APPM3021 Numerical Methods • Lab 2
Tyson Cross 1239448

Exercises

exercise1.m :

```
% APPM3021 Lab 2, Exercise 1

clc
clear all

% Input system of equations
rows = 8;
A = generateDiagonallyDominantMatrix(rows)
b = randi(10,rows,1)

% Iteration parameters
x_0 = zeros(length(b),1);
tol = 0.00001

% Iterative attempt at solution
tic;
[sol_jac, iter_jac] = jacobi(A,b,x_0,tol);
time_jac = toc;
correct_solution = A\b;

% Display results
displaySolution(sol_jac, iter_jac, tol, correct_solution, time_jac, 2)
```

When exercise1.m is run in the workspace, the following output is displayed to the command window:

```
A =

    1.0000    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250
    0.5000   41.0000    0.6667    0.5000    0.4000    0.3333    0.2857    0.2500
    0.3333    0.6667   71.0000    0.7500    0.6000    0.5000    0.4286    0.3750
    0.2500    0.5000    0.7500   21.0000    0.8000    0.6667    0.5714    0.5000
    0.2000    0.4000    0.6000    0.8000   21.0000    0.8333    0.7143    0.6250
    0.1667    0.3333    0.5000    0.6667    0.8333  -39.0000    0.8571    0.7500
    0.1429    0.2857    0.4286    0.5714    0.7143    0.8571   -9.0000    0.8750
    0.1250    0.2500    0.3750    0.5000    0.6250    0.7500    0.8750   41.0000


b =

     9
     8
     1
     7
     5
     5
     2
     9


tol =

   1.0000e-05



ans =

    8.8813
    0.0830
   -0.0321
    0.2205
    0.1435
   -0.0804
   -0.0440
    0.1898

The solution is correct
The solution is inaccurate by a maximum difference of 4.1411e-06
The solution has a norm of 9.6757e-07
The solution was calculated in 0.0079639 seconds
The solution converged within 6 iterations
```

---

exercise2.m :

```
% APPM3021 Lab 2, Exercise 2

clc
clear all

% Input system of equations
rows = 8;
```

```
A=generateDiagonallyDominantMatrix(rows)
b = randi(10,rows,1)

% Iteration parameters
x_0 = zeros(length(b),1);
tol = 0.00001

% Iterative attempt at solution
tic;
[sol_gss, iter_gss] = gaussSeidel(A,b,x_0,tol);
time_gss = toc;
correct_solution = A\b;

% Display results
displaySolution(sol_gss, iter_gss, tol, correct_solution, time_gss, 2)
```

When exercise2.m is run in the workspace, the following output is displayed to the command window:

```
A =

  -39.0000    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250
    0.5000  -49.0000    0.6667    0.5000    0.4000    0.3333    0.2857    0.2500
    0.3333    0.6667  -29.0000    0.7500    0.6000    0.5000    0.4286    0.3750
    0.2500    0.5000    0.7500  -29.0000    0.8000    0.6667    0.5714    0.5000
    0.2000    0.4000    0.6000    0.8000   11.0000    0.8333    0.7143    0.6250
    0.1667    0.3333    0.5000    0.6667    0.8333   11.0000    0.8571    0.7500
    0.1429    0.2857    0.4286    0.5714    0.7143    0.8571  -19.0000    0.8750
    0.1250    0.2500    0.3750    0.5000    0.6250    0.7500    0.8750  -19.0000


b =

     6
     7
    10
     8
     5
     9
     2
     1


tol =

   1.0000e-05



ans =

   -0.1547
   -0.1429
   -0.3345
   -0.2589
    0.4428
    0.8294
   -0.0710
   -0.0249

The solution is correct
The solution is inaccurate by a maximum difference of 7.3808e-07
The solution has a norm of 5.3409e-05
The solution was calculated in 0.0066872 seconds
The solution converged within 4 iterations
```

---

exercise3.m :

```
% APPM3021 Lab 2, Exercise 3

clc
clear all

% Input system of equations
rows = 8;
A=generateDiagonallyDominantMatrix(rows)
b = randi(10,rows,1)

% Iteration parameters
x_0 = zeros(length(b),1);
tol = 0.00001

% Iterative attempt at solution
tic
[sol_sor, iter_sor] = SOR(A,b,x_0,tol);
time_sor = toc;
correct_solution = A\b;

% Display results
displaySolution(sol_sor, iter_sor, tol, correct_solution, time_sor, 2)
```

When exercise3.m is run in the workspace, the following output is displayed to the command window:

```
A =

  -89.0000    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250
    0.5000  -69.0000    0.6667    0.5000    0.4000    0.3333    0.2857    0.2500
    0.3333    0.6667  -39.0000    0.7500    0.6000    0.5000    0.4286    0.3750
    0.2500    0.5000    0.7500  -39.0000    0.8000    0.6667    0.5714    0.5000
    0.2000    0.4000    0.6000    0.8000  -99.0000    0.8333    0.7143    0.6250
    0.1667    0.3333    0.5000    0.6667    0.8333   11.0000    0.8571    0.7500
    0.1429    0.2857    0.4286    0.5714    0.7143    0.8571  -79.0000    0.8750
    0.1250    0.2500    0.3750    0.5000    0.6250    0.7500    0.8750  -69.0000


b =

     7
     9
    10
     6
    10
     6
     6
     4


tol =

   1.0000e-05



ans =

   -0.0801
   -0.1329
   -0.2577
   -0.1548
   -0.1005
    0.5890
   -0.0742
   -0.0566

The solution is correct
The solution is inaccurate by a maximum difference of 4.4589e-08
The solution has a norm of 8.4817e-06
The solution was calculated in 0.007775 seconds
The solution converged within 4 iterations
```

---

exercise4.m :

```
% APPM3021 Lab 2, Exercise 4

clc
clear all
digits(32)
% dbstop if error

tol = 0.000001;

%% Generate
n = 100;
% A = generateDiagonallyDominantMatrix(n);
% dlmwrite('Data/matrix.txt',A,'precision',3);
% matrix2latexmatrix(A,'Data/matrix_values.tex');
A = dlmread('Data/matrix.txt');
b = randi(10,n,1);
x_0 = zeros(n,1);

%% Measure
tic
[sol_jac, iter_jac] = jacobi(A,b,x_0,tol);
time_jac = toc; tic;
[sol_gss, iter_gss] = gaussSeidel(A,b,x_0,tol);
time_gss = toc; tic;
[sol_sor, iter_sor] = SOR(A,b,x_0,tol);
time_sor = toc;

%% Relative Errors
error_jac = zeros(iter_jac,1);
error_gss = zeros(iter_gss,1);
error_sor = zeros(iter_sor,1);


for index=2:iter_jac+1
    difference = abs(sol_jac(:,index) - sol_jac(:,index-1));
    error_jac(index) = max(difference)/max(abs(sol_jac(:,index)));
end
for index=2:iter_gss+1
    difference = abs(sol_gss(:,index) - sol_gss(:,index-1));
    error_gss(index) = max(difference)/max(abs(sol_gss(:,index)));
end
```

```matlab
for index=2:iter_sor+1
    difference = abs(sol_sor(:,index) - sol_sor(:,index-1));
    error_sor(index) = max(difference)/max(abs(sol_sor(:,index)));
end

% remove the empty first entry
error_jac(1) = [];
error_gss(1) = [];
error_sor(1) = [];

%% Display setting and output setup
scr = get(groot,'ScreenSize');                              % screen resolution
fig1 =  figure('Position',...                               % draw figure
    [1 scr(4)*3/5 scr(3)*3.5/5 scr(4)*3/5]);
set(fig1,'numbertitle','off',...                            % Give figure useful title
    'name','Comparison of iterative matrix methods',...
    'Color','white');
set(fig1, 'MenuBar', 'none');                             % Make figure clean
set(fig1, 'ToolBar', 'none');
% fontName='CMU Serif';
fontName='Helvetica';
set(0,'defaultAxesFontName', fontName);                     % Make fonts pretty
set(0,'defaultTextFontName', fontName);
set(groot,'FixedWidthFontName', 'ElroNet Monospace')

%% Plot
p1 = semilogy(error_jac,...
        'Color',[0.18 0.18 0.9 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p2 = semilogy(error_gss,...
        'Color',[0.18 0.9 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p3 = semilogy(error_sor,...
        'Color',[0.9 0.18 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
% p4 = refline(0,tol);
% set(p4,'Color',[0.18 0.18 0.18 .6],...
%         'LineStyle',':',...
%         'LineWidth',1);
% hold on

% Title
title('Relative Error vs. Iterations',...
    'FontSize',14,...
    'FontName',fontName);

% Axes and labels
ax1 = gca;
% hold(ax1,'on');
ylabel('Relative Error',...
    'FontName',fontName,...
    'FontSize',14);%,...
xlabel('Number of Iterations',...
    'FontName',fontName,...
    'FontSize',14);
xlim(ax1,[1 iter_jac(1,1)]);
box(ax1,'off');
set(ax1,'FontSize',14,...
    'XTick',[0:5:iter_jac(1,1)],...
    'XTickLabelRotation',45,...
    'YMinorTick','on');hold on

% Legend
legend1 = legend({'Jacobi','Gauss-Seidel','SOR'},...
     'Position',[0.7    0.7    0.2    0.09],...
     'Box','off');
hold off
% epswrite('images/relative_error.eps');
```

Figure 1. Measured matrix (100x100)

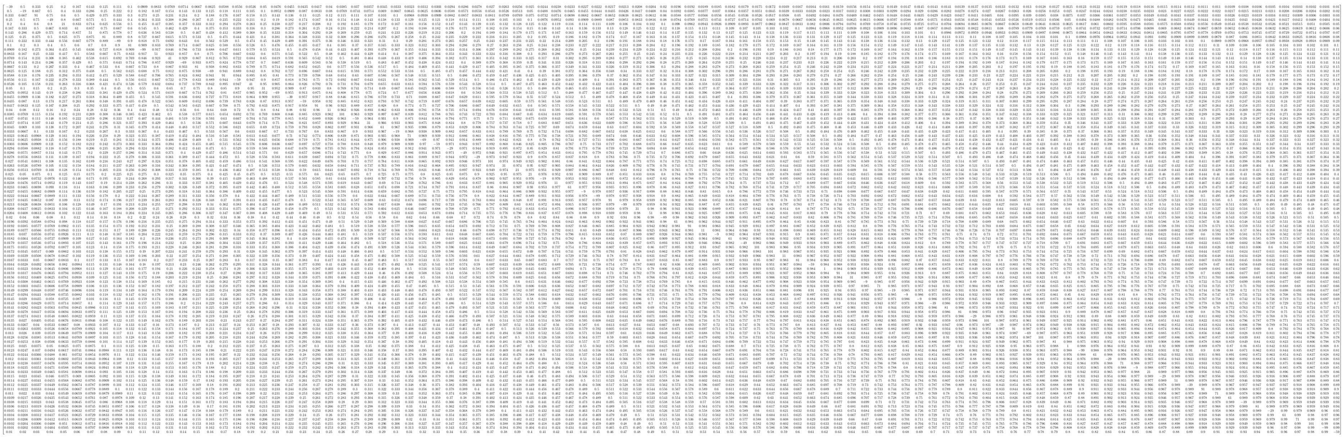
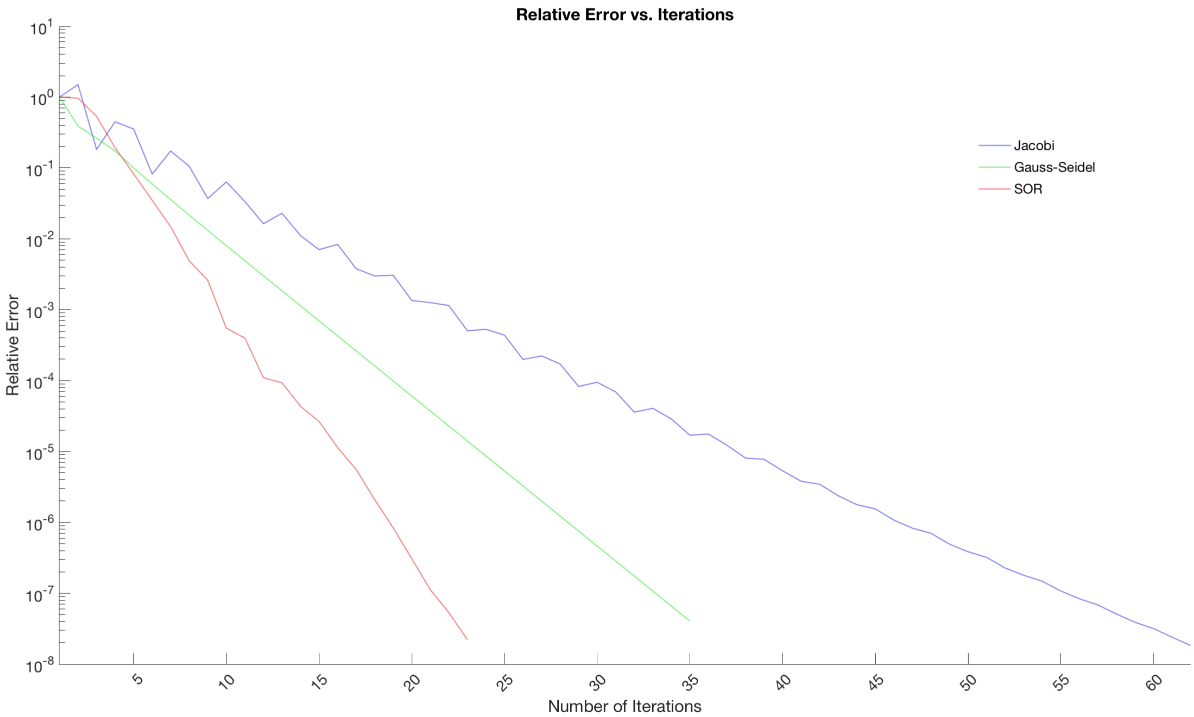


Figure 2. Comparison of iterative matrix methods



Observances:

```
The figure shown plots and compares the three methods, iterating on the same 100x100 matrix.
The matrix chosen is strongly diagonally dominant, with real, integer eigenvalues. The rcond() value for the
 matrix is > 0.1 and the spectral radius (rho) of coefficient matrix B (where Ax=Bx+c) is rho<1.
Iteratively solving a large, non-sparse matrix (requiring a high number of iterations) produces much more
 resolution in the curves, and more accurately represents the general character of the algorithmns, which can be
 hard to observe on small, simple or sparse matrices which converge in few iterations.

Jacobi is the slowest method for convergence, taking the most number of iterations to converge to a solution.

Gauss-Seidel is on average much faster then the Jacobi method, and can outperform the Successive Over-relaxation
 method when very few iterations (<5) are required.

The SOR method usually converges much faster than the other two methods, although for the first iterations, the
 early relative errors can be slightly larger than Gauss-Seidel. In general it produces much faster convergence,
 as can be seen in the example graph.
```

Questions

Question 1a)

```
% APPM3021 Lab 2, Question 1a

clc
clear all

% Input system of equations
A = [ 2,   1,  -1,  2;...
      4,   5,  -3,  6;...
     -2,   5,  -2,  6;...
      4,  11,  -4,  8]
 b = [5; 9; 4; 2]

% Iteration parameters
x_0 = zeros(length(b),1);
tol = 0.00001

% Iterative attempt at solution
if ~isSolvable(A)
    return
elseif ~converges(A)
    return
else
    tic;
    [sol_jac, iter_jac] = jacobi(A,b,x_0,tol);
    time_jac = toc; tic;
    [sol_gss, iter_gss] = gaussSeidel(A,b,x_0,tol);
    time_gss = toc; tic;
    [sol_sor, iter_sor] = SOR(A,b,x_0,tol);
    tims_sor = toc;
end

% Display results
correct_solution = A\b;
displaySolution(sol_jac, iter_jac, tol, correct_solution, time_jac, 1, 'Jacobi')
displaySolution(sol_gss, iter_gss, tol, correct_solution, time_gss, 1, 'Gauss-Seidel')
displaySolution(sol_sor, iter_sor, tol, correct_solution, time_sor, 1, 'SOR')
```

When question1a.m is run in the workspace, the following output is displayed to the command window:

```
A =

     2     1    -1     2
     4     5    -3     6
    -2     5    -2     6
     4    11    -4     8


b =

     5
     9
     4
     2


tol =

   1.0000e-05

The matrix will not iteratively converge to unique solution:
  ·The norm ||B||_inf is not less than 1
  ·The spectral radius rho(B) is not less than 1
```

Question 1b)

```
% APPM3021 Lab 2, Question 1b

clc
clear all

A = [ 3, 1, -1;...
     1, -4, 2;...
     -2, -1, 5]

 b = [3; -1; 2]

x_0 = zeros(length(b),1);
tol = 0.00001

if ~isSolvable(A)
    return
elseif ~converges(A)
```

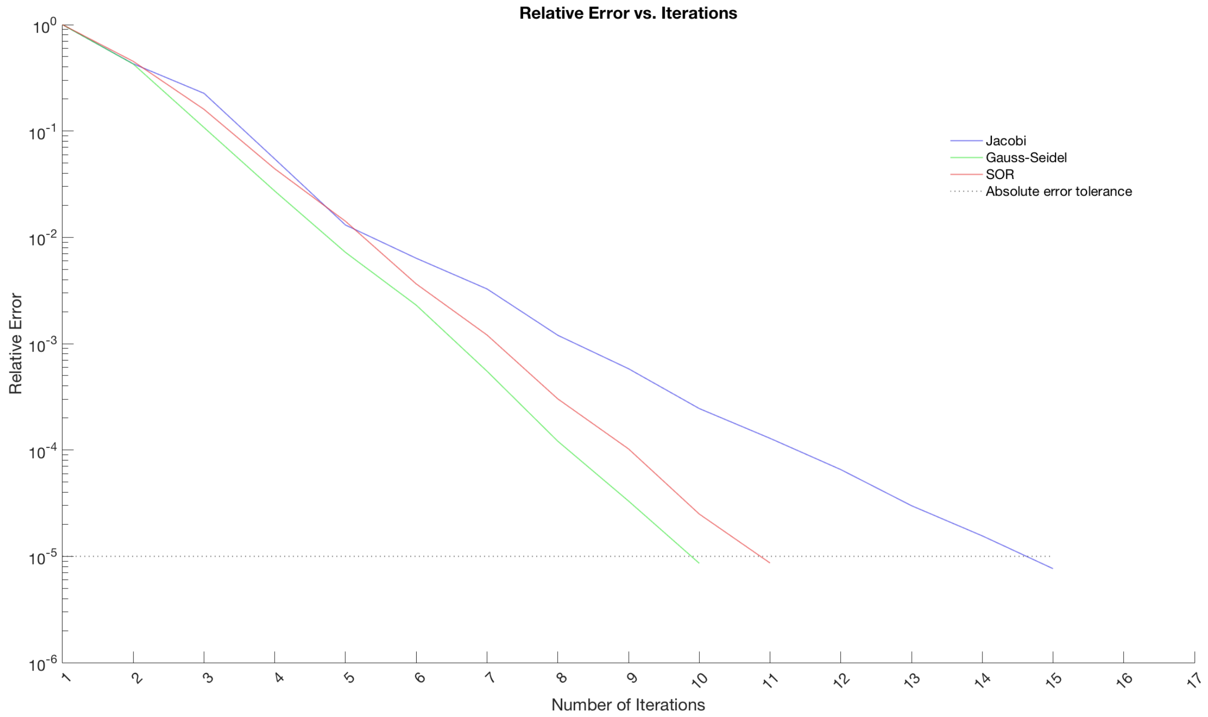
```
    return
else
    tic;
    [sol_jac, iter_jac] = jacobi(A,b,x_0,tol);
    time_jac = toc; tic;
    [sol_gss, iter_gss] = gaussSeidel(A,b,x_0,tol);
    time_gss = toc; tic;
    [sol_sor, iter_sor] = SOR(A,b,x_0,tol);
    time_sor = toc; tic;
end

%% Relative Errors
error_jac = zeros(iter_jac,1);
error_gss = zeros(iter_gss,1);
error_sor = zeros(iter_sor,1);


for index=2:iter_jac+1
    difference = abs(sol_jac(:,index) - sol_jac(:,index-1));
    error_jac(index) = max(difference)/max(abs(sol_jac(:,index)));
end
for index=2:iter_gss+1
    difference = abs(sol_gss(:,index) - sol_gss(:,index-1));
    error_gss(index) = max(difference)/max(abs(sol_gss(:,index)));
end
for index=2:iter_sor+1
    difference = abs(sol_sor(:,index) - sol_sor(:,index-1));
    error_sor(index) = max(difference)/max(abs(sol_sor(:,index)));
end

% remove the empty first entry
error_jac(1) = [];
error_gss(1) = [];
error_sor(1) = [];

correct_solution = A\b;

%% Output and check
displaySolution(sol_jac, iter_jac, tol, correct_solution, time_jac, 1, 'Jacobi')
displaySolution(sol_gss, iter_gss, tol, correct_solution, time_gss, 1, 'Gauss-Seidel')
displaySolution(sol_sor, iter_sor, tol, correct_solution, time_sor, 1, 'SOR')

%% Display setting and output setup
scr = get(groot,'ScreenSize');                             % screen resolution
fig2 =  figure('Position',...                              % draw figure
    [1 scr(4)*3/5 scr(3)*3.5/5 scr(4)*3/5]);
set(fig2,'numbertitle','off',...                           % Give figure useful title
    'name','Comparison of iterative matrix methods',...
    'Color','white');
set(fig2, 'MenuBar', 'none');                             % Make figure clean
set(fig2, 'ToolBar', 'none');
% fontName='CMU Serif';
fontName='Helvetica';
set(0,'defaultAxesFontName', fontName);                    % Make fonts pretty
set(0,'defaultTextFontName', fontName);
set(groot,'FixedWidthFontName', 'ElroNet Monospace')

%% Plot
p1 = semilogy(error_jac,...
        'Color',[0.18 0.18 0.9 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p2 = semilogy(error_gss,...
        'Color',[0.18 0.9 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p3 = semilogy(error_sor,...
        'Color',[0.9 0.18 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',1);
hold on
p4 = refline(0,tol);
set(p4,'Color',[0.18 0.18 0.18 .6],...
        'LineStyle',':',...
        'LineWidth',1);
hold on

% Title
title('Relative Error vs. Iterations',...
    'FontSize',14,...
    'FontName',fontName);

% Axes and labels
ax1 = gca;
% hold(ax1,'on');
ylabel('Relative Error',...
    'FontName',fontName,...
    'FontSize',14);%,...
xlabel('Number of Iterations',...
    'FontName',fontName,...
    'FontSize',14);
xlim(ax1,[1 iter_jac(1,1)+2]);
box(ax1,'off');
```

```
set(ax1,'FontSize',14,...
    'XTick',[0:1:iter_jac(1,1)+2],...
    'XTickLabelRotation',45,...
    'YMinorTick','on');hold on

% Legend
legend1 = legend({'Jacobi','Gauss-Seidel','SOR','Absolute error tolerance'},...
     'Position',[0.7    0.7    0.2    0.09],...
     'Box','off');
hold off
% epswrite('images/relative_error_1b.eps');
```

When question1b.m is run in the workspace, the following output is displayed to the command window:

```
A =

     3     1    -1
     1    -4     2
    -2    -1     5


b =

     3
    -1
     2


tol =

   1.0000e-05


Jacobi solution:

ans =

    1.0000
    1.0000
    1.0000

Jacobi solution is correct
Jacobi solution converged within 15 iterations

Gauss-Seidel solution:

ans =

    1.0000
    1.0000
    1.0000

Gauss-Seidel solution is correct
Gauss-Seidel solution converged within 10 iterations

SOR solution:

ans =

    1.0000
    1.0000
    1.0000

SOR solution is correct
SOR solution converged within 11 iterations
```

Figure 3. Comparison of iterative matrix methods



Question 2

```
% APPM3021 Lab 2, Question 2

clc
clear all

% Input system of equations
disp('Question 2')
disp(' ')

A = [ 4, -1,  0, -1,  0,  0,  0,  0,  0;...
     -1,  4, -1,  0, -1,  0,  0,  0,  0;...
      0, -1,  4,  0,  0, -1,  0,  0,  0;...
     -1,  0,  0,  4, -1,  0, -1,  0,  0;...
      0, -1,  0, -1,  4, -1,  0, -1,  0;...
      0,  0, -1,  0, -1,  4,  0,  0, -1;...
      0,  0,  0, -1,  0,  0,  4, -1,  0;...
      0,  0,  0,  0, -1,  0, -1,  4, -1;...
      0,  0,  0,  0,  0, -1,  0, -1,  4]
  disp(' ')

%% Question 2a)
disp('a)')
disp(' ')
if isDiagonallyDominant(A)
    disp('The matrix is diagonally dominant')
else
    disp('The matrix is not diagonally dominant')
end
disp(' ')
disp(' ')

%% Question 2b)
disp('b)')
disp(' ')
[L, D, U] = LDU(A);
B = -D\(L+U);
rho_j = max(abs(eig(B)));
rho_gs = rho_j^2;
disp('The spectral radii rho(B) of the Jacobi and Gauss-Seidel iteration matrices are as follows:')
disp(['rho_j  = ',num2str(rho_j)])
disp(['rho_gs = ',num2str(rho_gs) ])
disp(' ')
disp(' ')

%% Question 2c)
disp('c)')
disp(' ')
eigenvalue = max(abs(eig(B)));
omega = 2/(1+sqrt(1-eigenvalue^2));
disp(['The optimal value of the relaxation parameter omega is ', num2str(omega)]);
```

```
disp(' ')
disp(' ')

%% Question 2d)
disp('d)')
disp(' ')
rho_s = omega-1;
B_sor = (D+omega*L)\((1-omega)*D-omega*U);
rho_s2 = max(abs(eig(B_sor)));

disp(['The spectral radii rho(B) rho_s = omega - 1 = ', num2str(rho_s)]);
disp(['The spectral radii rho(B) of the SOR iteration matrix = max|lambda| = ', num2str(rho_s2)]);

if round(rho_s,4)==round(rho_s2,4)
    disp('(The values agree)')
else
    disp('(The values do NOT agree!)')
end
disp(' ')
disp(' ')

%% Question 2e)
disp('e)')
disp(' ')
n = length(A);
% setup the e_1 vector for the problem
e_1=zeros(n,1);
e_1(1,1)=1;
x_0 = zeros(n,1);
tol = 0.0001;

% Prediction
log_error = log(tol);
log_rho_j = log(rho_j);
log_rho_gs = log(rho_gs);
log_rho_s = log(rho_s);
i_j = log_error/log_rho_j;
i_gs = log_error/log_rho_gs;
i_s = log_error/log_rho_s;

disp('The predicted number of iterations is calculated for each method,')
disp(' using the spectral radii equations above, using the formula:')
disp(' ')
disp('    i <= log(error)/log(rho)')
disp(' ')
disp(['Using a tolerance/error threshold of ',num2str(tol),' for the predictions, then:'])
disp(' ')
disp(['i_j should be less than or equal to log(',...
    num2str(tol),')/log(',num2str(rho_j),') = ',...
    num2str(i_j), ' which is approximately ',...
    num2str(ceil(i_j))])
disp(['i_gs should be less than or equal to log(',...
    num2str(tol),')/log(',num2str(rho_gs),') = ',...
    num2str(i_gs), ' which is approximately ',...
    num2str(ceil(i_gs))])
disp(['i_s should be less than or equal to log(',...
    num2str(tol),')/log(',num2str(rho_s),') = ',...
    num2str(i_s), ' which is approximately ',...
    num2str(ceil(i_s))])
disp(' ')


%% Measure
disp('If the iterative solutions are computed from an initial zero vector then the results of each method are:')
disp(' ')
    tic
[sol_jac, iter_jac] = jacobi(A,e_1,x_0,tol);
    time_jac = toc; tic;
[sol_gss, iter_gss] = gaussSeidel(A,e_1,x_0,tol);
    time_gss = toc; tic;
[sol_sor, iter_sor] = SOR(A,e_1,x_0,tol);
    time_sor = toc;

% Display results
correct_solution = A\e_1;
disp(['Jacobi method converged in ', num2str(iter_jac),' iterations'])
disp(['Gauss-Seidel method converged in ', num2str(iter_gss),' iterations'])
disp(['SOR method converged in ', num2str(iter_sor),' iterations'])
disp(' ')
if iter_jac <= ceil(i_j)
disp('The predictions of the Jacobi method is within predicted bounds')
else disp('The predictions of the no. of iterations of Jacobi is larger than predected')
end

if iter_gss <= ceil(i_gs)
disp('The predictions of the Gauss-Seidel method is within predicted bounds')
else disp('The predictions of the no. of iterations of Gauss-Seidel is larger than predected')
end

if iter_sor <= ceil(i_s)
disp('The predictions of the SOR method is within predicted bounds')
else disp('The predictions of the no. of iterations of SOR is larger than predected')
end
```

The following method is used in Question 2e) to predict the number of iterations for each method:

$$\mathbf{e}^{(k)} = \overline{\mathbf{x}} - \overline{\mathbf{x}}^{(k+1)} = \left(\mathbf{B}\overline{\mathbf{x}} + \overline{\mathbf{c}}\right) - \left(\mathbf{B}\overline{\mathbf{x}}^{(k)} + \overline{\mathbf{c}}\right) = \mathbf{B}\left(\overline{\mathbf{x}} - \overline{\mathbf{x}}^{(k+1)}\right) = \mathbf{B}\mathbf{e}^{(k)} \quad (1)$$

$$\therefore \mathbf{e}^{(k)} = \mathbf{B}^k \mathbf{e}^{(0)} \quad (2)$$

Taking the norm of both sides:

$$||\mathbf{e}^{(k)}|| = ||\mathbf{B}^k \mathbf{e}^{(0)}|| \leq ||\mathbf{B}^k|| ||\mathbf{e}^{(0)}|| \leq ||\mathbf{B}||^k ||\mathbf{e}^{(0)}|| \quad (3)$$

We can also note that for convergence, $||\mathbf{B}|| < 1$ (4)

and that this is equivalent to the requirement $\rho(\mathbf{B}) = max|\lambda_j| < 1$ (5)

Note that for large value of k,

$$||\mathbf{e}^{(k+1)}|| = \rho ||\mathbf{e}^{(k)}||$$

We want to estimate the minimum number of iterations required to calculate, in order to reduce the relative error:

$$\rho^i < \epsilon \quad (6)$$

leading to

$$i \leq \frac{log(\epsilon)}{log(\rho)} \quad (7)$$

We will use equation 7 to estimate the number of iterations, noting the calculation of ρ for each method:

Jacobi: $\rho_j = max|\lambda_j|$

Gauss-Seidel: $\rho_{gs} = \rho_j^2$

SOR: $\rho_s = \omega - 1$, where $\omega = \dfrac{2}{1 + \sqrt{1 - \rho_j^2}}$

When question2.m is run in the workspace, the following output is displayed to the command window:

```
Question 2

A =

     4    -1     0    -1     0     0     0     0     0
    -1     4    -1     0    -1     0     0     0     0
     0    -1     4     0     0    -1     0     0     0
    -1     0     0     4    -1     0    -1     0     0
     0    -1     0    -1     4    -1     0    -1     0
     0     0    -1     0    -1     4     0     0    -1
     0     0     0    -1     0     0     4    -1     0
     0     0     0     0    -1     0    -1     4    -1
     0     0     0     0     0    -1     0    -1     4
```

```
a)

The matrix is diagonally dominant


b)

The spectral radii rho(B) of the Jacobi and Gauss-Seidel iteration matrices are as follows:
rho_j  = 0.70711
rho_gs = 0.5


c)

The optimal value of the relaxation parameter omega is 1.1716


d)

The spectral radii rho(B) rho_s = omega - 1 = 0.17157
The spectral radii rho(B) of the SOR iteration matrix = max|lambda| = 0.17157
(The values agree)


e)

The predicted number of iterations is calculated for each method,
 using the spectral radii equations above, using the formula:

    i <= log(error)/log(rho)

Using a tolerance/error threshold of 0.0001 for the predictions, then:

i_j should be less than or equal to log(0.0001)/log(0.70711) = 26.5754 which is approximately 27
i_gs should be less than or equal to log(0.0001)/log(0.5) = 13.2877 which is approximately 14
i_s should be less than or equal to log(0.0001)/log(0.17157) = 5.225 which is approximately 6

If the iterative solutions are computed from an initial zero vector then the results of each method are:

Jacobi method converged in 26 iterations
Gauss-Seidel method converged in 12 iterations
SOR method converged in 6 iterations

The predictions of the Jacobi method is within predicted bounds
The predictions of the Gauss-Seidel method is within predicted bounds
The predictions of the SOR method is within predicted bounds
```

Functions and Code

jacobi.m :

```
function [x,iterationCount] = jacobi(A,b,x_0,tol)
% Jacobi uses an iterative technique to estimate the solution
% to a given system of equations within a specified tolerance using
% the Jacobi method

if ~isSolvable(A)                                   % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

x = x_0;

% check convergence
if ~converges(A)
    disp('The matrix does not converge')
    iterationCount = 0;
    return
end

[L, D, U] = LDU(A);
correct_solution = A\b;

iterationCount = 1;
while true
    y=b-(L+U)*x(:,iterationCount);
    x(:,iterationCount+1)=D\y;
    err_norm = sum(abs(correct_solution - x(:,iterationCount+1)));
    if err_norm <= tol
        break;
    end
    if isnan(err_norm)
        error(['Solution at index(',num2str(iterationCount),' has NaN entry'])
    end
        if isinf(err_norm)
        error(['Solution at index(',num2str(iterationCount),' has Inf entry'])
    end
    iterationCount=iterationCount+1;
end

end
```

gaussSeidel.m :

```
function [X,iterationCount] = gaussSeidel(A,b,x_0,tol)
% gaussSeidel uses an iterative technique to estimate the solution
% to a given system of equations within a specified tolerance using
% the Gauss-Seidel method

if ~isSolvable(A)                                   % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

X = x_0;

% check convergence
if ~converges(A)
    disp('The matrix does not converge')
    iterationCount = 0;
    return
end

correct_solution = A\b;
[L, D, U] = LDU(A);
% B = -(D+L)\U;

iterationCount = 1;
while true
    y=b-U*X(:,iterationCount);
    X(:,iterationCount+1)=(L+D)\y;
    err_norm = sum(abs(correct_solution - X(:,iterationCount+1)));
    if err_norm <= tol
        break;
```

```
    end
    if isnan(err_norm)
        error(['Solution at index(',num2str(iterationCount),' has NaN entry'])
    end
        if isinf(err_norm)
        error(['Solution at index(',num2str(iterationCount),' has Inf entry'])
    end
    iterationCount=iterationCount+1;
end

end
```

SOR.m :

```
function [x,iterationCount] = SOR(A,b,x_0,tol)
% SOR uses an iterative technique to estimate the solution
% to a given system of equations within a specified tolerance using
% the Successive Over-relaxation (SOR) method

if ~isSolvable(A)                                   % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

x = x_0;

% check convergence
if ~converges(A)
    disp('The matrix does not converge')
    iterationCount = 0;
    return
end

correct_solution = A\b;
[L, D, U] = LDU(A);
B = D\(L+U);
eigenvalue = max(abs(eig(B)));
omega = 2/(1+sqrt(1-eigenvalue^2));
% B_sor = (D+omega*L)\((1-omega)*D-omega*U);

iterationCount = 1;
while true
    y = omega*b + ((1-omega)*D - omega*U)*x(:,iterationCount);
    x(:,iterationCount+1) = (D+omega*L)\y;
    err = sum(abs(correct_solution - x(:,iterationCount+1)));
    if err <= tol
        break;
    end
    if isnan(err)
        error(['Entry at index(',num2str(iterationCount),') has NaN entry'])
    end
    if isinf(err)
        error(['Entry at index(',num2str(iterationCount),') has Inf entry'])
    end
    iterationCount=iterationCount+1;
end

end
```

isSolvable.m :

```
function x = isSolvable( A )
% Checks if input matrix is square and non-singular

x = true;
n = size(A);
if n(1) ~= n(2)
    disp('Matrix is not square')
    x = false;
    return
end

if det(A)==0
    disp('Matrix is singular')
    x = false;
    return
end

if isnan(A)
    disp('Matrix contains NaN values')
    x = false;
    return
end

if isinf(A)
    disp('Matrix contains Inf values')
    x = false;
    return
end

end
```

converges.m :

```
function [ result ] = converges( A )
% Tests to see if a given iterative, coefficient matrix B will converge to a unique solution
% Where Ax = Bx + c

result = false;
[L, D, U] = LDU(A);
B = -D\(L+U);
reason1 = '';
reason2 = '';

if matrixNorm(B) < 1
    result = true;
else
    reason1 = '  ·The norm ||B||_inf is not less than 1 ';
end

rho = max(abs(eig(B)));
if rho < 1
    result = true;
else
    result = false;
    reason2 = '  ·The spectral radius rho(B) is not less than 1 ';
end

if ~result
    disp('The matrix will not iteratively converge to unique solution:')
    disp(reason1)
    disp(reason2)
end

end
```

LDU.m :

```
function [ L, D, U ] = LDU( A )
% LDU splits a given Matrix into
% L = strictly lower triangular matrix of A
% D = a matrix of only the diagonal entries of A
% U = strictly upper triangular matrix of A

L = tril(A,-1);
U = triu(A,1);
D = A-L-U;

end
```

isDiagonallyDominant.m :

```
function [ result ] = isDiagonallyDominant( A )
% Tests a matrix to see if it is diagnoally dominant

result = false;
[n,m] = size(A);

for i=1:n
    sum = 0;
    for j=1:m
        if i~=j
            sum = sum + A(i,j);
        end
    end
    if ~(abs(A(i,i)) > sum)
        result = false;
        return;
    else
        result = true;
    end
end

end
```

matrixNorm.m :

```
function [ output_norm ] = matrixNorm( A )
% Returns the norm of a given matrix

% if isSolvable(A)
    n = length(A);
% else disp('Matrix is not solvable')
% end

output_norm =[];
for row=1:n
    row_sum = 0;
    for column=1:n
        row_sum = row_sum + abs(A(row,column));
    end
```

```
output_norm = max([max(row_sum) output_norm]);
end

end
```

generateDiagonallyDominantMatrix.m :

```
function [ A ] = generateDiagonallyDominantMatrix( n)
% generateMatrix creates a single matrix of integers of size nxn
try_count=0;
rho=2;
redo=false;
A=[1,1;1,1];
while rho >= 1 || redo;
    try_count = try_count +1;
%     A=diag(randi([-10,10],n,1)*10) + randi(10,n,n) + ones(n,n);
    A = diag(randi([-10,10],n,1)*10)+gallery('lehmer',n);

    [L, D, U] = LDU(A);
    B = D\(L+U);
    if isSolvable(B)
        rho = max(abs(eig(B)));
    end
    if ~isSolvable(A) || ~isSolvable(B)
        redo=true;
    else redo=false;
    end
    if try_count > 100
        error('Unable to generate convergent matrix')
    end
%     disp(['A solvable:',num2str(isSolvable(A)),' with rcond =r',num2str(rcond(A))])
%     disp(['B solvable:',num2str(isSolvable(B)),' with rcond =r',num2str(rcond(B))])
end
end
```

displaySolution.m :

```
function displaySolution( solution, iterations, tolerance, correct_solution, timed, verbosity, method )
% Formats output and provides solution parameters and information

if nargin<7
   method = 'The';
end

% disp('___________________________________')
disp(' ')

if verbosity>=1
    round_error = abs(log10(tolerance))-1;
    if ~strcmp(method, 'The')
        disp([method,' solution:'])
    end
end

solution(:,end)

if verbosity>=1

    if isequal(round(solution(:,end),round_error),round(correct_solution,round_error))
        disp([method, ' solution is correct'])
    else disp([method, ' solution is incorrect:'])
        correct_solution
    end
end
if verbosity>=2
    if ~isequal(solution(:,end),correct_solution)
        disp([method,' solution is inaccurate by a maximum difference of ',...
            num2str(max(abs(solution(:,end)-correct_solution)))])
    end
    relative_norm = max(abs(solution(:,end) - solution(:,end-1)))/ max(abs(solution(:,end)));
    disp([method, ' solution has a norm of ', num2str(relative_norm)])
    disp([method, ' solution was calculated in ', num2str(timed), ' seconds'])
end
if verbosity>=1
    disp([method, ' solution converged within ', num2str(iterations), ' iterations'])
end

end
```