Table of Contents

MATH2019 (2019-2020) Coursework 3

NAME:JAKE DENTON STUDENT ID:14322189

```
clear variables
close all
clc
```

Question 1

Following code produces matrix Tj and vector cj and matrix Tg and vector cg for the Jacobi and Gauss-Seidel method, respectively.

```
% Set n
n = 4;

% Construction of A
% (using Matlab "diag" function)
A =  3 * diag(ones(1,n  )) ...
    -1 * diag(ones(1,n-1),-1) ...
    -1 * diag(ones(1,n-1), 1);

% Construction of b
b = [4; 2*ones(n-2,1); 4];

D=diag(diag(A)); %the first 3 lines decompose A into D, L and U
L=(-1)*tril(A,-1);
U=(-1)*triu(A,1);
invD=diag(1./diag(A));%computes inverse of diagonal matrix D (just 1/
(diagonal elements))
format short
Tj=invD*(L+U);%computes Jacobi matrix using definition from vector
 form of Jacobi method
disp('The Jacobi matrix is defined as Tj=');%Next two lines output the
 above result
disp(Tj);
cj=invD*b;%computes constant vector again using vector form of Jacobi
 method
disp('The constant vector in Jacobi method is defined as cj=');%Next
 two lines display the constant vector
```

```
disp(cj);
Tg=(D-L)\U;%computes Gauss-Seidel matrix using definition from vector
 form of Gauss-Seidel method
disp('The Gauss-Seidel matrix is defined as Tg=');
disp(Tg);%outputs Gauss-Seidel matrix
cg=(D-L)\b;%calculates constant vector from vector form of Gauss-
Seidel method
disp('The constant vector in Gauss-Seidel method is defined as cg=');
disp(cg);%outputs Gauss-Seidel constant vector
```

```
The Jacobi matrix is defined as Tj=
         0    0.3333         0         0
    0.3333         0    0.3333         0
         0    0.3333         0    0.3333
         0         0    0.3333         0

The constant vector in Jacobi method is defined as cj=
    1.3333
    0.6667
    0.6667
    1.3333

The Gauss-Seidel matrix is defined as Tg=
         0    0.3333         0         0
         0    0.1111    0.3333         0
         0    0.0370    0.1111    0.3333
         0    0.0123    0.0370    0.1111

The constant vector in Gauss-Seidel method is defined as cg=
    1.3333
    1.1111
    1.0370
    1.6790
```

Question 2

Following code produces two tables with output of the Jacobi and Gauss-Seidel method, respectively.

```
x0=[1;1;1;1];%defines initial approximation as that given in the
 question
Nmax=16;%question asks for 16 approximations
format long g
xJac=genIterMeth(Tj,cj,x0,Nmax); %calls genIterMeth to generate our
 approximation matrix (using Jacobi method)
x1J=xJac(1,:)';%next two lines collect the x1 and x2 approximations
 and transpose them for use in table function
x2J=xJac(2,:)';
K=0:1:16;%next two lines create column vector which will denote which
 approximation is which in our table
k=K';
JacobiTable=table(k,x1J,x2J);%produces table showing values of x1/x2
 over k
```

```
disp('Table showing how approximations of x1 and x2 evolve for Jacobi
 method:');
disp(JacobiTable);%outputs table
xGS=genIterMeth(Tg,cg,x0,Nmax);%the rest of this code repeats the
 above steps using the Gauss-Seidel approximations
x1GS=xGS(1,:)';
x2GS=xGS(2,:)';
GaussSeidelTable=table(k,x1GS,x2GS);
disp('Table showing how approximations of x1 and x2 evolve for Gauss-
Seidel method:');
disp(GaussSeidelTable);
```

Table showing how approximations of x1 and x2 evolve for Jacobi method:

k	x1J	x2J
0	1	1
1	1.66666666666667	1.33333333333333
2	1.77777777777778	1.66666666666667
3	1.88888888888889	1.81481481481481
4	1.93827160493827	1.90123456790123
5	1.96707818930041	1.94650205761317
6	1.98216735253772	1.97119341563786
7	1.99039780521262	1.98445358939186
8	1.99481786313062	1.99161713153483
9	1.99720571051161	1.99547833155515
10	1.99849277718505	1.99756134735559
11	1.9991871157852	1.99868470818021
12	1.9995615693934	1.99929060798847
13	1.99976353599616	1.99961739246062
14	1.99987246415354	1.99979364281893
15	1.99993121427298	1.99988870232416
16	1.99996290077472	1.99993997219904

Table showing how approximations of x1 and x2 evolve for Gauss-Seidel method:

k	x1GS	x2GS
0	1	1
1	1.66666666666667	1.55555555555556
2	1.85185185185185	1.79012345679012
3	1.93004115226337	1.93552812071331
4	1.9785093735711	1.98110044200579
5	1.99370014733526	1.99449609646226
6	1.99816536548742	1.99839869336389
7	1.9994662311213	1.99953418054769
8	1.99984472684923	1.99986449606696
9	1.99995483202232	1.99996058287976
10	1.99998686095992	1.99998853384798
11	1.99999617794933	1.99999666458044
12	1.99999888819348	1.99999902975091
13	1.99999967658364	1.99999971776166

```
    14    1.99999990592055    1.99999991789894
    15    1.99999997263298     1.9999999761174
    16    1.99999999203913    1.99999999305273
```

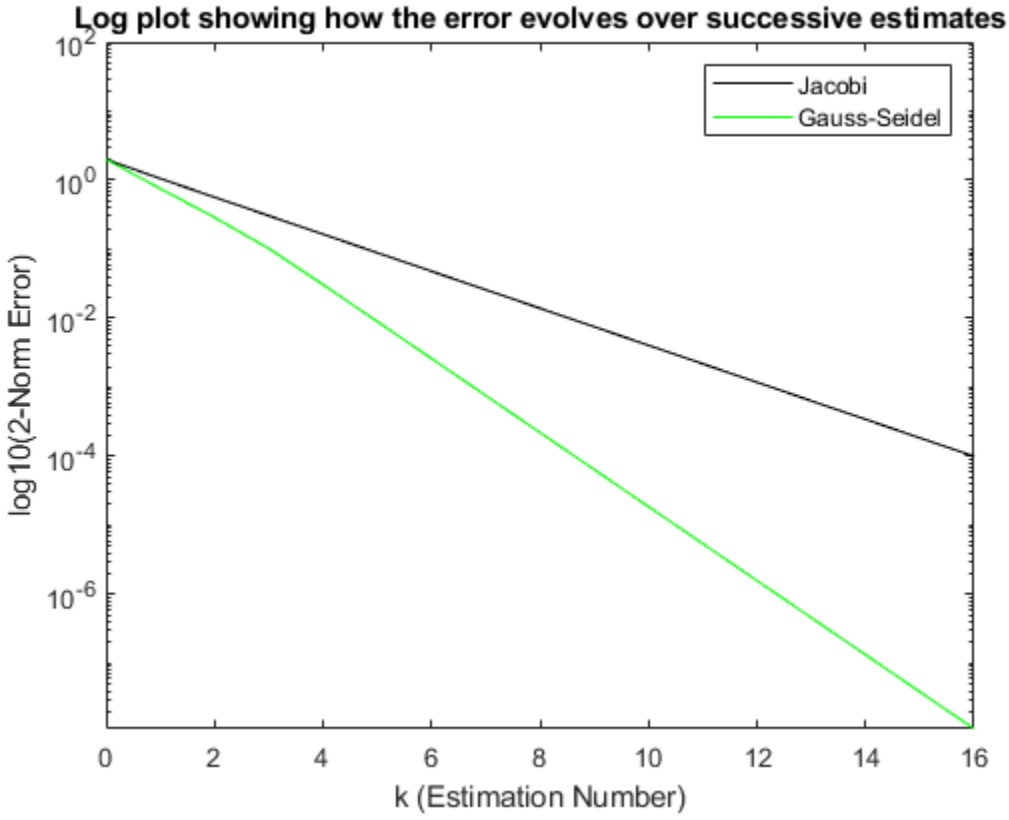
Question 3

Following code produces a figure with two plots for the 2-norm errors (for the Jacobi and Gauss-Seidel method).

```
l2J=zeros(17,1);%first two lines create zero vectors that we can fill
 with the error norms for each method
l2GS=zeros(17,1);
x=[2;2;2;2];%this is the exact solution given in the question
for i=1:17 %there are 17 estimate vectors to find the norm errors of
    errorJ=x-xJac(:,i);%finds error in i-th estimate vector for Jacobi
    errorGS=x-xGS(:,i);%finds error in i-th estimate vector for Gauss-
Seidel
    l2J(i)=norm(errorJ);%next two lines assign the i-th element of a
 vector with the appropriate 2-norm error for each method
    l2GS(i)=norm(errorGS);
end
semilogy(K,l2J,'k');%produces semilogy axes and plots Jacobi line
hold on
semilogy(K,l2GS,'g');%plots Gauss-Seidel line on same set of axes
title('Log plot showing how the error evolves over successive
 estimates');%rest of this portion of code formats figure
xlabel('k (Estimation Number)');
ylabel('log10(2-Norm Error)');
legend('Jacobi','Gauss-Seidel');
axis([0,16,0,100]);

% Answer to question on observed slopes: ...
logerrorJ=log10(l2J);%obtains the log (base10) of the error norms for
 Jacobi
slopeJ=diff(logerrorJ);%calculates the difference between adjacent
 elements in logerrorJ (gives slope between each pair of points)
alphaJ=mean(slopeJ);%sets alphaJ (approximate slope) as the arithmetic
 mean of the slopes between each pair of points
disp(['The approximate value for the slope (from the mean of the
 slopes of line segments between adjacent points) for Jacobi method is
 ',num2str(alphaJ,15)]);
logerrorGS=log10(l2GS);%rest of this code repeats the above for Gauss-
Seidel method
slopeGS=diff(logerrorGS);
alphaGS=mean(slopeGS);
disp(['The approximate value for the slope (from the mean of the
 slopes of line segments between adjacent points) for Gauss-Seidel
 method is ',num2str(alphaGS,15)]);

specJ=max(abs(eig(Tj)));
specGS=max(abs(eig(Tg)));
almostslopeJ=log10(specJ);
```

```
almostslopeGS=log10(specGS);
disp(['The approximate values (from theory using the spectral radius)
 are ',num2str(almostslopeJ,15),' and ',num2str(almostslopeGS,15),'
 for the Jacobian and Gauss-Seidel methods respectively']);


hold off
```

```
The approximate value for the slope (from the mean of the slopes
 of line segments between adjacent points) for Jacobi method is
 -0.268869616878455
The approximate value for the slope (from the mean of the slopes of
 line segments between adjacent points) for Gauss-Seidel method is
 -0.515544908985422
The approximate values (from theory using the spectral radius) are
 -0.268133614469684 and -0.536267228939367 for the Jacobian and Gauss-
Seidel methods respectively
```



Question 4*

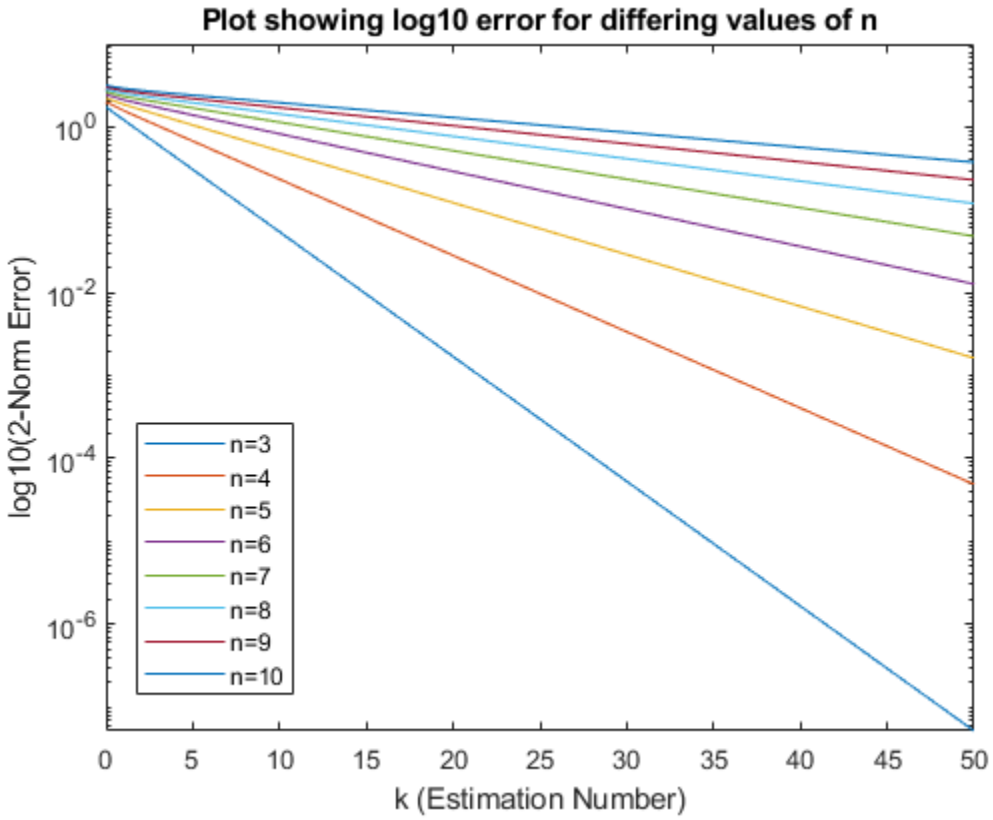
Answer to open question: It is known that the error norm |xk-x| is approximately equivalent to an expression involving the spectral radius of the transformation matrix as follows: |xk-x|(approx.)=p(Tj)^(k)*||x0-x|| where p(Tj) is the spectral radius of Tj. Therefore we should expect from this that if the value for this spectral radius is smaller convergence will be more rapid. It is easier to see this relationship using a semilog plot, as taking log to the base 10 of the above equation and noting that in this question, |x0-x|=n^(1/2) since x0-x=(-1)*ones(n,1) using their definitions. This gives us the linear

relationship log||xk-x||=klog(p(Tj))+(0.5)*logn. Clearly now the log10(p(Tj)) can be seen as the slope of the error line on a plot, so the MATLAB code below produces such plots for increasing n. It shows that as n increases, the error line approaches zero slope, that is equivalent to the spectral radius of the transformation matrix approaching value 1 from below (since the lines have negative gradient) and so we can conclude that the rapidity of convergence decreases as n increases since the spectral radius is increasing (see relationship above). The log 10 of spectral radii are also outputted for each n (gives approximate slope). Illustrative Matlab code below:

```
Legend=cell(7,1);%this is used so we can label the lines in the plot
for n=3:10 %I investigate convergence from n=3 to 10
    b = [2; zeros(n-2,1); 2];%first lines in loop define A,b and the
 other appropriate matrices that we need
    A =  2 * diag(ones(1,n  )) ...
    -1 * diag(ones(1,n-1),-1) ...
    -1 * diag(ones(1,n-1), 1);
    D=diag(diag(A));
    L=(-1)*tril(A,-1);
    U=(-1)*triu(A,1);
    invD=diag(1./diag(A));
    Tj=invD*(L+U);%this is the Jacobi matrix
    cj=invD*b;%this is the Jacobi constant vector
    x0=ones(n,1);%defines vector of initial estimates
    x=2*ones(n,1);%defines exact solution to system

    K=0:1:50;%generates vector for use in plot to show which estimate
 is used in the error
    Nmax=50;%want to investigate convergence from first 50 estimates
    xJac=genIterMeth(Tj,cj,x0,Nmax);%generates nx(Nmax+1) matrix of
 estimate column vectors
    l2J=zeros(51,1);%defines zero vector of appropriate length (more
 efficient to do this then assign values that replace zeros)
    for i=1:51%this for loop generates vector with l2-norm errors for
 each estimate
        errorJ=x-xJac(:,i);
        l2J(i)=norm(errorJ);
    end
    semilogy(K,l2J);%generates log10 plot of error against estimate
 number
    Legend{n-2}=strcat('n=',num2str(n));%this allows us to label each
 line with its respective n value
    hold all
    specTj=max(abs(eig(Tj)));%finds the spectral radius of the n-th
 Jacobi matrix
    slopes=log10(specTj);%takes log10 of this radius
    disp(['The value of the slope for n=',num2str(n), ' is
 approximately ' num2str(slopes,15)]);%outputs approximate slope from
 the error approx.
end
title('Plot showing log10 error for differing values of n');%rest of
 this code formats the graph
xlabel('k (Estimation Number)');
ylabel('log10(2-Norm Error)');
axis([0,50,0,10]);
legend(Legend,'location','best');
```

```
hold off
```

```
The value of the slope for n=3 is approximately -0.15051499783199
The value of the slope for n=4 is approximately -0.0920423554140024
The value of the slope for n=5 is approximately -0.0624693683041498
The value of the slope for n=6 is approximately -0.0452902153825133
The value of the slope for n=7 is approximately -0.0343846540790543
The value of the slope for n=8 is approximately -0.0270141835570636
The value of the slope for n=9 is approximately -0.0217936744549869
The value of the slope for n=10 is approximately -0.0179582012637099
```



Question 5

New method: Repetition of Q1, Q2, Q3

```
n=4;%first few lines redefine the vectors we are investigating
A =  3 * diag(ones(1,n  )) ...
    -1 * diag(ones(1,n-1),-1) ...
    -1 * diag(ones(1,n-1), 1);
b = [4; 2*ones(n-2,1); 4];
format short
D=diag(diag(A)); %next 3 lines decompose A into D, L and U
L=(-1)*tril(A,-1);
U=(-1)*triu(A,1);
```

```
toinv=0.8*D-L;%defines multiplier of xk in parameter-dependent method
matmulti=(-0.2)*D+U;%defines multiplier of xk-1 in parameter-dependent
 method
Tp=toinv\matmulti;%gives the transformation matrix for this method
disp('The parameter-dependent transformation matrix is defined as:');
disp(Tp);%outputs above result
cp=toinv\b;%calculates constant vector for this method
disp('The constant vector for the parameter-dependent method is
 defined as:');
disp(cp);%outputs above result
x0=[1;1;1;1];%defines initial approximation
Nmax=16;%this is the number of estimate vectors we require
format long g
xlambda=genIterMeth(Tp,cp,x0,Nmax);%calls function to produce matrix
 of estimate vectors
x1lambda=xlambda(1,:)';%next two lines extract the appropriate
 estimates and transpose the vectors
x2lambda=xlambda(2,:)';
K=0:1:16;
k=K';
lambdatable=table(k,x1lambda,x2lambda);%produces table with results
disp('Table showing how estimates evolve for parameter-dependent
 method (lambda=0.8):');
disp(lambdatable);%outputs table
l2p=zeros(17,1);%defines zero vector before input in loop (for
 efficiency)
x=[2;2;2;2];%defines the exact solution vector
for i=1:17%this for loop generates a vector of the l2-norms of the
 error in each estimate vector
    errorp=x-xlambda(:,i);
    l2p(i)=norm(errorp);
end
semilogy(K,l2p,'k');%plots curve
title('Log plot showing error the parameter-dependent
 method(lambda=0.8)');%next few lines format the plot
xlabel('k(estimation number)');
ylabel('log10(2-norm error)');
axis([0,16,0,25]);
logerrorp=log10(l2p);%takes log base 10 of the error norms so we can
 attempt to find average slope
slopep=diff(logerrorp);%next two lines find the average slope (using
 same method as I did for Jacobi/Gauss-Seidel which gave reasonably
 accurate values for slope)
alphap=mean(slopep);
disp(['The approximate value for the observed slope of the error line
 is ',num2str(alphap,15)]);%outputs result
speclambda=max(abs(eig(Tp)));%next two lines compute approx. slope
 using theory as before (just to compare for interest)
slopelamb=log10(speclambda);
disp(['The approximate value for the observed slope using theory
 involving the spectral radius is ',num2str(slopelamb,15)]);
```

```
The parameter-dependent transformation matrix is defined as:
   -0.2500    0.4167         0         0
```

```
   -0.1042   -0.0764    0.4167         0
   -0.0434   -0.0318   -0.0764    0.4167
   -0.0181   -0.0133   -0.0318   -0.0764
```

The constant vector for the parameter-dependent method is defined as:

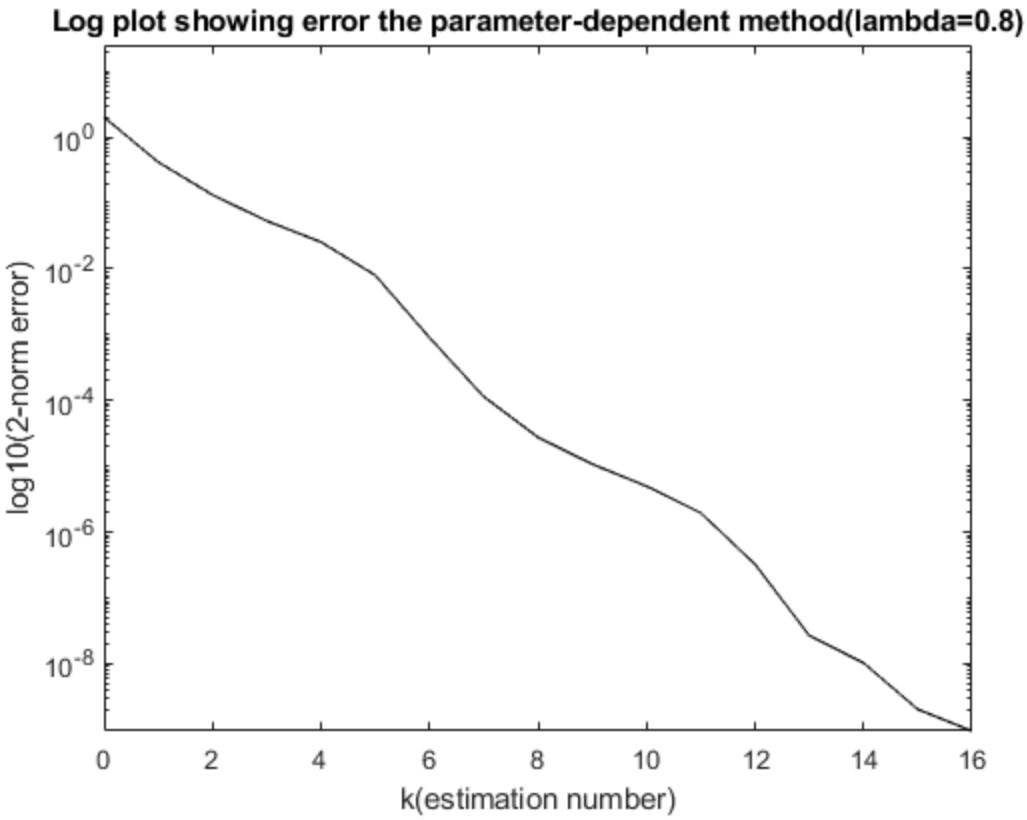
```
    1.6667
    1.5278
    1.4699
    2.2791
```

Table showing how estimates evolve for parameter-dependent method (lambda=0.8):

k	x1lambda	x2lambda
0	1	1
1	1.83333333333333	1.76388888888889
2	1.94328703703704	1.92496141975309
3	1.98291216563786	2.05045104059499
4	2.02529322550512	1.99766272376394
5	1.99270282852536	1.9969880190099
6	2.00056930078945	2.00056371710516
7	2.00009255692979	2.00003871412272
8	1.99999299165202	1.99999919854427
9	2.00000141814711	1.99998966456017
10	1.99999533902996	2.00000170696099
11	2.00000187647625	2.00000055064384
12	1.99999976031587	1.99999980176692
13	1.99999997732392	2.00000000454811
14	2.00000000756406	2.00000000335136
15	1.99999999950538	2.00000000152163
16	2.00000000075767	1.999999999424

The approximate value for the observed slope of the error line is -0.582551377490665

The approximate value for the observed slope using theory involving the spectral radius is -0.602059991327962



Question 6*

Answer to open question: From Q5, we are given a clue that for some range of lambda less than 1 (Gauss-Seidel method) convergence is quicker, since lambda=0.8 gave a more negative slope for the log10 error. Therefore to find the infimum of the set of lambda giving more rapid convergence, we can use a for loop that breaks as soon as the value of the spectral radius of Tp is less than that of the transformation matrix of Gauss-Seidel. This is done in the first for loop below. To test that this is the only region providing faster convergence I replaced the minimum for i by 1.00001 and the maximum with 100. This didn't break the for loop therefore it seems reasonable that this would be the only region of lambda that gives quicker convergence. Since it's intuitive that there must also be a range less than 1 with slower convergence than the Jacobi method due to the result obtained from the first for loop (under the constraint lambda>0), the supremum of this interval is found in the second for loop. Lastly, there is another region where convergence is slower where lambda>1, which is found in the final for loop. Illustrative Matlab code below:

```
n=4;%first few lines redefine the vectors we are investigating
A =  3 * diag(ones(1,n  )) ...
    -1 * diag(ones(1,n-1),-1) ...
    -1 * diag(ones(1,n-1), 1);
b = [4; 2*ones(n-2,1); 4];
format long g
D=diag(diag(A)); %next 3 lines decompose A into D, L and U
L=(-1)*tril(A,-1);
U=(-1)*triu(A,1);
Tg=(D-L)\U;%next 2 lines redefine transformation matrices for Jacobi/
Gauss-Seidel methods
Tj=D\(L+U);
```

```
specg=max(abs(eig(Tg)));%next 2 lines calculate the spectral radius of
 each transformation matrix
specj=max(abs(eig(Tj)));
for i=0.00001:0.00001:1 %first loop investigates quicker convergence
 (than Gauss-Seidel)
    toinv=i*D-L;%first few lines define the transformation matrix
 under this value of lambda
    matmulti=(i-1)*D+U;
    Tp=toinv\matmulti;
    specp=max(abs(eig(Tp)));%calculates spectral radius of the
 transformation matrix above
    if specp<specg %this condition finds the first lambda that gives a
 spectral radius less than that of Gauss-Seidel
        break
    end
end
disp(['The range of lambda giving quicker convergence is
 ',num2str(i),'<lambda<1']);
for j=0.00001:0.00001:1 %second loop investigates slower convergence
 (than Jacobi) in interval between 0 and 1
    toinv=j*D-L;%first few lines define the transformation matrix
 under this value of lambda
    matmulti=(j-1)*D+U;
    Tp=toinv\matmulti;
    specp=max(abs(eig(Tp)));%calculates spectral radius of the
 transformation matrix above
    if specp<specj %this condition finds the first lambda with
 spectral radius less than that of Jacobi (faster convergence)
        break
    end
end
disp(['The range of lambda with slower convergence between 0 and 1 is
 0<lambda<',num2str(j)]);
for k=1.00001:0.00001:3 %final loop investigates slower convergence
 for lambda>1 (than Jacobi)
    toinv=k*D-L;%first few lines define the transformation matrix
 under this value of lambda
    matmulti=(k-1)*D+U;
    Tp=toinv\matmulti;
    specp=max(abs(eig(Tp)));%calculates spectral radius of the
 transformation matrix above
    if specp>specj %this condition finds the first lambda with a
 spectral radius greater than that of Jacobi (slower convergence)
        break
    end
end
disp(['There is also slower convergence for lambda>',num2str(k)]);
```

```
The range of lambda giving quicker convergence is 0.77466<lambda<1
The range of lambda with slower convergence between 0 and 1 is
 0<lambda<0.64963
There is also slower convergence for lambda>1.311
```

Published with MATLAB® R2019a