1. Solution of linear systems of equations by direct methods

We consider the problem of finding a vector $x = (x_1, x_2, \cdots, x_n)^T$ satisfying the linear system of equations $Ax = b$, where $A = (a_{ij})$ is the square matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

and $b = (b_1, b_2, \cdots, b_n)^T$ is a given vector.

The issue is not finding an analytic solution (since Cramer's rule gives a formula for it if A is non-singular), but of finding an efficient computational algorithm. We shall consider this problem for two types of matrices: matrices with few zero elements and sparse matrices (many zero elements) and often with n large. Large sparse matrices typically occur when partial differential equations are discretized by numerical approximation schemes. We shall study two types of solution methods: direct and iterative.

1.1. Gaussian elimination and LU factorization.

Gaussian elimination and LU factorization. The direct methods we study are all variations of Gaussian elimination. In this approach, to solve the system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \equiv a_{1,n+1} \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \equiv a_{2,n+1} \\ &\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \equiv a_{n,n+1}, \end{aligned}$$

we subtract for $i = 2, \cdots, n$, a_{i1}/a_{11} times the first equation from the ith equation. We then obtain the derived system:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \equiv a_{1,n+1} \\ a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)} \equiv a_{2,n+1}^{(1)} \\ &\cdots\cdots \\ a_{n2}^{(1)}x_2 + \cdots + a_{nn}^{(1)}x_n &= b_n^{(1)} \equiv a_{n,n+1}^{(1)}, \end{aligned}$$

where the new coefficients $a_{ij}^{(1)} = a_{ij} - (a_{i1}/a_{11})a_{1j}$, $i = 2, \ldots, n$, $j = 2, \ldots, n+1$. If $a_{11} = 0$, then since A is assumed to be nonsingular, we may, by interchanging two rows, get a non-zero element in the upper left-hand corner.

Next, if $a_{22}^{(1)} \neq 0$, we subtract $a_{i2}^{(1)}/a_{22}^{(1)}$ times the second equation from the ith equation, $i = 3, \ldots, n$, and get the second derived system:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \equiv a_{1,n+1} \\
a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)} \equiv a_{2,n+1}^{(1)} \\
a_{33}^{(2)}x_3 + \cdots + a_{3n}^{(2)}x_n &= b_3^{(2)} \equiv a_{3,n+1}^{(2)} \\
&\cdots\cdots \\
a_{n3}^{(2)}x_3 + \cdots + a_{nn}^{(2)}x_n &= b_n^{(2)} \equiv a_{n,n+1}^{(2)},
\end{aligned}$$

where $a_{ij}^{(2)} = a_{ij}^{(1)} - (a_{i2}^{(1)}/a_{22}^{(1)})a_{2j}^{(1)}$, Again, if $a_{22}^{(1)} = 0$, we may interchange two rows to get a non-zero element in the $(2,2)$ position.

Continuing this process through $n-1$ steps, we arrive at the final system:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \equiv a_{1,n+1} \\
a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)} \equiv a_{2,n+1}^{(1)} \\
&\cdots\cdots \\
a_{nn}^{(n-1)}x_n &= b_n^{(n-1)} \equiv a_{n,n+1}^{(n-1)},
\end{aligned}$$

with the diagonal elements all non-zero, and

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - (a_{ik}^{(k-1)}/a_{kk}^{(k-1)})a_{kj}^{(k-1)},$$
$$k = 1, \ldots, n-1, \quad i = k+1, \ldots, n, \quad j = k+1, \ldots, n+1, \quad a_{ij}^{(0)} = a_{ij}.$$

From this form, the solution is then easily computed by back substitution, i.e,,

$$x_i = \frac{1}{a_{ii}^{(i-1)}}\left[b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)}x_j\right], \quad i = n, \ldots, 1.$$

The process leading to the upper triangular system is called Gaussian elimination. One could also consider a variant of this method, called Gauss-Jordan reduction that reduces all the non-diagonal elements to zero. However, if one counts the number of arithmetic operations needed to compute the solution by these two methods, one finds that the number of multiplications-divisions needed for the solution by Gaussian elimination is $n^3/3 + O(n^2)$, while the number needed using Gauss-Jordan reduction is $n^3/2 + O(n^2)$. Hence for n large, Gauss-Jordan reduction is not as efficient.

We now consider another variant, called Crout reduction, that is designed to reduce the number of intermediate quantities that must be retained in the process. The basic idea is to factor $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix. Once this is done, the solution of the system $Ax = b$ is found by solving $Lh = b$ and then $Ux = h$, both of which are fast, since this is accomplished using back substitution.

We first show why such a factorization is possible by relating it to the process of Gaussian elimination. Denote by A_1 the the matrix of coefficients in the first derived system obtained by Gaussian elimination. It is easy to check that $A_1 = L_1 A$, where

$$L_1 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ -a_{21}/a_{11} & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \\ -a_{n1}/a_{11} & 0 & \cdots & 1 \end{pmatrix}.$$

In general, if we denote the matrix of coefficients in the $i+1$st derived system by A_{i+1}, then $A_{i+1} = L_{i+1} A_i$, $i = 0, \dots, n-2$, where

$$L_{i+1} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & 0 & \cdots & 1 & 0 & \cdots & 0 & \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 & \\ 0 & 0 & \cdots & 0 & \frac{-a^{(i)}_{i+2,i+1}}{a^{(i)}_{i+1,i+1}} & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & 0 & \cdots & 0 & \frac{-a^{(i)}_{n,i+1}}{a^{(i)}_{i+1,i+1}} & 0 & \cdots & 1 \end{pmatrix}.$$

Finally, denoting the upper triangular matrix of coefficients in the final system by $U = (u_{ij})$, we have

$$U = L_{n-1} L_{n-2} \cdots L_1 A = \tilde{L} A,$$

where $\tilde{L}$ is lower triangular with ones on the diagonal. Since the product of two lower triangular matrices with ones on the diagonal is a matrix of the same form, and the inverse of $\tilde{L}$ is also of this form, we may write $\tilde{L}^{-1} = L$ and hence $A = LU$.

Remark 1: We have assumed in the derivation that $a^{(i)}_{i+1,i+1} \neq 0$. If this is not so, then the method breaks down. However, one can show that if A is non-singular, there exists a permutation matrix P (i.e., each row and column has exactly one entry equal to one and the rest zeroes) such that $LU = P^T A$. Thus, to solve $Ax = b$, we find a solution to the equivalent system: $P^T A x = P^T b$, which amounts to interchanging the rows.

Remark 2: If instead of using the matrices L_{i+1}, we use the identical matrices, but replacing the 1 in the $(i+1, i+1)$ position by $1/a^{(i)}_{i+1,i+1}$, then the algorithm proceeds in a similar way. However, in this case the factorization $A = LU$ will produce an upper triangular matrix U with ones on the diagonal, while L will be lower triangular, but not unit lower triangular (i.e., the diagonal elements will not necessarily be equal to one).

In matrix notation, we see that $LU = (LD)(D^{-1}U)$, where D is a diagonal matrix. We can choose D to make either LD or $D^{-1}U$ have ones on the diagonal. We now present an algorithm that produces an LU factorization of A with U unit upper triangular. The idea is simply that if $L = (l_{ij})$ is lower triangular and $U = (u_{ij})$ is unit upper triangular, then writing $A = (a_{ij})$,

$$a_{ij} = \sum_{k=1}^{\min(i,j)} l_{ik} u_{kj}.$$

Hence, we have

$$l_{ir}u_{rr} = a_{ir} - \sum_{k=1}^{r-1} l_{ik}u_{kr}, \quad i = r, \cdots, n, \qquad l_{rr}u_{rj} = a_{rj} - \sum_{k=1}^{r-1} l_{rk}u_{kj}, \quad j = r, \cdots, n.$$

Since $u_{rr} = 1$, we can define the elements l_{ir}, u_{rj} recursively by:

$$l_{ir} = a_{ir} - \sum_{k=1}^{r-1} l_{ik}u_{kr}, \qquad u_{rj} = [a_{rj} - \sum_{k=1}^{r-1} l_{rk}u_{kj}]/l_{rr}.$$

Using the convention that $\sum_{i=1}^{0} t_i = 0$ and putting in partial pivoting (i.e., interchange of rows to insure the factorization), we have the following algorithm.

For $r = 1, \cdots, n$ (incrementing by one), do steps (i) through (iv).

(i) Compute $l_{ir} = a_{ir} - \sum_{k=1}^{r-1} l_{ik}u_{kr}, \quad i = r, \cdots, n$, and overwrite l_{ir} on a_{ir} in the ith row and rth column of the matrix A.

(ii) Set $int_r \geq r$ equal to the smallest integer k for which $|l_{kr}| = \max_{r \leq j \leq n} |l_{jr}|$.

(iii) Interchange the entire rth and int_rth rows of the array. From this point on, these rows will be referred to by their new positions. Note: In practice, one does not physically interchange the rows, but instead keeps track of an index representing these changes.

(iv) Compute $u_{rj} = [a_{rj} - \sum_{k=1}^{r-1} l_{rk}u_{kj}]/l_{rr}$, $j = r+1, \cdots, n$ and overwrite u_{rj} on a_{rj} in the rth row and jth column of the matrix A.

Steps (ii) and (iii) are known as partial pivoting. We search the rth column (rows r through n) for the largest element in absolute value and use that element as the pivot to eliminate the other elements in that column (as in the Gaussian elimination method). In fact, partial pivoting is usually done, even when the LU factorization exists without doing it. We will consider this issue in more detail later.