APPM7440: Radial Basis Functions - Finite Differences April 30, 2015

Lab Report 3

Homework 3 *Author: Prasanth Prahladan(100817764)*

1 GA-RBF Standard Double Precision Implementation

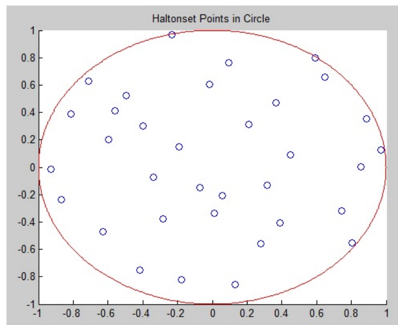


Figure 1: Haltonset node distribution in Circular Disc

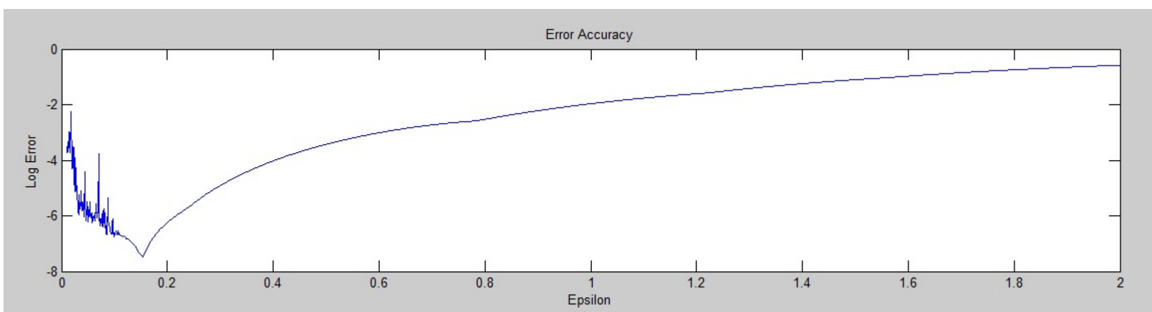


Figure 2: Log10(Error) variation with ϵ

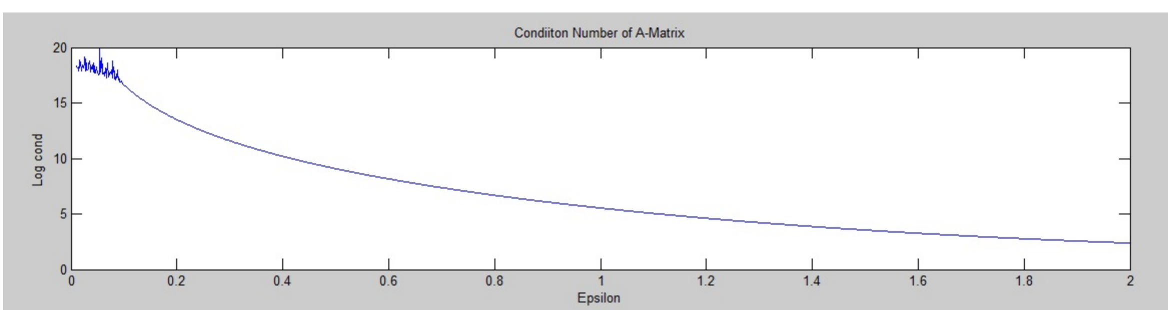


Figure 3: Variation of Condition Number of A matrix with ϵ

1.1 Matlab Implementation

```
% APPM 7440: HW#3
% Question 1: Double Precision Arithmetic
```

```
% RBF - DIRECT
% ==========================================================================
close all
clear all
clc

N = 42;         % Total sample points needed : Always Ensure EVEN!!

%%
% GENERATE POINTS WITHIN UNIT CIRCLE - can be done as a function.

% plot unit circle
theta = 0:0.01:2*pi;
circx = cos(theta);
circy = sin(theta);
fig1 = figure(1)
plot(circx,circy,'r')
hold on
%%
% Generate Uniformly Random distributed points in Unit Disc

rndTheta = 2*pi*rand(N,1);
rndRad1 = sqrt(rand(N,1));
x1 = rndRad1.*cos(rndTheta);
y1 = rndRad1.*sin(rndTheta);
scatter(x1,y1)
title('Uniformly Distributed Points in Unit Disc')
hold off

% xvecs = [x1' y1']; % Nx2 matrix.

%%
% Generate Random points using HaltonSet

rng default
p = haltonset(2,'Skip',1e3,'Leap', 1e2);
p = scramble(p,'RR2');      % Reverse radix scrambling
X0 = net(p,N);
X0 = -1*ones(size(X0)) + 2*X0;
R = sqrt(X0(:,1).^2 + X0(:,2).^2);
inds = find((R<=1));
x2 = X0(inds,1);
y2 = X0(inds,2);

% ----------------------------------------
fig2 = figure(2)
scatter(x2,y2,'b')
title('Haltonset Points in Circle')
hold on
plot(circx,circy,'r')
hold off


%%
% Choose the data-set you want from ABOVE
```

```matlab
% (x2,y2): Haltonset ; (x1,y1): RandomNumbers

% #COLUMNS = # Dimensions of the Spatial Vectors
% #ROWS = # Data Points

x0 = x2;
y0 = y2;

% x = floor(10*rand(1,5));
% y = floor(10*rand(1,5));

% Divide Data Points into GRID(TRAIN) and TEST LOCATIONS
x_trn = x0(1:N/2)';
y_trn = y0(1:N/2)';
Z_trn = [x_trn; y_trn]';
size(Z_trn)

x_tst = x0(N/2+1:end)';
y_tst = y0(N/2+1:end)';
Z_tst = [x_tst; y_tst]';


%%
% Checking for errors when the Test Data is same as Training Data
% Z_tst = Z_trn(2:4,:);
 Z_tst = Z_tst;
%% MAIN SECTION: Iterating over SHAPE PARAMTER

% Evaluate Test Function : Train Data
fvals_trn = evalTestFunction(Z_trn);
size(fvals_trn)

% Evaluate Test Function : Testing Data
fvals_tst = evalTestFunction(Z_tst);
size(fvals_tst)

Epsilon = 0.01:0.001:2;
% -------------------------------------------------------------------
% Epsilon = ones(size(Epsilon)); % Test Case for Accuracy
% When eps = 1, the Reciprocal Condition-Number should be closer to 1.
% -------------------------------------------------------------------
error = zeros(size(Epsilon));
Condition = zeros(size(Epsilon));
Lambda = zeros(size(Epsilon));
Fvals = zeros(size(Epsilon));

% -----------------------------------------------------------------------
for count = 1:length(Epsilon)
% Compute Lambda
% -----------------------------
% fprintf('Z_trn Dimensions \n')
% size(Z_trn)
epsilon = Epsilon(count);
A_trn = getRBFmatrix(Z_trn,epsilon,'GA');
% fprintf('Size A_trn \n')
```

```
% size(A_trn)
lambda = A_trn\fvals_trn;

% Evaluate the function at
feval_tst = evalRBFinterpolation(Z_tst,Z_trn,lambda, epsilon,'GA');

fprintf('Size feval_tst \t')
size(feval_tst)
fprintf('Size fvals_tst \t')
size(fvals_tst)

% Compute the parameters to plot.
Condition(count) = cond(A_trn);
error(count) = max(abs(feval_tst - fvals_tst));

% unnecessary params
Lambda(count) = max(abs(lambda));
Fvals(count) = max(abs(fvals_trn));

% break          % Used for Testing!

end


count
fig4 = figure(4)
subplot(2,1,1)
% plot(Epsilon, Lambda)
plot(log10(Lambda))          % TEST
xlabel('Epsilon')
ylabel('Log Max Lambda')
subplot(2,1,2)
% plot(Epsilon, Fvals)
plot(Fvals)          % TEST
xlabel('Epsilon')
ylabel('Fvals')

fig5 = figure(5)
subplot(2,1,1)
plot(Epsilon, log10(error))
% plot(error)          % TEST
title('Error Accuracy')
xlabel('Epsilon')
ylabel('Log Error')
subplot(2,1,2)
plot(Epsilon, log10(Condition))
% plot(Condition)      % TEST
title('Condiiton Number of A-Matrix')
xlabel('Epsilon')
ylabel('Log cond')



% ===========================================================================
% ===========================================================================
% APPM 7440: HW#3
```

```
% Question 1: Double Precision Arithmetic

function A = getRBFmatrix(Z,epsilon, RBFtype)
%{
Z           : Data point vectors
epsilon     : Shape Parameter
RBFtype     : Type of RBF used
%}

[rows,cols] = size(Z);
if cols ==1
    X = Z(:,1)'; % row vector
    [Ggrid, Gvar] = meshgrid(X,X);
    dX = Gvar - Ggrid;
    R2 = dX.^2;

elseif cols == 2 % 2-dimensional dataset
    % row-vectors
    X = Z(:,1)';
    Y = Z(:,2)';

    % grid -------------------------
    [gx, gy] = meshgrid(X,Y);
    GY = gy';
    GX = gx;

    % Compute RBF
    dX = GX' - GX;
    dY = GY' - GY;
    R2 = dX.^2 + dY.^2

else
     prinft('ERROR: Does not support greater than 2 Dimensions!')
end

% Create RBF Matrix
% ------------------------------
ONE = ones(size(R2));
switch RBFtype
     case 'GA' % RBF: Gaussian
         fprintf('Gaussian \n')
         A = exp(-(epsilon^2)*R2)

     case 'IMQ' % RBF: Inverse Multiquadric
         fprintf('Inverse Multiquadric \n')
         A = 1./(ONE + (epsilon^2)*R2).^(1/2);

     case 'IQ' % RBF: Inverse Quadratic
         fprintf('Inverse Quadratic \n')
         A = 1./(ONE + (epsilon^2)*R2);

     otherwise % RBF: Multi-Quadrics
         fprintf('Multi-Quadrics \n')
         A = (ONE + (epsilon^2)*R2).^(1/2);

```

```
end
return

% ==========================================================================
% ==========================================================================
% APPM 7440: HW#3
% Question 1: Double Precision Arithmetic

function feval = evalRBFinterpolation(Z_data, Z_grid,lambda, epsilon, RBFtype)
%{
Z_data      : Data point vectors
Z_grid      : Grid point vectors
lambda      : RBF Interpolation Coefficients
epsilon     : Shape Parameter
RBFtype     : Type of RBF used
%}

% MQ-RBF/ GA-RBF methodology for function interpolation
% and computing error.
% ----------------------------------
% Compute as a function of shape-parameter
% ----------------------------------
% meshgrid and ndgrid to compute the A matrix
% ----------------------------------

[r,c] = size(Z_data);
if c ==1 % 1-Dimensional Data
    X_data = Z_data(:,1)';
    X_grid = Z_grid(:,1)';

    G = ndgrid(X_grid, X_data);
    GridX = G';
    DataX = ndgrid(X_data, X_grid);

    dX = DataX - GridX;

    R2 = dX.^2;

elseif c == 2 % 2-dimensional dataset
    X_data = Z_data(:,1)'
    Y_data = Z_data(:,2)'

    X_grid = Z_grid(:,1)'
    Y_grid = Z_grid(:,2)'

    % ============================================
    G = ndgrid(X_grid, X_data);
    GridX = G'
    DataX = ndgrid(X_data, X_grid)


    G = ndgrid(Y_grid, Y_data);
    GridY = G'
    DataY = ndgrid(Y_data, Y_grid)

```

```
    % =============================
    % Compute RBF

    dX = DataX - GridX;
    dY = DataY - GridY;
    R2 = dX.^2 + dY.^2;

else
    fprintf('>2-Dimensions NOT supported!')
end

% Create RBF Matrix
% ------------------------------
ONE = ones(size(R2));
switch RBFtype
    case 'GA' % RBF: Gaussian
        fprintf('Gaussian \n')
        A = exp(-(epsilon^2)*R2)

    case 'IMQ' % RBF: Inverse Multiquadric
        fprintf('Inverse Multiquadric \n')
        A = 1./(ONE + epsilon^2*R2).^(1/2);

    case 'IQ' % RBF: Inverse Quadratic
        fprintf('Inverse Quadratic \n')
        A = 1./(ONE + epsilon^2*R2);

    otherwise % RBF: Multi-Quadrics
        fprintf('Multi-Quadrics \n')
        A = (ONE + epsilon^2*R2).^(1/2);
end

feval = A*lambda

return
```

2 GA-RBF: Variable Precision Arithmetic

2.1 Matlab Implementation

```
% APPM 7440: HW#3
% Question 2: VARIABLE Precision Arithmetic
% RBF - DIRECT
% ==========================================================================
close all
clear all
clc

N = 42;          % Total sample points needed : Always Ensure EVEN!!

%%
% GENERATE POINTS WITHIN UNIT CIRCLE - can be done as a function.

% plot unit circle
theta = 0:0.01:2*pi;
```

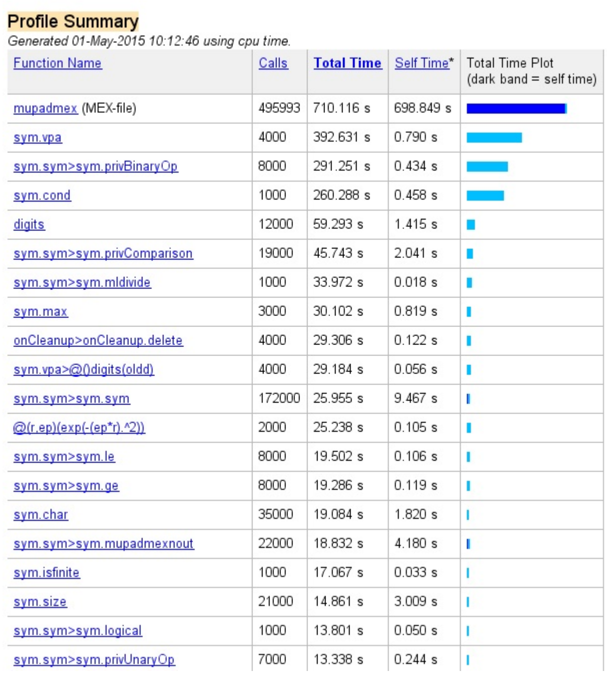
```
circx = cos(theta);
circy = sin(theta);

fig1 = figure(1)
plot(circx,circy,'r')

%%
% Generate Random points using HaltonSet

rng default
p = haltonset(2,'Skip',1e3,'Leap', 1e2);
p = scramble(p,'RR2');       % Reverse radix scrambling
X0 = net(p,N);
X0 = -1*ones(size(X0)) + 2*X0;
R = sqrt(X0(:,1).^2 + X0(:,2).^2);
inds = find((R<=1));
x2 = X0(inds,1);
y2 = X0(inds,2);

% ------------------------------------------------
fig2 = figure(2)
% subplot(2,1,1)
scatter(x2,y2,'b')
title('Haltonset Points in Circle')
hold on
plot(circx,circy,'r')
hold off

%%
% Choose the data-set you want from ABOVE
% (x2,y2): Haltonset ; (x1,y1): RandomNumbers

% #COLUMNS = # Dimensions of the Spatial Vectors
% #ROWS = # Data Points

x0 = sym(x2);
y0 = sym(y2);

% Divide Data Points into GRID(TRAIN) and TEST LOCATIONS
% Grid-Training Data
x_trn = x0(1:N/2)';
y_trn = y0(1:N/2)';
Z_trn = [x_trn; y_trn]';
Z_trn = sym(Z_trn);

% Testing Data
x_tst = x0(N/2+1:end)';
y_tst = y0(N/2+1:end)';
Z_tst = [x_tst; y_tst]';
Z_tst = sym(Z_tst);

%% MAIN SECTION: Iterating over SHAPE PARAMTER

% Describe Test Function
testFunc = @(Z)( sym(59) ./ sym(67 + ( sym(Z(:,1)) + 1/sym(7) ).^2 + ( sym(Z(:,2)
```

```
    ) - 1/sym(11) ).^2 ) );

% Evaluate Test Function : Train Data
fvals_trn = testFunc(Z_trn);
fvals_trn = sym(fvals_trn);
% fprintf('\nSize fvals_trn \t'); size(fvals_trn)

% Evaluate Test Function : Testing Data
fvals_tst = testFunc(Z_tst);
fvals_tst = sym(fvals_tst);
% fprintf('\nSize fvals_tst \t'); size(fvals_tst)

% ==================================================
% Evlauation for Training Data

X_trn = sym(Z_trn(:,1));
Y_trn = sym(Z_trn(:,2));

% grid -------------------------
[gx, gy] = meshgrid(X_trn,Y_trn);
GY = sym(gy');
GX = sym(gx);

% Compute RBF
dX = sym(GX' - GX);
dY = sym(GY' - GY);
R2Trn = sym(dX.^2 + dY.^2);
rTrn = sym(sqrt(R2Trn));
% fprintf('\nSize rTrn \t');size(rTrn)

% ==========================================
% Computations for Testing Data
X_tst = sym(Z_tst(:,1));
Y_tst = sym(Z_tst(:,2));

[nodeX,gridX] = ndgrid(X_tst,X_trn);
[nodeY,gridY] = ndgrid(Y_tst,Y_trn);

R2tst = (sym(nodeX)-sym(gridX)).^2 + (sym(nodeY) - sym(gridY)).^2;
rTst = sqrt(sym(R2tst));
% fprintf('\nSize rTst \t');size(rTst)
% ==================================================
% ----------------------------------------------------------------
% Epsilon = ones(size(Epsilon)); % Test Case for Accuracy
% When eps = 1, the Reciprocal Condition-Number should be closer to 1.
% ----------------------------------------------------------------
Epsilon = 0.001:0.001:1;

error = zeros(size(Epsilon));
Condition = zeros(size(Epsilon));
Lambda = zeros(size(Epsilon));
Fvals = zeros(size(Epsilon));
% ==================================================
% Description of RBF Function
fi = @(r,ep)(exp(-(ep*r).^2)); % GA-RBF
```

```

ACC = 2^10; % VPA Precision Digits
profile ON
% ------------------------------------------------------------------------
for count = 1:length(Epsilon)
% count = 1;

    epsilon = Epsilon(count);
    fprintf('epsilon \t %f \n',epsilon)
    epsilon = sym(epsilon);

    % Compute Lambda
    % ------------------------------
    A_trn = fi(rTrn,epsilon);

    nmA_trn = vpa(A_trn,ACC);
%     fprintf('\nSize A_trn \t');size(A_trn)

%     invA_trn = vpa(inv(A_trn),ACC);
%     fprintf('\nSize invA_trn \t');size(invA_trn)

%     fprintf('\nSize fvals_trn \t');size(fvals_trn)

    lambda = nmA_trn\fvals_trn;
    lambda = sym(lambda);
%     fprintf('\n Size lambda \t');size(lambda)

    % Evaluate the function at Test Points
    % ------------------------------
    A_tst = fi(rTst,epsilon);
%     A_tst = sym(A_tst);
%     fprintf('\nSize A_tst \t');size(A_tst)

    feval_tst = A_tst*lambda;
    feval_tst = sym(feval_tst);
%     fprintf('\nSize feval_tst \t');size(feval_tst)

    % Compute the parameters to plot.
    % ------------------------------
    Condition(count) = cond(nmA_trn);
    err = vpa(feval_tst - fvals_tst , ACC);
    error(count) = max(abs(err));

    % unnecessary params
    Lambda(count) = max(abs(vpa(lambda,ACC)));
    Fvals(count) = max(abs(vpa(fvals_trn,ACC)));

end

profile VIEWER

count


fig4 = figure(4)
```

```
subplot(2,1,1)
% plot(Epsilon, Lambda)
plot(log10(Lambda))           % TEST
xlabel('Epsilon')
ylabel('Log Max Lambda')
subplot(2,1,2)
% plot(Epsilon, Fvals)
plot(Fvals)           % TEST
xlabel('Epsilon')
ylabel('Fvals')

fig5 = figure(5)
subplot(2,1,1)
plot(Epsilon, log10(error))
% plot(error)           % TEST
title('Error Accuracy')
xlabel('Epsilon')
ylabel('Log Error')
subplot(2,1,2)
plot(Epsilon, log10(Condition))
% plot(Condition)       % TEST
title('Condiiton Number of A-Matrix')
xlabel('Epsilon')
ylabel('Log cond')
```

2.2 Plots and Results



Profile Summary

Generated 01-May-2015 10:12:46 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
mupadmex (MEX-file)	495993	710.116 s	698.849 s	
sym.vpa	4000	392.631 s	0.790 s	
sym.sym>sym.privBinaryOp	8000	291.251 s	0.434 s	
sym.cond	1000	260.288 s	0.458 s	
digits	12000	59.293 s	1.415 s	
sym.sym>sym.privComparison	19000	45.743 s	2.041 s	
sym.sym>sym.mldivide	1000	33.972 s	0.018 s	
sym.max	3000	30.102 s	0.819 s	
onCleanup>onCleanup.delete	4000	29.306 s	0.122 s	
sym.vpa>@()digits(oldd)	4000	29.184 s	0.056 s	
sym.sym>sym.sym	172000	25.955 s	9.467 s	
@(r,ep)(exp(-(ep*r).^2))	2000	25.238 s	0.105 s	
sym.sym>sym.le	8000	19.502 s	0.106 s	
sym.sym>sym.ge	8000	19.286 s	0.119 s	
sym.char	35000	19.084 s	1.820 s	
sym.sym>sym.mupadmexnout	22000	18.832 s	4.180 s	
sym.isfinite	1000	17.067 s	0.033 s	
sym.size	21000	14.861 s	3.009 s	
sym.sym>sym.logical	1000	13.801 s	0.050 s	
sym.sym>sym.privUnaryOp	7000	13.338 s	0.244 s	

Figure 4: Profile Times

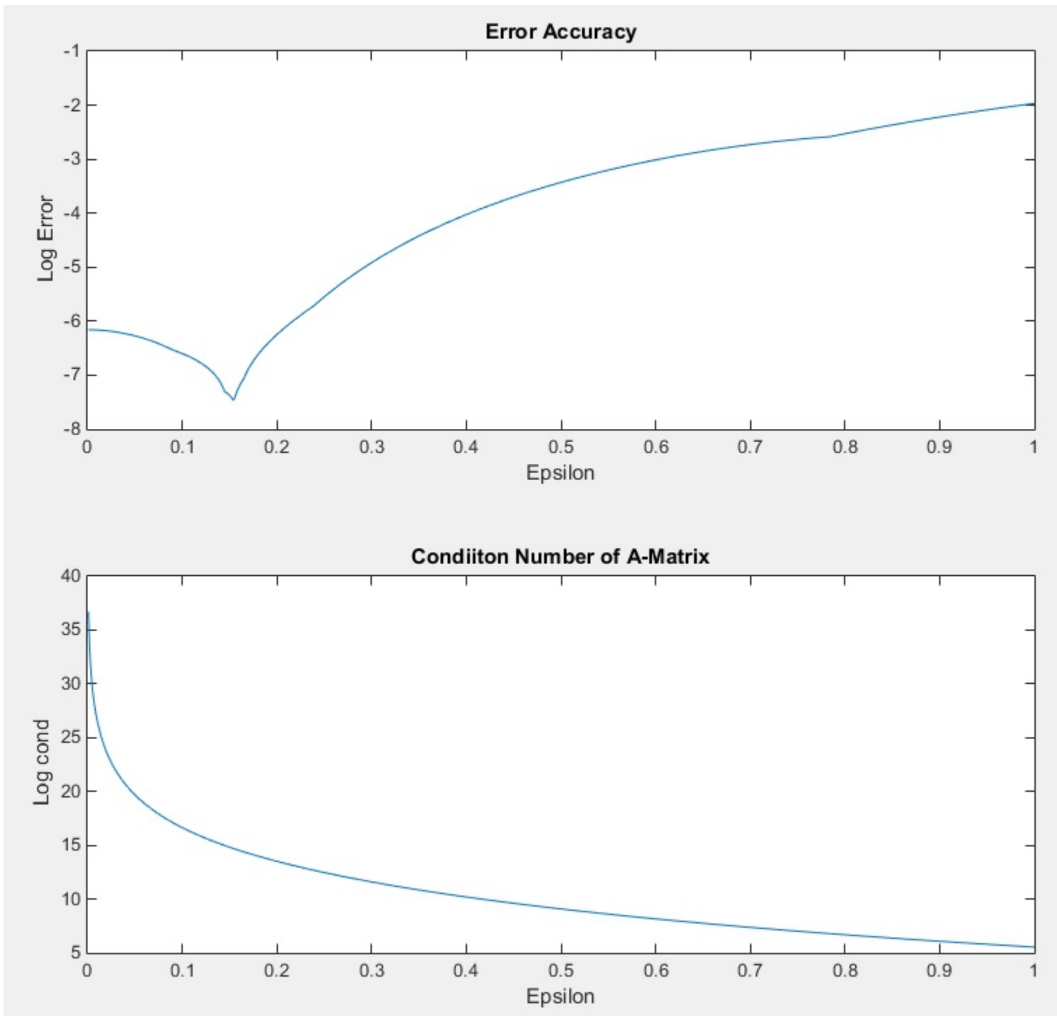


Figure 5: log10(Error) and Condition Number Plots