Chapter F11

Sparse Linear Algebra

Contents

1 Scope of the Chapter

This chapter provides routines for the solution of large sparse systems of simultaneous linear equations. These include **iterative** methods for real nonsymmetric and symmetric linear systems. Some further direct methods are currently available in Chapter F01 and Chapter F04, and will be added to this chapter at future marks.

For a wider selection of routines for sparse linear algebra, users are referred to the Harwell Sparse Matrix Library (available from NAG), especially for direct methods for solving linear systems (see Section 2.2).

2 Background to the Problems

This section is only a brief introduction to the solution of sparse linear systems. For a more detailed discussion see for example Duff *et al.* [2] for direct methods, or Barrett *et al.* [1] for iterative methods.

2.1 Sparse Matrices and Their Storage

A matrix A may be described as **sparse** if the number of zero elements is sufficiently large that it is worthwhile using algorithms which avoid computations involving zero elements.

If A is sparse, and the chosen algorithm requires the matrix coefficients to be stored, a significant saving in storage can often be made by storing only non-zero elements. A number of different formats may be used to represent sparse matrices economically. These differ according to the amount of storage required, the amount of indirect addressing required for fundamental operations such as matrix–vector products, and their suitability for vector and/or parallel architectures. For a survey of some of these storage formats see Barrett *et al.* [1].

Some of the routines in this chapter have been designed to be independent of the matrix storage format. This allows the user to choose his or her own preferred format, or to avoid storing the matrix altogether. Other routines are black boxes, which are easier to use, but are based on fixed storage formats. Two such fixed formats are currently catered for. These are known as coordinate storage (CS) format, and symmetric coordinate storage (SCS) format.

2.1.1 Coordinate storage (CS) format

This storage format represents a sparse nonsymmetric matrix A, with NNZ non-zero elements, in terms of a real array A and two integer arrays IROW and ICOL. These arrays are all of rank 1 and of dimension at least NNZ. A contains the non-zero elements themselves, while IROW and ICOL store the corresponding row and column indices respectively.

For example, the matrix

$$A = \begin{pmatrix} 1 & 2 & -1 & -1 & -3 \\ 0 & -1 & 0 & 0 & -4 \\ 3 & 0 & 0 & 0 & 2 \\ 2 & 0 & 4 & 1 & 1 \\ -2 & 0 & 0 & 0 & 1 \end{pmatrix}$$

might be represented in the arrays A, IROW and ICOL as

A = (1,2,−1,−1,−3,−1,−4,3,2,2,4,1,1,−2,1)

IROW = (1,1,1,1,1,2,2,3,3,4,4,4,4,5,5)

ICOL = (1,2,3,4,5,2,5,1,5,1,3,4,5,1,5)

Notes

(i) The general format specifies no ordering of the array elements, but some routines may impose a specific ordering. For example, the non-zero elements may be required to be ordered by increasing row index and by increasing column index within each row, as in the example above. A utility routine is provided to order the elements appropriately.

(ii) With this storage format it is possible to enter duplicate elements. These may be interpreted in various ways (raising an error, ignoring all but the first entry, all but the last, or summing, for example).

2.1.2 Symmetric coordinate storage (SCS) format

This storage format is suitable for symmetric matrices, and is identical to the CS format described in Section 2.1.1, except that only the lower triangular non-zero elements are stored. Thus, for example, the matrix

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 & -1 & 2 \\ 1 & 5 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & -1 \\ 0 & 2 & 1 & 3 & 1 & 0 \\ -1 & 0 & 0 & 1 & 4 & 0 \\ 2 & 0 & -1 & 0 & 0 & 3 \end{pmatrix}$$

might be represented in the arrays A, IROW and ICOL as

A = (4,1,5,2,2,1,3,−1,1,4,2,−1,3)
IROW = (1,2,2,3,4,4,4,5,5,5,6,6,6)
ICOL = (1,1,2,3,2,3,4,1,4,5,1,3,6)

2.2 Direct Methods

Direct methods for the solution of the linear algebraic system

$$Ax = b \tag{1}$$

aim to determine the solution vector x in a fixed number of arithmetic operations, which is determined a priori by the number of unknowns. For example, an LU factorization of A followed by forward and backward substitution is a direct method for (1).

If the matrix A is sparse it is possible to design direct methods which exploit the sparsity pattern and are therefore much more computationally efficient than the algorithms in Chapter F07, which in general take no account of sparsity. However, if the matrix is very large and sparse, then **iterative** methods (see Section 2.3) are generally more efficient still.

This chapter currently provides direct methods for sparse real nonsymmetric, and symmetric positive-definite systems. Further direct methods may be found in Chapter F01, Chapter F04 and Chapter F07, and will be introduced into Chapter F11 at a future mark. For more details see Section 3.4.

2.3 Iterative Methods

In contrast to the direct methods discussed in Section 2.2 **iterative** methods for (1) approach the solution through a sequence of approximations, until some user-specified termination criterion is met or until some predefined maximum number of iterations has been carried out. The number of iterations required for convergence is not generally known in advance, as it depends on the accuracy required, and on the matrix A — its sparsity pattern, conditioning and eigenvalue spectrum.

Faster convergence can often be achieved using a **preconditioner** (Golub and Van Loan [3], Barrett *et al.* [1]). A preconditioner maps the original system of equations onto a different system

$$\bar{A}\bar{x} = \bar{b}, \tag{2}$$

which hopefully exhibits better convergence characteristics: for example, the condition number of the matrix $\bar{A}$ may be better than that of A, or it may have eigenvalues of greater multiplicity.

An unsuitable preconditioner or no preconditioning at all may result in a very slow rate or lack of convergence. However, preconditioning involves a trade-off between the reduction in the number of iterations required for convergence and the additional computational costs per iteration. Also, setting up a preconditioner may involve non-negligible overheads. The application of preconditioners to nonsymmetric and symmetric systems of equations is further considered in Section 2.4 and Section 2.5.

2.4 Iterative Methods for Nonsymmetric Linear Systems

Many of the most effective iterative methods for the solution of (1) lie in the class of non-stationary **Krylov subspace methods** [1]. For **nonsymmetric** matrices this class includes the restarted generalized minimum residual (RGMRES) method [8], conjugate gradient squared (CGS) method [10], and stabilized bi-conjugate gradient (Bi-CGSTAB) method [11], [9]. Here we just give a brief overview of these algorithms as implemented in Chapter F11. For full details see Section 3 of the document for F11BAF.

RGMRES is based on the Arnoldi method, which explicitly generates an orthogonal basis for the Krylov subspace span$\{A^k r_0\}$, $k = 0, 1, 2, \ldots$, where r_0 is the initial residual. The solution is then expanded onto the orthogonal basis so as to minimize the residual norm. For nonsymmetric matrices the generation of the basis requires a 'long' recurrence relation, resulting in prohibitive computational and storage costs. RGMRES limits these costs by restarting the Arnoldi process from the latest available residual every m iterations. The value of m is chosen in advance and is fixed throughout the computation. Unfortunately, an optimum value of m cannot easily be predicted.

CGS is a development of the bi-conjugate gradient method where the nonsymmetric Lanczos method is applied to reduce the coefficient matrix to real tridiagonal form: two bi-orthogonal sequences of vectors are generated starting from the initial residual r_0 and from the *shadow residual* $\hat{r}_0$ corresponding to the arbitrary problem $A^T\hat{x} = \hat{b}$, where $\hat{b}$ is chosen so that $r_0 = \hat{r}_0$. In the course of the iteration, the residual and shadow residual $r_i = P_i(A)r_0$ and $\hat{r}_i = P_i(A^T)\hat{r}_0$ are generated, where P_i is a polynomial of order i, and bi-orthogonality is exploited by computing the vector product $\rho_i = (\hat{r}_i, r_i) = (P_i(A^T)\hat{r}_0, P_i(A)r_0) = (\hat{r}_0, P_i^2(A)r_0)$. Applying the 'contraction' operator $P_i(A)$ twice, the iteration coefficients can still be recovered without advancing the solution of the shadow problem, which is of no interest. The CGS method often provides fast convergence; however, there is no reason why the contraction operator should also reduce the once reduced vector $P_i(A)r_0$: this can lead to a highly irregular convergence.

Bi-CGSTAB (ℓ)) is similar to the CGS method. However, instead of generating the sequence $\{P_i^2(A)r_0\}$, it generates the sequence $\{Q_i(A)P_i(A)r_0\}$ where the $Q_i(A)$ are polynomials chosen to minimize the residual *after* the application of the contraction operator $P_i(A)$. Two main steps can be identified for each iteration: an OR (Orthogonal Residuals) step where a basis of order ℓ is generated by a Bi-CG iteration and an MR (Minimum Residuals) step where the residual is minimized over the basis generated, by a method akin to GMRES. For $\ell = 1$, the method corresponds to the Bi-CGSTAB method of van der Vorst [11]. For $\ell > 1$, more information about complex eigenvalues of the iteration matrix can be taken into account, and this may lead to improved convergence and robustness. However, as ℓ increases, numerical instabilities may arise.

Faster convergence can usually be achieved by using a **preconditioner**. A *left* preconditioner M^{-1} can be used by the RGMRES and CGS methods, such that $\bar{A} = M^{-1}A \sim I_n$ in (2), where I_n is the identity matrix of order n; a *right* preconditioner M^{-1} can be used by the Bi-CGSTAB (ℓ) method, such that $\bar{A} = AM^{-1} \sim I_n$. These are formal definitions, used only in the design of the algorithms; in practice, only the means to compute the matrix–vector products $v = Au$ and $v = A^T u$ (the latter only being required when an estimate of $\|A\|_1$ or $\|A\|_\infty$ is computed internally), and to solve the preconditioning equations $Mv = u$ are required, that is, explicit information about M, or its inverse is not required at any stage.

Preconditioning matrices M are typically based on incomplete factorizations [6], or on the approximate inverses occurring in stationary iterative methods (see Young [12]). A common example is the **incomplete LU factorization**

$$M = PLDUQ = A - R$$

where L is lower triangular with unit diagonal elements, D is diagonal, U is upper triangular with unit diagonals, P and Q are permutation matrices, and R is a remainder matrix. A **zero-fill** incomplete LU factorization is one for which the matrix

$$S = P(L + D + U)Q$$

has the same pattern of non-zero entries as A. This is obtained by discarding any **fill** elements (non-zero elements of S arising during the factorization in locations where A has zero elements). Allowing some of these fill elements to be kept rather than discarded generally increases the accuracy of the factorization at the expense of some loss of sparsity. For further details see [1].

2.5 Iterative Methods for Symmetric Linear Systems

Two of the best known iterative methods applicable to symmetric linear systems are the conjugate gradient (CG) method [4], [3] and a Lanczos type method based on SYMMLQ [7].

For the CG method the matrix A should ideally be positive-definite. The application of CG to indefinite matrices may lead to failure, or to lack of convergence. The SYMMLQ method is suitable for both positive-definite and indefinite symmetric matrices. It is more robust than CG, but less efficient when A is positive-definite.

Both methods start from the residual $r_0 = b - Ax_0$, where x_0 is an initial estimate for the solution (often $x_0 = 0$), and generate an orthogonal basis for the Krylov subspace span$\{A^k r_0\}$, for $k = 0, 1, \ldots$, by means of three-term recurrence relations (Golub and Van Loan [3]). A sequence of symmetric tridiagonal matrices $\{T_k\}$ is also generated. Here and in the following, the index k denotes the iteration count. The resulting symmetric tridiagonal systems of equations are usually more easily solved than the original problem. A sequence of solution iterates $\{x_k\}$ is thus generated such that the sequence of the norms of the residuals $\{\|r_k\|\}$ converges to a required tolerance. Note that, in general, the convergence is not monotonic.

In exact arithmetic, after n iterations, this process is equivalent to an orthogonal reduction of A to symmetric tridiagonal form, $T_n = Q^T AQ$; the solution x_n would thus achieve exact convergence. In finite-precision arithmetic, cancellation and round-off errors accumulate causing loss of orthogonality. These methods must therefore be viewed as genuinely iterative methods, able to converge to a solution **within a prescribed tolerance**.

The orthogonal basis is not formed explicitly in either method. The basic difference between the two methods lies in the method of solution of the resulting symmetric tridiagonal systems of equations: the CG method is equivalent to carrying out an LDL^T (Cholesky) factorization whereas the Lanczos method (SYMMLQ) uses an LQ factorization.

A preconditioner for these methods must be **symmetric and positive-definite**, i.e., representable by $M = EE^T$, where M is non-singular, and such that $\bar{A} = E^{-1}AE^{-T} \sim I_n$ in (2), where I_n is the identity matrix of order n. These are formal definitions, used only in the design of the algorithms; in practice, only the means to compute the matrix-vector products $v = Au$ and to solve the preconditioning equations $Mv = u$ are required.

Preconditioning matrices M are typically based on incomplete factorizations [5], or on the approximate inverses occurring in stationary iterative methods (see Young [12]). A common example is the **incomplete Cholesky factorization**

$$M = PLDL^T P^T = A - R$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements, D is diagonal and R is a remainder matrix. A **zero-fill** incomplete Cholesky factorization is one for which the matrix

$$S = P(L + D + L^T)P^T$$

has the same pattern of non-zero entries as A. This is obtained by discarding any **fill** elements (non-zero elements of S arising during the factorization in locations where A has zero elements). Allowing some of these fill elements to be kept rather than discarded generally increases the accuracy of the factorization at the expense of some loss of sparsity. For further details see [1].

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 Types of Routine Available

The routines available in this chapter divide essentially into three types: basic routines, utility routines and black boxes.

Basic routines are grouped in suites of three, and implement the underlying iterative method. Each suite comprises a set-up routine, a solver, and a routine to return additional information. The solver routine is independent of the matrix storage format (indeed the matrix need not be stored at all) and the

type of preconditioner. It uses **reverse communication**, i.e., it returns repeatedly to the calling program with the parameter IREVCM set to specified values which require the calling program to carry out a specific task (either to compute a matrix-vector product or to solve the preconditioning equation), to signal the completion of the computation or to allow the calling program to monitor the solution. Reverse communication has the following advantages.

(i) Maximum flexibility in the representation and storage of sparse matrices. All matrix operations are carried out outside the solver routine, thereby avoiding the need for a complicated interface with enough flexibility to cope with all types of storage schemes and sparsity patterns. This applies also to preconditioners.
(ii) Enhanced user interaction: the progress of the solution can be closely monitored by the user and tidy or immediate termination can be requested. This is useful, for example, when alternative termination criteria are to be employed or in case of failure of the external routines used to perform matrix operations.

At present there are two suites of basic routines, for real symmetric, and for real nonsymmetric systems, respectively.

Utility routines perform such tasks as initializing the preconditioning matrix M, solving linear systems involving M, or computing matrix-vector products, for particular preconditioners and matrix storage formats. Used in combination basic routines and utility routines therefore provide iterative methods with a considerable degree of flexibility; allowing the user to select from different termination criteria, monitor the approximate solution, and compute various diagnostic parameters. The tasks of computing the matrix-vector products and dealing with the preconditioner are removed from the user, but at the expense of sacrificing some flexibility in the choice of preconditioner and matrix storage format.

Black box routines call basic and utility routines in order to provide easy-to-use routines for particular preconditioners and sparse matrix storage formats. They are much less flexible than the basic routines, but do not use reverse communication, and may be suitable in many simple cases.

The structure of this chapter has been designed to cater for as many types of application as possible. If a black box exists which is suitable for a given application you are recommended to use it. If you then decide you need some additional flexibility it is easy to achieve this by using basic and utility routines which reproduce the algorithm used in the black box, but allow more access to algorithmic control parameters and monitoring. If you wish to use a preconditioner or storage format for which no utility routines are provided, you must call basic routines, and provide your own utility routines.

3.2 Iterative Methods for Nonsymmetric Linear Systems

The suite of basic routines F11BAF, F11BBF and F11BCF implements either RGMRES, CGS, or Bi-CGSTAB(ℓ), for the iterative solution of the sparse nonsymmetric linear system $Ax = b$. These routines allow a choice of termination criteria and the norms used in them, allow monitoring of the approximate solution, and can return estimates of the norm of A and the largest singular value of the preconditioned matrix $\bar{A}$.

In general, it is not possible to recommend one of these methods in preference to another. RGMRES is popular, but requires the most storage, and can easily stagnate when the size m of the orthogonal basis is too small, or the preconditioner is not good enough. CGS can be the fastest method, but the computed residuals can exhibit instability which may greatly affect the convergence and quality of the solution. Bi-CGSTAB(ℓ) seems robust and reliable, but it can be slower than the other methods. Some further discussion of the relative merits of these methods can be found in [1].

The utility routines provided for nonsymmetric matrices use the co-ordinate storage (CS) format described in Section 2.1.1. F11DAF computes a preconditioning matrix based on incomplete LU factorization, and F11DBF solves linear systems involving the preconditioner generated by F11DAF. The amount of fill-in occurring in the incomplete factorization can be controlled by specifying either the level of fill, or the drop tolerance. Partial or complete pivoting may optionally be employed, and the factorization can be modified to preserve row-sums.

F11DDF is similar to F11DBF, but solves linear systems involving the preconditioner corresponding to symmetric successive-over-relaxation (SSOR). The value of the relaxation parameter ω must currently be supplied by the user. Automatic procedures for choosing ω will be included in the chapter at a future mark.

F11XAF computes matrix-vector products for nonsymmetric matrices stored in ordered CS format. An additional utility routine F11ZAF orders the non-zero elements of a sparse nonsymmetric matrix stored in general CS format.

The black box routine F11DCF makes calls to F11BAF, F11BBF, F11BCF, F11DBF and F11XAF, to solve a sparse nonsymmetric linear system, represented in CS format, using RGMRES, CGS, or Bi-CGSTAB(ℓ), with incomplete LU preconditioning. F11DEF is similar, but has options for no preconditioning, Jacobi preconditioning or SSOR preconditioning.

3.3 Iterative Methods for Symmetric Linear Systems

The suite of basic routines F11GAF, F11GBF and F11GCF implement either the conjugate gradient (CG) method, or a Lanczos method based on SYMMLQ, for the iterative solution of the sparse symmetric linear system $Ax = b$. If A is known to be positive-definite the CG method should be chosen; the Lanczos method is more robust but less efficient for positive-definite matrices. These routines allow a choice of termination criteria and the norms used in them, allow monitoring of the approximate solution, and can return estimates of the norm of A and the largest singular value of the preconditioned matrix $\bar{A}$.

The utility routines provided for symmetric matrices use the symmetric co-ordinate storage (SCS) format described in Section 2.1.2. F11JAF computes a preconditioning matrix based on incomplete Cholesky factorization, and F11JBF solves linear systems involving the preconditioner generated by F11JAF. The amount of fill-in occurring in the incomplete factorization can be controlled by specifying either the level of fill, or the drop tolerance. Diagonal Markowitz pivoting may optionally be employed, and the factorization can be modified to preserve row-sums.

F11JDF is similar to F11JBF, but solves linear systems involving the preconditioner corresponding to symmetric successive-over-relaxation (SSOR). The value of the relaxation parameter ω must currently be supplied by the user. Automatic procedures for choosing ω will be included in the chapter at a future mark.

F11XEF computes matrix-vector products for symmetric matrices stored in ordered SCS format. An additional utility routine F11ZBF orders the non-zero elements of a sparse symmetric matrix stored in general SCS format.

The black box routine F11JCF makes calls to F11GAF, F11GBF, F11GCF, F11JBF and F11XEF, to solve a sparse symmetric linear system, represented in SCS format, using a conjugate gradient or Lanczos method, with incomplete Cholesky preconditioning. F11JEF is similar, but has options for no preconditioning, Jacobi preconditioning or SSOR preconditioning.

3.4 Direct Methods

Chapter F11 does not currently provide any routines **specifically** designed for direct solution of sparse linear systems. However, the arguments of F11DAF and F11DBF may be chosen in such a way as to produce a direct solution.

The routine F11DBF solves a linear system involving the incomplete LU preconditioning matrix

$$M = PLDUQ = A - R$$

generated by F11DAF, where P and Q are permutation matrices, L is lower triangular with unit diagonal elements, U is upper triangular with unit diagonal elements, D is diagonal and R is a remainder matrix.

If A is non-singular, a call to F11DAF with LFILL < 0 and DTOL $= 0.0$ results in a zero remainder matrix R and a **complete** factorization. A subsequent call to F11DBF will therefore result in a direct method for real sparse nonsymmetric systems.

If A is known to be symmetric positive-definite, F11JAF and F11JBF may similarly be used to give a direct solution. For further details see Section 8.4 of the document for F11JAF.

Routines specifically designed for direct solution of sparse linear systems can currently be found in Chapter F01, Chapter F04 and Chapter F07. In particular, the following routines allow the direct solution of nonsymmetric systems:

Band	F07BDF and F07BEF
Almost block-diagonal	F01LHF and F04LHF
Tridiagonal	F01LEF and F04LEF or F04EAF
Sparse	F01BRF (or F01BSF) and F04AXF

and the following routines allow the direct solution of symmetric positive-definite systems:

Band	F07HDF and F07HEF
Variable band (skyline)	F01MCF and F04MCF
Tridiagonal	F04FAF

4 Index

5 References

[1] Barrett R, Berry M, Chan T F, Demmel J, Donato J, Dongarra J, Eijkhout V, Pozo R, Romine C and van der Vorst H (1994) *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* SIAM, Philadelphia

[2] Duff I S, Erisman A M, Reid J K (1986) *Direct Methods for Sparse Matrices* Oxford University Press, London

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

[4] Hestenes M and Stiefel E (1952) Methods of conjugate gradients for solving linear systems *J. Res. Nat. Bur. Stand.* **49** 409–436

[5] Meijerink J and van der Vorst H (1977) An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix *Math. Comput.* **31** 148–162

[6] Meijerink J and van der Vorst H (1981) Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems *J. Comput. Phys.* **44** 134–155

[7] Paige C C and Saunders M A (1975) Solution of sparse indefinite systems of linear equations *SIAM J. Numer. Anal.* **12** 617–629

[8] Saad Y and Schultz M (1986) GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **7** 856–869

[9] Sleijpen G L G and Fokkema D R (1993) BiCGSTAB(ℓ) for linear equations involving matrices with complex spectrum *ETNA* **1** 11–32

[10] Sonneveld P (1989) CGS, a fast Lanczos-type solver for nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **10** 36–52

[11] van der Vorst H (1989) Bi–CGSTAB, A fast and smoothly converging variant of Bi–CG for the solution of nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **13** 631–644

[12] Young D (1971) *Iterative Solution of Large Linear Systems* Academic Press, New York