2 Elimination methods for solving linear equations

2.1 Overview

In this lecture we will take another look at linear equations. More importantly, we are going to start using computers to do the hard work for us.

In this lecture, we are going to write a program, which can solve a set of linear equations, using the method of Gaussian elimination. The program you will meet in this lecture will contain the following elements.

We want to write a program (a program or subprogram is referred to as a function in Matlab), which will take a matrix A, and a right-hand side vector b

as inputs, and give back a vector which contains the solution to A$\underline{x}$=$\underline{b}$.

```
function [x] = GaussianEliminate(A, b)

%work out the number of equations
N = length(b)



return
```

comment

We need to write our program in a text file and make sure that it has the extension "GaussianEliminate.m". To call the program in Matlab, we use would write in the command line:

```
GaussianEliminate(Matrix, RightHandSide)
```

Note : in these lecture notes, computer code will in a different font, and enclosed in a box.

2.2 Gaussian elimination

You will have probably seen Gaussian elimination before, so hopefully this will just be a review. Here we will attempt to write an algorithm which can be performed by a computer. Also, for the time being we will assume that we have a square system of equations with a unique solution to our equations. A robust algorithm would check to make sure there was a solution before proceeding.

Lets consider the system of equations

$$A\underline{x} = \underline{b} \tag{2-1}$$

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2-2}$$

We could write this in augmented form as

$$\left[\begin{array}{ccc|c} A_{11} & A_{12} & A_{13} & b_1 \\ A_{21} & A_{22} & A_{23} & b_2 \\ A_{31} & A_{32} & A_{33} & b_3 \end{array}\right] \tag{2-3}$$

Whilst we go through the method of solving these equations in the handout, we will go through (in parallel, on the board) the following example:

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

Now we can do our row operations (equivalent to adding multiples of equations together to eliminate variables, or if you prefer, to multiplying both sides of Eq. 2-1 by an elementary matrix which performs the same row operation).

Now we begin trying to reduce our system of equations to a simpler form. Using row 1, we can eliminate element A_{21} by subtracting $A_{21}/A_{11} = d_{21}$ times row 1 from row 2. Row 1 is the **pivot** row and A_{11} is the **pivot** element

$$\left[\begin{array}{ccc|c} A_{11} & A_{12} & A_{13} & b_1 \\ A_{21} & A_{22} & A_{23} & b_2 \\ A_{31} & A_{32} & A_{33} & b_3 \end{array}\right] \qquad (2\text{-}3)$$

The following operations

A_{21} -> $A_{21} - A_{11}d_{21}$

A_{22} -> $A_{22} - A_{12}d_{21}$

A_{23} -> $A_{23} - A_{13}d_{21}$

b_2->$b_2 - b_1d$

```
d      = A(2,1)/A(1,1)
A(2,1) = A(2,1) – A(1,1)*d
A(2,2) = A(2,2) – A(1,2)*d
A(2,3) = A(2,3) – A(1,3)*d
b(2) = b(2) – b(1)*d
```

give

$$\left[\begin{array}{ccc|c} A_{11} & A_{12} & A_{13} & b_1 \\ 0 & A_{22}' & A_{23}' & b_2' \\ A_{31} & A_{32} & A_{33} & b_3 \end{array}\right] \qquad (2\text{-}4)$$

Similarly, we can perform a similar operation to eliminate A_{31} by subtracting $A_{31}/A_{11} = d_{31}$ times row 1 from row 2.

A_{31} -> $A_{31} - A_{11}d_{31}$

A_{32} -> $A_{32} - A_{12}d_{31}$

A_{33} -> $A_{33} - A_{13}d_{31}$

b_3->$b_3 - b_1d$

```
d      = A(3,1)/A(1,1)
A(3,1) = A(3,1) – A(1,1)*d
A(3,2) = A(3,2) – A(1,2)*d
A(3,3) = A(3,3) – A(1,3)*d
b(3) = b(3) – b(1)*d
```

$$\left[\begin{array}{ccc|c} A_{11} & A_{12} & A_{13} & b_1 \\ 0 & A_{22}' & A_{23}' & b_2' \\ 0 & A_{32}' & A_{33}' & b_3' \end{array}\right] \qquad (2\text{-}5)$$

Having made all the elements below the pivot in this column zero, we move onto the next column. The pivot is now the element A_{22}'

Now we can eliminate A_{32}’ using row 2

$d_{32} = A_{32}'/A_{22}'$

$A_{32}' -> A_{32}' - A_{22}'d_{32}$

$A_{33}' -> A_{33}' - A_{23}'d_{32}$

b_3->$b_3 - b_2d$

```
d      = A(3,2)/A(2,2)
A(3,2) = A(3,2) – A(2,2)*d
A(3,3) = A(3,3) – A(2,3)*d
b(3)   = b(3) – b(2)*d
```

giving

$$\left[\begin{array}{ccc|c} A_{11} & A_{12} & A_{13} & b_1 \\ 0 & A_{22}' & A_{23}' & b_2' \\ 0 & 0 & A_{33}'' & b_3'' \end{array}\right] \qquad (2\text{-}6)$$

Now that the system of equations is in a triangular form, the solution can readily obtained by back-substitution. Notice, that to get to this stage, we worked on each column in turn, eliminating all the elements below the diagonal (pivot) element.

Now we have an upper triangular system, we can use back substitution

$x_3 = b_3''/A_{33}''$

$x_2 = (b_2'-A_{23}'x_3)/A_{22}'$

$x_1 = (b_1-A_{12}x_2- A_{13}x_3)/A_{11}$

```
x(3)   = b(3)/A(3,3);
x(2)   = (b(2)-A(2,3)*x(3))/A(2,2);
x(1)   = (b(1)-A(1,2)*x(2)-
     A(1,3)*x(3))/A(1,1);
```

2.2.1 Writing the program - Gaussian elimination

Rather than type out each command into Matlab, we can use "for loops", to loop through these operations. We can also take advantage of Matlab's ability to work with whole rows and columns of a matrix. If you want to access an entire row or column of a matrix, you can use do the following:

`A(1,:)` $= [A_{11},A_{12},A_{13}]$

`A(:,2)` $= [A_{12},A_{22},A_{32}]^T$

or parts of a matrix

`A(1,2:end)` $= [A_{12},A_{13}]$

So a row operation, e.g. subtracting 2*row 1 from row i is:

```
A(i,:) = A(i,:) –2* A(1,:)
```

We need two loops, one to loop through the columns (we find the diagonal element on each column and eliminate each element below it) and an inner loop to go through each row under the pivot element.

```
for column=1:(N-1)
 %work on all the rows below the diagonal element
   for row = (column+1):N
   % the row operation goes here
   end%loop through rows
end %loop through columns
```

If we add the code to do a row operation within these loops our program looks like:

```
function [x] = GaussianEliminate(A, b)
%work out the number of equations
N = length(b)
%Gaussian elimination
for column=1:(N-1)
 %work on all the rows below the diagonal element
 for row = (column+1):N

   %work out the value of d
   d = A(row,column)/A(column,column);
   %do the row operation
      A(row,:) = A(row,:)-d*A(column,:)
      b(row)   = b(row)-d*b(column)
   end%loop through rows
end %loop through columns

return
```

We still haven't finished, but we are half way there.

2.2.2 Writing the program - Back substitution

For the back substitution, we start at the bottom row and work upwards. For each row we need to do:

$$x_i = \left(b_i - \sum_{j=i+1 \text{ to } N} A_{i,j} x_j \right) \Big/ A_{i,i}$$

We can write a for loop to accomplish the back-substitution

```
%back substitution
for row=N:-1:1
    x(row) = b(row);
    for i=(row+1):N
        x(row) = x(row)-A(row,i)*x(i);
    end
    x(row) = x(row)/A(row,row);
end
```

Adding the back substitution the main program gives:

```
function [x] = GaussianEliminate(A, b)
%work out the number of equations
N = length(b)
%Gaussian elimination
for column=1:(N-1)
 %work on all the rows below the diagonal element
 for row = (column+1):N
   %work out the value of d
   d = A(row,column)/A(column,column);
      %do the row operation
      A(row,:) = A(row,:)-d*A(column,:)
      b(row)   = b(row)-d*b(column)
 end%loop through rows
end %loop through columns

%back substitution
for row=N:-1:1
    x(row) = b(row);
    for i=(row+1):N
        x(row) = x(row)-A(row,i)*x(i);
    end
    x(row) = x(row)/A(row,row);
end
%return the answer
x = x';
return
```

2.3 Partial pivoting

This basically means row swapping. Before performing elimination on a column, it is advantageous to swap rows and promote the row with the largest value in that column to the diagonal.

e.g. if our matrix A is

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

Our algorithm would fail (divide by zero). But we can swap row 1, with row 2, which has the largest absolute value in this column. (note that we must also swap the rows in the RHS vector *b*)

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

Now our algorithm can proceed as normal.

If we always swap rows to move the largest element to diagonal, the values of d_{ij} will always be less than one. This reduces the amount of accumulated numerical error in the calculation.

(Note: we would also have to swap the rows in the right hand side vector too, since all we are doing is re-ordering the equations)

2.4 Why we shouldn't use this algorithm?

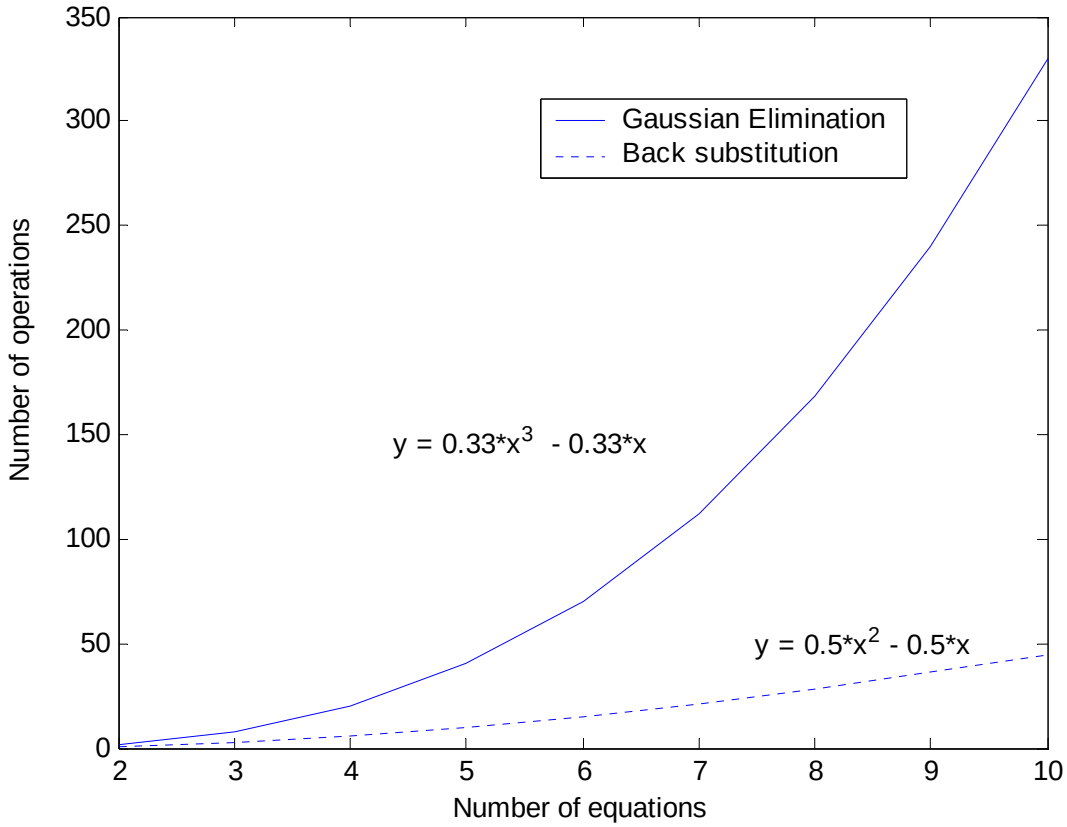
There any several reasons why we should not use this algorithm.

1. Matlab will compute the solution to $A\underline{x} = \underline{b}$ using its own (more efficient solvers) with the command `A\b`
2. Our algorithm contains lots of for loops. A peculiarity of Matlab makes for loops very slow, but vector or matrix operations very fast. The vector or matrix operations are compiled. However, each step in our algorithm (including each loop of the for loop) must first be interpreted before execution.
3. There are fundamental problems with Gaussian elimination.

2.5 Problems with Gaussian elimination

We can modify the algorithm to work out how many subtraction + multiplication operations it performs for a given size of matrix A.

```
NumOperations = 0;

%Gaussian elimination
for column=1:(N-1)



 %work on all the rows below the diagonal element
 for row = (column+1):N

   %work out the value of d
   d = A(row,column)/A(column,column);

   %do the row operation
   A(row,column:end) = A(row,column:end)-
                          ... d*A(column,column:end);
   b(row)            = b(row)-d*b(column);

   NumOperations = NumOperations+N-column+1;

 end%loop through rows
end %loop through columns
```



We find that the number of operations required to perform Gaussian elimination is $O(N^3)$, where N is the number of equations. For large systems of equations Gaussian elimination is very inefficient – even for the fastest super computer!

On the other hand, the number of operations required for back-substitution scales with $O(N^2)$. Back substitution is much more efficient for large systems than Gaussian elimination.

2.6 LU decomposition

Suppose we want to solve the same set of equations but with several different right hand sides. We don't want to use the expense of Gaussian elimination every time we solve the equations, ideally we would like to only do the Gaussian elimination once.

e.g. $A\underline{x}_1 = \underline{b}_1$, $A\underline{x}_2 = \underline{b}_2$, $A\underline{x}_3 = \underline{b}_3$

which may be written as

$$A\begin{bmatrix} \vdots & \vdots & \vdots \\ \underline{x}_1 & \underline{x}_2 & \underline{x}_3 \\ \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ \underline{b}_1 & \underline{b}_2 & \underline{b}_3 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

If by Gaussian elimination, we could factor our matrix A into two matrices, L and U, such that

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ * & 1 & 0 \\ * & * & 1 \end{bmatrix}\begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix}, \qquad (2\text{-}7)$$

$$A = LU$$

we could then solve for each right hand side using only forward, followed by backward substitution.

$A\underline{x} = \underline{b}$

$LU\underline{x} = \underline{b}$

let $\underline{y} = U\underline{x}$

$L\underline{y} = \underline{b}$ (Forward substitution)

Then solve

$U\underline{x} = \underline{y}$ (Backward substitution)

Lets start with the matrix A from our previous example

When we eliminate the element A_{21} (subtract $d_{21} = A_{21}/A_{11}$ times row 1 from row 2), we can keep multiplying by a matrix that undoes this row operation, so that the product remains equal to A.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} - d_{21}A_{12} & A_{23} - d_{21}A_{13} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

When we eliminate the element A_{31} (subtract $d_{31} = A_{31}/A_{11}$ times row 1 from row 3), we can multiply by a matrix that undoes this row operation, so that the product remains equal to A.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ d_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22}' = A_{22} - d_{21}A_{12} & A_{23}' = A_{23} - d_{21}A_{13} \\ 0 & A_{32}' = A_{32} - d_{31}A_{12} & A_{33}' = A_{33} - d_{31}A_{12} \end{bmatrix}$$

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22}' = A_{22} - d_{21}A_{12} & A_{23}' = A_{23} - d_{21}A_{13} \\ 0 & A_{32}' = A_{32} - d_{31}A_{12} & A_{33}' = A_{33} - d_{31}A_{12} \end{bmatrix}$$

When we eliminate the element A_{32} (subtract $d_{32} = A_{32}/A_{22}$ times row 2 from row 3), we can keep multiply by a matrix that undoes this row operation, so that the product remains equal to A.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & d_{32} & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22}' & A_{23}' \\ 0 & 0 & A_{33}'' = A_{33}' - d_{32} A_{23}' \end{bmatrix}$$

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & d_{32} & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22}' & A_{23}' \\ 0 & 0 & A_{33}'' = A_{33}' - d_{32} A_{23}' \end{bmatrix}$$

I.e. we have performed LU decomposition.

In practice we don't bother recording the matrices, we can just write down the matrix L.

2.6.1 *What happens if we have to do a row swap during the Gaussian elimination?*

Suppose we had reached the following stage in the elimination process

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22}' & A_{23}' \\ 0 & A_{32}' & A_{33}' \end{bmatrix}$$

and we needed to exchange rows 2 and 3.

This is a permutation matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{31} & 0 & 1 \\ d_{21} & 1 & 0 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{32}' & A_{33}' \\ 0 & A_{22}' & A_{23}' \end{bmatrix}$$

Multiplying by a permutation matrix will swap the rows of a matrix. The permutation matrix is just an identity matrix, whose rows have been interchanged.

We can then continue as normal.

2.6.2 A recipe for doing LU decomposition

1. Write down a permutation matrix, P (which is initially the identity matrix)

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Write down the Matrix you want to decompose

$$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

3. Starting with column 1, row swap to promote the largest value in the column to the diagonal. Do exactly the same row swap to the Matrix P.

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4. Eliminate all the elements below the diagonal in column 1. **Record the multiplier *d* used for elimination where you create the zero**.

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1.5 & -0.5 \end{bmatrix}$$ Here we did row 3 = row 3 - 0.5*row 1

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0.5 & 1.5 & -0.5 \end{bmatrix}$$

5. Move onto the next column. Swap rows to move the largest element to the diagonal. (Do the same row swap to P)

$$\begin{bmatrix} 2 & 1 & 1 \\ 0.5 & 1.5 & -0.5 \\ 0 & 1 & 1 \end{bmatrix} \qquad P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

6. Eliminate the elements below the diagonal

$$\begin{bmatrix} 2 & 1 & 1 \\ 0.5 & 1.5 & -0.5 \\ 0 & 2/3 & 4/3 \end{bmatrix}$$

row 3 = row 3 - 2/3*row 2

7. Repeat 5 and 6 for all columns

8. Write down L, U and P

$$\begin{bmatrix} 2 & 1 & 1 \\ 0.5 & 1.5 & -0.5 \\ 0 & 2/3 & 4/3 \end{bmatrix}$$

implies

$$U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1.5 & -0.5 \\ 0 & 0 & 4/3 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0 & 2/3 & 1 \end{bmatrix}$$

and $P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$

2.7 Summary

- Gaussian elimination can be slow (the number of operations required to do Gaussian elimination is $O(N^3)$ where N is the number of equations.
- Back substitution only takes $O(N^2)$
- If we want to solve the set of equations repeatedly with different right hand sides, LU decomposition means that we don't have to do Gaussian elimination every time. This is a considerable saving.
- Matlab has inbuilt Matlab routines for solving linear equations (`\`) and LU decomposition (`LU`). They will generally be better than any you write yourself

2.8 Appendix: Code to do Gaussian elimination with partial pivoting

```
function [x] = GaussianEliminate(A,b)
% Solves Ax = b by Gaussian elimination


%work out the number of equations
N = length(b);
%Gaussian elimination
for column=1:(N-1)

    %swap rows so that the row we are using to eliminate
    %the entries in the rows below is larger than the
    %values to be eliminated.
    [dummy,index] = max(abs(A(column:end,column)));
    index=index+column-1;
    temp  = A(column,:);
    A(column,:) = A(index,:);
    A(index,:)  = temp;
    temp = b(column)
    b(column)= b(index);
    b(index) = temp;


    %work on all the rows below the diagonal element
    for row =(column+1):N


        %work out the value of d
        d = A(row,column)/A(column,column);

        %do the row operation (result displayed on screen)
        A(row,column:end) = A(row,column:end)-d*A(column,column:end) ;
          b(row)          = b(row)-d*b(column);
    end%loop through rows
end %loop through columns


%back substitution

for row=N:-1:1

x(row) = b(row);

    for i=(row+1):N
          x(row) = x(row)-A(row,i)*x(i);
    end

x(row) = x(row)/A(row,row);

end


x = x'

return
```