Tom Green reg number 1202807

Lab 3

1) Starting with the code provided in GE_simple.m, add the missing lines. Test your program on the set of equations:

$$6x_1 + 6x_2 + 6x_3 = -6$$
$$3x_1 + 9x_2 + 12x_3 = 0$$
$$12x_1 + 9x_2 + 18x_3 = 24$$

The code that I changed in the GE_simple.m file was:

```
A(j,k) = A(j,k) + mult*A(i,k);
b(j) = b(j) + mult*b(i);
x(i) = (b(i)-sum)/A(i,i);
```

Then in the original load_save.m file the matrix was:

```
A = [1,1,1;      2,3,4;      3,2,4]; b = [6;16;17]; save Adata A b
```

So I firstly ran the GE_simple file on this matrix to check that it was working and I got this:

x =

3
2
1

Then I changed the load_save.m file to fit the equations in question 1 in and I got the correct answers of:

x =

0.2857
-4.8571
3.5714

2) Now turn to the code provided in GE_PP.m, which is intended to be a modification of the GE_simple.m to include partial pivoting. Add the missing lines of code and test whether you get the same result as in question 1.

The code that I changed in the file GE_PP was:

```
pivot_row = j;
A(i,k) = A(pivot_row,k);
A(pivot_row,k) = temp;
b(i) = b(pivot_row);
b(pivot_row) = temp;
mult = -A(j,i)/A(i,i);
A(j,k) = A(j,k) + mult*A(i,k);
b(j) = b(j) + mult*b(i);
x(n) = b(n) / A(n,n);
x(i) = (b(i)-sum)/A(i,i);
```

Just like in the first question I ran my code for the original matrix to check that it was working and I got the same answers as in the first question. I then ran the code for the equations given in question 1 to get these answers:
x =

0.2857
-4.8571
3.5714

Which are in fact the same as in question 1.

2) **The Vandermonde matrix (V) of size p+1 depends on a set of p+1 distinct values $\{\propto_i\}_{i=0}^{p}$ and is created like this:**

$$V = \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ \propto_0^p & \cdots & \propto_p^p \end{pmatrix}$$

For p=5, create a Vandermonde matrix from the last digit (d) of your registration number by defining

$$\propto_i = \frac{1}{i + 3 - 0.1 * d}, for\ i = 0,1 \ldots p$$

Let z be the vector defined by $z_i = p + 2 - i$ for i=1,2…p+1 and let w be defined by w=Vz solev the system

Vx=w

Using Gaussian Elimination with and without partial pivoting. Repeat the exercise with p=10 and p=15.

The code that I changed in the VM.m file was:

```
d=7;
alpha(i) = 1/(i+3-0.1*d);
```

The code that I used for this question was:

```
d=7;
p = 10;

alpha = zeros(p+1,1);
z=zeros(p+1,1);    %creating z vector, size


V = zeros(p+1,p+1);

for i = 1:p+1
  alpha(i) = 1/(i+3-0.1*d);
  z(i)=p+2-i;            %define the z vector
end
```

```
for i = 1:p+1              %defining the v vector
  V(1,i) = 1.0;
  for j = 1:p
    V(j+1,i) = alpha(i)^j;
  end
end

%defining w vector

% checking solution using matrix inverse


w=V*z;
xstar=inv(V)*w;
error1=xstar-z

%copying v and w into A and b

A=V;
b=w;

%checking that matrix A is square

[m,n] = size(A)
 if (m ~= n) %---------1------------
   disp('A must be square')
 else      %---------1------------

 %      BEGINNING of SIMPLE GAUSS NUMERICAL PROCEDURE

     % Elimination phase

   for i = 1:(n-1)   % Eliminate x_i
     for j = (i+1):n
       mult = -A(j,i)/A(i,i);
       for k = i:n
         A(j,k) = A(j,k) + mult*A(i,k);  % COMPLETE THIS LINE
       end
       b(j) = b(j) + mult*b(i);    % COMPLETE THIS LINE
     end
   end


   % Back substitution phase

   x = zeros(n,1);
   x(n) = b(n) / A(n,n);
   for i = (n-1):-1:1
     sum = 0.0;
     for j = (i+1):n
       sum = sum + A(i,j) * x(j);
     end
     x(i) = (b(i)-sum)/A(i,i);
   end

   xsimple=x; %copied solution

   error_simple=xsimple-z
```

```
%      END of SIMPLE GAUSS NUMERICAL PROCEDURE


%copying v and w into A and b

A=V;
b=w;

%checking that matrix A is square

[m,n] = size(A)
 if (m ~= n) %---------1------------
   disp('A must be square')
 else      %---------2------------

 %      BEGINNING of Partial GAUSS NUMERICAL PROCEDURE
% Elimination phase

   for i = 1:(n-1)   % Eliminate x_i


     % First, find the largest of the potential pivots
     % Start from the i_th row and work downwards

     pivot = abs(A(i,i));
     pivot_row = i;
     for j = (i+1):n
       if (abs(A(j,i)) > pivot)
         pivot = abs(A(j,i));
         pivot_row = j;
       end
     end


     % If the chosen row is not the i_th row, swap the equations

     if (i ~= pivot_row)
       for k = 1:n
         temp = A(i,k);
         A(i,k) = A(pivot_row,k);
         A(pivot_row,k) = temp;
       end
       temp = b(i)
       b(i) = b(pivot_row)
       b(pivot_row) = temp
     end


     % Now proceed with the elimination of x_i

     for j = (i+1):n
       mult = -A(j,i)/A(i,i);
       for k = i:n
         A(j,k) = A(j,k) + mult*A(i,k);
       end
       b(j) = b(j) + mult*b(i);
     end
   end
```

```
   % Back substitution phase

   x = zeros(n,1);
   x(n) = b(n) / A(n,n);
   for i = (n-1):-1:1
     sum = 0;
     for j = (i+1):n
       sum = sum + A(i,j) * x(j);
     end
     x(i) = (b(i)-sum)/A(i,i);
   end



%      end of Partial GAUSS NUMERICAL PROCEDURE
 end         %---------2------------

xpivoting=x;

error_pivoting=xpivoting-z

norm_error1=norm(error1)
norm_error_simple=norm(error_simple)
norm_error_pivoting=norm(error_pivoting)


eps=2
iteration=0
while (eps+1>1)
    eps=eps/1.0001;
  iteration=iteration+1;
end
eps=eps*2;
eps
end %---------1------------
```

This code does basically everything all in one, it creates the Vandermonde Matrix and then runs it through the simple Gauss elimination and then the partial pivoting one. If you want it to go through just one then I have labeled each section i.e. if you want just the partial pivoting then you can either delete the simple code or just % it all.

For p=5 the Matrix V, z and w that I got was:

```
V =

    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    0.3030    0.2326    0.1887    0.1587    0.1370    0.1205
    0.0918    0.0541    0.0356    0.0252    0.0188    0.0145
    0.0278    0.0126    0.0067    0.0040    0.0026    0.0017
    0.0084    0.0029    0.0013    0.0006    0.0004    0.0002
    0.0026    0.0007    0.0002    0.0001    0.0000    0.0000

z =
```

```
     6
     5
     4
     3
     2
     1

w =

   21.0000
    4.6063
    1.0914
    0.2756
    0.0731
    0.0201
```

The program then gave an x values of:

```
x =

    6.0000
    5.0000
    4.0000
    3.0000
    2.0000
    1.0000
```

The program gives error values of:

```
norm_error1 =

   1.4929e-10


norm_error_simple =

   6.7740e-12


norm_error_pivoting =

   6.8484e-12
```

Proof that it works is:

```
V*x

ans =

   21.0000
    4.6063
    1.0914
    0.2756
    0.0731
    0.0201
```

which is exactly the same as w listed above

When I increase p to 10 I get these results for both methods:

```
x =
```

```
   11.0000
   10.0000
    9.0000
    8.0000
    7.0000
    5.9999
    5.0002
    3.9997
    3.0002
    1.9999
    1.0000


V =

    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
1.0000    1.0000    1.0000    1.0000    1.0000
    0.3030    0.2326    0.1887    0.1587    0.1370    0.1205
0.1075    0.0971    0.0885    0.0813    0.0752
    0.0918    0.0541    0.0356    0.0252    0.0188    0.0145
0.0116    0.0094    0.0078    0.0066    0.0057
    0.0278    0.0126    0.0067    0.0040    0.0026    0.0017
0.0012    0.0009    0.0007    0.0005    0.0004
    0.0084    0.0029    0.0013    0.0006    0.0004    0.0002
0.0001    0.0001    0.0001    0.0000    0.0000
    0.0026    0.0007    0.0002    0.0001    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
    0.0008    0.0002    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
    0.0002    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
    0.0001    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000    0.0000    0.0000    0.0000

norm_error1 =

   5.2117e-04


norm_error_simple =

   3.9839e-04


norm_error_pivoting =

   3.9778e-04


eps =

     2
```

Finally for p=15 I get:

```
norm_error1 =
```

```
  102.6131


norm_error_simple =

   81.5894


norm_error_pivoting =

  132.7909


eps =

     2

x =

   16.0000
   15.0000
   14.0000
   13.0003
   11.9944
   11.0476
    9.8199
    8.9061
   11.9684
  -11.6949
   53.7422
  -71.3971
   82.7335
  -47.8945
   20.8108
   -2.0367
```

This shows that the number of pivots as you increase the value of p will also increase with a larger amount of errors.

4a) **Is –E diagonally domianant?**

-E is diagonally dominant because the absolute values of the matricies will not change.

4b) **IS E+F diagonally dominant by rows?**

E+F is not diagonally dominant as you can have opposite signed elements that can cancel each other out.

4c) **Does your answer change if all of the values in E and F are non-negative?**

Yes, if all of the values of E and F are non-negative then when added together they would not be able to cancel each other out but only add to make a larger number.

4d) **Does your answer change if all of the values in E and F are non-positive?**

No the answer does not change because if they are all negative then when added together they will keep the same order of size. The answer does not change if they are all the same signs.

5) **Starting with the code provided in Jacobi.m fill in the missing lines. Test your program by using it on the example problem on slides 71-73 in the module notes. What does the norm function on lines 16 and 36 do? Why is it being used?**

The lines of code that I used for this program were:

```
sum = sum + A(i,j) * x(j);
for j = i+1:n
sum = sum + A(i,j) * x(j);
x(i) = (b(i)-sum)/A(i,i);
```

The problem in the lecture notes is:

$10x_1+4x_2+x_3=21$
$2x_1 + 20x_2 + 3x_3 = 51$
$1x_1 + 3x_2 + 10x_3 = 37$

When I create CData A b to be:

```
A = [10,4,1;
     2,20,3;
     1,3,10];
b = [21;51;37];
```

and run the Jacobi program I get the correct answer of:

```
x =

   1.0000
   2.0000
   3.0000
```

after 8 iterations

The norm function is this program is used to show that the program is converging as it takes the norm of both the iteration and the one before to show whether the numbers are close to each other.

6) **Let c be the next to last digit of your registration number-but if c=0 then let c=3. Define b=2c and a =7c+0.3d and create the following two 5x5 matricies:**

```
F =

   23.1000    6.0000         0         0         0
```

```
    6.0000   23.1000   -6.0000         0         0
         0   -6.0000   23.1000    6.0000         0
         0         0    6.0000   23.1000   -6.0000
         0         0         0   -6.0000   23.1000

G =

     3     0     0     0     0
     0    -3     0     0     0
     0     0    -3     0     0
     0     0     0    -3     0
     0     0     0     0     3
```

Now form the following matrix M

```
M =

     F     G     0     0     0
     G     F    -G     0     0
     0    -G     F     G     0
     0     0     G     F    -G
     0     0     0    -G     F
```

Compute the vector defined by $z_i = 13 - i$ for i=1,2…25 and then calculate v=Mz and solve

Mx=v

Using Jacobi's method record the number of itertaions and show how quickly the process converged.

The code that I used for this question is as follows:

```
eps=1.e-6
 cc=3;
 dd=7;
 bb=2*cc;
 aa=7*cc+0.3*dd;

 z0=zeros(5,5);

 F=[aa bb 0 0 0;bb aa -bb 0 0;0 -bb aa bb 0;0 0 bb aa -bb;0 0 0 -bb
aa];

 G=[cc 0 0 0 0;0 -cc 0 0 0;0 0 -cc 0 0;0 0 0 -cc 0;0 0 0 0 cc];

 M=[F G z0 z0 z0;G F -G z0 z0;z0 -G F G z0;z0 z0 G F -G;z0 z0 z0 -G
F];

 z=rand([25,1]);

 for i=1:25
 z(i,1)=13-i;
 end

 v=M*z;
 A=M;
 b=v;
 [m,n] = size(A);
 if (m ~= n)
```

```
   disp('A must be square')
 else

   x = zeros(n,1);

   prev_x = ones(n,1);
   diff = norm(x - prev_x);

   e=A*x-b;

   ediff=norm(e);

   iterations = 0;

   disp(['Iteration= ' int2str(iterations) ' Diff= ' num2str(diff)
' norm(e)= ' num2str(norm(e))]);

   diffall=max(diff, ediff);


   while ( ( (iterations < 100000) & (diffall > eps) ) ),

     iterations = iterations + 1;
     prev_x = x;

     for i = 1:n
       sum = 0;
       for j = 1:(i-1)
         sum = sum + A(i,j) * x(j);     % COMPLETE THIS LINE
       end
       for j = i+1:n                   % COMPLETE THIS LINE
         sum = sum + A(i,j) * x(j);     % COMPLETE THIS LINE
       end

       x(i) = (b(i)-sum)/A(i,i);
     end
     diff = norm(x - prev_x);

     e=A*x-b;
     ediff=norm(e);

    diffall=max(diff, ediff);

     disp(['Iteration= ' int2str(iterations) ' Diff= ' num2str(diff)
' norm(e)= ' num2str(norm(e))]);
end   % of while loop
   disp('Solution is: ')
   x
   disp(' ')
   disp('Number of iterations: ')
   disp(iterations)

 end
```

This gives the vector z to be

```
z =

    12
```

```
    11
    10
     9
     8
     7
     6
     5
     4
     3
     2
     1
     0
    -1
    -2
    -3
    -4
    -5
    -6
    -7
    -8
    -9
   -10
   -11
   -12
```

The code then gives v to be:

```
v =

  364.2000
  248.1000
  204.0000
  207.9000
  139.8000
  227.7000
  120.6000
   73.5000
   74.4000
   75.3000
   22.2000
   65.1000
   18.0000
   18.9000
  -70.2000
  -63.3000
 -110.4000
 -157.5000
 -156.6000
  -95.7000
 -229.8000
 -207.9000
 -258.0000
 -260.1000
 -190.2000
```

Then calculates x to be equal to z. It takes a total of 19 iterations to get this answer with these results:

```
Iteration= 1 Diff= 34.9896 norm(e)= 187.5998
Iteration= 2 Diff= 8.3906 norm(e)= 52.282
Iteration= 3 Diff= 2.2486 norm(e)= 9.9974
```

```
Iteration= 4 Diff= 0.48119 norm(e)= 3.3056
Iteration= 5 Diff= 0.18066 norm(e)= 1.2304
Iteration= 6 Diff= 0.068598 norm(e)= 0.45816
Iteration= 7 Diff= 0.02574 norm(e)= 0.16808
Iteration= 8 Diff= 0.0094137 norm(e)= 0.059785
Iteration= 9 Diff= 0.0033394 norm(e)= 0.02085
Iteration= 10 Diff= 0.001164 norm(e)= 0.0072106
Iteration= 11 Diff= 0.00040306 norm(e)= 0.0024955
Iteration= 12 Diff= 0.0001399 norm(e)= 0.00087083
Iteration= 13 Diff= 4.902e-05 norm(e)= 0.00030806
Iteration= 14 Diff= 1.7424e-05 norm(e)= 0.00011082
Iteration= 15 Diff= 6.2994e-06 norm(e)= 4.0586e-05
Iteration= 16 Diff= 2.3179e-06 norm(e)= 1.512e-05
Iteration= 17 Diff= 8.6717e-07 norm(e)= 5.7199e-06
Iteration= 18 Diff= 3.2921e-07 norm(e)= 2.1921e-06
Iteration= 19 Diff= 1.2653e-07 norm(e)= 8.49e-07
```

The norm and the diff start to converge after about 6 or 7 iterations.

If we then change the diagonal elements of F to be 2a we get an F of:

```
F =

   46.2000    6.0000         0         0         0
    6.0000   46.2000   -6.0000         0         0
         0   -6.0000   46.2000    6.0000         0
         0         0    6.0000   46.2000   -6.0000
         0         0         0   -6.0000   46.2000
```

Which changes the value of v to be:

```
v =

  641.4000
  502.2000
  435.0000
  415.8000
  324.6000
  389.4000
  259.2000
  189.0000
  166.8000
  144.6000
   68.4000
   88.2000
   18.0000
   -4.2000
 -116.4000
 -132.6000
 -202.8000
 -273.0000
 -295.2000
 -257.4000
 -414.6000
 -415.8000
 -489.0000
 -514.2000
 -467.4000
```

This gets the results in a lower number of iterations at 11 getting these results:

```
Iteration= 0 Diff= 5 norm(e)= 1677.5285
Iteration= 1 Diff= 35.7696 norm(e)= 203.1203
Iteration= 2 Diff= 4.4951 norm(e)= 26.7399
Iteration= 3 Diff= 0.57024 norm(e)= 3.0215
Iteration= 4 Diff= 0.069153 norm(e)= 0.45546
Iteration= 5 Diff= 0.010566 norm(e)= 0.055327
Iteration= 6 Diff= 0.0012913 norm(e)= 0.0066167
Iteration= 7 Diff= 0.00015328 norm(e)= 0.00067455
Iteration= 8 Diff= 1.5597e-05 norm(e)= 7.1025e-05
Iteration= 9 Diff= 1.6867e-06 norm(e)= 9.3081e-06
Iteration= 10 Diff= 2.2452e-07 norm(e)= 1.2656e-06
Iteration= 11 Diff= 3.0459e-08 norm(e)= 1.6192e-07
```