SOLVING LINEAR SYSTEMS

We want to solve the linear system

\[ a_{1,1}x_1 + \cdots + a_{1,n}x_n = b_1 \]
\[ \vdots \]
\[ a_{n,1}x_1 + \cdots + a_{n,n}x_n = b_n \]

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 put it onto a computer, however, we must be more precise than is generally the case in high school algebra.

We begin with the linear system

\[ 3x_1 - 2x_2 - x_3 = 0 \quad (E1) \]
\[ 6x_1 - 2x_2 + 2x_3 = 6 \quad (E2) \]
\[ -9x_1 + 7x_2 + x_3 = -1 \quad (E3) \]
\[ 3x_1 - 2x_2 - x_3 = 0 \quad (E1) \]
\[ 6x_1 - 2x_2 + 2x_3 = 6 \quad (E2) \]
\[ -9x_1 + 7x_2 + x_3 = -1 \quad (E3) \]

[1] Eliminate \( x_1 \) from equations \((E2)\) and \((E3)\). Subtract 2 times \((E1)\) from \((E2)\); and subtract \(-3\) times \((E1)\) from \((E3)\). This yields

\[ 3x_1 - 2x_2 - x_3 = 0 \quad (E1) \]
\[ 2x_2 + 4x_3 = 6 \quad (E2) \]
\[ x_2 - 2x_3 = -1 \quad (E3) \]

[2] Eliminate \( x_2 \) from equation \((E3)\). Subtract \(\frac{1}{2}\) times \((E2)\) from \((E3)\). This yields

\[ 3x_1 - 2x_2 - x_3 = 0 \quad (E1) \]
\[ 2x_2 + 4x_3 = 6 \quad (E2) \]
\[ -4x_3 = -4 \quad (E3) \]

Using back substitution, solve for \( x_3, x_2, \) and \( x_1 \), obtaining

\[ x_3 = x_2 = x_1 = 1 \]
In the computer, we work on the arrays rather than on the equations. To illustrate this, we repeat the preceding example using array notation.

The original system is \( Ax = b \), with

\[
A = \begin{bmatrix}
  3 & -2 & -1 \\
  6 & -2 & 2 \\
  -9 & 7 & 1 \\
\end{bmatrix}, \quad b = \begin{bmatrix}
  0 \\
  6 \\
  -1 \\
\end{bmatrix}
\]

We often write these in combined form as an augmented matrix:

\[
[A \mid b] = \begin{bmatrix}
  3 & -2 & -1 & 0 \\
  6 & -2 & 2 & 6 \\
  -9 & 7 & 1 & -1 \\
\end{bmatrix}
\]

In step 1, we eliminate \( x_1 \) from equations 2 and 3. We multiply row 1 by 2 and subtract it from row 2; and we multiply row 1 by -3 and subtract it from row 3. This yields

\[
\begin{bmatrix}
  3 & -2 & -1 & 0 \\
  0 & 2 & 4 & 6 \\
  0 & 1 & -2 & -1 \\
\end{bmatrix}
\]
\[
\begin{bmatrix}
3 & -2 & -1 & 0 \\
0 & 2 & 4 & 6 \\
0 & 1 & -2 & -1 \\
\end{bmatrix}
\]

In step 2, we eliminate \(x_2\) from equation 3. We multiply row 2 by \(\frac{1}{2}\) and subtract from row 3. This yields

\[
\begin{bmatrix}
3 & -2 & -1 & 0 \\
0 & 2 & 4 & 6 \\
0 & 0 & -4 & -4 \\
\end{bmatrix}
\]

Then we proceed with back substitution as previously.
For the general case, we reduce

\[
[A \mid b] = \begin{bmatrix}
  a^{(1)}_{1,1} & \cdots & a^{(1)}_{1,n} & b^{(1)}_{1} \\
  \vdots & \ddots & \vdots & \vdots \\
  a^{(1)}_{n,1} & \cdots & a^{(1)}_{n,n} & b^{(1)}_{n} 
\end{bmatrix}
\]

in \( n - 1 \) steps to the form

\[
\begin{bmatrix}
  a^{(1)}_{1,1} & \cdots & a^{(1)}_{1,n} & b^{(1)}_{1} \\
  0 & \ddots & \vdots & \vdots \\
  \vdots & \ddots & \ddots & \vdots \\
  0 & \cdots & 0 & a^{(n)}_{n,n} & b^{(n)}_{n}
\end{bmatrix}
\]

More simply, this is equivalent to the matrix-vector equation \( UX = g \):

\[
\begin{bmatrix}
  u_{1,1} & \cdots & u_{1,n} \\
  0 & \ddots & \vdots \\
  \vdots & \ddots & \ddots \\
  0 & \cdots & 0 & u_{n,n}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  \vdots \\
  \vdots \\
  x_n
\end{bmatrix}
= 
\begin{bmatrix}
  g_1 \\
  \vdots \\
  \vdots \\
  g_n
\end{bmatrix}
\]
This is the linear system

\[ u_{1,1}x_1 + u_{1,2}x_2 + \cdots + u_{1,n-1}x_{n-1} + u_{1,n}x_n = g_1 \]
\[ \vdots \]
\[ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n = g_{n-1} \]
\[ u_{n,n}x_n = g_n \]

We solve for \( x_n \), then \( x_{n-1} \), and backwards to \( x_1 \). This process is called \textit{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 \( k = n-1, \ldots, 1 \). What we have done here is simply a more carefully defined and methodical version of what you have done in high school algebra.
How do we carry out the conversion of
\[
\begin{bmatrix}
  a_{1,1}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\
  \vdots & \ddots & \vdots \\
  a_{n,1}^{(1)} & \cdots & a_{n,n}^{(1)} & b_n^{(1)}
\end{bmatrix}
\]
to
\[
\begin{bmatrix}
  a_{1,1}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\
  0 & \ddots & \vdots \\
  \vdots & \ddots & \vdots \\
  0 & \cdots & 0 & a_{n,n}^{(n)} & b_n^{(n)}
\end{bmatrix}
\]
To help us keep track of the steps of this process, we will denote the initial system by
\[
[A^{(1)} | b^{(1)}] = \begin{bmatrix}
  a_{1,1}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\
  \vdots & \ddots & \vdots \\
  a_{n,1}^{(1)} & \cdots & a_{n,n}^{(1)} & b_n^{(1)}
\end{bmatrix}
\]
Initially we will make the assumption that every \textit{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

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

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

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

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

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

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

$$a_{i,1}^{(2)} = a_{i,1}^{(1)} - m_{i,1}a_{1,1}^{(1)} = 0, \quad i = 2, \ldots, n$$
The augmented matrix now is

$$[A^{(2)} | b^{(2)}] = \begin{bmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ 0 & a_{2,2}^{(2)} & a_{2,n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n,2}^{(2)} & a_{n,n}^{(2)} & b_n^{(2)} \end{bmatrix}$$

Step $k$: Assume that for $i = 1, \ldots, k-1$ the unknown $x_i$ has been eliminated from equations $i+1$ thru $n$. We have the augmented matrix

$$[A^{(k)} | b^{(k)}] = \begin{bmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\ 0 & a_{2,2}^{(2)} & a_{2,n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & 0 & a_{k,k}^{(k)} & a_{k,n}^{(k)} & b_k^{(k)} \\ 0 & \cdots & 0 & a_{n,k}^{(k)} & a_{n,n}^{(k)} & b_n^{(k)} \end{bmatrix}$$
We want to eliminate unknown $x_k$ from equations $k+1$ thru $n$. Begin by defining the multipliers

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

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 $n$.

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

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

for $i = k + 1, \ldots, n$. This yields the augmented matrix
\[
\begin{bmatrix}
A^{(k+1)} & b^{(k+1)} \\
\end{bmatrix}:
\begin{bmatrix}
a^{(1)}_{1,1} & \cdots & a^{(1)}_{1,n} & b^{(1)}_1 \\
0 & \ddots & \vdots & \vdots \\
0 & a^{(k)}_{k,k} & a^{(k)}_{k,k+1} & \cdots & a^{(k)}_{k,n} & b^{(k)}_k \\
0 & 0 & a^{(k+1)}_{k+1,k+1} & \cdots & a^{(k+1)}_{k+1,n} & b^{(k+1)}_{k+1} \\
0 & 0 & 0 & a^{(n)}_{n,k+1} & \cdots & a^{(n)}_{n,n} & b^{(n)}_n \\
\end{bmatrix}
\]

Doing this for \( k = 1, 2, \ldots, n - 1 \) leads to the upper triangular system with the augmented matrix

\[
\begin{bmatrix}
a^{(1)}_{1,1} & \cdots & a^{(1)}_{1,n} & b^{(1)}_1 \\
0 & \ddots & \vdots & \vdots \\
0 & 0 & a^{(n)}_{n,n} & b^{(n)}_n \\
\end{bmatrix}
\]

We later remove the assumption

\[a^{(k)}_{k,k} \neq 0, \quad k = 1, 2, \ldots, n\]
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?

- How does the machine architecture affect the implementation of this algorithm.
PARTIAL PIVOTING

Recall the reduction of

\[
[A^{(1)} \ | \ b^{(1)}] = \begin{bmatrix}
    a^{(1)}_{1,1} & \cdots & a^{(1)}_{1,n} & b^{(1)}_1 \\
    \vdots & \ddots & \vdots & \vdots \\
    a^{(1)}_{n,1} & \cdots & a^{(1)}_{n,n} & b^{(1)}_n
\end{bmatrix}
\]

to

\[
[A^{(2)} \ | \ b^{(2)}] = \begin{bmatrix}
    a^{(1)}_{1,1} & a^{(1)}_{1,2} & \cdots & a^{(1)}_{1,n} & b^{(1)}_1 \\
    0 & a^{(2)}_{2,2} & \cdots & a^{(2)}_{2,n} & b^{(2)}_2 \\
    \vdots & \vdots & \ddots & \vdots & \vdots \\
    0 & a^{(2)}_{n,2} & \cdots & a^{(2)}_{n,n} & b^{(2)}_n
\end{bmatrix}
\]

What if \( a^{(1)}_{1,1} = 0 \)? In that case we look for an equation in which the \( x_1 \) is present. To do this in such a way as to avoid zero the maximum extant possible, we do the following.
Look at all the elements in the first column,

\[ a_{1,1}^{(1)}, a_{2,1}^{(1)}, \ldots, a_{n,1}^{(1)} \]

and pick the largest in size. Say it is

\[ a_{k,1}^{(1)} = \max_{j=1,\ldots,n} a_{j,1}^{(1)} \]

Then interchange equations 1 and \( k \), which means interchanging rows 1 and \( k \) in the augmented matrix \([A^{(1)} \mid b^{(1)}]\). Then proceed with the elimination of \( x_1 \) from equations 2 thru \( n \) as before.

Having obtained

\[
\begin{bmatrix}
  a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\
  0 & a_{2,2}^{(2)} & a_{2,n}^{(2)} & b_2^{(2)} \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & a_{n,2}^{(2)} & \cdots & a_{n,n}^{(2)} & b_n^{(2)}
\end{bmatrix}
\]

what if \( a_{2,2}^{(2)} = 0 \)? Then we proceed as before.
Among the elements
\[ a_{2,2}^{(2)}, a_{3,2}^{(2)}, \ldots, a_{n,2}^{(2)} \]
pick the one of largest size:

\[ |a_{k,2}^{(2)}| = \max_{j=2,\ldots,n} |a_{j,2}^{(2)}| \]

Interchange rows 2 and \( k \). Then proceed as before to eliminate \( x_2 \) from equations 3 thru \( n \), thus obtaining

\[
[A^{(3)} | b^{(3)}] = \begin{bmatrix}
    a_{1,1}^{(1)} & a_{1,2}^{(1)} & a_{1,3}^{(1)} & \cdots & a_{1,n}^{(1)} & b_1^{(1)} \\
    0 & a_{2,2}^{(2)} & a_{2,3}^{(2)} & \cdots & a_{2,n}^{(2)} & b_2^{(2)} \\
    0 & 0 & a_{3,3}^{(3)} & \cdots & a_{3,n}^{(3)} & b_3^{(3)} \\
    \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
    0 & 0 & a_{n,3}^{(n)} & \cdots & a_{n,n}^{(n)} & b_n^{(n)}
\end{bmatrix}
\]

This is done at every stage of the elimination process. This technique is called \underline{partial pivoting}, and it is a part of most Gaussian elimination programs (including the one in the text).
Consequences of partial pivoting. Recall the definition of the elements obtained in the process of eliminating $x_1$ from equations 2 thru $n$.

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

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

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

for $i = 2, \ldots, n$. By our definition of the pivot element $a_{1,1}^{(1)}$, we have

$$|m_{i,1}| \leq 1, \quad i = 2, \ldots, n$$

Thus in the calculation of $a_{i,j}^{(2)}$ and $b_{i}^{(2)}$, we have that the elements do not grow rapidly in size. This is in comparison to what might happen otherwise, in which the multipliers $m_{i,1}$ might have been very large. This property is true of the multipliers at very step of the elimination process:

$$|m_{i,k}| \leq 1, \quad i = k + 1, \ldots, n, \quad k = 1, \ldots, n - 1$$
The property

\[ |m_{i,k}| \leq 1, \quad i = k + 1, \ldots, n \]

leads to good error propagation properties in Gaussian elimination with partial pivoting. The only error in Gaussian elimination is that derived from the rounding errors in the arithmetic operations. For example, at the first elimination step (eliminating \( x_1 \) from equations 2 thru \( n \)),

\[
\begin{align*}
    a_{i,j}^{(2)} &= a_{i,j}^{(1)} - m_{i,1} a_{1,j}^{(1)}, \quad j = 2, \ldots, n \\
    b_i^{(2)} &= b_i^{(1)} - m_{i,1} b_1^{(1)}
\end{align*}
\]

The above property on the size of the multipliers prevents these numbers and the errors in their calculation from growing as rapidly as they might if no partial pivoting was used.

As an example of the improvement in accuracy obtained with partial pivoting, see the example on pages 217-218.
OPERATION COUNTS

One of the major ways in which we compare the efficiency of different numerical methods is to count the number of needed arithmetic operations. For solving the linear system

\[ a_{1,1}x_1 + \cdots + a_{1,n}x_n = b_1 \]
\[ \vdots \]
\[ a_{n,1}x_1 + \cdots + a_{n,n}x_n = b_n \]

using Gaussian elimination, we have the following operation counts.

1. A → U, where we are converting \( Ax = b \) to \( Ux = g \):

<table>
<thead>
<tr>
<th>Operation</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divisions</td>
<td>( \frac{n(n - 1)}{2} )</td>
</tr>
<tr>
<td>Additions</td>
<td>( \frac{n(n - 1)(2n - 1)}{6} )</td>
</tr>
<tr>
<td>Multiplications</td>
<td>( \frac{n(n - 1)(2n - 1)}{6} )</td>
</tr>
</tbody>
</table>
2. \( b \rightarrow g \):

\[
\begin{align*}
\text{Additions} & \quad \frac{n(n - 1)}{2} \\
\text{Multiplications} & \quad \frac{n(n - 1)}{2}
\end{align*}
\]

3. Solving \( Ux = g \):

\[
\begin{align*}
\text{Divisions} & \quad \frac{n}{n(n - 1)} \\
\text{Additions} & \quad \frac{n(n - 1)}{2} \\
\text{Multiplications} & \quad \frac{n(n - 1)}{2}
\end{align*}
\]

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. We assume the latter; and then the operation costs are as follows.

\[
\begin{align*}
MD(A \rightarrow U) & = \frac{n(n^2 - 1)}{3} \\
MD(b \rightarrow g) & = \frac{n(n - 1)}{2} \\
MD(\text{Find } x) & = \frac{n(n + 1)}{2}
\end{align*}
\]
\[ AS(A \rightarrow U) = \frac{n(n - 1)(2n - 1)}{6} \]
\[ AS(b \rightarrow g) = \frac{n(n - 1)}{2} \]
\[ AS(\text{Find } x) = \frac{n(n - 1)}{2} \]

Thus the total number of operations is

\[
\begin{array}{c}
\text{Additions} \\
\text{Multiplications and Divisions}
\end{array}
\begin{array}{c}
\frac{2n^3 + 3n^2 - 5n}{6} \\
\frac{n^3 + 3n^2 - n}{3}
\end{array}
\]

Both are around \( \frac{1}{3}n^3 \), and thus the total operations account is approximately

\[ \frac{2}{3}n^3 \]

What happens to the cost when \( n \) is doubled?
Solving $Ax = b$ and $Ax = c$. What is the cost? Only the modification of the right side is different in these two cases. Thus the additional cost is
\[
\begin{pmatrix}
MD(b \rightarrow g) \\
MD(\text{Find } x)
\end{pmatrix}
= n^2
\]
\[
\begin{pmatrix}
AS(b \rightarrow g) \\
AS(\text{Find } x)
\end{pmatrix}
= n(n - 1)
\]

The total is around $2n^2$ operations, which is quite a bit smaller than $\frac{2}{3}n^3$ when $n$ is even moderately large, say $n = 100$.

Thus one can solve the linear system $Ax = c$ at little additional cost to that for solving $Ax = b$. This has important consequences when it comes to estimation of the error in computed solutions.
Arrays in Fortran must be declared in a DIMENSION statement or something equivalent. Then the compiler arranges for the storage of the array in the computer memory, column by column. The first column is stored, then the second column follows it in memory, then the third, and so on. The “machine dimension” of the array is often larger than the amount you are using. For example, you may write

```
DIMENSION A(100,100), B(100)
```

whereas you may actually be using only the square portion in the first 10 rows and columns of the dimensioned array. This means the machine dimension of the array is often larger than the actual dimension of the matrices you are using. This has implications when passing arrays from one program section to another, say from the main program to some subroutine. We illustrate this with the book’s program LINSYS.
DYNAMIC STORAGE ALLOCATION

In Fortran 90, it is possible to dynamically create storage of just the size needed; and later when no longer need, this storage space can be deleted from memory. This uses the ALLOCATE statement and other associated statements. Consult a manual on Fortran 90. An example follows for computing a matrix-vector multiplication.