, f08gtc*, f08jsc`
selected eigenvalues and eigenvectors (bisection and inverse iteration)	`f08gsc, f08jjc, f08jxc, f08guc*`

Complex Hermitian banded matrix

all eigenvalues and eigenvectors (using divide and conquer)	`f08hqc`
all eigenvalues and eigenvectors (using QR algorithm)	`f08hsc*, f08jsc`

3.1.4 Generalized symmetric-definite eigenvalue problems

Functions 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 functions allow the matrices to be stored either conventionally or in packed storage. The positive-definite matrix B must first be factorized using a function from Chapter f07. There is also a function 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 function in Chapter f08 is provided.

	Reduce to standard problem	**Reduce to standard problem (packed storage)**	**Reduce to standard problem (band matrices)**
real symmetric matrices	`f08sec` `DSYGST`	`f08tec` `DSPGST`	`f08uec` `DSBGST`
complex Hermitian matrices	`f08ssc` `ZHEGST`	`f08tsc` `ZHPGST`	`f08usc` `ZHBGST`

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

real symmetric-definite problem	`f07fdc, f08sec*, f08fec, f08jfc`
real symmetric-definite problem, packed storage	`f07gdc, f08tec*, f08gec, f08jfc`
real symmetric-definite banded problem	`f08ufc*, f08uec*, f08hec, f08jfc`
complex Hermitian-definite problem	`f07frc, f08ssc*, f08fsc, f08jfc`
complex Hermitian-definite problem, packed storage	`f07grc, f08tsc*, f08gsc, f08jfc`
complex Hermitian-definite banded problem	`f08utc*, f08usc*, f08hsc, f08jfc`

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; functions from Chapter f16 may be used for this.

3.1.5 Nonsymmetric eigenvalue problems

Functions 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 functions do not form the matrix Q explicitly; additional functions are provided to generate Q, or to apply it to another matrix, as with the functions 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.

Functions are also provided to balance the matrix before reducing it to Hessenberg form, as described in Section 2.12.6. Companion functions 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	f08nec DGEHRD	f08nfc DORGHR	f08ngc DORMHR	f08nhc DGEBAL	f08njc DGEBAK
complex matrices	f08nsc ZGEHRD	f08ntc ZUNGHR	f08nuc ZUNMHR	f08nvc ZGEBAL	f08nwc ZGEBAK

Functions 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 functions 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	f08pec DHSEQR	f08pkc DHSEIN	f08qkc DTREVC	f08qlc DTRSNA
complex matrices	f08psc ZHSEQR	f08pxc ZHSEIN	f08qxc ZTREVC	f08qyc ZTRSNA

Finally functions are provided for re-ordering the Schur factorization, so that eigenvalues appear in any desired order on the diagonal of the Schur form. The functions nag_dtrexc (f08qfc) and nag_ztrexc (f08qtc) simply swap two diagonal elements or blocks, and may need to be called repeatedly to achieve a desired order. The functions nag_dtrsen (f08qgc) and nag_ztrsen (f08quc) 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 functions 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	f08qfc DTREXC	f08qgc DTRSEN
complex matrices	f08qtc ZTREXC	f08quc 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	f08nec, f08nfc*, f08pec
real matrix, all eigenvalues and selected eigenvectors	f08nec, f08pec, f08pkc, f08ngc*
real matrix, all eigenvalues and eigenvectors (with balancing)	f08nhc*, f08nec, f08nfc, f08pec, f08pkc, f08njc
complex matrix, all eigenvalues and Schur factorization	f08nsc, f08ntc*, f08psc
complex matrix, all eigenvalues and selected eigenvectors	f08nsc, f08psc, f08pxc, f08nuc*
complex matrix, all eigenvalues and eigenvectors (with balancing)	f08nvc*, f08nsc, f08ntc, f08psc, f08pxc, f08nwc

3.1.6 Generalized nonsymmetric eigenvalue problems

Functions 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 functions 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).

Functions are also provided to balance a general matrix pair before reducing it to generalized Hessenberg form, as described in Section 2.12.8. Companion functions 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	f08wec DGGHRD	f08whc DGGBAL	f08wjc DGGBAK
complex matrices	f08wsc ZGGHRD	f08wvc ZGGBAL	f08wwc ZGGBAK

Functions 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	f08xec DHGEQZ	f08ykc DTGEVC
complex matrices	f08xsc ZHGEQZ	f08yxc 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)	f08whc, f08aec, f08agc, f08wec, f08xec*
real matrix pair, all eigenvalues and generalized Schur factorization	f08aec, f08agc, f08afc, f08wec, f08xec
real matrix pair, all eigenvalues and eigenvectors (with balancing)	f08whc, f08aec, f08agc, f16qhc, f16qfc, f08afc, f08wec, f08xec, f08ykc*, f08wjc
complex matrix pair, all eigenvalues (with balancing)	f08wvc, f08asc, f08auc, f08wsc, f08xsc*
complex matrix pair, all eigenvalues and generalized Schur factorization	f08asc, f08auc, f08atc, f08wsc, f08xsc
complex matrix pair, all eigenvalues and eigenvectors (with balancing)	f08wvc, f08asc, f08auc, f16thc, f16tfc, f08atc, f08wsc, f08xsc, f08yxc*, f08wwc

3.1.7 Sylvester's equation

Functions 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 functions described in Section 3.1.5. For more details, see the documents for the functions listed below.

	solve Sylvester's equation
real matrices	f08qhc DTRSYL
complex matrices	f08qvc ZTRSYL

3.2 NAG Names and LAPACK Names

As well as the NAG function short name (beginning f08-), the tables in Section 3.1 show the LAPACK function names in double precision.

The functions may be called either by their NAG short names or by their NAG long names which contain their double precision LAPACK names.

References to Chapter f08 functions in the manual normally include the LAPACK double precision names, for example nag_dgeqrf (f08aec). The LAPACK function names follow a simple scheme (which is similar to that used for the BLAS in Chapter f16). 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
D – real, double precision
C – complex, single precision
Z – complex, double precision

– 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 function nag_dgeqrf performs a QR factorization of a real general matrix in a single precision implementation of the Library.

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 f16 and Chapter f07, but different schemes for packed, band and tridiagonal storage are used in a few older functions in Chapter f01, Chapter f03, Chapter f03 and Chapter f04.

In the examples below, $*$ indicates an array element which need not be set and is not referenced by the functions. The examples illustrate only the relevant leading rows and columns of the arrays.

3.3.1 Conventional storage

Matrices may be stored column-wise or row-wise as described in Section 2.2.1.4 of the Essential Introduction: a matrix A is stored in a one-dimensional array **a**, with matrix element $a_{i,j}$ stored column-wise in array element $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ or row-wise in array element $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ where **pda** is the principle dimension of the array (i.e., the stride separating row or column elements of the matrix respectively). Most functions in this chapter contain the **order** argument which can be set to **Nag_ColMajor** for column-wise storage or **Nag_RowMajor** for row-wise storage of matrices. Where

groups of functions are intended to be used together, the value of the **order** argument passed must be consistent throughout.

If a matrix is **triangular** (upper or lower, as specified by the argument **uplo**), only the elements of the relevant triangle are stored; the remaining elements of the array need not be set. Such elements are indicated by * in the examples below.

For example, when $n = 3$:

order	uplo	Triangular matrix A	Storage in array a
Nag_ColMajor	**Nag_Upper**	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ & a_{22} & a_{23} \\ & & a_{33} \end{pmatrix}$	$a_{11}\ *\ *\ a_{12}\ a_{22}\ *\ a_{13}\ a_{23}\ a_{33}$
Nag_RowMajor	**Nag_Upper**	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ & a_{22} & a_{23} \\ & & a_{33} \end{pmatrix}$	$a_{11}\ a_{12}\ a_{13}\ *\ a_{22}\ a_{23}\ *\ *\ a_{33}$
Nag_ColMajor	**Nag_Lower**	$\begin{pmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$	$a_{11}\ a_{21}\ a_{31}\ *\ a_{22}\ a_{32}\ *\ *\ a_{33}$
Nag_RowMajor	**Nag_Lower**	$\begin{pmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$	$a_{11}\ *\ *\ a_{21}\ a_{22}\ *\ a_{31}\ a_{32}\ a_{33}$

functions 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 $n = 3$:

order	uplo	Hermitian matrix A	Storage in array a
Nag_ColMajor	**Nag_Upper**	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ \bar{a}_{12} & a_{22} & a_{23} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} \end{pmatrix}$	$a_{11}\ *\ *\ a_{12}\ a_{22}\ *\ a_{13}\ a_{23}\ a_{33}$
Nag_RowMajor	**Nag_Upper**	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ \bar{a}_{12} & a_{22} & a_{23} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} \end{pmatrix}$	$a_{11}\ a_{12}\ a_{13}\ *\ a_{22}\ a_{23}\ *\ *\ a_{33}$
Nag_ColMajor	**Nag_Lower**	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} \\ a_{21} & a_{22} & \bar{a}_{32} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$	$a_{11}\ a_{21}\ a_{31}\ *\ a_{22}\ a_{32}\ *\ *\ a_{33}$
Nag_RowMajor	**Nag_Lower**	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} \\ a_{21} & a_{22} & \bar{a}_{32} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$	$a_{11}\ *\ *\ a_{21}\ a_{22}\ *\ a_{31}\ a_{32}\ a_{33}$

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 or rows in a one-dimensional array. In Chapters f07 and f08, arrays which hold matrices in packed storage have names ending in p. The storage of matrix elements $a_{i,j}$ are stored in the packed array **ap** as follows:

if **uplo** = **Nag_Upper** then

if **order** = **Nag_ColMajor**, a_{ij} is stored in **ap**$[(i-1)+j(j-1)/2]$ for $i \leq j$;

if **order** = **Nag_RowMajor**, a_{ij} is stored in **ap**$[(j-1)+(2n-i)(i-1)/2]$ for $i \leq j$;

if **uplo** = **Nag_Lower** then

if **order** = **Nag_ColMajor**, a_{ij} is stored in **ap**$[(i-1)+(2n-j)(j-1)/2]$ for $j \leq i$;

if **order** = **Nag_RowMajor**, a_{ij} is stored in **ap**$[(j-1)+i(i-1)/2]$ for $j \leq i$.

For example:

order	uplo	Triangle of matrix A	Packed storage in array ap
Nag_ColMajor	**Nag_Upper**	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ & a_{22} & a_{23} \\ & & a_{33} \end{pmatrix}$	$a_{11}\ \underbrace{a_{12}a_{22}}\ \underbrace{a_{13}a_{23}a_{33}}$
Nag_RowMajor	**Nag_Upper**	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ & a_{22} & a_{23} \\ & & a_{33} \end{pmatrix}$	$\underbrace{a_{11}a_{12}a_{13}}\ \underbrace{a_{22}a_{23}}\ a_{33}$
Nag_ColMajor	**Nag_Lower**	$\begin{pmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$	$\underbrace{a_{11}a_{21}a_{31}}\ \underbrace{a_{22}a_{32}}\ a_{33}$
Nag_RowMajor	**Nag_Lower**	$\begin{pmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$	$a_{11}\ \underbrace{a_{21}a_{22}}\ \underbrace{a_{31}a_{32}\ a_{33}}$

Note that for real symmetric matrices, packing the upper triangle by columns is equivalent to packing the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the upper triangle by rows. (For complex Hermitian matrices, the only difference is that the off-diagonal elements are conjugated.)

3.3.3 Band storage

A band matrix with k_l sub-diagonals and k_u super-diagonals may be stored compactly in a notional two-dimensional array with $k_l + k_u + 1$ rows and n columns if stored column-wise or n rows and $k_l + k_u + 1$ columns if stored row-wise. In column-major order, elements of a column of the matrix are stored contiguously in the array, and elements of the diagonals of the matrix are stored with constant stride (i.e., in a row of the two-dimensional array). In row-major order, elements of a row of the matrix are stored contiguously in the array, and elements of a diagonal of the matrix are stored with constant stride (i.e., in a column of the two-dimensional array). These storage schemes should only be used in practice if k_l, $k_u \ll n$, although the functions in Chapter f07 and Chapter f08 work correctly for all values of k_l and k_u. In Chapter f07 and Chapter 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:

if **order** = **Nag_ColMajor**, a_{ij} is stored in **ab**$[(k_u + i - j) \times$ **ldab** $+ j]$;

if **order** = **Nag_RowMajor**, a_{ij} is stored in **ab**$[(k_l + j - i) \times$ **pdab** $+ i]$;

where **pdab** $\geq k_l + k_u + 1$ is the stride between diagonal elements and where $\max(1, i - k_l) \leq j \leq \min(n, i + k_u)$.

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

Band matrix A	**Band storage in array ab**	
	order = Nag_ColMajor	**order = Nag_RowMajor**
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{pmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{pmatrix}$	$\begin{pmatrix} * & * & a_{11} & a_{12} \\ * & a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} & a_{45} \\ a_{53} & a_{54} & a_{55} & * \end{pmatrix}$

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 functions.

Triangular band matrices are stored in the same format, with either $k_l = 0$ if upper triangular, or $k_u = 0$ if lower triangular.

For symmetric or Hermitian band matrices with k sub-diagonals or super-diagonals, only the upper or lower triangle (as specified by **uplo**) need be stored:

if **uplo** = **Nag_Upper** then

if **order** = **Nag_ColMajor**, a_{ij} is stored in **ab**$[(j-1) \times$ **pdab** $+ k + i - j]$;

if **order** = **Nag_RowMajor**, a_{ij} is stored in **ab**$[(i-1) \times$ **pdab** $+ j - i]$;

for $\max(1, j-k) \le i \le j$;

if **uplo** = **Nag_Lower** then

if **order** = **Nag_ColMajor**, a_{ij} is stored in **ab**$[(j-1) \times$ **pdab** $+ i - j]$;

if **order** = **Nag_RowMajor**, a_{ij} is stored in **ab**$[(i-1) \times$ **pdab** $+ k + j - i]$;

for $j \le i \le \min(n, j+k)$;

where **pdab** $\ge k + 1$ is the stride separating diagonal matrix elements in the array **ab**.

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

uplo	**Hermitian band matrix** A	**Band storage in array a**	
		order = Nag_ColMajor	**order = Nag_RowMajor**
Nag_Upper	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} & \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} & a_{35} \\ & \bar{a}_{24} & \bar{a}_{34} & a_{44} & a_{45} \\ & & \bar{a}_{35} & \bar{a}_{45} & a_{55} \end{pmatrix}$	$\begin{pmatrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{pmatrix}$	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{22} & a_{23} & a_{24} \\ a_{33} & a_{34} & a_{35} \\ a_{44} & a_{45} & * \\ a_{55} & * & * \end{pmatrix}$
Nag_Lower	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & & \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} & \bar{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \bar{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{pmatrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{pmatrix}$	$\begin{pmatrix} * & * & a_{11} \\ * & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} \\ a_{42} & a_{43} & a_{44} \\ a_{53} & a_{54} & a_{55} \end{pmatrix}$

Note that different storage schemes for band matrices are used by some functions in Chapter f01, Chapter f03, Chapter f03 and Chapter f04.

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 functions 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 functions are defined by the algorithm to have real diagonal elements – in QR factorization, for example.

If such matrices are supplied as input to f08 functions, 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 functions, 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 functions 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 functions in Chapter f01, Chapter f02, Chapter f04 and Chapter f16) 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

In addition to the **order** argument of type **Nag_OrderType**, most functions in this Chapter have one or more option arguments of various types; only options of the correct type may be supplied.

For example,

```
f08fec(Nag_RowMajor,Nag_Upper,...)
```

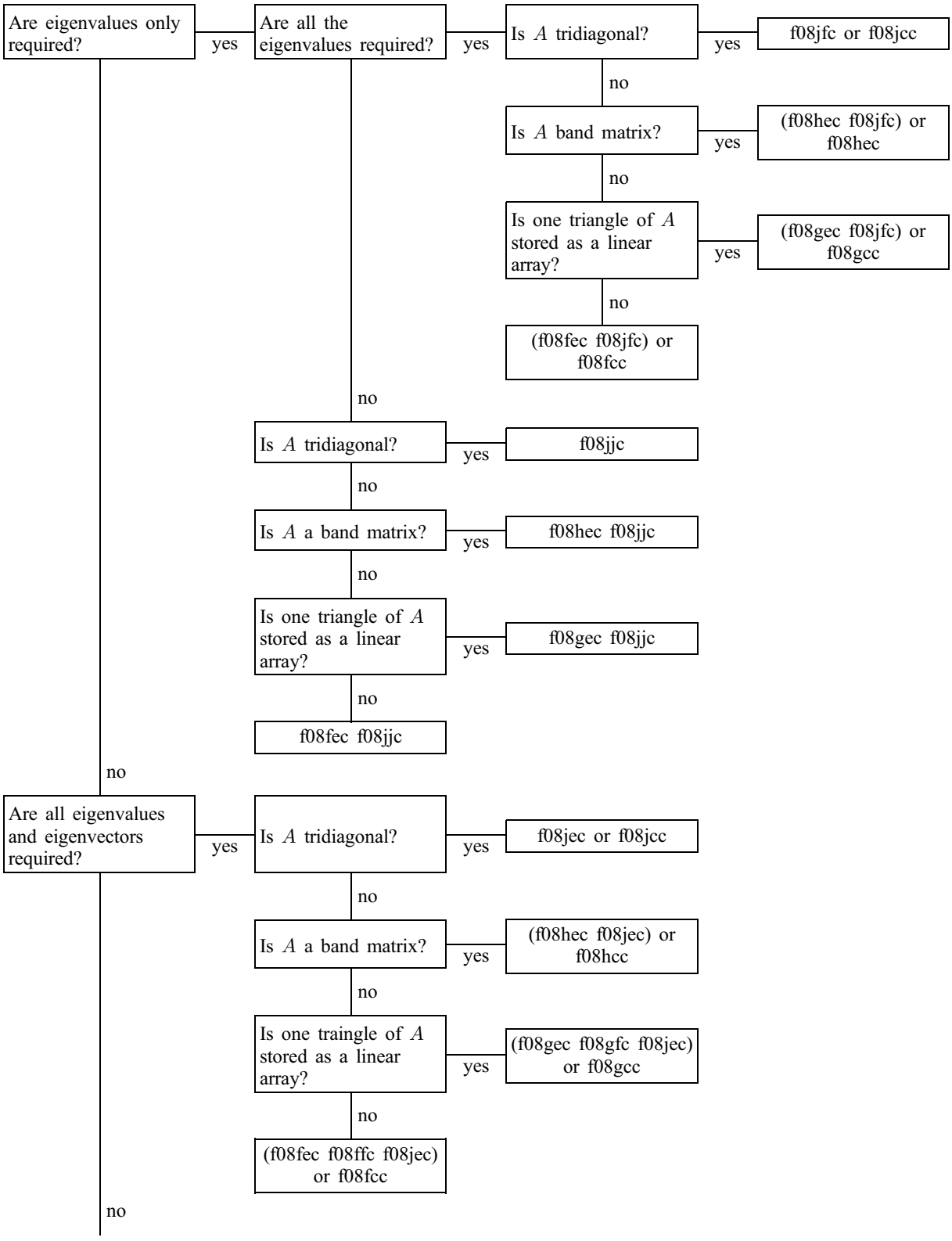
3.4.2 Problem dimensions

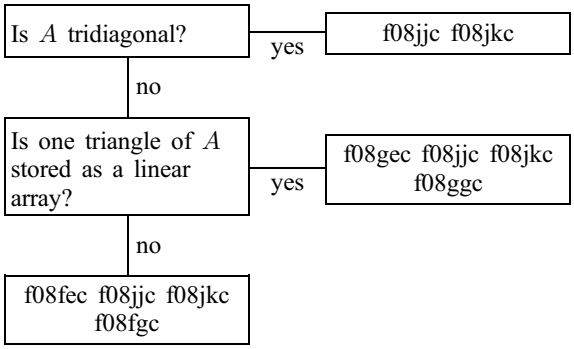
It is permissible for the problem dimensions (for example, M or N) to be passed as zero, in which case the computation (or part of it) is skipped. Negative dimensions are regarded as an error.

4 Decision Tree

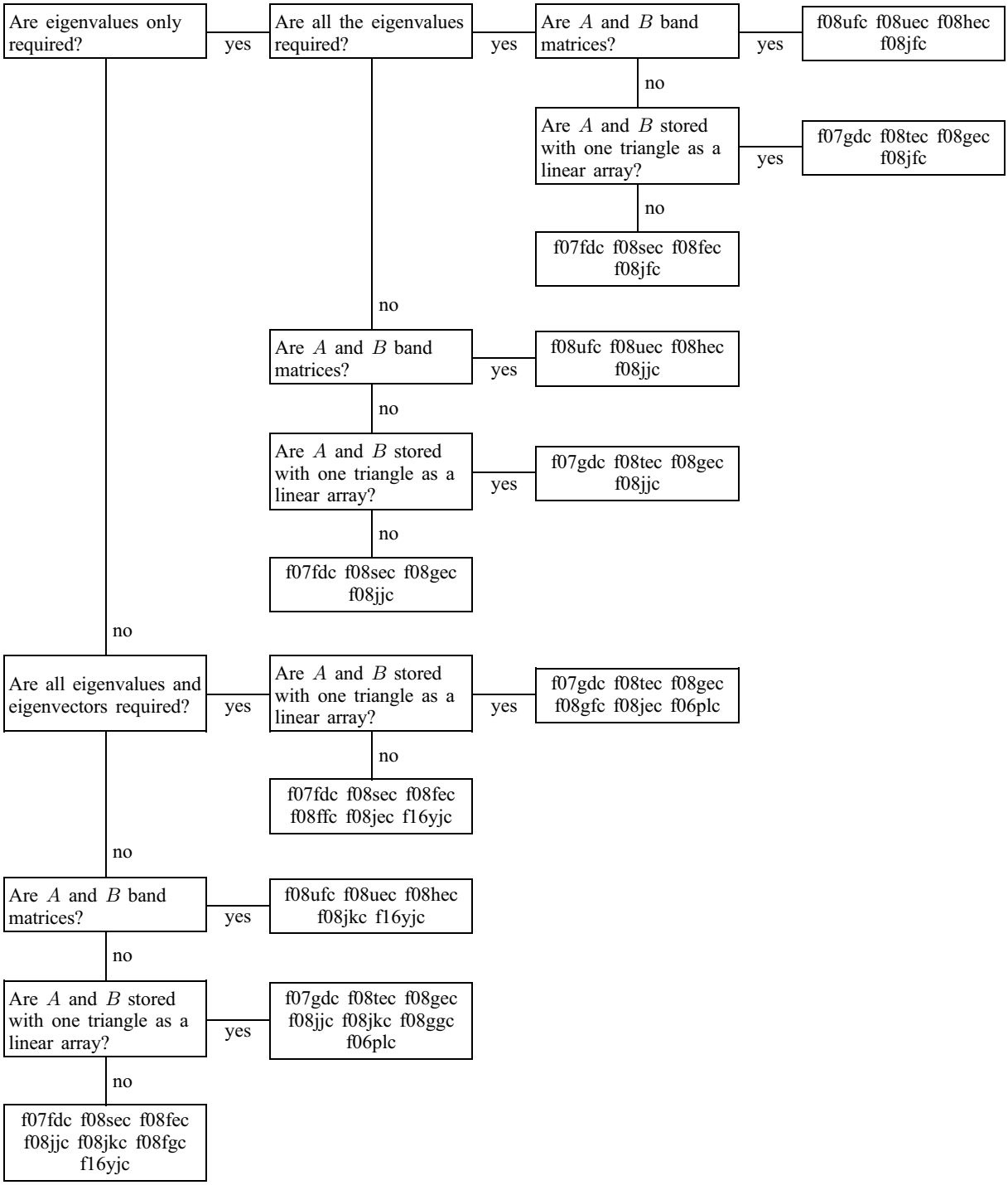
4.1 General purpose functions (eigenvalues and eigenvectors)

Tree 1: Real Symmetric Eigenvalue Problems



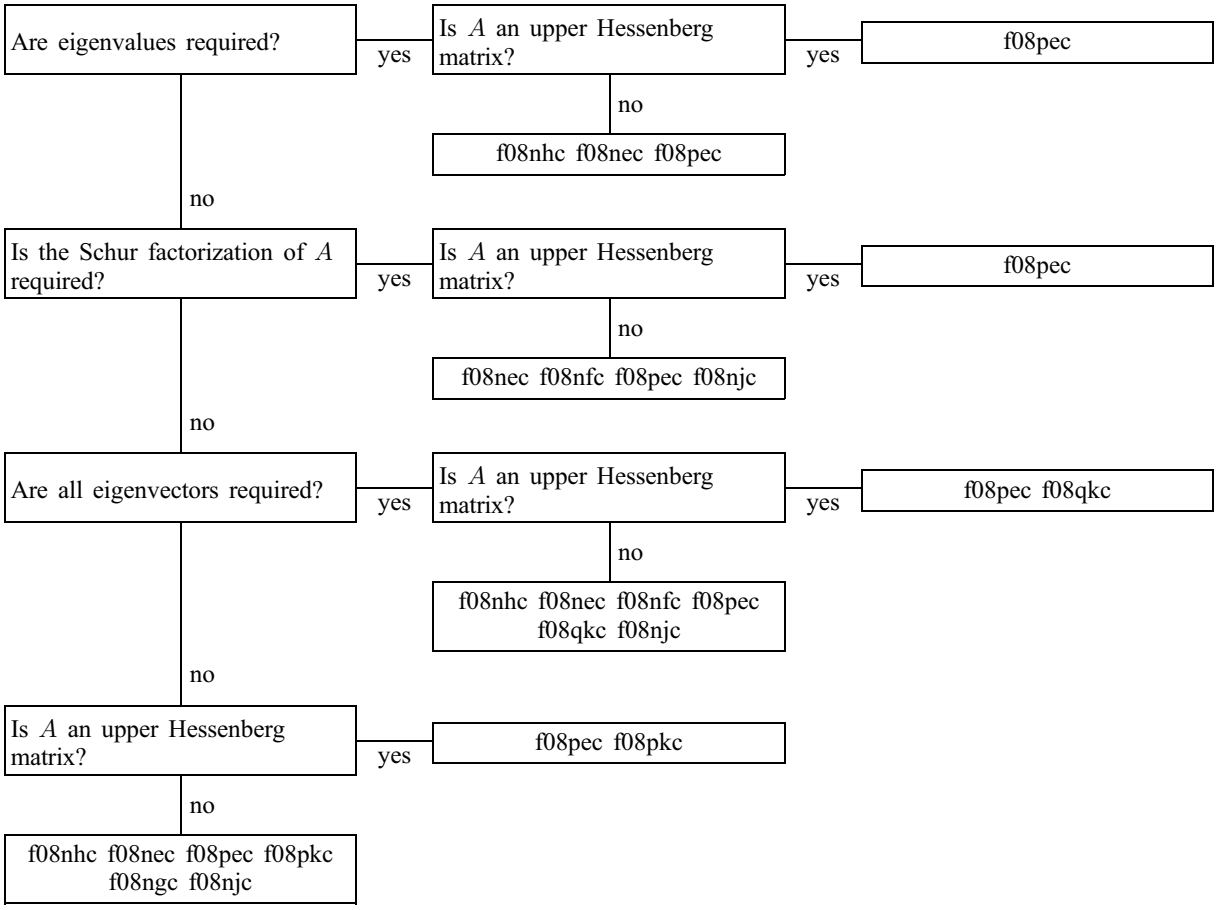

Is A tridiagonal?
yes
f08jjc f08jkc
no
Is one triangle of A stored as a linear array?
yes
f08gec f08jjc f08jkc f08ggc
no
f08fec f08jjc f08jkc f08fgc

Tree 2: Real Generalized Symmetric-definite Eigenvalue Problems

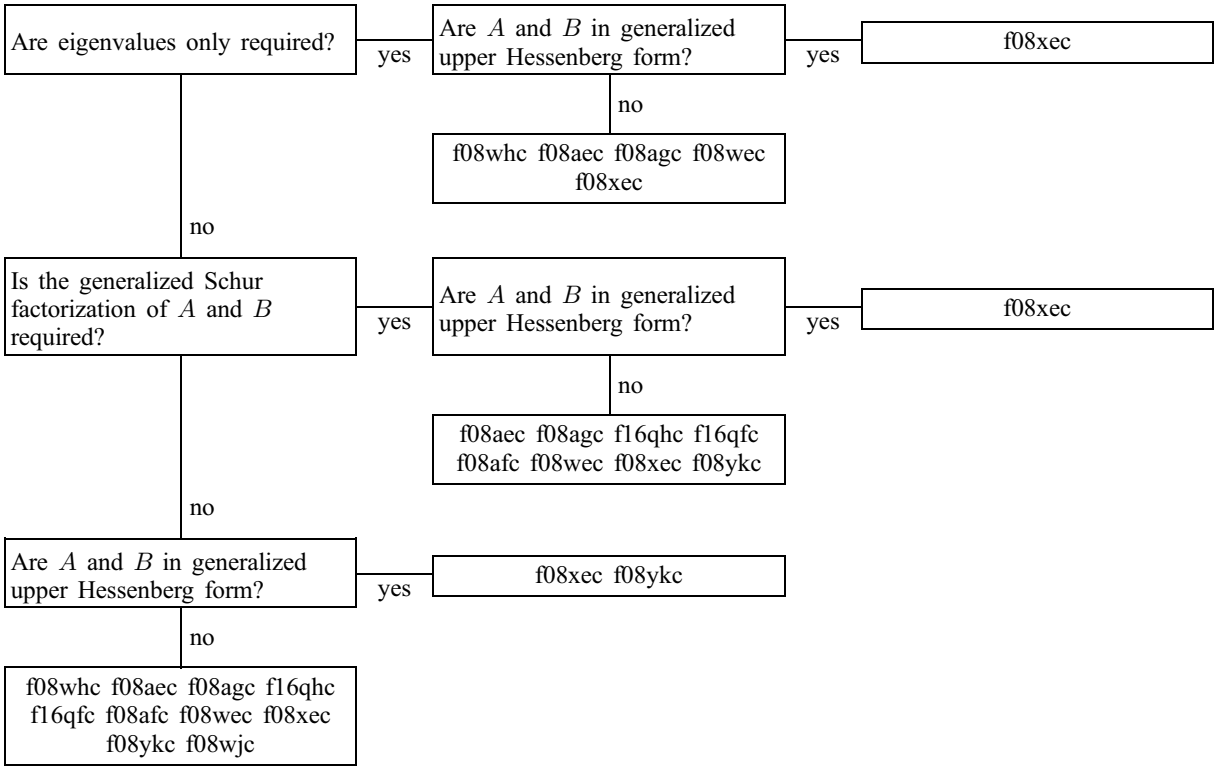


Note: the functions for band matrices only handle the problem $Ax = \lambda Bx$; the other functions handle all three types of problems ($Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$) except that, if the problem is $BAx = \lambda x$ and eigenvectors are required, f06phc must be used instead of f06plc, and f06yfc instead of f16yjc.

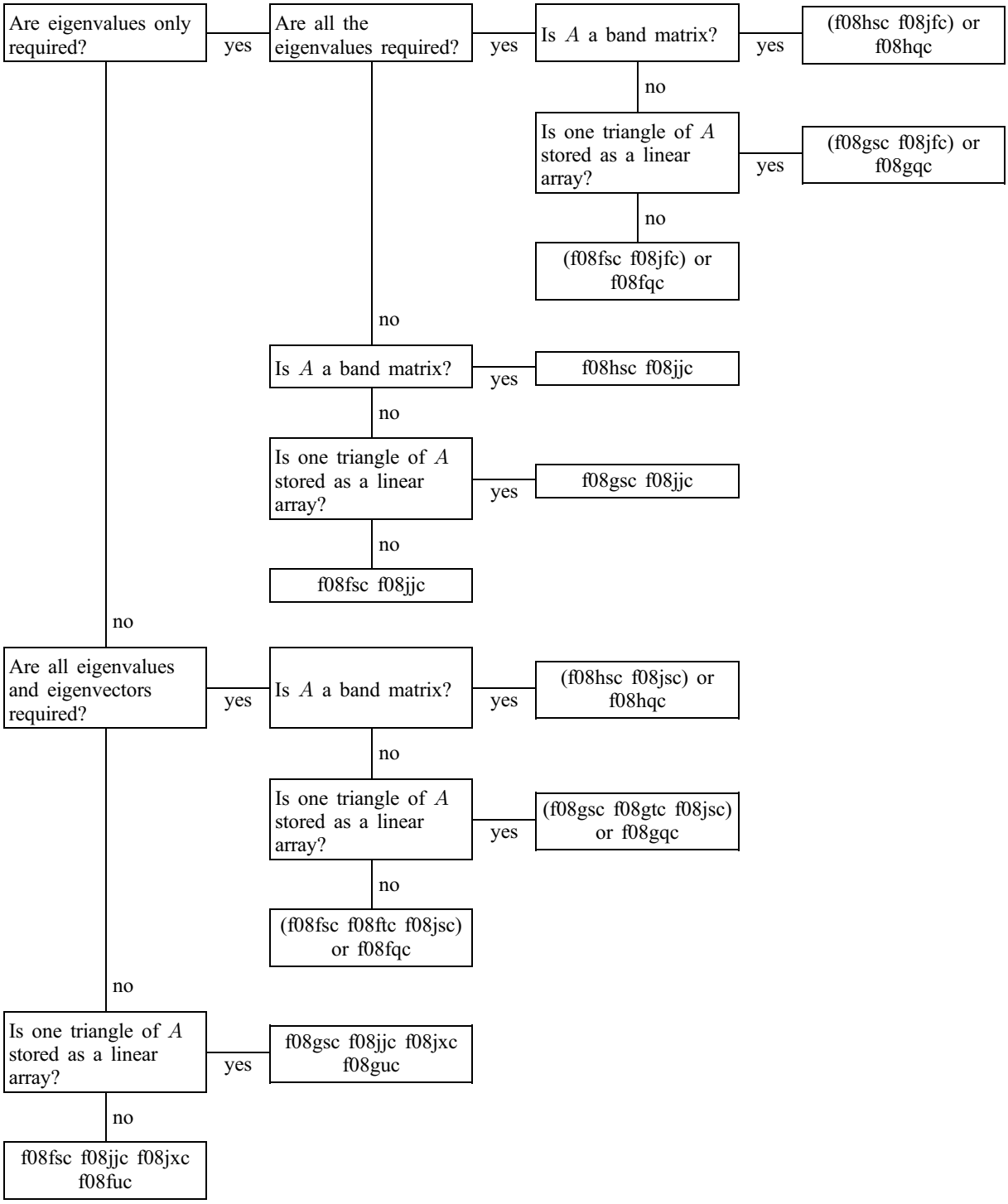
Tree 3: Real Nonsymmetric Eigenvalue Problems



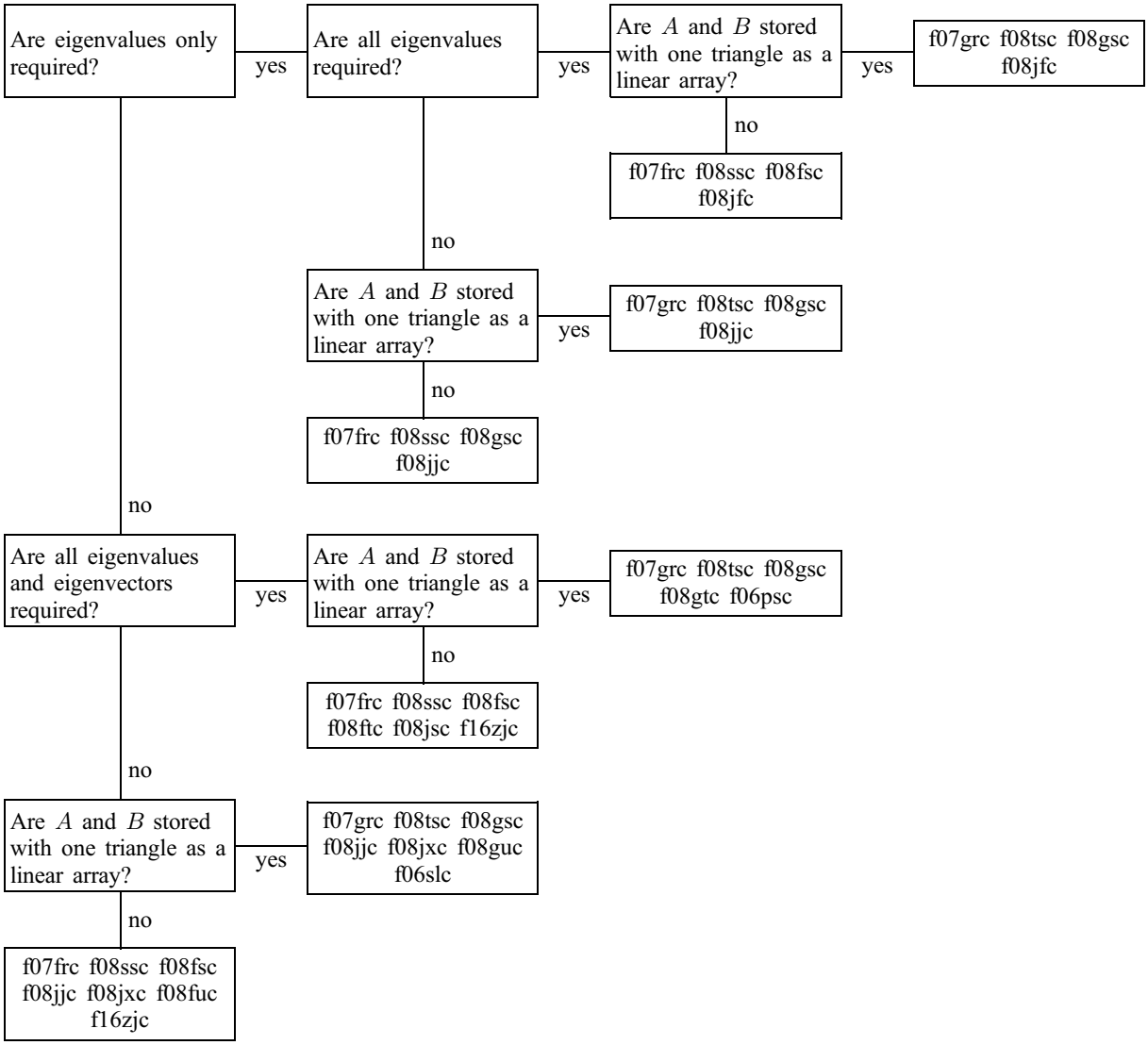
Tree 4: Real Generalized Nonsymmetric Eigenvalue Problems



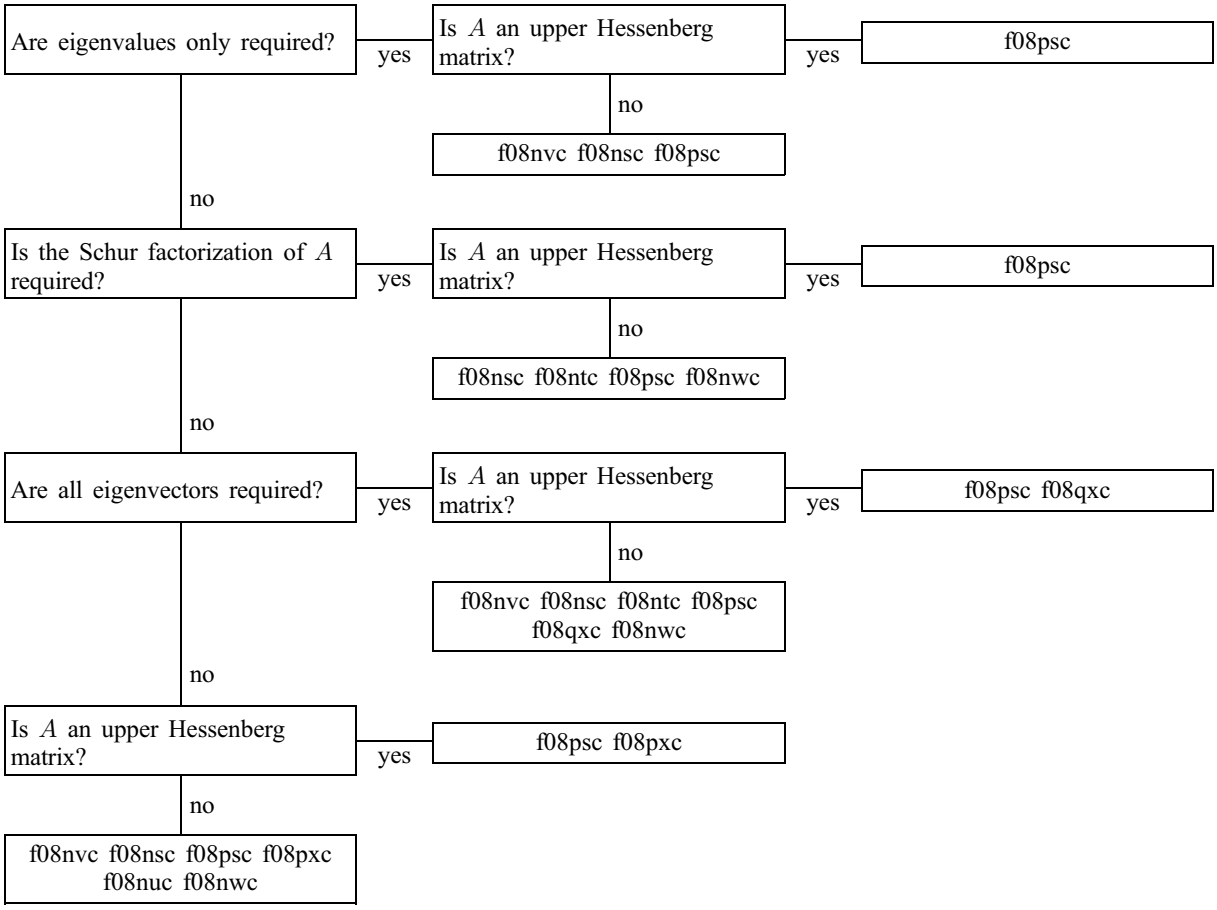
Tree 5: Complex Hermitian Eigenvalue Problems



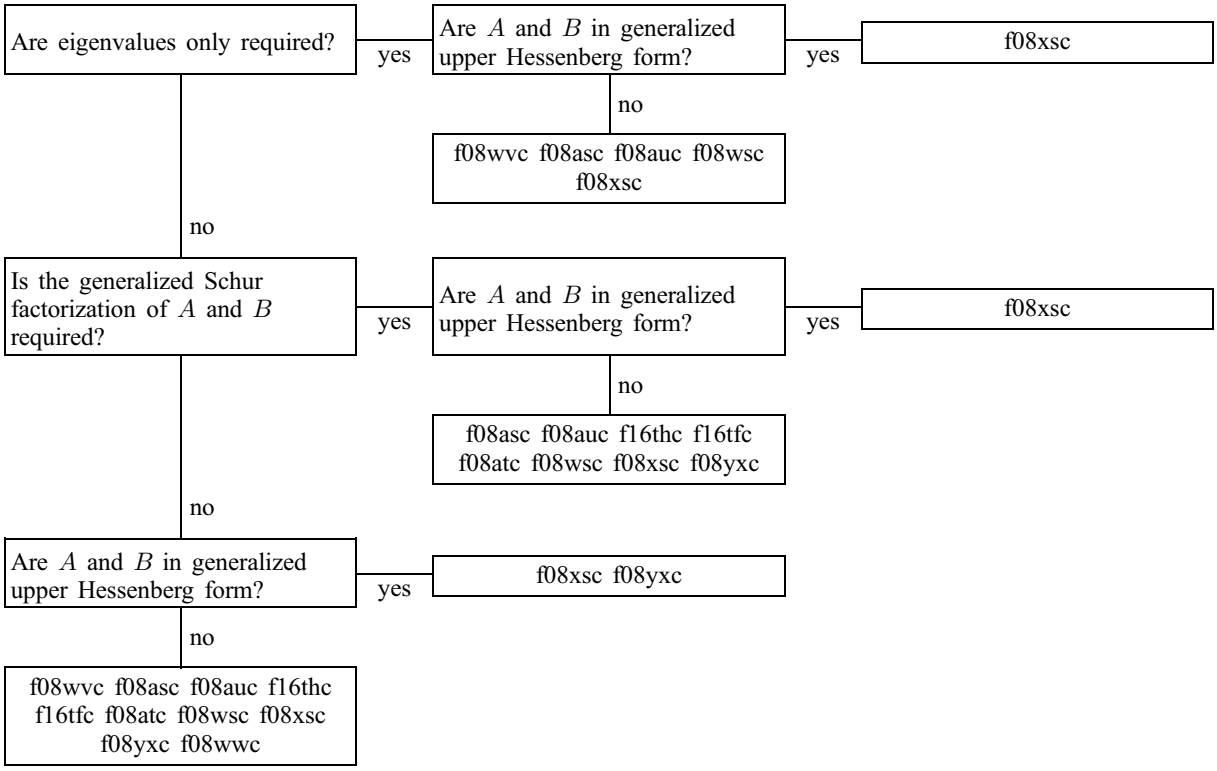
Tree 6: Complex Generalized Hermitian-definite Eigenvalue Problems



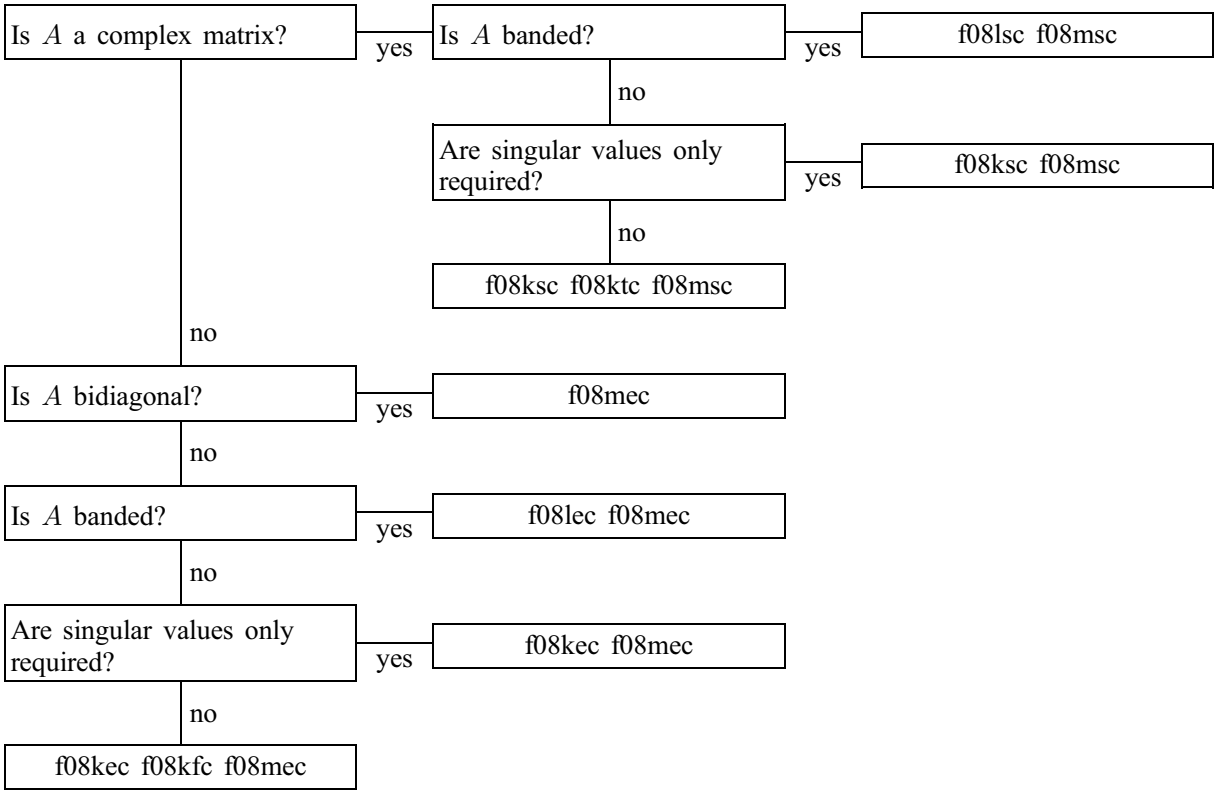
Tree 7: Complex non-Hermitian Eigenvalue Problems



Tree 8: Complex Generalized non-Hermitian Eigenvalue Problems



4.2 General purpose functions (singular value decomposition)



5 Index

real bidiagonal matrix:
singular value decomposition:
after reduction from complex general matrix ..... nag_zbdsqr (f08msc)
after reduction from real general matrix ..... nag_dbdsqr (f08mec)
real symmetric matrix:
eigenvalues and eigenvectors:
band matrix:
all eigenvalues and eigenvectors by a divide and conquer algorithm nag_dsbevd (f08hcc)
general matrix:
all eigenvalues and eigenvectors by a divide and conquer algorithm nag_dsyevd (f08fcc)
all eigenvalues and eigenvectors by a divide and conquer algorithm using packed storage nag_dspevd (f08gcc)
eigenvalues only:
band matrix:
all eigenvalues by the Pal–Walker–Kahan variant of the QL or QR algorithm nag_dsbevd (f08hcc)
general matrix:
all eigenvalues by the Pal–Walker–Kahan variant of the QL or QR algorithm nag_dsyevd (f08fcc)
all eigenvalues by the Pal–Walker–Kahan variant of the QL or QR algorithm using packed storage ..... nag_dspevd (f08gcc)
real symmetric tridiagonal matrix:
eigenvalues and eigenvectors:
after reduction from complex Hermitian matrix:
all eigenvalues and eigenvectors ..... nag_zsteqr (f08jsc)
selected eigenvectors by inverse iteration ..... nag_zstein (f08jxc)
all eigenvalues and eigenvectors ..... nag_dsteqr (f08jec)
all eigenvalues and eigenvectors by a divide and conquer algorithm after reduction from real symmetric matrix: ..... nag_dstevd (f08jcc)
selected eigenvectors by inverse iteration ..... nag_dstein (f08jkc)
eigenvalues only:
all eigenvalues by root-free QR algorithm ..... nag_dsterf (f08jfc)
all eigenvalues by the Pal–Walker–Kahan variant of the QL or QR algorithm nag_dstevd (f08jcc)
selected eigenvalues by bisection ..... nag_dstebz (f08jjc)
real upper Hessenberg matrix, reduced from real general matrix:
eigenvalues and Schur factorization ..... nag_dhseqr (f08pec)
selected right and/or left eigenvectors by inverse iteration ..... nag_dhsein (f08pkc)
Eigenvalues and generalized Schur factorization
complex generalized upper Hessenberg form ..... nag_zhgeqz (f08xsc)
real generalized upper Hessenberg form ..... nag_dhgeqz (f08xec)
Left and right eigenvectors of a pair of matrices:
complex upper triangular matrices ..... nag_ztgevc (f08yxc)
real quasi-triangular matrices ..... nag_dtgevc (f08ykc)
LQ factorization and related operations:
complex matrices:
apply unitary matrix ..... nag_zunmlq (f08axc)
factorization ..... nag_zgelqf (f08avc)
form all or part of unitary matrix ..... nag_zunglq (f08awc)
real matrices:
apply orthogonal matrix ..... nag_dormlq (f08akc)
factorization ..... nag_dgelqf (f08ahc)
form all or part of orthogonal matrix ..... nag_dorglq (f08ajc)
Operations on Schur factorization of a general matrix:
complex matrix:
compute left and/or right eigenvectors ..... nag_ztrevc (f08qxc)
estimate sensitivities of eigenvalues and/or eigenvectors ..... nag_ztrsna (f08qyc)
re-order Schur factorization ..... nag_ztrexc (f08qtc)

- re-order Schur factorization, compute basis of invariant subspace, and estimate sensitivities — nag_ztrsen (f08quc)

real matrix:

- compute left and/or right eigenvectors — nag_dtrevc (f08qkc)
- estimate sensitivities of eigenvalues and/or eigenvectors — nag_dtrsna (f08qlc)
- re-order Schur factorization — nag_dtrexc (f08qfc)
- re-order Schur factorization, compute basis of invariant subspace, and estimate sensitivities — nag_dtrsen (f08qgc)

QR factorization and related operations:

complex matrices:

- apply unitary matrix — nag_zunmqr (f08auc)
- factorization — nag_zgeqrf (f08asc)
- form all or part of unitary matrix — nag_zungqr (f08atc)

real matrices:

- apply orthogonal matrix — nag_dormqr (f08agc)
- factorization — nag_dgeqrf (f08aec)
- form all or part of orthogonal matrix — nag_dorgqr (f08afc)

Reduction of a pair of general matrices to generalized upper Hessenberg form

- orthogonal reduction, real matrices — nag_dgghrd (f08wec)
- unitary reduction, complex matrices — nag_zgghrd (f08wsc)

Reduction of eigenvalue problems to condensed forms, and related operations:

complex general matrix to upper Hessenberg form:

- apply orthogonal matrix — nag_zunmhr (f08nuc)
- form orthogonal matrix — nag_zunghr (f08ntc)
- reduce to Hessenberg form — nag_zgehrd (f08nsc)

complex Hermitian band matrix to real symmetric tridiagonal form — nag_zhbtrd (f08hsc)

complex Hermitian matrix to real symmetric tridiagonal form:

- apply unitary matrix — nag_zunmtr (f08fuc)
- form unitary matrix — nag_zungtr (f08ftc)
- reduce to tridiagonal form — nag_zhetrd (f08fsc)

complex rectangular band matrix to real upper bidiagonal form — nag_zgbbrd (f08lsc)

complex rectangular matrix to real bidiagonal form:

- apply unitary matrix — nag_zunmbr (f08kuc)
- form unitary matrix — nag_zungbr (f08ktc)
- reduce to bidiagonal form — nag_zgebrd (f08ksc)

real general matrix to upper Hessenberg form:

- apply orthogonal matrix — nag_dormhr (f08ngc)
- form orthogonal matrix — nag_dorghr (f08nfc)
- reduce to Hessenberg form — nag_dgehrd (f08nec)

real rectangular band matrix to upper bidiagonal form — nag_dgbbrd (f08lec)

real rectangular matrix to bidiagonal form:

- apply orthogonal matrix — nag_dormbr (f08kgc)
- form orthogonal matrix — nag_dorgbr (f08kfc)
- reduce to bidiagonal form — nag_dgebrd (f08kec)

real symmetric band matrix to symmetric tridiagonal form — nag_dsbtrd (f08hec)

real symmetric matrix to symmetric tridiagonal form:

- apply orthogonal matrix — nag_dormtr (f08fgc)
- form orthogonal matrix — nag_dorgtr (f08ffc)
- reduce to tridiagonal form — nag_dsytrd (f08fec)

Reduction of generalized eigenproblems to standard eigenproblems:

complex Hermitian-definite banded generalized eigenproblem $Ax = \lambda Bx$ — nag_zhbgst (f08usc)

complex Hermitian-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$ — nag_zhegst (f08ssc)

real symmetric-definite banded generalized eigenproblem $Ax = \lambda Bx$ — nag_dsbgst (f08uec)

real symmetric-definite generalized eigenproblem $Ax = \lambda Bx$, $ABx = \lambda x$ or $BAx = \lambda x$ — nag_dsygst (f08sec)

Solve reduced form of Sylvester matrix equation:

- complex matrices — nag_ztrsyl (f08qvc)
- real matrices — nag_dtrsyl (f08qhc)

Split Cholesky factorization:
complex Hermitian positive-definite band matrix ..... `nag_zpbstf (f08utc)`
real symmetric positive-definite band matrix ..... `nag_dpbstf (f08ufc)`
Transform eigenvectors of a pair of matrices
from complex balanced to those supplied to nag_zggbal (f08wvc) ..... `nag_zggbak (f08wwc)`
from real balanced to those supplied to nag_dggbal (f08whc) ..... `nag_dggbak (f08wjc)`

6 Functions 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 (1998) *The Symmetric Eigenvalue Problem* SIAM, Philadelphia

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

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