Chapter F08

Least-squares and Eigenvalue Problems (ScaLAPACK)

Contents

1 Scope of the Chapter

At this release, this chapter provides routines for the solution of linear least-squares and eigenvalue problems. It provides routines for:

QR factorization

Routines associated with the solution of linear least-squares problems

Eigenvalues and eigenvectors of real symmetric matrices

The routines in this chapter are derived from the ScaLAPACK project (see Blackford *et al.* [2]) and can handle **real** and **complex dense** matrices. They have been designed to be efficient on a wide range of parallel computers.

2 Background to the Problems

This section gives a brief introduction to the numerical solution of linear least-squares problems. Consult a standard textbook, for example Golub and Van Loan [3], for a more thorough discussion.

2.1 Linear Least-squares Problems

The **linear least-squares problem** is

$$\min_x \|b - Ax\|_2 \tag{1}$$

where A is an m by n matrix, b is a given m-element vector and x is the n-element solution vector. In the most usual case $m \geq n$ and $\mathrm{rank}(A) = n$, so that A has **full rank** and in this case the solution to the problem of (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 $\mathrm{rank}(A) = m$, 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 $\mathrm{rank}(A) < \min(m,n)$ – 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 contains computational routines that can be combined with routines 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

Two routines are provided for factorizing a general real or complex 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

$$\|x\|_2 = \|Qx\|_2.$$

They help to maintain numerical stability because they do not amplify rounding errors. Orthogonal factorizations are used in the solution of linear least-squares 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

$$A = Q\,(R_1 R_2)$$

where R_1 is upper triangular and R_2 is rectangular.

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

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

for real A, 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. Similarly,

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

for complex A, where

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

In either case, 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

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

2.3 Eigenvalue Problems

2.3.1 Symmetric Eigenvalue Problem

Let A be a real square symmetric matrix of order n. The symmetric eigenvalue problem is to find eigenvalues, λ, and corresponding eigenvectors, $z \neq 0$, such that

$$Az = \lambda z. \qquad (2)$$

The phrase 'eigenvalue problem' is sometimes abbreviated to **eigenproblem**.

The eigenvalues λ are all real, and the eigenvectors can be chosen to be mutually orthogonal. That is, we can write

$$Az_i = \lambda_i z_i \text{ for } i = 1, \ldots, n$$

or equivalently:

$$AZ = Z\Lambda \qquad (3)$$

where Λ is a real diagonal matrix whose diagonal elements λ_i are the eigenvalues, and Z is a real orthogonal matrix whose columns z_i are the eigenvectors. This implies that $z_i^T z_j = 0$ if $i \neq j$, and $\|z_i\|_2 = 1$ where z_i^T denotes the transpose of the vector z_i.

Equation (3) can be rewritten

$$A = Z\Lambda Z^T. \tag{4}$$

This is known as the **eigendecomposition** or **spectral factorization** of A.

Eigenvalues of a real symmetric matrix are well conditioned, that is, they are not unduly sensitive to perturbations in the original matrix A. The sensitivity of an eigenvector depends on how small the gap is between its eigenvalue and any other eigenvalue; the smaller the gap, the more sensitive the eigenvector.

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

(1) The real symmetric matrix A is reduced to **real tridiagonal form** T as the decomposition $A = QTQ^T$ with Q orthogonal and T symmetric tridiagonal.

(2) Eigenvalues 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 $A = Z\Lambda Z^T$.

2.3.2 Hermitian Eigenvalue Problem

The Hermitian eigenvalue problem is similar to the symmetric eigenvalue problem. In the Hermitian eigenvalue problem, the matrix A is complex but the eigenvalues of A are real. However, the eigenvectors z_i, $i = 1, \ldots, n$ are, in general, complex. The eigendecomposition is given by

$$A = Z\Lambda Z^H \tag{5}$$

where the matrix Z is now unitary. That is, $z_i^H z_j = 0$ if $i \neq j$, and $\|z_i\|_2 = 1$ where z_i^H represents the complex conjugate transpose of the vector z_i.

2.4 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. 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 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 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.

Finally, we remark on the accuracy of the bounds when they are large. Relative errors like $\|\hat{x} - x\|/\|x\|$ 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.

2.4.1 Least-squares problems

The conventional error analysis of linear least-squares problems goes as follows. The problem is to find the solution x minimizing $\|b - Ax\|_2$. Let $\hat{x}$ be the solution computed using the method 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. Let $\kappa_2(A) = \sigma_{\max}(A)/\sigma_{\min}(A)$, $\rho = \|b - Ax\|_2$, and $\sin(\theta) = \rho/\|b\|_2$, where $\sigma_i(A)$ denotes the ith singular value of A. 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 [3] 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

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

or

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

if A is complex. By solving this linear systems (see Chapter F07) componentwise error bounds can also be obtained (see Golub and Van Loan [3]).

2.4.2 The symmetric and Hermitian eigenproblem

The usual error analysis of the symmetric and Hermitian eigenproblem is as follows Koelbel *et al.* [7].

The computed eigendecomposition $\hat{Z}\hat{\Lambda}\hat{Z}$ is nearly the exact eigendecomposition of $A + E$, i.e., $A + E = (\hat{Z} + \delta\hat{Z})\hat{\Lambda}(\hat{Z} + \delta\hat{Z})^H$ 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. 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

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

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

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

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

$$\theta(\hat{S}, S) \lesssim \frac{p(n)\epsilon\|A\|_2}{\mathrm{gap}_I}$$

where

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

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

In the special case of a real symmetric tridiagonal matrix T, routines in this chapter can compute the eigenvalues and eigenvectors much more accurately. See Anderson *et al.*[1] for further details.

2.5 Block Algorithms

The routines in this chapter use what is termed a **block-partitioned algorithm**. This means that at each major step of the algorithm a **block** of rows or columns is updated, and most of the computation is performed by matrix–matrix operations on these blocks. Blocks are distributed among the participating processors in a cyclic two-dimensional **block** fashion (see Section 2.6). The matrix–matrix operations are performed by calls to the Level-3 PBLAS (Parallel BLAS) (see Blackford *et al.* [2]), which rely on the communication primitives provided by the Basic Linear Algebra Communication Subprograms or BLACS (Blackford *et al.* [2]). See Golub and Van Loan [3] or Anderson *et al.* [1] for more information about block-partitioned algorithms. The performance of a block-partitioned algorithm varies to some extent with the block size – that is, the number of rows or columns per block.

2.6 Usage

The routines in this chapter use a **cyclic two-dimensional block distribution** for all matrices and vectors, in order to try to minimise data movement. This distribution is such that row blocks of the matrix are distributed in wrapped fashion to the associated row of the logical grid of processors and, likewise, column blocks are distributed in wrapped fashion to the associated column of the logical grid of processors. Here the terms row block and column block refer to one or more contiguous rows or columns of a matrix which are treated as a single entity from the algorithmic point of view. For those familiar with High Performance Fortran (HPF) terminology this is equivalent to !HPF$ DISTRIBUTE CYCLIC(MB), CYCLIC(NB) where MB and NB are the row and column blocking factors of the matrix distribution. See Koelbel *et al.* [6].

	1	2	3	4	5	6	7	8	9	10	11	12
1	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
2	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
3	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
4	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
5	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
6	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
7	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
8	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
9	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
10	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}
11	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}	{0,0}	{0,1}	{0,2}
12	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}	{1,0}	{1,1}	{1,2}

Figure 1
Block distribution over a 2 by 3 logical grid of processors

Figure 1 shows how blocks of a matrix are distributed over a 2 by 3 logical grid of processors: the matrix has 12 column and 12 row blocks; the column and row indices in Figure 1 indicate row and column blocks, respectively. Each box contains the index of the processor storing that particular block. The shading is provided only as a visual aid to highlight the processor template and has no other meaning.

Figure 2 shows the same distribution from the processors' point of view. Each of the larger boxes in Figure 2 is labelled by the index of the processor which stores the blocks that the box contains. The indices of the rows and columns in Figure 2 denote the indices of the row and column blocks.

2.6.1 Reduction to tridiagonal form (the Symmetric Eigenvalue Problem)

The eigenproblem algorithm in this chapter reduces an n by n real symmetric matrix A to tridiagonal form T by an orthogonal similarity transformation Q

$$Q^T AQ = T.$$

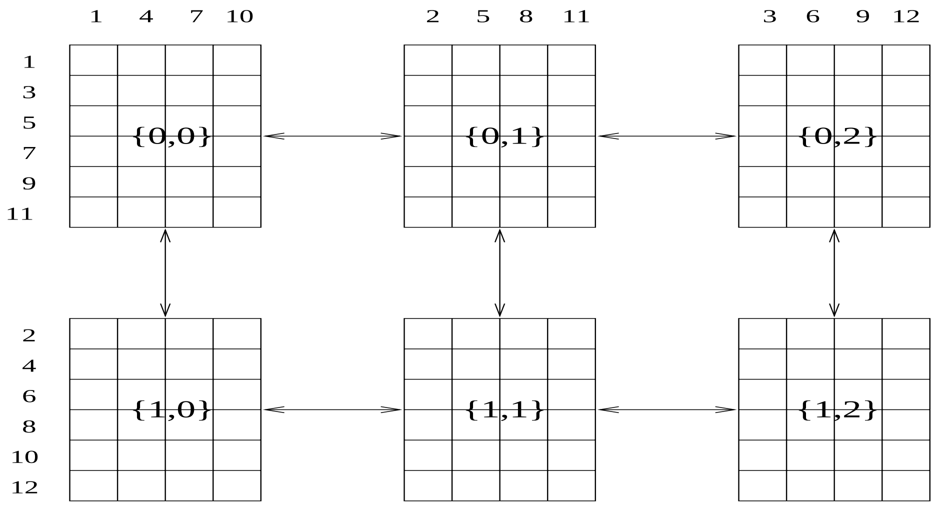

Figure 2
Data distribution from the processors' point of view

Since A is real symmetric, only the the upper triangular part or the lower triangular part is required.

The diagonal elements of the tridiagonal matrix T are represented by a vector d of length n and the off-diagonal elements by a vector e. On exit, the vector d is distributed in the cyclic one-dimensional block form across each logical processor row of the two-dimensional logical processor grid. The vector e is similarly distributed.

The orthogonal matrix Q is not formed explicitly but is represented as a product of $n-1$ elementary reflectors. If the data in the upper triangular part of the symmetric matrix A_s is used in the computation (i.e., the argument UPLO = 'U'), the matrix Q is represented as a product of elementary reflectors

$$Q = H_{n-1} \dots H_2 H_1.$$

Each H_i has the form

$$H_i = I - \tau_i v^{(i)} (v^{(i)})^T$$

where τ_i is a real scalar, and $v^{(i)}$ is a real vector with $v_j^{(i)} = 0$, $j = i+1, \dots, n$ and $v_i^{(i)} = 1$; $v_j^{(i)}$, $j = 1, \dots, i-1$ is stored on exit in A(1:$i-2$,$i+1$), and τ_i in TAU($i-1$). The contents of the array A on exit are illustrated by the following example with $m = 6$:

$$\begin{bmatrix} d_1 & e_1 & v_1^{(2)} & v_1^{(3)} & v_1^{(4)} & v_1^{(5)} \\ & d_2 & e_2^{(2)} & v_2^{(3)} & v_2^{(4)} & v_2^{(5)} \\ & & d_3 & e_3^{(3)} & v_3^{(4)} & v_3^{(5)} \\ & & & d_4 & e_4 & v_4^{(5)} \\ & & & & d_5 & e_5 \\ & & & & & d_6 \end{bmatrix}.$$

where τ_i is a real scalar, and $v^{(i)}$ is a real vector with $v_j^{(i)} = 0$, $j = 1, \dots, i$ and $v_{i+1}^{(i)} = 1$; $v_j^{(i)}$, $j = i+2, \dots, n$ is stored on exit in A($i+2:n, i$), and τ_i in TAU($i-1$).

In the lower triangular case (UPLO = 'L'), the contents of the array A on exit are in the form:

$$\begin{bmatrix} d_1 & & & & & \\ e_1 & d_2 & & & & \\ v_3^{(1)} & e_2 & d_3 & & & \\ v_4^{(1)} & v_4^{(2)} & e_3 & d_4 & & \\ v_5^{(1)} & v_5^{(2)} & v_5^{(3)} & e_4 & d_5 & \\ v_6^{(1)} & v_6^{(2)} & v_6^{(3)} & v_6^{(4)} & e_5 & d_6 \end{bmatrix}.$$

2.6.2 Reduction to triangular form (the Hermitian Eigenvalue Problem)

This is identical to that of the Symmetric Eigenvalue Problem except that the transpose operation $(.)^T$ is now replaced by the complex conjugate operation $(.)^H$.

Note that the tridiagonal matrix is real symmetric.

2.7 References

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

[2] Blackford L S, Choi J, Cleary A, D'Azevedo E, Demmel J, Dhillon I, Dongarra J, Hammarling S, Henry G, Petitet A, Stanley K, Walker D and Whaley R C (1997) ScaLAPACK Users' Guide *SIAM* 3600 University City Science Center, Philadelpia, PA 19104-2688, USA. URL: http://www.netlib.org/scalapack/slug/scalapack_slug.html

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

[4] Gropp W, Lusk E and Skjellum A (1999) *Using MPI: Portable Programming with the Message-passing Interface* The MIT Press, Cambridge, MA, USA (2nd Edition)

[5] Hestenes M R (1958) Inversion of matrices by biorthogonalization and related results *J. SIAM* **6** 51–90

[6] Koelbel C H, Loveman D B, Schreiber R S, Steele Jr. G L, and Zosel M E (1994) *The High Performance Fortran Handbook* The MIT Press, Cambridge, MA, USA

[7] Parlett B N (1980) *The Symmetric Eigenvalue Problem* Prentice–Hall

[8] Snir M, Otto S W, Huss-Lederman S, Walker D W and Dongarra J J (1998) *MPI - The Complete Reference: Volume 1 - The MPI Core* The MIT Press, Cambridge, MA, USA (2nd Edition)

[9] Wilkinson J H (1965) *The Algebraic Eigenvalue Problem* Oxford University Press, London

3 Recommendations on Choice and Use of Available Routines

3.1 Available Routines

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

The tables in Section 3.4 shows the routines which are provided in this chapter (Chapter F08). Each entry in the table gives the NAG name and the ScaLAPACK double precision name.

3.1.1 *QR* factorization

Routines are provided to perform the following computations:

(a) QR factorization
(b) Generation of the orthogonal or unitary matrix Q after the QR factorization
(c) Application of the orthogonal or unitary matrix Q to a general matrix without forming Q explicitly

To solve a least-squares problem, first the QR factorization is performed, then the right-hand side is transformed by applying Q^T or Q^H from the left; finally the resulting system of triangular equations can be solved using the PBLAS (Parallel BLAS) routine PDTRSM or PZTRSM (see Blackford *et al.* [2]).

3.1.2 The Symmetric Eigenvalue Problem

Routines are provided to perform the following computations:

(a) Reduction of a real symmetric matrix to a real symmetric tridiagonal form.
(b) Computation of eigenvalues of a real symmetric tridiagonal matrix.
(c) Computation of eigenvectors of a real symmetric tridiagonal matrix.
(d) Transformation of the eigenvectors of the tridiagonal matrix to the original matrix in step (a).
(e) Multiplication of a real matrix by a real orthogonal matrix.

3.1.3 The Hermitian Eigenvalue Problem

Routines are provided to perform the following computations:

(a) Reduction of a complex Hermitian matrix to a real symmetric tridiagonal form.
(b) Computation of eigenvalues of a real symmetric tridiagonal matrix.
(c) Computation of eigenvectors of a real symmetric tridiagonal matrix.
(d) Transformation of the eigenvectors of the tridiagonal matrix to the original matrix in step (a).
(e) Multiplication of a complex matrix by a complex unitary matrix.

3.2 NAG Names and ScaLAPACK Names

As well as the NAG routine name (beginning F08), the table in Section 3.4 shows the ScaLAPACK double precision routine names. The routines may be called either by their NAG names or by their ScaLAPACK names. References to Chapter F08 routines in the Manual normally include the ScaLAPACK double precision names, for example, F08AEFP (PDGEQRF).

The ScaLAPACK routine names follow a simple scheme. Each name has the structure **PXYYZZZ**, where the components have various meanings. With respect to the routines in this chapter the components have the following meanings:

X the second letter indicates the data type (real or complex) and precision

D real, double precision (in Fortran, `DOUBLE PRECISION`)

Z complex, double precision (in Fortran, `COMPLEX*16`)

(for real and complex, single precision, S and C respectively)

YY the third and fourth letters indicate the type of the matrix A, for example:

GE general

OR orthogonal

UN unitary

ZZZ the last three letters indicate the computation performed, for example:

QRF QR factorization

GQR generate the orthogonal or unitary matrix Q after the QR factorization

MQR apply the orthogonal or unitary matrix Q to a general matrix without forming Q explicitly

Thus PDGEQRF performs the QR factorization of a real general matrix.

3.3 Parameters Conventions

3.3.1 Option parameters

Most routines in this chapter have one or more option parameters, of type CHARACTER. The descriptions in Section 4 of the routine documents refer only to upper-case values (for example 'N' or 'T'); however in every case, the corresponding lower-case characters may be supplied (with the same meaning). Any other value is illegal.

A longer character string can be passed as the actual parameter, making the calling program more readable, but only the first character is significant. For example:

```
        CALL F08AGFP ('left', 'transpose', . . . )
```

3.3.2 Problem dimensions

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

3.3.3 Matrix data

In all routines in this chapter the local elements of a matrix are stored in a one-dimensional array. For example, the local elements of the m_A by n_A matrix A can be stored in the real array A($*$). However, it is more convenient to consider A as a two-dimensional array of dimension (LDA,$*$), where LDA must be greater than or equal to the number of rows of A stored in the specific row of the processor grid and the array A must have a number of columns greater than or equal to the number of columns of A stored in the specific column of the processor grid. Further information about the distribution of the matrix over the participating processors are encapsulated in an integer array called an **array descriptor**. Such a descriptor is associated with each distributed matrix. The entries of the descriptor uniquely determine the mapping of the matrix entries onto local processors' memories. Moreover, with the exception of the local leading dimension and the Library context, the descriptor array elements are global, characterising the distributed matrix. As an example, in the QR factorization and the associated routines, a descriptor array of dimension (9), say IDESCA, would store the following information:

IDESCA(1)	descriptor type: for cyclic two-dimensional block distribution this must be set to 1;
IDESCA(2)	the Library context, usually returned by a call to the Library Grid initialisation routine Z01AAFP;
IDESCA(3)	m_A, the number of rows of A;
IDESCA(4)	n_A, the number of columns of A;
IDESCA(5)	M_b, the blocking factor used to distribute the rows of A, i.e., the number of rows stored in a block;
IDESCA(6)	N_b, the blocking factor used to distribute the columns of A, i.e., the number of columns stored in a block;
IDESCA(7)	the processor row index over which the first row of A is distributed;
IDESCA(8)	the processor column index over which the first column of A is distributed;
IDESCA(9)	the leading dimension (LDA) of the local array A storing the local blocks of A.

In general, the descriptor array **does not change** throughout the life cycle of the matrix with which it is associated.

It is possible to reference an m by n **submatrix** of A, for example $A(i_A : m + i_A - 1, j_A : n + j_A - 1)$, by specifying the four parameters IA = i_A, JA = j_A, M = m and N = n, in the interface of the routine called.

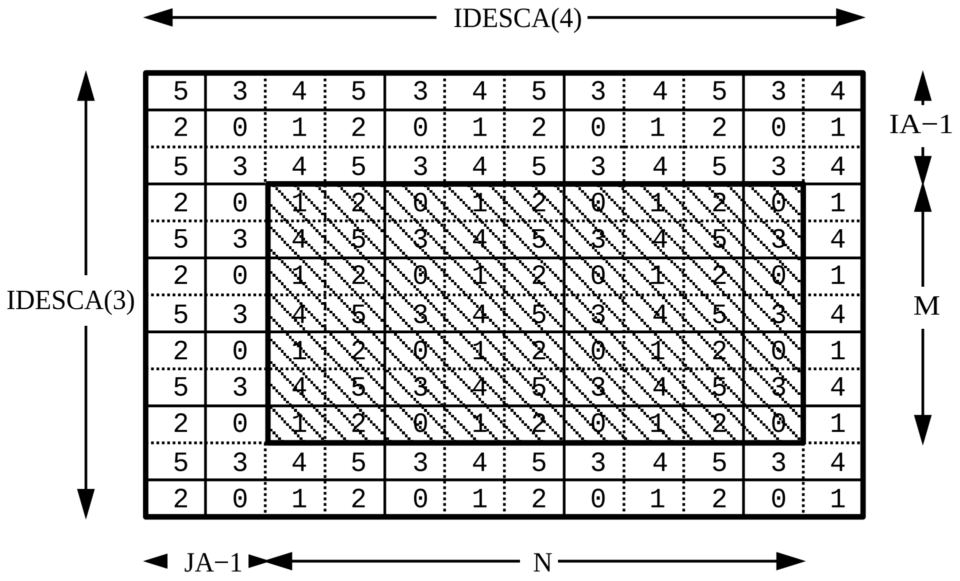

Figure 3
Referencing a submatrix

Figure 3 illustrates graphically the meaning and use of the parameters M, N, IA and JA and of some of the entries of the array descriptor IDESCA. The case depicted shows a matrix distributed over a 2 × 3 logical grid of processes. The first row of the matrix is stored in the second row of the grid (IDESCA(7) = 1), and the first column is stored in the third column of the grid (IDESCA(8) = 2). The index within each block refers to the processor (numbered from 0 to 5) which stores that block.

If the **whole** matrix is referenced, then, obviously, IA = 1, JA = 1 M = IDESCA(3) = m_A and N = IDESCA(4) = n_A.

In general, submatrices must start on the boundary between blocks, i.e., mod(IA−1,M_b) = 0 and mod(JA−1,N_b) = 0. Exceptions to this rule are described in the documents for the individual routines.

3.3.4 Error-handling and the diagnostic parameter INFO

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

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

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

INFO = 0 indicates successful termination;

INFO = $-(i*100+j)$ indicates an error in the jth component of the ith argument (for example, a component of an array descriptor);

INFO = $-i$ indicates an error in the ith argument;

INFO > 0 indicates an error detected during execution.

It is **essential** to test INFO on exit from the routine (this corresponds to a **soft failure** in terms of the usual error-handling terminology used for the rest of the Library), both for argument errors and errors detected during execution.

If INFO $\neq 0$ explanatory error messages are output from the root processor (or processor $\{0,0\}$ when the root processor is not available) on the current error message unit (as defined by X04AAF).

It should also be noted that calling routine Z02EAFP to reduce the amount of error checking does not disable all the argument checking in F08 routines. Some global argument checks will be omitted, but for compatibility with ScaLAPACK, the checks performed by ScaLAPACK are retained (see Blackford *et al.* [2] for details).

3.4 Table of Available Routines

3.4.1 *QR* factorization

	Factorize	**Generate matrix Q**	**Apply matrix Q**
QR factorization, real matrices	F08AEFP PDGEQRF	F08AFFP PDORGQR	F08AGFP PDORMQR
QR factorization, complex matrices	F08ASFP PZGEQRF	F08ATFP PZUNGQR	F08AUFP PZUNMQR

3.4.2 The Symmetric Eigenvalue Problem (SEP)

	Tridiagonalize	**Compute eigenvalues of tridiagonal matrix**	**Compute eigenvectors of tridiagonal matrix**	**Compute eigenvectors of original matrix**
SEP	F08FEFP PDSYTRD	F08JJP PDSTFEBZ	F08JKP PDSTFEIN	F08FGFP PDORMTR

3.4.3 The Hermitian Eigenvalue Problem (HEP)

	Tridiagonalize	**Compute eigenvalues of tridiagonal matrix**	**Compute eigenvectors of tridiagonal matrix**	**Compute eigenvectors of original matrix**
HEP	F08FSFP	F08JJFP	F08JXFP	F08FUFP
	PZHETRD	PDSTEBZ	PZSTEIN	PZUNMTR

Each entry gives

the NAG routine name

the double precision ScaLAPACK routine name.