SOLVING LINEAR SYSTEMS

Linear systems $Ax = b$ occur widely in applied mathematics. They occur as direct formulations of "real world" problems; but more often, they occur as a part of the numerical analysis of some other problem. As examples of the latter, we have the numerical solution of systems of nonlinear equations, ordinary and partial differential equations, integral equations, and the solution of optimization problems.

There are many ways of classifying linear systems.

Size: Small, moderate and large. This of course varies with the machine you are using. Most PCs are now being sold with a memory of 64 to 128 megabytes (MB), my own HP workstation has 768 MB, and the Weeg computer center's SGI shared memory parallel computer has a memory of 2 gigabytes (GB).

For a matrix A of order $n \times n$, it will take $8n^2$ bytes to store it in double precision. Thus a matrix of order 4000 will need around 128 MB of storage. The latter would be too large for most present day PCs, if the matrix was to be stored in the computer's memory.

Sparse vs. Dense. Many linear systems have a matrix A in which almost all the elements are zero. These matrices are said to be *sparse*. For example, it is quite common to work with tridiagonal matrices

$$A = \begin{bmatrix} a_1 & c_1 & 0 & \cdots & 0 \\ b_2 & a_2 & c_2 & 0 & \vdots \\ 0 & b_3 & a_3 & c_3 & \\ \vdots & & & \ddots & \\ 0 & \cdots & & b_n & a_n \end{bmatrix}$$

in which the order is 10^4 or much more. For such matrices, it does not make sense to store the zero elements; and the sparsity should be taken into account when solving the linear system $Ax = b$. Also, the sparsity need not be as regular as in this example.

Special properties: Often the matrix will have some other special properties. For example, A might be symmetric, tridiagonal, *banded*,

$$a_{i,j} = 0 \quad \text{for} \quad |i-j| > p$$

or *positive definite,*

$$(Ax, x) > 0 \quad \text{for all } x \in \mathbb{C}^n,\ x \neq 0$$

All of these properties can be used to develop more efficient and accurate methods for solving $Ax = b$.

METHODS OF SOLUTION

There are two general categories of numerical methods for solving $Ax = b$.

Direct Methods: These are methods with a finite number of steps; and they end with the exact solution x, provided that all arithmetic operations are exact. The most used of these methods is *Gaussian elimination*, which we will begin with. There are other direct methods, and we will study them later in connection with solving the matrix eigenvalue problem.

Iteration Methods: These are used in solving all types of linear systems, but they are most commonly used with large sparse systems, especially those produced by discretizing partial differential equations. This is an extremely active area of research.

SOLVING LINEAR SYSTEMS

We want to solve the linear system $Ax = b$:

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

This will be done by the method used in beginning algebra, by successively eliminating unknowns from equations, until eventually we have only one equation in one unknown. This process is known as *Gaussian elimination*. To help us keep track of the steps of this process, we will denote the initial system by $A^{(1)}x = b^{(1)}$:

$$\begin{bmatrix} a_{1,1}^{(1)} & \cdots & a_{1,n}^{(1)} \\ \vdots & \ddots & \vdots \\ a_{n,1}^{(1)} & \cdots & a_{n,n}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix}$$

Initially we will make the assumption that every *pivot element* will be nonzero; and later we remove this assumption.

Step 1. We will eliminate x_1 from equations 2 thru n. Begin by defining the *multipliers*

$$m_{i,1} = \frac{a_{i,1}^{(1)}}{a_{1,1}^{(1)}}, \quad i = 2, ..., n$$

Here we are assuming the *pivot element* $a_{1,1}^{(1)} \neq 0$. Then in succession, multiply $m_{i,1}$ times row 1 (called the *pivot row*) and subtract the result from row i. This yields new matrix elements

$$a_{i,j}^{(2)} = a_{i,j}^{(1)} - m_{i,1} a_{1,j}^{(1)}, \quad j = 2, ..., n$$

$$b_i^{(2)} = b_i^{(1)} - m_{i,1} b_1^{(1)}$$

for $i = 2, ..., n$.

Note that the index j does not include $j = 1$. The reason is that with the definition of the multiplier $m_{i,1}$, it is automatic that

$$a_{i,1}^{(2)} = a_{i,1}^{(1)} - m_{i,1} a_{1,1}^{(1)} = 0, \quad i = 2, ..., n$$

The linear system is now denoted by $A^{(2)}x = b^{(2)}$:

$$\begin{bmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n}^{(1)} \\ 0 & a_{2,2}^{(2)} & & a_{2,n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n,2}^{(2)} & \cdots & a_{n,n}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix}$$

Step k: Assume that for $i = 1, ..., k-1$ the unknown x_i has been eliminated from equations $i+1$ thru n. We have the system $A^{(k)}x = b^{(k)}$

$$\begin{bmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & & \cdots & & a_{1,n}^{(1)} \\ 0 & a_{2,2}^{(2)} & & \cdots & & a_{2,n}^{(2)} \\ & \ddots & \ddots & & & \vdots \\ \vdots & & 0 & a_{k,k}^{(k)} & \cdots & a_{k,n}^{(k)} \\ & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n,k}^{(k)} & \cdots & a_{n,n}^{(k)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ \vdots \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ \vdots \\ b_n^{(k)} \end{bmatrix}$$

We want to eliminate unknown x_k from equations $k+1$ thru n.

Begin by defining the multipliers

$$m_{i,k} = \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}, \quad i = k+1, ..., n$$

The pivot element is $a_{k,k}^{(k)}$, and we assume it is nonzero. Using these, we eliminate x_k from equations $k+1$ thru n. Multiply $m_{i,k}$ times row k (the *pivot row*) and subtract from row i, for $i = k+1$ thru n.

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - m_{i,k} a_{k,j}^{(k)}, \quad j = k+1, ..., n$$

$$b_i^{(k+1)} = b_i^{(k)} - m_{i,k} b_k^{(k)}$$

for $i = k+1, ..., n$. This yields the linear system $A^{(k)}x = b^{(k)}$:

$$\begin{bmatrix} a_{1,1}^{(1)} & & & \cdots & & a_{1,n}^{(1)} \\ 0 & \ddots & & & & \vdots \\ & & a_{k,k}^{(k)} & a_{k,k+1}^{(k)} & \cdots & a_{k,n}^{(k)} \\ \vdots & & 0 & a_{k+1,k+1}^{(k+1)} & & a_{k+1,n}^{(k+1)} \\ & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n,k+1}^{(k+1)} & \cdots & a_{n,n}^{(k+1)} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ x_{k+1} \\ \vdots \\ x_n \end{bmatrix}$$

$$= \begin{bmatrix} b_1^{(1)} \\ \vdots \\ b_k^{(k)} \\ b_{k+1}^{(k+1)} \\ \vdots \\ b_n^{(k+1)} \end{bmatrix}$$

Doing this for $k = 1, 2, ..., n$ leads to an upper triangular system of linear equations $A^{(n)}x = b^{(n)}$:

$$\begin{bmatrix} a_{1,1}^{(1)} & \cdots & & a_{1,n}^{(1)} \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & a_{n,n}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ \vdots \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

At each step, we have assumed for the pivot element that

$$a_{k,k}^{(k)} \neq 0$$

Later we remove this assumption. For now, we consider the solution of the upper triangular system $A^{(n)}x = b^{(n)}$.

To avoid the superscripts, and to emphasize we are now solving an upper triangular system, denote this system by

$$Ux = g$$

$$\begin{bmatrix} u_{1,1} & \cdots & & u_{1,n} \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & u_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} g_1 \\ \vdots \\ \vdots \\ g_n \end{bmatrix}$$

This is the linear system

$$\begin{aligned} u_{1,1}x_1 + u_{1,2}x_2 + \cdots + u_{1,n-1}x_{n-1} + u_{1,n}x_n &= g_1 \\ &\vdots \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= g_{n-1} \\ u_{n,n}x_n &= g_n \end{aligned}$$

$$\begin{aligned} u_{1,1}x_1 + u_{1,2}x_2 + \cdots + u_{1,n-1}x_{n-1} + u_{1,n}x_n &= g_1 \\ &\vdots \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= g_{n-1} \\ u_{n,n}x_n &= g_n \end{aligned}$$

We solve for x_n, then x_{n-1}, and backwards to x_1. This process is called *back substitution*.

$$x_n = \frac{g_n}{u_{n,n}}$$

$$u_k = \frac{g_k - \left\{u_{k,k+1}x_{k+1} + \cdots + u_{k,n}x_n\right\}}{u_{k,k}}$$

for $k = n-1, ..., 1$.

Examples of the process of producing $A^{(n)}x = b^{(n)}$ and of solving $Ux = g$ are given in the text. They are simply a more carefully defined and methodical version of what you have done in high school algebra.

QUESTIONS

- How do we remove the assumption on the pivot elements?

- How many operations are involved in this procedure?

- How much error is there in the computed solution due to rounding errors in the calculations?

- Later (if there is time), how does the machine architecture affect the implementation of this algorithm.

Before doing this, we consider a result on factoring matrices that is associated with Gaussian elimination.

THE LU FACTORIZATION

Using the multipliers from Gaussian elimination, define the lower triangular matrix

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{2,1} & 1 & 0 & & \\ m_{3,1} & m_{3,2} & 1 & & \vdots \\ \vdots & & & \ddots & \\ m_{n,1} & m_{n,2} & \cdots & m_{n,n-1} & 1 \end{bmatrix}$$

Recall the upper triangular matrix $U = A^{(n)}$. Then

$$A = LU$$

This is called the *LU-factorization* of the matrix A.

Proof. The proof is obtained by looking at the (i, j) element of LU,

$$(LU)_{i,j} = \begin{bmatrix} m_{i,1}, ..., m_{i,i-1}, 1, 0, ..., 0 \end{bmatrix} \begin{bmatrix} u_{1,j} \\ \vdots \\ u_{j,j} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

We consider separately the cases of $i \leq j$ and $i > j$. With the first case, we have more nonzero elements in the column $U_{*,j}$ than in the row $L_{i,*}$. Thus

$$(LU)_{i,j} = m_{i,1}u_{1,j} + \cdots + m_{i,i-1}u_{i-1,j} + u_{i,j}$$

The elements $u_{i,j}$ are given by

$$u_{i,j} = a_{i,j}^{(i)}, \quad j = i, ..., n, \quad i = 1, ..., n$$

Recall that

$$a_{i,j}^{(2)} = a_{i,j}^{(1)} - m_{i,1}a_{1,j}^{(1)}, \quad j = 2, ..., n$$

for $i = 2, ..., n$. Then

$$m_{i,1}u_{1,j} = m_{i,1}a_{1,j}^{(1)} = a_{i,j}^{(1)} - a_{i,j}^{(2)}$$

Similarly,

$$\begin{aligned} m_{i,2}u_{2,j} = m_{i,2}a_{2,j}^{(2)} &= a_{i,j}^{(2)} - a_{i,j}^{(3)} \\ &\vdots \\ m_{i,i-1}u_{i-1,j} = m_{i,i-1}a_{i-1,j}^{(i-1)} &= a_{i,j}^{(i-1)} - a_{i,j}^{(i)} \end{aligned}$$

Combining these results,

$$\begin{aligned} (LU)_{i,j} &= m_{i,1}u_{1,j} + \cdots + m_{i,i-1}u_{i-1,j} + u_{i,j} \\ &= \left(a_{i,j}^{(1)} - a_{i,j}^{(2)}\right) + \left(a_{i,j}^{(2)} - a_{i,j}^{(3)}\right) \\ &\quad + \cdots + \left(a_{i,j}^{(i-1)} - a_{i,j}^{(i)}\right) + a_{i,j}^{(i)} \\ &= a_{i,j}^{(1)} = a_{i,j} \end{aligned}$$

The second case, in which $i > j$, is given in the text.

EXAMPLE

In the text, we consider the linear system

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ -1 & -3 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 2 \end{bmatrix}$$

Gaussian elimination leads to

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & -2 & 1 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ \frac{1}{2} \end{bmatrix}$$

with multipliers

$$m_{2,1} = 2, \quad m_{3,1} = -1, \quad m_{3,2} = \frac{1}{2}$$

Then

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & -2 & 1 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ -1 & -3 & 0 \end{bmatrix}$$

In Matlab, you can find the LU factorization of a matrix using the function *lu*. For example,

$$[L \quad U \quad P] = lu(A)$$

yields a lower triangular matrix L, an upper triangular matrix U, and a *permuation matrix* P, for which

$$LU = PA$$

The reason for this arises in a modification of Gaussian elimination, one which eliminates our assumption regarding the pivot elements being nonzero and simultaneously improves the error behaviour of Gaussian elimination.

AN OPERATIONS COUNT

How many operations are expended in solving $Ax = b$ using Gaussian elimination? To calculate this, we must return to the formal definitions, of both $A^{(k)}x = b^{(k)}$ and the solution of $Ux = g$.

Step 1. Multipliers: $n - 1$ divisions.
$A^{(1)} \longrightarrow A^{(2)}$: $(n-1)^2$ additions and multiplications.
[Note that we consider additions and subtractions as the same type of operation.]
$b^{(1)} \longrightarrow b^{(2)}$: $n - 1$ additions and multiplications.

Step 2. Multipliers: $n - 2$ divisions.
$A^{(2)} \longrightarrow A^{(3)}$: $(n-2)^2$ additions and multiplications.
$b^{(2)} \longrightarrow b^{(3)}$: $n - 2$ additions and multiplications.

Step $n-1$. Multipliers: 1 division.
$A^{(n)} \longrightarrow A^{(n-1)}$: 1 addition and multiplication.
$b^{(n-1)} \longrightarrow b^{(n)}$: 1 addition and multiplication.

Thus we have the following totals for the transformation of A to $U = A^{(n)}$.
Divisions. $(n-1) + (n-2) + \cdots + 1$
Additions. $(n-1)^2 + (n-2)^2 + \cdots + 1$
Multiplications. $(n-1)^2 + (n-2)^2 + \cdots + 1$

To add these up, we need the following general formulas.

$$\sum_{j=1}^{p} j = \frac{p(p+1)}{2}$$

$$\sum_{j=1}^{p} j^2 = \frac{p(p+1)(2p+1)}{6}$$

These can proven using mathematical induction.

With these, the number of operations for $A \longrightarrow U$ are as follows.

Divisions	$\dfrac{n(n-1)}{2}$
Additions	$\dfrac{n(n-1)(2n-1)}{6}$
Multiplications	$\dfrac{n(n-1)(2n-1)}{6}$

Similarly, we can calulate the cost of $b^{(1)} \longrightarrow b^{(n)}$ as

Additions	$\dfrac{n(n-1)}{2}$
Multiplications	$\dfrac{n(n-1)}{2}$

The cost of solving $Ux = g$ is

Divisions	n
Additions	$\dfrac{n(n-1)}{2}$
Multiplications	$\dfrac{n(n-1)}{2}$

On some machines, the cost of a division is much more than that of a multiplication; whereas on others there is not any important difference.

For this last case, we usually combine the additions and subtractions to obtain the following costs.

$$\begin{aligned} MD(A \to U) &= \frac{n\left(n^2-1\right)}{3} \\ MD(b^{(1)} \to g) &= \frac{n(n-1)}{2} \\ MD(\text{Find } x) &= \frac{n(n+1)}{2} \end{aligned}$$

For additions and subtractions, we have

$$\begin{aligned} AS(A \to U) &= \frac{n(n-1)(2n-1)}{6} \\ AS(b^{(1)} \to g) &= \frac{n(n-1)}{2} \\ AS(\text{Find } x) &= \frac{n(n-1)}{2} \end{aligned}$$

Summarizing, the cost of factoring the matrix A into L and U involves approximately $\frac{1}{3}n^3$ multiplications and divisions, and approximately $\frac{1}{3}n^3$ additions and subtractions. The total number of arithmetic operations is approximately

$$\frac{2}{3}n^3$$

Thus as the size of a linear system is doubled, the cost of factoring the matrix increases by a factor of about 8.

Once the matrix has been factored, the cost of modifying the right side b to g, and of then solving for x, is

$$\begin{aligned} MD(b \to x) &= n^2 \\ AS(b \to x) &= n(n-1) \doteq n^2 \end{aligned}$$

Thus the principal cost is the solution of a linear system $Ax = b$ is in the factorization of the matrix A into LU; and the subsequent solving for x is much smaller if n is a moderately large number.

We often wish to solve linear systems $Ax = b$ with varying vectors b, and A unchanged. For such cases, it is important to keep the information used in producing L and U, as this this is the costly portion of the solution process.

In the machine, store the elements $u_{i,j}$ and $m_{i,j}$ into the portions of A with the same subscripts. This can be done, as those positions are no longer being changed; and for the portion below the diagonal, those elements are being set to zero in A.

To solve m systems

$$Ax^{(\ell)} = b^{(\ell)}, \quad \ell = 1, ..., m$$

for m different right hand vectors $b^{(\ell)}$, the operations cost will be

$$\begin{aligned} MD &= \frac{n\left(n^2-1\right)}{3} + mn^2 \doteq \tfrac{1}{3}n^3 + mn^2 \\ AS &= \frac{n(n-1)(2n-1)}{2} + mn(n-1) \doteq \tfrac{1}{3}n^3 + mn^2 \end{aligned}$$

CALCULATING AN INVERSE

To find the inverse of a matrix A amounts to solving the equation

$$AX = I$$

In partioned form,

$$A[X_{*,1}, ..., X_{*,n}] = \left[e^{(1)}, ..., e^{(n)}\right]$$

which yields the n equivalent linear systems

$$AX_{*,k} = e^{(k)}, \quad k = 1, ..., n$$

Thus the cost of calculating the inverse $X = A^{-1}$ is approximately

$$\frac{2}{3}n^3 + 2n^3 = \frac{8}{3}n^3$$

This is only around 4 times the cost of solving a single system $Ax = b$.