# ASSIGNMENT-3

## 21MAT204

MIS-3
Professor- Dr.Neethu Mohan

# 1. Prove and verify the following using MATLAB.

a) Nonzero eigenvalues of $A^TA$ and $AA^T$ are same.

**Code: For 2x2 matrix**

Q1. a)

```
%For 2x2 matrix

A = [6 7;8 5]
A_trans = transpose(A)
B = mtimes(A,A_trans)
eig(B)

%Similar eigenvalue can be obtained if one uses SVD in descending order
svd(B)

C = mtimes(A_trans,A)
eig(C)

%Similar eigenvalue can be obtained if one uses SVD in descending order
svd(C)
```

mtimes(A,B) – It is used for matrix multiplication in MATLAB. It will check for correct matrix dimension to follow. It will multiply A with B

Eig(B) – Gives the eigen value of the selected matrix B in form of diagonal matrix or a vector and depending upon parameter one can also get the right and left eigenvector for that matrix using eig funcntion. It can only be applied to square matrix

Svd(B) – Gives the singular value decomposition in form of U∑V' . It is applied to arbitrary sized matrix. It is not a compulsion to use it on square matrix.

## Output: For 2x2 matrix –

Q1. a)

```
%For 2x2 matrix

A = [6 7;8 5]
A_trans = transpose(A)
B = mtimes(A,A_trans)
eig(B)

%Similar eigenvalue can be obtained if one uses SVD in descending order
svd(B)

C = mtimes(A_trans,A)
eig(C)

%Similar eigenvalue can be obtained if one uses SVD in descending order
svd(C)
```

*Hence, we can see that eigenvalues for both matrix A^TA and AA^T is same that is of form 2x2.*

```
B = 2x2
    85    83
    83    89

ans = 2x1
    3.9759
  170.0241

ans = 2x1
  170.0241
    3.9759

C = 2x2
   100    82
    82    74

ans = 2x1
    3.9759
  170.0241

ans = 2x1
  170.0241
    3.9759
```

## Code: For 3x3 matrix —

```
%For 3x3 matrix

A = [1 6 2;8 5 4;6 3 1];
A_trans = transpose(A);
B = mtimes(A_trans,A)
eig(B)
%Similar eigenvalue can be obtained if one uses SVD
svd(B)

C = mtimes(A,A_trans)
eig(C)
%Similar eigenvalue can be obtained if one uses SVD
svd(C)
```

## Output: For 3x3 matrix –

```
%For 3x3 matrix

A = [1 6 2;8 5 4;6 3 1];
A_trans = transpose(A);
B = mtimes(A_trans,A)
eig(B)
%Similar eigenvalue can be obtained if one uses SVD
svd(B)

C = mtimes(A,A_trans)
eig(C)
%Similar eigenvalue can be obtained if one uses SVD
svd(C)
```

*Hence, we can see that eigenvalues for both matrix A^TA and AA^T is same that is of form 3x3.*

```
B = 3x3
   101    64    40
    64    70    35
    40    35    21

ans = 3x1
    1.7392
   20.0259
  170.2350

ans = 3x1
  170.2350
   20.0259
    1.7392

C = 3x3
    41    46    26
    46   105    67
    26    67    46

ans = 3x1
    1.7392
   20.0259
  170.2350
```

b)

$$Let\ \lambda_1, \lambda_2, \ldots, \lambda_n \text{ be eigenvalues and } v_1, v_2, \ldots, v_n \text{ be orthonormal eigenvectors of } A^T A$$

$$Prove\ That\ Total\ variation = \sum_{i=1\ to\ n} \lambda_i = \|A\|_F^2$$

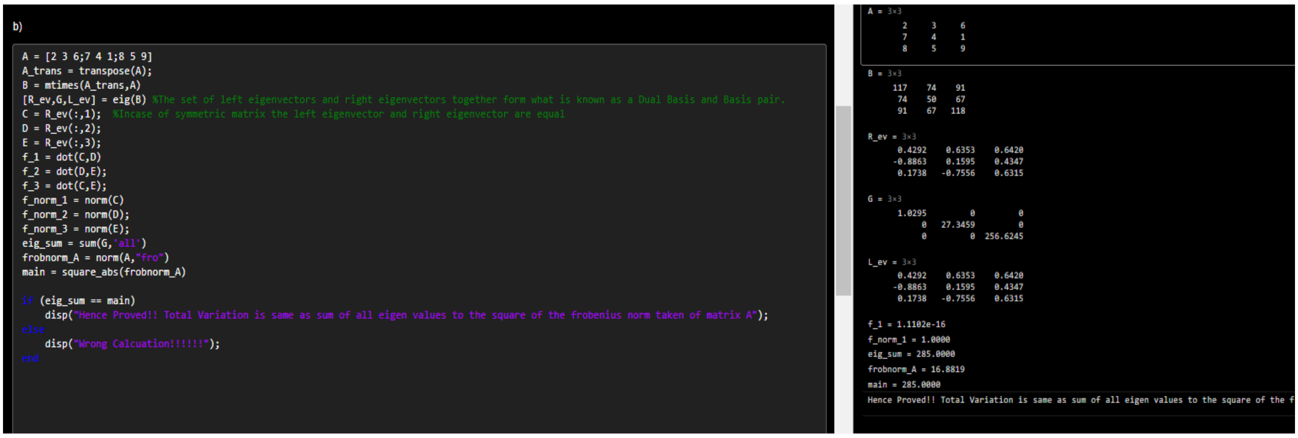
CODE:

b)

```
A = [2 3 6;7 4 1;8 5 9]
A_trans = transpose(A);
B = mtimes(A_trans,A);
[R_ev,G,L_ev] = eig(B) %The set of left eigenvectors and right eigenvectors together form what is known as a Dual Basis and Basis pair.
C = R_ev(:,1);  %Incase of symmetric matrix the left eigenvector and right eigenvector are equal
D = R_ev(:,2);
E = R_ev(:,3);
f_1 = dot(C,D)
f_2 = dot(D,E);
f_3 = dot(C,E);
f_norm_1 = norm(C)
f_norm_2 = norm(D);
f_norm_3 = norm(E);
eig_sum = sum(G,'all')
frobnorm_A = norm(A,"fro")
main = square_abs(frobnorm_A)

if (eig_sum == main)
    disp("Hence Proved!! Total Variation is same as sum of all eigen values to the square of the frobenius norm taken of matrix A");
else
    disp("Wrong Calcuation!!!!!!");
end
```

OUTPUT: Using eig() – {Please zoom in the picture for the results}




b)

```
A = [2 3 6;7 4 1;8 5 9]
A_trans = transpose(A);
B = mtimes(A_trans,A)
[R_ev,G,L_ev] = eig(B) %The set of left eigenvectors and right eigenvectors together form what is known as a Dual Basis and Basis pair.
C = R_ev(:,1);  %Incase of symmetric matrix the left eigenvector and right eigenvector are equal
D = R_ev(:,2);
E = R_ev(:,3);
f_1 = dot(C,D)
f_2 = dot(D,E);
f_3 = dot(C,E);
f_norm_1 = norm(C)
f_norm_2 = norm(D);
f_norm_3 = norm(E);
eig_sum = sum(G,'all')
frobnorm_A = norm(A,"fro")
main = square_abs(frobnorm_A)

if (eig_sum == main)
    disp("Hence Proved!! Total Variation is same as sum of all eigen values to the square of the frobenius norm taken of matrix A");
else
    disp("Wrong Calcuation!!!!!!");
end
```


```
A = 3×3
     2     3     6
     7     4     1
     8     5     9

B = 3×3
   117    74    91
    74    50    67
    91    67   118

R_ev = 3×3
    0.4292    0.6353    0.6420
   -0.8863    0.1595    0.4347
    0.1738   -0.7556    0.6315

G = 3×3
    1.0295         0         0
         0   27.3459         0
         0         0  256.6245

L_ev = 3×3
    0.4292    0.6353    0.6420
   -0.8863    0.1595    0.4347
    0.1738   -0.7556    0.6315

f_1 = 1.1102e-16
f_norm_1 = 1.0000
eig_sum = 285.0000
frobnorm_A = 16.8819
main = 285.0000
Hence Proved!! Total Variation is same as sum of all eigen values to the square of the f
```

Initially, from eig(B) we are getting right eigenvector (in form of matrix because of three eigen values the matrix has) , diagonal matrix which have eigenvalue and left eigenvector matrix. Then we are going for confirmation regarding vectors that whether they are orthonormal or not.

This can be done by checking whether the eigenvector is orthogonal or not and norm of that eigenvector is 1 or not.

In line 13, we are creating a variable named eig_sum which will store the sum of all the eigenvalue of $A^T A$.

Now we will calculate the square of frobenius norm using norm() function and will keep it inside an if condition.

We could have used [U S V] = svd(B) instead of [R_ev,G,L_ev] = eig(B) but eig(B) helps in achieving much accurate result in terms of decimal point but the answer will remain unchanged.

We are now going to cross-verify the same.

CODE:

```
A = [2 3 6;7 4 1;8 5 9];
A_trans = transpose(A);
B = mtimes(A_trans,A)
[U S V] = svd(B) %The set of left singularvectors and right singularvectors together form what is known as a Dual Basis and Basis pair.
C = U(:,1);  %Incase of symmetric matrix the left singularvector and right singularvector are equal
D = U(:,2);
E = U(:,3);
f_1 = round(dot(C,D),1)
f_2 = round(dot(D,E),1)
f_3 = round(dot(C,E),1)
f_norm_1 = norm(C)
f_norm_2 = norm(D)
f_norm_3 = norm(E)
eig_sum = sum(G,'all')
frobnorm_A = norm(A,"fro")
main = square_abs(frobnorm_A)

if (eig_sum == main)
    disp("Hence Proved!! Total Variation is same as sum of all eigen values to the square of the frobenius norm taken of matrix A");
else
    disp("Wrong Calcuation!!!!!!");
end
```

OUTPUT: Using svd() – {Please zoom in the picture for the results}


```
A = [2 3 6;7 4 1;8 5 9];
A_trans = transpose(A);
B = mtimes(A_trans,A)
[U S V] = svd(B) %The set of left singularvectors and right singularvectors together form what is known as a Dual Basis and Basis pair.
C = U(:,1);  %Incase of symmetric matrix the left singularvector and right singularvector are equal
D = U(:,2);
E = U(:,3);
f_1 = round(dot(C,D),1)
f_2 = round(dot(D,E),1)
f_3 = round(dot(C,E),1)
f_norm_1 = norm(C)
f_norm_2 = norm(D)
f_norm_3 = norm(E)
eig_sum = sum(G,'all')
frobnorm_A = norm(A,"fro")
main = square_abs(frobnorm_A)

if (eig_sum == main)
    disp("Hence Proved!! Total Variation is same as sum of all eigen values to the square of the frobenius norm taken of matrix A");
else
    disp("Wrong Calcuation!!!!!!");
end
```


```
c)
```

```
B = 3×3
    117    74    91
     74    50    67
     91    67   118

U = 3×3
    -0.6420    0.6353    0.4292
    -0.4347    0.1595   -0.8863
    -0.6315   -0.7556    0.1738

S = 3×3
   256.6245         0         0
          0   27.3459         0
          0         0    1.0295

V = 3×3
    -0.6420    0.6353    0.4292
    -0.4347    0.1595   -0.8863
    -0.6315   -0.7556    0.1738

f_1 = 0
f_2 = 0
f_3 = 0
f_norm_1 = 1.0000
f_norm_2 = 1.0000
f_norm_3 = 1
eig_sum = 285.0000
frobnorm_A = 16.8819
main = 285.0000

Hence Proved!! Total Variation is same as sum of all eigen values to the square of the fr
```

c)

$Let\ \lambda_1, \lambda_2, ..., \lambda_n$ be eigenvalues and $v_1, v_2, .., v_n$ be orthonormal eigenvectors of $A^T A$

$Show\ that\ Av_1, Av_2, .., Av_n$ are orthogonal and are eigen vectors of $AA^T$

CODE:

c)

```
A = [2 3 6;7 4 1;8 5 9]
A_trans = transpose(A);
B = mtimes(A_trans,A)
[R_ev,G] = svd(B,'vector')

norm(A)
otherscal = norm(A)-norm(A,Inf)

Matrix_mult_1_RC = mtimes(A,R_ev(:,1));
Optimal_Matrix_mult_1_RC = Matrix_mult_1_RC/norm(A)

Matrix_mult_2_RC = mtimes(A,R_ev(:,2));
Optimal_Matrix_mult_2_RC = Matrix_mult_2_RC/otherscal

Matrix_mult_3_RC = mtimes(A,R_ev(:,3));
Optimal_Matrix_mult_3_RC = Matrix_mult_3_RC

f_1 = round(dot(Optimal_Matrix_mult_1_RC, Optimal_Matrix_mult_2_RC),1)
f_2 = round(dot(Optimal_Matrix_mult_2_RC, Optimal_Matrix_mult_3_RC),1)
f_3 = round(dot(Optimal_Matrix_mult_3_RC, Optimal_Matrix_mult_1_RC),1)


C = mtimes(A,A_trans)
[R_ev_1,G_1] = svd(C,'vector')

EV_1 = R_ev_1(:,1)
EV_2 = R_ev_1(:,2)
EV_3 = R_ev_1(:,3)
```

Using svd() we are getting right eigenvector and vector which contains three different eigenvalues.

Matrix_mult_1_RC contains the vector which one gets after multiplying it by matrix A

Optimal vector is one which we will get when we divide it by the norm

f_1 is having dot product which is used to prove that $Av_1$, $Av_2$ and $Av_3$ are orthogonal.

After proving orthogonality we can check $Av_1$, $Av_2$ and $Av_3$ with eigenvector of $AA^T$ , we will be getting approximate equality for eigenvectors.

Output:

```
R_ev = 3x3
     -0.6420    0.6353    0.4292
     -0.4347    0.1595   -0.8863
     -0.6315   -0.7556    0.1738

G = 3x1
    256.6245
     27.3459
      1.0295

ans = 16.0195
otherscal = -5.9805
Optimal_Matrix_mult_1_RC = 3x1
     -0.3981
     -0.4285
     -0.8111

Optimal_Matrix_mult_2_RC = 3x1
      0.4656
     -0.7239
      0.1539

Optimal_Matrix_mult_3_RC = 3x1
     -0.7579
     -0.3671
      0.5660

f_1 = 0
f_2 = 0
f_3 = 0
C = 3x3
      49    32    85
      32    66    85
      85    85   170
```

```
C = 3x3
      49    32    85
      32    66    85
      85    85   170

R_ev_1 = 3x3
     -0.3981    0.5325   -0.7470
     -0.4285   -0.8279   -0.3618
     -0.8111    0.1760    0.5578

G_1 = 3x1
    256.6245
     27.3459
      1.0295

EV_1 = 3x1
     -0.3981
     -0.4285
     -0.8111

EV_2 = 3x1
      0.5325
     -0.8279
      0.1760

EV_3 = 3x1
     -0.7470
     -0.3618
      0.5578
```

Q2. Demonstrate Jacobi and Gauss-Seidel and SOR iterations for the following data.

$$A=\begin{bmatrix} -4 & 2 & 1 & 0 & 0 \\ 1 & -4 & 1 & 1 & 0 \\ 2 & 1 & -4 & 1 & 2 \\ 0 & 1 & 1 & -4 & 1 \\ 0 & 0 & 1 & 2 & -4 \end{bmatrix}, b=\begin{bmatrix} -4 \\ 11 \\ -16 \\ 11 \\ -4 \end{bmatrix}$$

a) Using Jacobi Iteration

CODE:

**Jacobi Implementation**

```
A = [-4 2 1 0 0;1 -4 1 1 0;2 1 -4 1 2;0 1 1 -4 1;0 0 1 2 -4]
B = [-4;11;-16;11;-4]
x_old = [0;0;0;0;0]

fprintf('Convergence before using jacobi iteration')
    conv = norm(B - (mtimes(A,x_old)))

nrow = size(A);
i = 1;
j = 1;

for j = 1:nrow(:,1)
    for i = 1:nrow(:,1)
        xnew(i) = x_old(i) + (B(i) - dot(A(i,:),x_old))/A(i,i);
    end
    fprintf('X Vector after %d iteration \n',j)
    x_old = xnew
    fprintf('Convergence using jacobi after %d iteration \n',j)
    conv = norm(B - (mtimes(A,x_old)))
end
```

X_old is the first iteration vector which can be taken randomly and here it is taken as [0,0,0,0,0].

We are calculating norm just before the iteration using norm().

Dimensionality of matrix A is helping us in setting up the number of iterations of the FOR loop

We are going to use two FOR loops where the first loop will change the row and the second loop will change the column and then store that x_new vector in x_old vector for repetitive iterations over the new vector.

We are updating the x_old vector after every iteration and we are printing norm after every iteration.

Formula:

## Jacobi implementation

$$x_i^{k+1} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^k\right]$$

We can simplify above for fast computation

$$x_i^{k+1} = x_i^k + \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i}^{n} a_{ij}x_j^k\right]$$

$$x_i^{k+1} = x_i^k + \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{j=n} a_{ij}x_j^k\right]$$ **Based on ith equation, ith variable $x_i$ is updated**

$$= x_i^k + \frac{1}{a_{ii}}\left[b_i - \left(\mathit{dotproduct}\ \text{of ith row of A with old x vector}\right)\right]$$

Output:

```
A = 5×5
    -4     2     1     0     0
     1    -4     1     1     0
     2     1    -4     1     2
     0     1     1    -4     1
     0     0     1     2    -4

B = 5×1
    -4
    11
   -16
    11
    -4

x_old = 5×1
     0
     0
     0
     0
     0

Convergence before using jacobi iteration
conv = 23.0217
```

```
X Vector after 1 iteration
x_old = 5×1
     1.0000
    -2.7500
     4.0000
    -2.7500
     1.0000

Convergence using jacobi after 1 iteration
conv = 4.1079
X Vector after 2 iteration
x_old = 5×1
     0.6250
    -2.1875
     3.6250
    -2.1875
     0.6250

Convergence using jacobi after 2 iteration
conv = 1.1558
X Vector after 3 iteration
x_old = 5×1
     0.8125
    -2.2344
     3.5312
    -2.2344
     0.8125

Convergence using jacobi after 3 iteration
conv = 0.7109
```

```
X Vector after 4 iteration
x_old = 5×1
     0.7656
    -2.2227
     3.6953
    -2.2227
     0.7656

Convergence using jacobi after 4 iteration
conv = 0.3612
X Vector after 5 iteration
x_old = 5×1
     0.8125
    -2.1904
     3.6543
    -2.1904
     0.8125

Convergence using jacobi after 5 iteration
conv = 0.2598
```

b) Using Gauss-Seidel

Code:

**Gauss Seidel Implementation**

```
A = [-4 2 1 0 0;1 -4 1 1 0;2 1 -4 1 2;0 1 1 -4 1;0 0 1 2 -4];
B = [-4;11;-16;11;-4];
x_old = [0;0;0;0;0];

fprintf('Convergence before using gauss seidel iteration')
    conv = norm(B - (mtimes(A,x_old)))

nrow = size(A);
i = 1;
j = 1;

for j = 1:nrow(:,1)
    xnew = x_old;
    for i = 1:nrow(:,1)
        xnew(i) = xnew(i) + (B(i) - dot(A(i,:),xnew))/A(i,i);
    end
    fprintf('X Vector after %d iteration \n',j)
    x_old = xnew
    fprintf('Convergence using gauss seidel after %d iteration \n',j)
    conv = norm(B - (mtimes(A,x_old)))
end
```

When compared with code jacobi implementation, one can see that the iteration is taking place considering the new value of x vector rather than taking the old vector itself. X_Vector is getting updated on a regular basis. This helps in accelerating the convergence in comparison to jacobi.

# Gauss-Seidel (GS) iteration

Use the latest update

$$
\begin{aligned}
a_{11}\boxed{x_1} + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}\boxed{x_2} + \cdots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}\boxed{x_n} &= b_n
\end{aligned}
\qquad
x^0 = \begin{bmatrix} x_1^0 \\ x_2^0 \\ \vdots \\ x_n^0 \end{bmatrix}
$$

$$
\begin{aligned}
x_1^1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2^0 - \cdots - a_{1n}x_n^0) \\
x_2^1 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1^1 - a_{23}x_3^0 - \cdots - a_{2n}x_n^0) \\
x_n^1 &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1^1 - a_{n2}x_2^1 - \cdots - a_{nn-1}x_{n-1}^1)
\end{aligned}
$$

$$
x_i^{k+1} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k\right]
$$

Output:

```
Convergence before using gauss seidel iteration
conv = 23.0217
X Vector after 1 iteration
x_old = 5×1
      1.0000
     -2.5000
      3.8750
     -2.4062
      0.7656

Convergence using gauss seidel after 1 iteration
conv = 2.1851
X Vector after 2 iteration
x_old = 5×1
      0.7188
     -2.2031
      3.5898
     -2.2119
      0.7915

Convergence using gauss seidel after 2 iteration
conv = 0.4058
```

```
X Vector after 3 iteration
x_old = 5×1
      0.7959
     -2.2065
      3.6891
     -2.1815
      0.8315

Convergence using gauss seidel after 3 iteration
conv = 0.1979
X Vector after 4 iteration
x_old = 5×1
      0.8190
     -2.1684
      3.7378
     -2.1498
      0.8596

Convergence using gauss seidel after 4 iteration
conv = 0.1750
X Vector after 5 iteration
x_old = 5×1
      0.8503
     -2.1404
      3.7824
     -2.1246
      0.8833

Convergence using gauss seidel after 5 iteration
conv = 0.1441
```

c) Using Successive Over Relaxation method

Code:

**Successive Over Relaxation Method**

```
A = [-4 2 1 0 0;1 -4 1 1 0;2 1 -4 1 2;0 1 1 -4 1;0 0 1 2 -4];
B = [-4;11;-16;11;-4];
x_old = [0;0;0;0;0];

fprintf('Convergence before using SOR iteration')
    conv = norm(B - (mtimes(A,x_old)))

nrow = size(A);
i = 1;
j = 1;
omega = 1.18;

for j = 1:nrow(:,1)
    xnew = x_old;
    for i = 1:nrow(:,1)
        xnew(i) = xnew(i) + omega*(B(i) - dot(A(i,:),xnew))/A(i,i);
    end
    fprintf('X Vector after %d iteration \n',j)
    x_old = xnew
    fprintf('Convergence using SOR after %d iteration \n',j)
    conv = norm(B - (mtimes(A,x_old)))
end
```

When compared to jacobi and gauss-seidel, one can notice that while updating x_vector for each row there is an omega which is being multiplied to again accelerate the convergence. Here omega is around 1.2.There is a range of omega or which convergence takes place faster and slower accordingly.

# SOR

$$x_i^{k+1} = x_i^k + \omega \delta_i^k$$

$$x_i^{k+1} = x_i^k + \omega \frac{1}{a_{ii}} \left[ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i}^{n} a_{ij} x_j^k \right]$$

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

$1<\omega<2$ over relaxation (faster convergence)
$0<\omega<1$ under relaxation (slower convergence)
There is an optimum value for $\omega$
Find it by trial and error (usually around 1.6)

Output:

```
Convergence before using SOR iteration
conv = 23.0217
X Vector after 1 iteration
x_old = 5×1
    1.1800
   -2.8969
    4.5616
   -2.7539
    0.9009

Convergence using SOR after 1 iteration
conv = 6.1248
X Vector after 2 iteration
x_old = 5×1
    0.6041
   -2.0121
    3.3809
   -2.0797
    0.7882

Convergence using SOR after 2 iteration
conv = 1.9041
```

```
X Vector after 3 iteration
x_old = 5×1
    0.8815
   -2.2389
    3.8225
   -2.1710
    0.8849

Convergence using SOR after 3 iteration
conv = 0.5714
X Vector after 4 iteration
x_old = 5×1
    0.8280
   -2.1105
    3.7795
   -2.1008
    0.8962

Convergence using SOR after 4 iteration
conv = 0.2803
X Vector after 5 iteration
x_old = 5×1
    0.9007
   -2.1042
    3.8594
   -2.0847
    0.9272

Convergence using SOR after 5 iteration
conv = 0.1118
```

*THANK YOU!!*