Chapter 2. Solution of Linear Algebraic Equations

2.0 Introduction

A set of linear algebraic equations looks like this:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1N}x_N &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2N}x_N &= b_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3N}x_N &= b_3 \\
\cdots \qquad & \quad \cdots \\
a_{M1}x_1 + a_{M2}x_2 + a_{M3}x_3 + \cdots + a_{MN}x_N &= b_M
\end{aligned}
\tag{2.0.1}
$$

Here the N unknowns x_j, $j = 1, 2, \ldots, N$ are related by M equations. The coefficients a_{ij} with $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$ are known numbers, as are the *right-hand side* quantities b_i, $i = 1, 2, \ldots, M$.

Nonsingular versus Singular Sets of Equations

If $N = M$ then there are as many equations as unknowns, and there is a good chance of solving for a unique solution set of x_j's. Analytically, there can fail to be a unique solution if one or more of the M equations is a linear combination of the others, a condition called *row degeneracy*, or if all equations contain certain variables only in exactly the same linear combination, called *column degeneracy*. (For square matrices, a row degeneracy implies a column degeneracy, and vice versa.) A set of equations that is degenerate is called *singular*. We will consider singular matrices in some detail in §2.6.

Numerically, at least two additional things can go wrong:

- While not exact linear combinations of each other, some of the equations may be so close to linearly dependent that roundoff errors in the machine render them linearly dependent at some stage in the solution process. In this case your numerical procedure will fail, and it can tell you that it has failed.

- Accumulated roundoff errors in the solution process can swamp the true solution. This problem particularly emerges if N is too large. The numerical procedure does not fail algorithmically. However, it returns a set of x's that are wrong, as can be discovered by direct substitution back into the original equations. The closer a set of equations is to being singular, the more likely this is to happen, since increasingly close cancellations will occur during the solution. In fact, the preceding item can be viewed as the special case where the loss of significance is unfortunately total.

Much of the sophistication of complicated "linear equation-solving packages" is devoted to the detection and/or correction of these two pathologies. As you work with large linear sets of equations, you will develop a feeling for when such sophistication is needed. It is difficult to give any firm guidelines, since there is no such thing as a "typical" linear problem. But here is a rough idea: Linear sets with N as large as 20 or 50 can be routinely solved in single precision (32 bit floating representations) without resorting to sophisticated methods, *if* the equations are not close to singular. With double precision (60 or 64 bits), this number can readily be extended to N as large as several hundred, after which point the limiting factor is generally machine time, not accuracy.

Even larger linear sets, N in the thousands or greater, can be solved when the coefficients are sparse (that is, mostly zero), by methods that take advantage of the sparseness. We discuss this further in §2.7.

At the other end of the spectrum, one seems just as often to encounter linear problems which, by their underlying nature, are close to singular. In this case, you *might* need to resort to sophisticated methods even for the case of $N = 10$ (though rarely for $N = 5$). Singular value decomposition (§2.6) is a technique that can sometimes turn singular problems into nonsingular ones, in which case additional sophistication becomes unnecessary.

Matrices

Equation (2.0.1) can be written in matrix form as

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{2.0.2}$$

Here the raised dot denotes matrix multiplication, **A** is the matrix of coefficients, and **b** is the right-hand side written as a column vector,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ & \cdots & & \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_M \end{bmatrix} \tag{2.0.3}$$

By convention, the first index on an element a_{ij} denotes its row, the second index its column. For most purposes you don't need to know how a matrix is stored in a computer's physical memory; you simply reference matrix elements by their two-dimensional addresses, e.g., $a_{34} =$ `a[3][4]`. We have already seen, in §1.2, that this `C` notation can in fact hide a rather subtle and versatile physical storage scheme, "pointer to array of pointers to rows." You might wish to review that section at this point. Occasionally it is useful to be able to peer through the veil, for example to pass a whole row `a[i][j], j=1,...,N` by the reference `a[i]`.

Tasks of Computational Linear Algebra

We will consider the following tasks as falling in the general purview of this chapter:

- Solution of the matrix equation $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ for an unknown vector $\mathbf{x}$, where $\mathbf{A}$ is a square matrix of coefficients, raised dot denotes matrix multiplication, and $\mathbf{b}$ is a known right-hand side vector (§2.1–§2.10).
- Solution of more than one matrix equation $\mathbf{A} \cdot \mathbf{x}_j = \mathbf{b}_j$, for a set of vectors $\mathbf{x}_j$, $j = 1, 2, \ldots$, each corresponding to a different, known right-hand side vector $\mathbf{b}_j$. In this task the key simplification is that the matrix $\mathbf{A}$ is held constant, while the right-hand sides, the $\mathbf{b}$'s, are changed (§2.1–§2.10).
- Calculation of the matrix $\mathbf{A}^{-1}$ which is the matrix inverse of a square matrix $\mathbf{A}$, i.e., $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{A}^{-1} \cdot \mathbf{A} = \mathbf{1}$, where $\mathbf{1}$ is the identity matrix (all zeros except for ones on the diagonal). This task is equivalent, for an $N \times N$ matrix $\mathbf{A}$, to the previous task with N different $\mathbf{b}_j$'s $(j = 1, 2, \ldots, N)$, namely the unit vectors ($\mathbf{b}_j$ = all zero elements except for 1 in the jth component). The corresponding $\mathbf{x}$'s are then the columns of the matrix inverse of $\mathbf{A}$ (§2.1 and §2.3).
- Calculation of the determinant of a square matrix $\mathbf{A}$ (§2.3).

If $M < N$, or if $M = N$ but the equations are degenerate, then there are effectively fewer equations than unknowns. In this case there can be either no solution, or else more than one solution vector $\mathbf{x}$. In the latter event, the solution space consists of a particular solution $\mathbf{x}_p$ added to any linear combination of (typically) $N - M$ vectors (which are said to be in the nullspace of the matrix $\mathbf{A}$). The task of finding the solution space of $\mathbf{A}$ involves

- Singular value decomposition of a matrix $\mathbf{A}$.

This subject is treated in §2.6.

In the opposite case there are more equations than unknowns, $M > N$. When this occurs there is, in general, no solution vector $\mathbf{x}$ to equation (2.0.1), and the set of equations is said to be *overdetermined*. It happens frequently, however, that the best "compromise" solution is sought, the one that comes closest to satisfying all equations simultaneously. If closeness is defined in the least-squares sense, i.e., that the sum of the squares of the differences between the left- and right-hand sides of equation (2.0.1) be minimized, then the overdetermined linear problem reduces to a (usually) solvable linear problem, called the

- Linear least-squares problem.

The reduced set of equations to be solved can be written as the $N \times N$ set of equations

$$(\mathbf{A}^T \cdot \mathbf{A}) \cdot \mathbf{x} = (\mathbf{A}^T \cdot \mathbf{b}) \tag{2.0.4}$$

where $\mathbf{A}^T$ denotes the transpose of the matrix $\mathbf{A}$. Equations (2.0.4) are called the *normal equations* of the linear least-squares problem. There is a close connection between singular value decomposition and the linear least-squares problem, and the latter is also discussed in §2.6. You should be warned that direct solution of the normal equations (2.0.4) is not generally the best way to find least-squares solutions.

Some other topics in this chapter include

- Iterative improvement of a solution (§2.5)
- Various special forms: symmetric positive-definite (§2.9), tridiagonal (§2.4), band diagonal (§2.4), Toeplitz (§2.8), Vandermonde (§2.8), sparse (§2.7)
- Strassen's "fast matrix inversion" (§2.11).

Standard Subroutine Packages

We cannot hope, in this chapter or in this book, to tell you everything there is to know about the tasks that have been defined above. In many cases you will have no alternative but to use sophisticated black-box program packages. Several good ones are available, though not always in C. LINPACK was developed at Argonne National Laboratories and deserves particular mention because it is published, documented, and available for free use. A successor to LINPACK, LAPACK, is now becoming available. Packages available commercially (though not necessarily in C) include those in the IMSL and NAG libraries.

You should keep in mind that the sophisticated packages are designed with very large linear systems in mind. They therefore go to great effort to minimize not only the number of operations, but also the required storage. Routines for the various tasks are usually provided in several versions, corresponding to several possible simplifications in the form of the input coefficient matrix: symmetric, triangular, banded, positive definite, etc. If you have a large matrix in one of these forms, you should certainly take advantage of the increased efficiency provided by these different routines, and not just use the form provided for general matrices.

There is also a great watershed dividing routines that are *direct* (i.e., execute in a predictable number of operations) from routines that are *iterative* (i.e., attempt to converge to the desired answer in however many steps are necessary). Iterative methods become preferable when the battle against loss of significance is in danger of being lost, either due to large N or because the problem is close to singular. We will treat iterative methods only incompletely in this book, in §2.7 and in Chapters 18 and 19. These methods are important, but mostly beyond our scope. We will, however, discuss in detail a technique which is on the borderline between direct and iterative methods, namely the iterative improvement of a solution that has been obtained by direct methods (§2.5).

CITED REFERENCES AND FURTHER READING:

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

Gill, P.E., Murray, W., and Wright, M.H. 1991, *Numerical Linear Algebra and Optimization*, vol. 1 (Redwood City, CA: Addison-Wesley).

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), Chapter 4.

Dongarra, J.J., et al. 1979, *LINPACK User's Guide* (Philadelphia: S.I.A.M.).

Coleman, T.F., and Van Loan, C. 1988, *Handbook for Matrix Computations* (Philadelphia: S.I.A.M.).

Forsythe, G.E., and Moler, C.B. 1967, *Computer Solution of Linear Algebraic Systems* (Englewood Cliffs, NJ: Prentice-Hall).

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag).

Westlake, J.R. 1968, *A Handbook of Numerical Matrix Inversion and Solution of Linear Equations* (New York: Wiley).

Johnson, L.W., and Riess, R.D. 1982, *Numerical Analysis*, 2nd ed. (Reading, MA: Addison-Wesley), Chapter 2.

Ralston, A., and Rabinowitz, P. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill), Chapter 9.

2.1 Gauss-Jordan Elimination

For inverting a matrix, *Gauss-Jordan elimination* is about as efficient as any other method. For solving sets of linear equations, Gauss-Jordan elimination produces *both* the solution of the equations for one or more right-hand side vectors $\mathbf{b}$, and also the matrix inverse $\mathbf{A}^{-1}$. However, its principal weaknesses are (i) that it requires all the right-hand sides to be stored and manipulated at the same time, and (ii) that when the inverse matrix is *not* desired, Gauss-Jordan is three times slower than the best alternative technique for solving a single linear set (§2.3). The method's principal strength is that it is as stable as any other direct method, perhaps even a bit more stable when full pivoting is used (see below).

If you come along later with an additional right-hand side vector, you can multiply it by the inverse matrix, of course. This does give an answer, but one that is quite susceptible to roundoff error, not nearly as good as if the new vector had been included with the set of right-hand side vectors in the first instance.

For these reasons, Gauss-Jordan elimination should usually not be your method of first choice, either for solving linear equations or for matrix inversion. The decomposition methods in §2.3 are better. Why do we give you Gauss-Jordan at all? Because it is straightforward, understandable, solid as a rock, and an exceptionally good "psychological" backup for those times that something is going wrong and you think it *might* be your linear-equation solver.

Some people believe that the backup is more than psychological, that Gauss-Jordan elimination is an "independent" numerical method. This turns out to be mostly myth. Except for the relatively minor differences in pivoting, described below, the actual sequence of operations performed in Gauss-Jordan elimination is very closely related to that performed by the routines in the next two sections.

For clarity, and to avoid writing endless ellipses $(\cdots)$ we will write out equations only for the case of four equations and four unknowns, and with three different right-hand side vectors that are known in advance. You can write bigger matrices and extend the equations to the case of $N \times N$ matrices, with M sets of right-hand side vectors, in completely analogous fashion. The routine implemented below is, of course, general.