APPM3021 Numerical Methods • Lab 2
Tyson Cross 1239448

Exercises

exercise1.m :

```
% APPM3021 Lab 2, Exercise 1

clc
clear all

rows = 8;
A=generateDiagonallyDominantMatrix(rows)
b = randi(10,rows,1)

x_0 = zeros(length(b),1);
tol = 0.00001
round_error = abs(log10(tol))-1;

[solution, iterations] = jacobiMethod(A,b,x_0,tol);

% Output and check
solution(:,end)
correct_solution = A\b;
if isequal(round(solution(:,end),round_error),round(correct_solution,round_error))
    disp('Solution is correct')
else disp('Solution is incorrect:')
    correct_solution
end
if ~isequal(solution(:,end),correct_solution)
    disp(['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(['Solution has a norm of ', num2str(relative_norm)])
disp(['Solution converged within ', num2str(iterations), ' iterations'])
```

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

```
A =

  -59.0000    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250
    0.5000   -9.0000    0.6667    0.5000    0.4000    0.3333    0.2857    0.2500
    0.3333    0.6667  101.0000    0.7500    0.6000    0.5000    0.4286    0.3750
    0.2500    0.5000    0.7500   11.0000    0.8000    0.6667    0.5714    0.5000
    0.2000    0.4000    0.6000    0.8000    1.0000    0.8333    0.7143    0.6250
    0.1667    0.3333    0.5000    0.6667    0.8333  -59.0000    0.8571    0.7500
    0.1429    0.2857    0.4286    0.5714    0.7143    0.8571    1.0000    0.8750
    0.1250    0.2500    0.3750    0.5000    0.6250    0.7500    0.8750   31.0000


b =

     7
     4
     4
    10
     1
     9
    10
     8


tol =

   1.0000e-05


ans =

   -0.1164
   -0.3792
    0.0332
    0.8948
```

```
  -13.0783
   -0.0535
   19.0087
   -0.0248

Solution is incorrect:

correct_solution =

   -0.1164
   -0.3792
    0.0332
    0.8948
  -13.0783
   -0.0535
   19.0086
   -0.0248

Solution is inaccurate by a maximum difference of 4.0353e-06
Solution has a norm of 4.7349e-07
Solution converged within 61 iterations
```

exercise2.m :

```
% APPM3021 Lab 2, Exercise 2

clc
clear all

rows = 8;
A=generateDiagonallyDominantMatrix(rows)
b = randi(10,rows,1)

x_0 = zeros(length(b),1);
tol = 0.00001
round_error = abs(log10(tol))-1;

[solution, iterations] = gaussSeidel(A,b,x_0,tol);

% Output and check
solution(:,end)
correct_solution = A\b;
if isequal(round(solution(:,end),round_error),round(correct_solution,round_error))
    disp('Solution is correct')
else disp('Solution is incorrect:')
    correct_solution
end
if ~isequal(solution(:,end),correct_solution)
    disp(['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(['Solution has a norm of ', num2str(relative_norm)])
disp(['Solution converged within ', num2str(iterations), ' iterations'])
```

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

```
A =

  -79.0000    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   41.0000    0.8000    0.6667    0.5714    0.5000
    0.2000    0.4000    0.6000    0.8000  -79.0000    0.8333    0.7143    0.6250
    0.1667    0.3333    0.5000    0.6667    0.8333   51.0000    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   31.0000


b =

     5
     8
     8
    10
     9
     4
```

```
     7
     2


tol =

   1.0000e-05


ans =

   -0.0649
   -0.1656
   -0.2753
    0.2527
   -0.1139
    0.0815
   -0.0884
    0.0682

Solution is correct
Solution is inaccurate by a maximum difference of 8.0113e-07
Solution has a norm of 0.00034069
Solution converged within 3 iterations
```

exercise3.m :

```
% APPM3021 Lab 2, Exercise 3

clc
clear all

rows = 8;
A=generateDiagonallyDominantMatrix(rows)
b = randi(10,rows,1)

x_0 = zeros(length(b),1);
tol = 0.00001
round_error = abs(log10(tol))-1;

[solution, iterations] = SOR(A,b,x_0,tol);

% Output and check
solution(:,end)
correct_solution = A\b;
if isequal(round(solution(:,end),round_error),round(correct_solution,round_error))
    disp('Solution is correct')
else disp('Solution is incorrect:')
    correct_solution
end
if ~isequal(solution(:,end),correct_solution)
    disp(['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(['Solution has a norm of ', num2str(relative_norm)])
disp(['Solution converged within ', num2str(iterations), ' iterations'])
```

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

```
A =

  -99.0000    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250
    0.5000   51.0000    0.6667    0.5000    0.4000    0.3333    0.2857    0.2500
    0.3333    0.6667    1.0000    0.7500    0.6000    0.5000    0.4286    0.3750
    0.2500    0.5000    0.7500    1.0000    0.8000    0.6667    0.5714    0.5000
    0.2000    0.4000    0.6000    0.8000   81.0000    0.8333    0.7143    0.6250
    0.1667    0.3333    0.5000    0.6667    0.8333   21.0000    0.8571    0.7500
    0.1429    0.2857    0.4286    0.5714    0.7143    0.8571   21.0000    0.8750
    0.1250    0.2500    0.3750    0.5000    0.6250    0.7500    0.8750   81.0000


b =

     9
     6
```

```
     2
     3
     9
     1
     5
     2


tol =

   1.0000e-05


ans =

   -0.0838
    0.0928
   -0.6053
    3.3055
    0.0818
   -0.0536
    0.1590
    0.0051

Solution is correct
Solution is inaccurate by a maximum difference of 4.7989e-06
Solution has a norm of 4.5984e-06
Solution converged within 13 iterations
```

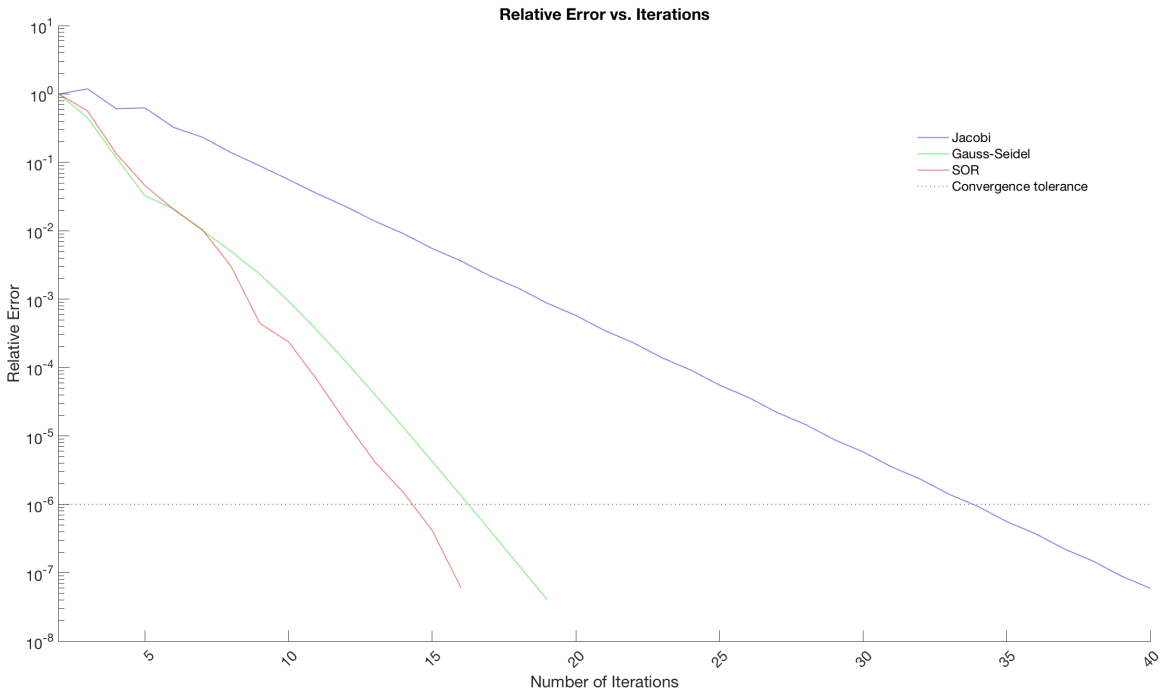
exercise4.m :

```
% APPM3021 Lab 2, Exercise 4

clc
clear all
digits(32)
% dbstop if error

n = 100;
tol = 0.000001;

%% Generate
A = generateDiagonallyDominantMatrix(n);
b = randi(10,n,1);
x_0 = zeros(n,1);


%% Measure
[sol_jac, iter_jac] = jacobiMethod(A,b,x_0,tol);
[sol_gss, iter_gss] = gaussSeidel(A,b,x_0,tol);
[sol_sor, iter_sor] = SOR(A,b,x_0,tol);

%% 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



%% 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,[2 iter_jac(1,1)]);
box(ax1,'off');
set(ax1,'FontSize',14,...
    'XTick',[0:5:iter_jac(1,1)],...
    'XTickLabelRotation',45,...
    'YMinorTick','on','YScale','log');hold on

% Legend
legend1 = legend({'Jacobi','Gauss-Seidel','SOR','Convergence tolerance'},...
     'Position',[0.7    0.7    0.2    0.09],...
     'Box','off');
hold off
% export (fix for missing CMU fonts in eps export)
% export_fig relative_error.eps
% epswrite('images/relative_error.eps');
% epsembedfont('relative_error.eps','+CMU Serif=>mwa_cmr10')
```

Running exercise4.m creates a figure. A snapshot is displayed below.

Figure 1. 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<2.
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 method, although when the number of iterations are small, it can perform poorly compared to the other two algorithms. In general it produces much faster convergence, as can be seen in the example graph.

Questions

Question 1a)

Functions and Code

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
```

jacobiMethod.m :

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

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

[L, D, U] = LDU(A);
x = x_0;

check = A\b;

iterationCount = 1;
while true
    y=b-(L+U)*x(:,iterationCount);
    x(:,iterationCount+1)=D\y;
    err_norm = sum(abs(check-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
    err(strcat('Matrix is not solvable'))
end

[L, D, U] = LDU(A);
X = x_0;

correct_solution = A\b;
```

```
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
    err(strcat('Matrix is not solvable'))
end

[L, D, U] = LDU(A);
x = x_0;

correct_solution = A\b;
B = D\(L+U);
eigenvalue = max(abs(eig(B)));
omega = 2/(1+sqrt(1-eigenvalue^2));

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
```

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 diagononal entries of A
% U = strictly upper triangular matrix of A

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

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
```