Chapter 5
Linear Equations

1 Scope of the Chapter

This chapter provides procedures for solving systems of linear equations.

Separate modules are provided to handle systems with special structure; this offers possibilities for greater efficiency, more economical storage and increased reliability.

All the procedures in this chapter are generic procedures which can handle either real or complex data.

2 Available Modules

Module 5.1: `nag_gen_lin_sys` — **General systems of linear equations**

Provides procedures for:

- solving a system of linear equations with one or many right-hand sides $Ax = b$ or $AX = B$, where the coefficient matrix A is a *general* square matrix;
- computing an *LU* factorization of a general square matrix and solving a system of equations, using a previously computed factorization. (These are lower-level procedures, intended for more experienced users.)

Optional facilities are provided in these procedures for evaluating the determinant of A, estimating the condition number of A and computing error bounds on the solution.

Module 5.2: `nag_sym_lin_sys` — **Symmetric systems of linear equations**

Provides procedures for:

- solving a system of linear equations with one or many right-hand sides $Ax = b$ or $AX = B$, where the coefficient matrix A is *real symmetric*, *complex symmetric* or *complex Hermitian.* If A is known to be *positive definite*, this is treated as a special case;
- computing a Bunch–Kaufman factorization of a symmetric or Hermitian matrix, or a Cholesky factorization of a positive definite matrix, and solving a system of equations, using a previously computed factorization. (These are lower-level procedures, intended for more experienced users.)

Optional facilities are provided in these procedures for evaluating the determinant of A, estimating the condition number of A and computing error bounds on the solution.

Module 5.3: `nag_tri_lin_sys` — **Triangular systems of linear equations**

Provides procedures for:

- solving a system of linear equations with one or many right-hand sides $Ax = b$ or $AX = B$, where the coefficient matrix A is upper or lower *triangular*, with optional facilities for computing error bounds on the solution;
- estimating the condition number of an upper or lower triangular matrix.
- evaluating the determinant of an upper or lower triangular matrix.

Module 5.4: `nag_gen_bnd_lin_sys` — **General banded systems of linear equations**

Provides procedures for:

- solving a system of linear equations with one or many right-hand sides $Ax = b$ or $AX = B$, where the coefficient matrix A is a *general* square *banded* matrix;
- computing an LU factorization of a general square band matrix and solving a system of equations, using a previously computed factorization. (These are lower-level procedures, intended for more experienced users.)

Optional facilities are provided in these procedures for evaluating the determinant of A, estimating the condition number of A and computing error bounds on the solution.

Module 5.5: `nag_sym_bnd_lin_sys` — **Symmetric positive definite banded systems of linear equations**

Provides procedures for:

- solving a system of linear equations with one or many right-hand sides $Ax = b$ or $AX = B$, where the coefficient matrix A is a *real symmetric*, or *complex Hermitian*, *positive definite band* matrix.
- computing a Cholesky factorization of a symmetric or Hermitian positive definite matrix, and solving a system of equations, using a previously computed factorization. (These are lower-level procedures, intended for more experienced users.)

Optional facilities are provided in these procedures for evaluating the determinant of A, estimating the condition number of A and computing error bounds on the solution.

Module 5.6: `nag_sparse_prec` — **Sparse matrix preconditioner set-up and solve**

Provides procedures for:

- initializing a sparse Jacobi preconditioner;
- initializing a sparse SSOR preconditioner;
- initializing a sparse incomplete LU preconditioner;
- solving a system of linear equations $Mz = r$, where M is supplied as a previously initialized sparse preconditioner.

Module 5.7: `nag_sparse_lin_sys` — **Sparse linear system iterative solvers**

Provides procedures for:

- the iterative solution of a general sparse system of linear equations $Ax = b$, where A is real non-symmetric or complex non-Hermitian.

3 Background

3.1 Direct Methods

For the direct methods of modules `nag_gen_lin_sys`, `nag_sym_lin_sys`, `nag_tri_lin_sys`, `nag_gen_bnd_lin_sys` and `nag_sym_bnd_lin_sys`, this chapter provides optional facilities for obtaining information about the *accuracy* of the computed solution to a system of linear equations, and about the *sensitivity* of the system to perturbations in the original data. The rest of this section explains the relevant concepts and mathematical background. For more details, see Section 2.7 of Golub and Van Loan [3] or Chapter 4 of Anderson *et al.* [1].

For simplicity we fix on systems with a single right-hand side, $Ax = b$. If there are several right-hand sides, the remarks apply to each right-hand side individually.

Perturbation Theory

Frequently in practical problems the data A and b are not known exactly, and then it is important to understand how uncertainties or perturbations in the data can affect the solution. Some systems are highly sensitive: a small perturbation in the data can result in a large change in the solution. Such systems are called *ill conditioned*.

This can be expressed in more precise mathematical terms. If x is the exact solution to $Ax = b$, and $x + \delta x$ is the solution to a perturbed problem $(A + \delta A)(x + \delta x) = b + \delta b$, then

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right) + \cdots \text{(2nd order terms)} \tag{1}$$

where $\kappa(A)$ is the *condition number* of A defined by

$$\kappa(A) = \|A\| \, \|A^{-1}\|. \tag{2}$$

In other words, $\kappa(A)$ can be regarded as an amplification factor, which describes how relative errors in the data may be amplified in the solution. An ill conditioned matrix has a large value of $\kappa(A)$. If A is singular, we define $\kappa(A) = \infty$.

To compute $\kappa(A)$ directly from the definition (2) is expensive. Therefore the procedures in this chapter *estimate* it by a method which is much cheaper, and in practice almost always reliable.

Because of the risk of overflow if A is singular or almost so, the procedures actually return an estimate of its *reciprocal* $1/\kappa(A)$, which is zero if A is singular.

Error Analysis

Perturbation theory can be applied to analyse the effects of *rounding errors* introduced by computation in finite precision.

The effects of rounding errors can be shown to be equivalent to perturbations in the original data, such that $\|\delta A\|/\|A\|$ and $\|\delta b\|/\|b\|$ are usually at most $p(n)\epsilon$, where $\epsilon =$ `EPSILON(1.0_`*wp*`)`, and $p(n)$ is an increasing function of n, which is seldom larger than $10n$ (although in theory it can be as large as 2^{n-1}). This is a *backward error bound*, because it relates the effect of rounding errors *backward* to equivalent perturbations in the original data.

A *forward error bound* analyses rounding errors in terms of their *forward* effect on the computed solution, in other words, it gives a bound on the error in the computed solution $\|\delta x\|/\|x\|$. The equation (1) can be used to convert a backward error bound on $\|\delta A\|/\|A\|$ and $\|\delta b\|/\|b\|$ to a forward error bound on $\|\delta x\|/\|x\|$ (though this may sometimes be unduly pessimistic; see Section 3.3).

If the system is ill conditioned, rounding errors in the computation may result in a computed solution which deviates quite markedly from the exact solution. If the condition number is of the order of `1/EPSILON(1.0_`*wp*`)` or larger, then the errors in the computed solution may be as large as, or larger than, the solution itself; there may be no meaningful digits at all in the computed solution. In this case A is said to be *singular to working precision*.

Computable Error Bounds

Error bounds derived from (1) have two limitations. First they are *norm wise* error bounds: they are expressed in terms of vector and matrix norms, and so they are dominated by the largest elements in the data. They do not reflect any special structure in A and b, for example a pattern of elements which are known to be zero. Second, they can be unduly pessimistic because they are based solely on the condition number of A, and do not take into account the properties of b.

The procedures in this chapter provide options for computing error bounds which do not suffer from these limitations.

First, they compute a *componentwise* backward error bound. This bounds the *relative* perturbation in *each component* of A and b which would make the computed solution $\hat{x}$ exact. Formally, if $\hat{x}$ is the exact solution of a perturbed system of equations

$$(A + \delta A)\hat{x} = b + \delta b,$$

then the backward error bound is an upper bound on

$$\max_{i,j,k}\left(\frac{|\delta a_{ij}|}{|a_{ij}|},\frac{|\delta b_k|}{|b_k|}\right).$$

Second, the procedures compute a forward error bound, which takes into account the properties of the right-hand side b and is sometimes much sharper than the norm wise bound based on (1). This bound is defined by

$$\frac{\|x-\hat{x}\|_\infty}{\|\hat{x}\|_\infty}\le\frac{\|\,|A^{-1}|\;|r|\,\|_\infty}{\|x\|_\infty},$$

where r is the residual $b-A\hat{x}$, and $|r|$ and $|A^{-1}|$ denote the vector and matrix whose elements are the absolute values of the elements of r and A^{-1} respectively. The norm $\|\,|A^{-1}|\;\;|r|\,\|_\infty$ is estimated cheaply (without computing A^{-1}) by a modification of the method used to estimate $\kappa(A)$.

Iterative Refinement

If $\hat{x}$ is an approximate computed solution, and r is the corresponding residual $b-A\hat{x}$, then a process of *iterative refinement* of $\hat{x}$ can be defined as follows, starting with $x^{(0)}=\hat{x}$:

```
for $i=0,1,\ldots$, until convergence
    compute $r^{(i)}=b-Ax^{(i)}$;
    solve $Ad^{(i)}=r^{(i)}$ for $d^{(i)}$;
    compute $x^{(i+1)}=x^{(i)}+d^{(i)}$.
```

This process can guarantee a small backward error (except in rare cases when A is very ill conditioned or when A and x are sparse in such a way that $|A|\;|x|$ has a component which is zero or very small). The iterations continue until the backward error has been reduced as much as possible; usually only one iteration is needed, and at most five are allowed.

The procedures in this chapter always perform iterative refinement if error bounds are requested.

Note that iterative refinement cannot guarantee a small forward error unless additional precision is used to compute $r^{(i)}$; this facility is not currently provided in the Library.

3.2 Iterative Methods for Sparse Linear Systems

Many of the most effective iterative methods for the solution of a sparse linear system

$$Ax=b \tag{3}$$

lie in the class of non-stationary Krylov subspace methods (see Barrett *et al.* [2]). At this release methods for real non-symmetric and complex non-Hermitian matrices are available, with or without preconditioning (see modules `nag_sparse_prec` and `nag_sparse_lin_sys`).

Preconditioning

Faster convergence of an iterative solver for a linear system can often be achieved by using a preconditioner, where (3) is replaced by the modified system

$$\bar{A}\bar{x}=\bar{b}. \tag{4}$$

A *left* preconditioner M^{-1} is chosen such that $\bar{A}=M^{-1}A\sim I_n$ in (4), where I_n is the identity matrix of order n, while a *right* preconditioner M^{-1} is such that $\bar{A}=AM^{-1}\sim I_n$. Preconditioning matrices M are typically based on incomplete factorizations (see Meijerink and van der Vorst [4]), or on the approximate inverses occurring in stationary iterative methods (see Young [5]). A common example is the incomplete *LU* factorization

$$M=PLDUQ=A-R,$$

where L is unit lower triangular, D is diagonal, U is unit upper triangular, 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.* [2].

Convergence

Iterative methods for (3) 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 reached. 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. Note that, in general, convergence, when it occurs, is not monotonic. The procedures provide a choice of termination criteria and the norms used in them. They allow monitoring of the approximate solution and can return estimates of the norm of A and the largest singular value of the preconditioned matrix $\bar{A}$.

4 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] 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

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

[4] 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

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