Chapter F04

Simultaneous Linear Equations

Contents

1 Scope of the Chapter

This chapter is concerned with the solution of the matrix equation $AX = B$, where B may be a single vector or a matrix of multiple right-hand sides. The matrix A may be a general real or complex square matrix, or a real symmetric or complex Hermitian positive-definite matrix. It may also be real rectangular, in which case a least-squares solution is obtained.

2 Background to the Problems

A set of linear equations may be written in the form

$$Ax = b$$

where the known matrix A is of size n by n (n rows and columns), the known right-hand vector b has n components (n rows and one column), and the required solution vector x has n components (n rows and one column). There may also be r vectors $b^{(i)}$, $i = 1, 2, ..., r$ on the right-hand side and the equations may then be written as

$$AX = B,$$

the required matrix X having as its r columns the solutions of $Ax^{(i)} = b^{(i)}$, $i = 1, 2, ..., r$. The routines in this chapter deal with the latter case, but for clarity only the case $r = 1$ is discussed here.

The most common problem, the determination of the unique solution of $Ax = b$, occurs when A is not singular, that is rank$(A) = n$. This is discussed in Section 2.1. The next most common problem, discussed in Section 2.2 below, is the determination of the least-squares solution of $Ax \simeq b$ required when $m > n$ and rank(A) $= n$, i.e. the determination of the vector x which minimizes the norm of the residual vector $q = b - Ax$. These are currently the only problems addressed by this chapter. The routines in this chapter call routines from Chapter F07 and Chapter F08 to perform the underlying computations (see Chapters F07 and F08 for further information).

2.1 Unique Solution of $Ax = b$

The routines in this chapter solve this particular problem. The computation starts with the triangular decomposition $A = PLU$, where L and U are respectively lower and upper triangular matrices and P is a permutation matrix, chosen so as to ensure that the decomposition is numerically stable. The solution is then obtained by solving in succession the simpler equations

$$Ly = P^T b, \;\; Ux = y,$$

the first by forward-substitution and the second by back-substitution.

If A is real symmetric and positive-definite then $U = L^T$, whereas if A is complex Hermitian and positive-definite $U = L^H$; in both cases P is the identity matrix (i.e., no permutations are necessary). Otherwise L has unit diagonal elements.

Due to rounding errors the computed 'solution' x, is only an approximation to the true solution $\hat{x}$, say. This approximation will sometimes be satisfactory, agreeing with $\hat{x}$ to several figures, but if the problem is ill-conditioned then $\hat{x}$ and x may have few or even no figures in common, and at this stage there is no means of estimating the 'accuracy' of x.

There are various possible approaches to estimating the accuracy of a computed solution, including the use of **iterative refinement** and **condition estimation**. See Golub and Van Loan [3] and Anderson *et al.* [1] for further information.

An alternative way to obtain useful information about the accuracy of a solution is to solve two sets of equations, one with the given coefficients, which are assumed to be known with certainty to ℓ figures, and one with the coefficients rounded to $(\ell - 1)$ figures, and to count the number of figures to which the two solutions agree. In ill-conditioned problems this can be surprisingly small and even zero.

2.2 The Least-squares Solution of $Ax \simeq b, m > n, \text{rank}(A) = n$

The least-squares solution is the vector x which minimizes the sum of squares of the residuals

$$s = (b - Ax)^T (b - Ax) = \|b - Ax\|_2^2.$$

The solution is obtained in two steps:

(i) Householder Transformations are used to perform a QR factorization of the matrix A so that

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where R is a non-singular n by n upper triangular matrix and 0 is a zero matrix of shape $m - n$ by n. Similarly b is transformed as

$$Q^T b = c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix},$$

with c_1 having n rows and c_2 having $(m - n)$ rows.

(ii) The required least-squares solution is obtained by back-substitution in the equation

$$Rx = c_1.$$

Again due to rounding errors the computed solution $\hat{x}$ is only an approximation to x, and its accuracy can be estimated using the techniques mentioned in Section 2.1.

2.3 Calculating the Inverse of a Matrix

The routines in this chapter can also be used to calculate the inverse of a square matrix A by solving the equation

$$AX = I$$

where I is the identity matrix. However, solving the equations $AX = B$ by calculation of the inverse of the coefficient matrix A, i.e., by $X = A^{-1}B$, is **definitely not recommended**.

2.4 Data Distribution

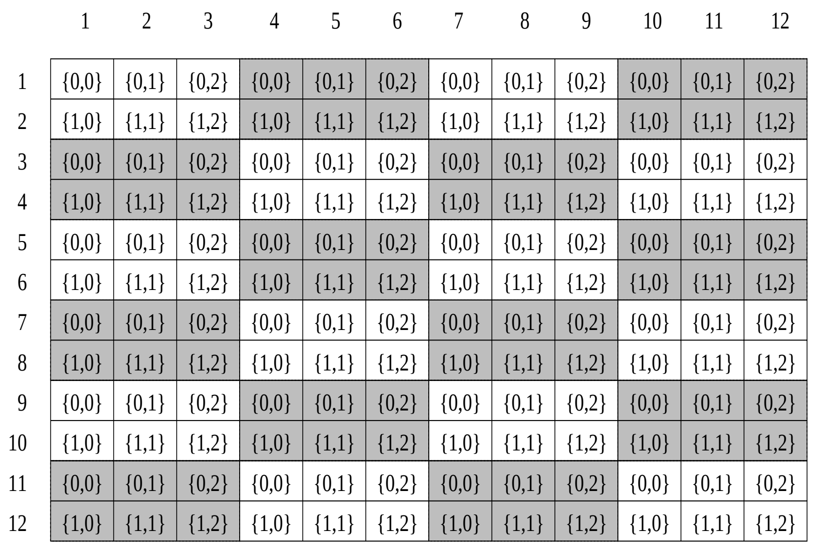
The routines in this chapter use a **cyclic 2-d block distribution** for the matrices, in order to try to minimize data movement. This distribution is such that row blocks of the matrix are distributed in cyclic fashion to the associated row of logical processors and column blocks are distributed in cyclic fashion to the associated column of logical 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 [4].

Figure 1 and Figure 2 illustrate the cyclic 2-d block distribution of a matrix consisting of twelve row and column blocks on a two by three grid. The row blocks (and the column blocks) of the matrix are numbered from 1 to 12.

2.5 References

[1] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A, Ostrouchov S and Sorensen D (1995) *LAPACK Users' Guide* (2nd Edition) SIAM, Philadelphia

[2] Choi J, Demmel J, Dhillon I, Dongarra J J, Ostrouchov S, Petitet A, Stanley K, Walker D W and Whaley R C (1995) ScaLAPACK: a portable linear algebra library for distributed memory computers–design issues and performance *LAPACK Working Note No. 95 (Technical Report CS-95-283)* Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA. Published as: Dongarra J J, Masden K and Waśniewski J (Ed), (Proceedings of the Second International Workshop, PARA '95, Lyngby, Denmark) *Applied Parallel Computing* Springer-Verlag, Berlin, Germany 95–106.
URL: http://www.netlib.org/lapack/lawns/lawn95.ps



	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.

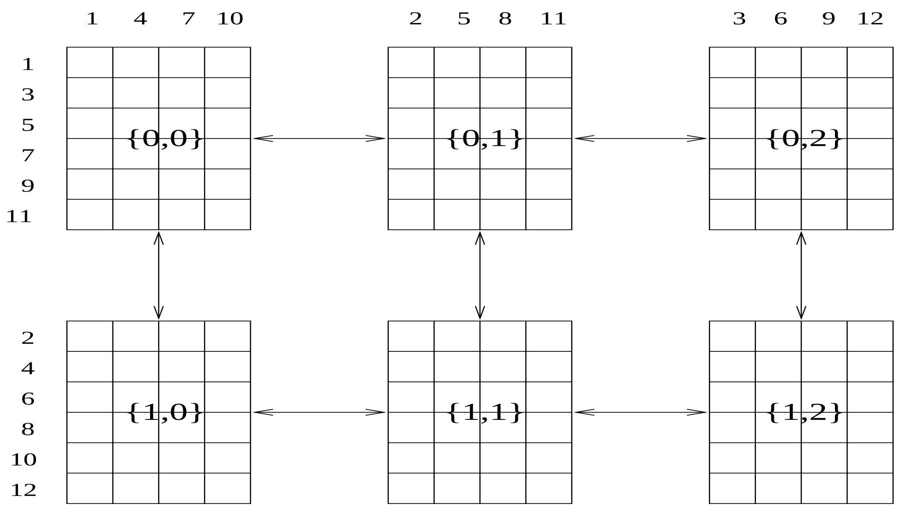


Figure 2
Data distribution from the processors' point of view.

[3] Golub G H and Van Loan C F (1989) *Matrix Computations* Johns Hopkins University Press (2nd Edition), Baltimore

[4] 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

[5] (1996) *ScaLAPACK Users Guide* (Draft), To be published. Department of Computer Science, University of Tennessee, 104 Ayres Hall, Knoxville, TN 37996-1301, USA.
URL: http://www.netlib.org/scalapack/scalapack_ug.ps

3 Recommendations on Choice and Use of Available Routines

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

3.1 Black Box and General Purpose Routines

The five routines in this chapter are categorised as Black Box routines, in that they solve the linear equations $Ax^{(i)} = b^{(i)}$, $i = 1, 2, ..., r$ in a single call with the matrix A and the right-hand sides b_i being supplied as data. These are the simplest routines to use for solving linear systems and are suitable when all the right-hand sides are known in advance, when the matrices A and B are distributed with the first block in the top left-hand processor of the Library Grid, and have the same blocking factor.

General purpose routines require a previous call to a routine to factorize the matrix A. This factorization can then be used repeatedly to solve the equations for one or more right-hand sides which may be generated in the course of the computation. The Black Box routines simply call a factorization routine and then a general purpose routine to solve the equations. There are currently no general purpose routines in this chapter, but such routines are available in Chapter F07, and these routines also allow a little more flexibility with the distribution of the matrices.

F04EBFP solves a real general system of linear equations and F04FBFP solves a system of real symmetric positive-definite linear equations. Similarly, F04ECFP solves a complex general system of linear equations and F04FCFP solves a system of complex Hermitian positive-definite linear equations. F04GBFP solves a real linear least-squares problem.

3.2 Data Distribution Routines

There are routines in Chapters F01 and X04 to aid in the distribution of data in the cyclic 2-d block format required by the routines in this chapter, and in the subsequent collection of the results.

For real matrices, F01ZSFP can be used to distribute a matrix, X04BGFP can be used to read data from an external file and X04BHFP can be used to output the results to an external file. For complex matrices, F01ZXFP can be used to distribute a matrix, X04BVFP can be used to read data from an external file and X04BWFP can be used to output the results to an external file.