Chapter f11 – Sparse Linear Algebra

1. Scope of the Chapter

This chapter provides routines for the iterative solution of large sparse nonsymmetric and symmetric systems of simultaneous linear equations.

2. Background

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.* (1986) for direct methods, or Barrett *et al.* (1994) 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.* (1994).

Only two storage schemes are used by the routines in this chapter. These are coordinate storage (CS) format and symmetric coordinate storage (SCS) format. These formats specify 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 in Section 2.1.1. below. A utility routine is provided to order the elements appropriately. With these storage formats 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).

2.1.1. 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

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

might be represented in the arrays **a**, **irow** and **icol** as

$$\mathbf{a} = (1, 2, -1, -1, -3, -1, -4, 3, 2, 2, 4, 1, 1, -2, 1)$$

$$\mathbf{irow} = (1, 1, 1, 1, 1, 2, 2, 3, 3, 4, 4, 4, 4, 5, 5)$$

$$\mathbf{icol} = (1, 2, 3, 4, 5, 2, 5, 1, 5, 1, 3, 4, 5, 1, 5)$$

2.1.2. Symmetric Coordinate Storage (SCS) Format

This storage format represents a sparse symmetric matrix A, with **nnz** non-zero lower triangular 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 lower triangular elements themselves, while **irow** and **icol** store the corresponding row and column indices respectively.

For example, the matrix

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

might be represented in the arrays **a**, **irow** and **icol** as

$$\mathbf{a} = (4, 1, 5, 2, 2, 1, 3, -1, 1, 4, 2, -1, 3)$$

$$\mathbf{irow} = (1, 2, 2, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6)$$

$$\mathbf{icol} = (1, 1, 2, 3, 2, 3, 4, 1, 4, 5, 1, 3, 6)$$

2.2. Iterative Methods

Iterative methods for the solution of the linear algebraic system

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

aim to determine the solution vector x 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 (1989), Barrett *et al.* (1994)). 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 2.3 and 2.4.

2.3. 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* (Barrett *et al.* (1994)). For *nonsymmetric* matrices this class includes the restarted generalized minimum residual (RGMRES) method (Paige and Saunders (1975)), conjugate gradient squared (CGS) method (Sonneveld (1989)), and stabilized bi-conjugate gradient (Bi-CGSTAB) method (van der Vorst (1989), and Sleijpen and Fokkema (1993)). Here we just give a brief overview of these algorithms as implemented in this chapter.

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 (van der Vorst (1989)). 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 $v = A^Tu$ (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 (Meijerink and van der Vorst (1981)), or on the approximate inverses occurring in stationary iterative methods (Young (1971)). 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 (Barrett *et al.* (1994)).

2.4. Iterative Methods for Symmetric Linear Systems

Two of the best known iterative methods applicable to symmetric linear systems are the conjugate gradient (CG) method (Hestenes and Stiefel (1952), and Golub and Van Loan (1989)) and a Lanczos type method based on SYMMLQ (Paige and Saunders (1975)).

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 (1989)). 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 (Meijerink J and van der Vorst H (1977)), or on the approximate inverses occurring in stationary iterative methods (Young (1971)). 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 Barrett *et al.* (1994).

2.5. Routines for Nonsymmetric Linear Systems

All routines for nonsymmetric linear systems in this chapter use the coordinate storage (CS) format described in Section 2.1.1.

In general it is not possible to recommend one of these methods in preference to another. RMGRES 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 Barrett *et al.* (1994).

Routine nag_sparse_nsym_fac (f11dac) computes a preconditioning matrix based on incomplete LU factorisation. The amount of fill-in occurring in the incomplete factorisation can be controlled by specifying either the level of fill or the drop tolerance. Partial or complete pivoting may optionally be employed and the factorisation can be modified to preserve row-sums.

Routine nag_sparse_nsym_fac_sol (f11dcc) uses the incomplete preconditioning matrix generated by nag_sparse_nsym_fac (f11dac) to solve a sparse nonsymmetric linear system, using RMGRES, CGS, or Bi-CGSTAB(ℓ).

Routine nag_sparse_nsym_sol (f11dec) is similar to nag_sparse_nsym_fac_sol (f11dcc) but has options for no preconditioning, SSOR preconditioning or Jacobi preconditioning.

The utility routine nag_sparse_nsym_sort (f11zac) orders the non-zero elements of a sparse nonsymmetric matrix stored in general CS format.

2.6. Routines for Symmetric Linear Systems

All routines for symmetric linear systems in this chapter use the symmetric coordinate storage (SCS) format described in Section 2.1.2.

Routine nag_sparse_sym_chol_fac (f11jac) computes a preconditioning matrix based on incomplete Cholesky factorisation. The amount of fill-in occurring in the incomplete factorisation can be controlled by specifying either the level of fill or the drop tolerance. Partial or complete pivoting may optionally be employed and the factorisation can be modified to preserve row-sums.

Routine nag_sparse_sym_chol_sol (f11jcc) uses the incomplete preconditioning matrix generated by nag_sparse_sym_chol_fac (f11jac) to solve a sparse symmetric linear system, using a conjugate gradient or Lanczos method.

Routine nag_sparse_sym_sol (f11jec) is similar to nag_sparse_nsym_sol (f11dec) but has options for no preconditioning, SSOR preconditioning or Jacobi preconditioning.

The utility routine nag_sparse_sym_sort (f11zbc) orders the non-zero elements of a symmetric matrix stored in general SCS format.

3. References

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.

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

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

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

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.

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

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.

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

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

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.

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

4. Available Functions

Routines for nonsymmetric linear systems:

RGMRES, CGS or Bi-CGSTAB(ℓ) solver with incomplete *LU* preconditioning	`f11dcc`
RGMRES, CGS or Bi-CGSTAB(ℓ) solver with Jacobi, SSOR, or no preconditioning	`f11dec`
Incomplete *LU* factorization	`f11dac`
Sort routine for nonsymmetric matrices in CS format	`f11zac`

Routines for symmetric linear systems:

CG or SYMMLQ solver with incomplete Cholesky preconditioning	`f11jcc`
CG or SYMMLQ solver with Jacobi, SSOR, or no preconditioning	`f11jec`
Incomplete Cholesky factorization	`f11jac`
Sort routine for symmetric matrices in SCS format	`f11zbc`