

Lab Report 3

Homework 3

Author: Prasanth Prahladan(100817764)

1 GA-RBF Standard Double Precision Implementation

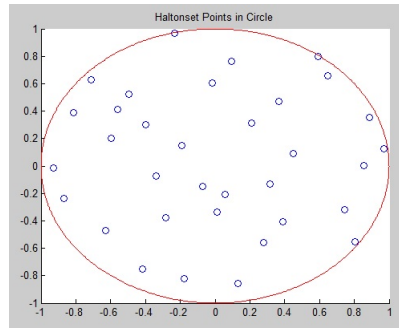
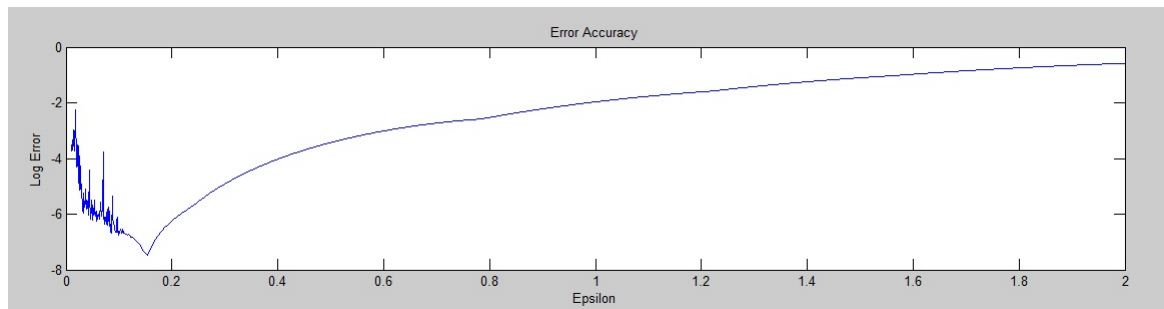
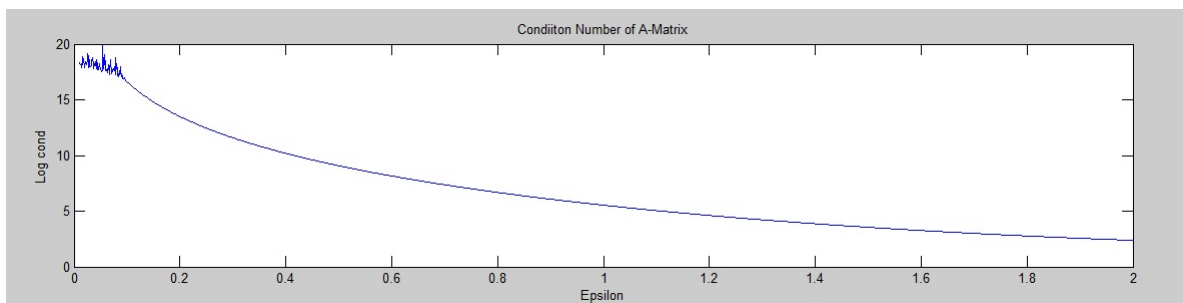


Figure 1: Haltonset node distribution in Circular Disc

Figure 2: $\text{Log}_{10}(\text{Error})$ variation with ϵ Figure 3: Variation of Condition Number of A matrix with ϵ

1.1 Matlab Implementation

```

1 % APPM 7440: HW#3
2 % Question 1: Double Precision Arithmetic

```

```

3 % RBF – DIRECT
4 % =====
5 close all
6 clear all
7 clc
8
9 N = 42;           % Total sample points needed : Always Ensure EVEN!!
10
11 %%
12 % GENERATE POINTS WITHIN UNIT CIRCLE – can be done as a function.
13
14 % plot unit circle
15 theta = 0:0.01:2*pi;
16 circx = cos(theta);
17 circy = sin(theta);
18 fig1 = figure(1)
19 plot(circx, circy, 'r')
20 hold on
21 %%
22 % Generate Uniformly Random distributed points in Unit Disc
23
24 rndTheta = 2*pi*rand(N,1);
25 rndRad1 = sqrt(rand(N,1));
26 x1 = rndRad1.*cos(rndTheta);
27 y1 = rndRad1.*sin(rndTheta);
28 scatter(x1,y1)
29 title('Uniformly Distributed Points in Unit Disc')
30 hold off
31
32 % xvecs = [x1' y1']; % Nx2 matrix.
33
34 %%
35 % Generate Random points using HaltonSet
36
37 rng default
38 p = haltonset(2, 'Skip', 1e3, 'Leap', 1e2);
39 p = scramble(p, 'RR2'); % Reverse radix scrambling
40 X0 = net(p,N);
41 X0 = -1*ones(size(X0)) + 2*X0;
42 R = sqrt(X0(:,1).^2 + X0(:,2).^2);
43 inds = find((R<=1));
44 x2 = X0(inds,1);
45 y2 = X0(inds,2);
46
47 % =====
48 fig2 = figure(2)
49 scatter(x2,y2, 'b')
50 title('Haltonset Points in Circle')
51 hold on
52 plot(circx, circy, 'r')
53 hold off
54
55
56 %%
57 % Choose the data-set you want from ABOVE

```

```

58 % (x2,y2): Haltonset ; (x1,y1): RandomNumbers
59
60 % #COLUMNS = # Dimensions of the Spatial Vectors
61 % #ROWS = # Data Points
62
63 x0 = x2;
64 y0 = y2;
65
66 % x = floor(10*rand(1,5));
67 % y = floor(10*rand(1,5));
68
69 % Divide Data Points into GRID(TRAIN) and TEST LOCATIONS
70 x_trn = x0(1:N/2)';
71 y_trn = y0(1:N/2)';
72 Z_trn = [x_trn; y_trn]';
73 size(Z_trn)
74
75 x_tst = x0(N/2+1:end)';
76 y_tst = y0(N/2+1:end)';
77 Z_tst = [x_tst; y_tst]';
78
79
80 %%
81 % Checking for errors when the Test Data is same as Training Data
82 % Z_tst = Z_trn(2:4,:);
83 Z_tst = Z_tst;
84 %% MAIN SECTION: Iterating over SHAPE PARAMETER
85
86 % Evaluate Test Function : Train Data
87 fvals_trn = evalTestFunction(Z_trn);
88 size(fvals_trn)
89
90 % Evaluate Test Function : Testing Data
91 fvals_tst = evalTestFunction(Z_tst);
92 size(fvals_tst)
93
94 Epsilon = 0.01:0.001:2;
95 % -----
96 % Epsilon = ones(size(Epsilon)); % Test Case for Accuracy
97 % When eps = 1, the Reciprocal Condition-Number should be closer to 1.
98 % -----
99 error = zeros(size(Epsilon));
100 Condition = zeros(size(Epsilon));
101 Lambda = zeros(size(Epsilon));
102 Fvals = zeros(size(Epsilon));
103
104 % -----
105 for count = 1:length(Epsilon)
106 % Compute Lambda
107 % -----
108 % fprintf('Z_trn Dimensions \n')
109 % size(Z_trn)
110 epsilon = Epsilon(count);
111 A_trn = getRBFmatrix(Z_trn, epsilon, 'GA');
112 % fprintf('Size A_trn \n')

```

```

113 % size(A_trn)
114 lambda = A_trn\fvals_trn;
115
116 % Evaluate the function at
117 feval_tst = evalRBFinterpolation(Z_tst,Z_trn,lambda, epsilon,'GA');
118
119 fprintf('Size feval_tst \t')
120 size(feval_tst)
121 fprintf('Size fvals_tst \t')
122 size(fvals_tst)
123
124 % Compute the parameters to plot.
125 Condition(count) = cond(A_trn);
126 error(count) = max(abs(feval_tst - fvals_tst));
127
128 % unnecessary params
129 Lambda(count) = max(abs(lambda));
130 Fvals(count) = max(abs(fvals_trn));
131
132 % break          % Used for Testing!
133
134 end
135
136 count
137 fig4 = figure(4)
138 subplot(2,1,1)
139 % plot(Epsilon, Lambda)
140 plot(log10(Lambda))          % TEST
141 xlabel('Epsilon')
142 ylabel('Log Max Lambda')
143 subplot(2,1,2)
144 % plot(Epsilon, Fvals)
145 plot(Fvals)          % TEST
146 xlabel('Epsilon')
147 ylabel('Fvals')
148
149 fig5 = figure(5)
150 subplot(2,1,1)
151 plot(Epsilon, log10(error))
152 % plot(error)          % TEST
153 title('Error Accuracy')
154 xlabel('Epsilon')
155 ylabel('Log Error')
156 subplot(2,1,2)
157 plot(Epsilon, log10(Condition))
158 % plot(Condition)          % TEST
159 title('Condiiton Number of A-Matrix')
160 xlabel('Epsilon')
161 ylabel('Log cond')
162
163
164
165 % =====
166 % =====
167 % APPM 7440: HW#3

```

```

168 % Question 1: Double Precision Arithmetic
169
170 function A = getRBFmatrix(Z, epsilon , RBFtype)
171 %{
172 Z           : Data point vectors
173 epsilon     : Shape Parameter
174 RBFtype     : Type of RBF used
175 %}
176
177 [rows,cols] = size(Z);
178 if cols ==1
179     X = Z(:,1)'; % row vector
180     [Ggrid, Gvar] = meshgrid(X,X);
181     dX = Gvar - Ggrid;
182     R2 = dX.^2;
183
184 elseif cols == 2 % 2-dimensional dataset
185     % row-vectors
186     X = Z(:,1)';
187     Y = Z(:,2)';
188
189     % grid -----
190     [gx, gy] = meshgrid(X,Y);
191     GY = gy';
192     GX = gx;
193
194     % Compute RBF
195     dX = GX' - GX;
196     dY = GY' - GY;
197     R2 = dX.^2 + dY.^2
198
199 else
200     printf('ERROR: Does not support greater than 2 Dimensions!')
201 end
202
203 % Create RBF Matrix
204 % -----
205 ONE = ones(size(R2));
206 switch RBFtype
207     case 'GA' % RBF: Gaussian
208         fprintf('Gaussian \n')
209         A = exp(-(epsilon^2)*R2)
210
211     case 'IMQ' % RBF: Inverse Multiquadric
212         fprintf('Inverse Multiquadric \n')
213         A = 1./(ONE + (epsilon^2)*R2).^(1/2);
214
215     case 'IQ' % RBF: Inverse Quadratic
216         fprintf('Inverse Quadratic \n')
217         A = 1./(ONE + (epsilon^2)*R2);
218
219     otherwise % RBF: Multi-Quadrics
220         fprintf('Multi-Quadrics \n')
221         A = (ONE + (epsilon^2)*R2).^(1/2);
222

```

```

223 end
224 return
225
226 % =====
227 % =====
228 % APPM 7440: HW#3
229 % Question 1: Double Precision Arithmetic
230
231 function feval = evalRBFinterpolation(Z_data, Z_grid, lambda, epsilon, RBFtype)
232 %{
233 Z_data      : Data point vectors
234 Z_grid      : Grid point vectors
235 lambda      : RBF Interpolation Coefficients
236 epsilon     : Shape Parameter
237 RBFtype     : Type of RBF used
238 %}
239
240 % MQ-RBF/ GA-RBF methodology for function interpolation
241 % and computing error.
242 % -----
243 % Compute as a function of shape-parameter
244 % -----
245 % meshgrid and ndgrid to compute the A matrix
246 % -----
247
248 [r, c] = size(Z_data);
249 if c == 1 % 1-Dimensional Data
250     X_data = Z_data(:,1)';
251     X_grid = Z_grid(:,1)';
252
253     G = ndgrid(X_grid, X_data);
254     GridX = G';
255     DataX = ndgrid(X_data, X_grid);
256
257     dX = DataX - GridX;
258
259     R2 = dX.^2;
260
261 elseif c == 2 % 2-dimensional dataset
262     X_data = Z_data(:,1)';
263     Y_data = Z_data(:,2)';
264
265     X_grid = Z_grid(:,1)';
266     Y_grid = Z_grid(:,2)';
267
268 % =====
269 G = ndgrid(X_grid, X_data);
270 GridX = G';
271 DataX = ndgrid(X_data, X_grid);
272
273
274 G = ndgrid(Y_grid, Y_data);
275 GridY = G';
276 DataY = ndgrid(Y_data, Y_grid);
277

```

```

278 % =====
279 % Compute RBF
280
281 dX = DataX - GridX;
282 dY = DataY - GridY;
283 R2 = dX.^2 + dY.^2;
284
285 else
286     fprintf(' >2-Dimensions NOT supported! ')
287 end
288
289 % Create RBF Matrix
290 % =====
291 ONE = ones(size(R2));
292 switch RBFtype
293     case 'GA' % RBF: Gaussian
294         fprintf('Gaussian \n')
295         A = exp(-(epsilon^2)*R2)
296
297     case 'IMQ' % RBF: Inverse Multiquadric
298         fprintf('Inverse Multiquadric \n')
299         A = 1./(ONE + epsilon^2*R2).^(1/2);
300
301     case 'IQ' % RBF: Inverse Quadratic
302         fprintf('Inverse Quadratic \n')
303         A = 1./(ONE + epsilon^2*R2);
304
305     otherwise % RBF: Multi-Quadrics
306         fprintf('Multi-Quadrics \n')
307         A = (ONE + epsilon^2*R2).^(1/2);
308 end
309
310 feval = A*lambda
311
312 return

```

2 GA-RBF: Variable Precision Arithmetic

2.1 Matlab Implementation

```

1 % APPM 7440: HW#3
2 % Question 2: VARIABLE Precision Arithmetic
3 % RBF - DIRECT
4 % =====
5 close all
6 clear all
7 clc
8
9 N = 42; % Total sample points needed : Always Ensure EVEN!!
10
11 %%
12 % GENERATE POINTS WITHIN UNIT CIRCLE - can be done as a function.
13
14 % plot unit circle
15 theta = 0:0.01:2*pi;

```

```

16 circx = cos(theta);
17 circy = sin(theta);
18
19 fig1 = figure(1)
20 plot(circx, circy, 'r')
21
22 %%
23 % Generate Random points using HaltonSet
24
25 rng default
26 p = haltonset(2, 'Skip', 1e3, 'Leap', 1e2);
27 p = scramble(p, 'RR2'); % Reverse radix scrambling
28 X0 = net(p, N);
29 X0 = -1*ones(size(X0)) + 2*X0;
30 R = sqrt(X0(:,1).^2 + X0(:,2).^2);
31 inds = find((R<=1));
32 x2 = X0(inds,1);
33 y2 = X0(inds,2);
34
35 % -----
36 fig2 = figure(2)
37 % subplot(2,1,1)
38 scatter(x2, y2, 'b')
39 title('Haltonset Points in Circle')
40 hold on
41 plot(circx, circy, 'r')
42 hold off
43
44 %%
45 % Choose the data-set you want from ABOVE
46 % (x2,y2): Haltonset ; (x1,y1): RandomNumbers
47
48 % #COLUMNS = # Dimensions of the Spatial Vectors
49 % #ROWS = # Data Points
50
51 x0 = sym(x2);
52 y0 = sym(y2);
53
54 % Divide Data Points into GRID(TRAIN) and TEST LOCATIONS
55 % Grid-Training Data
56 x_trn = x0(1:N/2)';
57 y_trn = y0(1:N/2)';
58 Z_trn = [x_trn; y_trn]';
59 Z_trn = sym(Z_trn);
60
61 % Testing Data
62 x_tst = x0(N/2+1:end)';
63 y_tst = y0(N/2+1:end)';
64 Z_tst = [x_tst; y_tst]';
65 Z_tst = sym(Z_tst);
66
67 %% MAIN SECTION: Iterating over SHAPE PARAMETER
68
69 % Describe Test Function
70 testFunc = @(Z)( sym(59) ./ sym(67 + ( sym(Z(:,1)) + 1/sym(7) ).^2 + ( sym(Z(:,2))

```



```

    ) - 1/sym(11) ).^2 ) );
71
72 % Evaluate Test Function : Train Data
73 fvals_trn = testFunc(Z_trn);
74 fvals_trn = sym(fvals_trn);
75 % fprintf('\nSize fvals_trn \t'); size(fvals_trn)
76
77 % Evaluate Test Function : Testing Data
78 fvals_tst = testFunc(Z_tst);
79 fvals_tst = sym(fvals_tst);
80 % fprintf('\nSize fvals_tst \t'); size(fvals_tst)
81
82 % =====
83 % Evlauation for Training Data
84
85 X_trn = sym(Z_trn(:,1));
86 Y_trn = sym(Z_trn(:,2));
87
88 % grid -----
89 [gx, gy] = meshgrid(X_trn, Y_trn);
90 GY = sym(gy');
91 GX = sym(gx);
92
93 % Compute RBF
94 dX = sym(GX' - GX);
95 dY = sym(GY' - GY);
96 R2Trn = sym(dX.^2 + dY.^2);
97 rTrn = sym(sqrt(R2Trn));
98 % fprintf('\nSize rTrn \t'); size(rTrn)
99
100 % =====
101 % Computations for Testing Data
102 X_tst = sym(Z_tst(:,1));
103 Y_tst = sym(Z_tst(:,2));
104
105 [nodeX, gridX] = ndgrid(X_tst, X_trn);
106 [nodeY, gridY] = ndgrid(Y_tst, Y_trn);
107
108 R2tst = (sym(nodeX)-sym(gridX)).^2 + (sym(nodeY) - sym(gridY)).^2;
109 rTst = sqrt(sym(R2tst));
110 % fprintf('\nSize rTst \t'); size(rTst)
111 % =====
112 % -----
113 % Epsilon = ones(size(Epsilon)); % Test Case for Accuracy
114 % When eps = 1, the Reciprocal Condition-Number should be closer to 1.
115 % -----
116 Epsilon = 0.001:0.001:1;
117
118 error = zeros(size(Epsilon));
119 Condition = zeros(size(Epsilon));
120 Lambda = zeros(size(Epsilon));
121 Fvals = zeros(size(Epsilon));
122 % =====
123 % Description of RBF Function
124 fi = @(r,ep)(exp(-(ep*r).^2)); % GA-RBF

```

```

125
126 ACC = 2^10; % VPA Precision Digits
127 profile ON
128 % -----
129 for count = 1:length(Epsilon)
130 % count = 1;
131
132     epsilon = Epsilon(count);
133     fprintf('epsilon \t %f \n',epsilon)
134     epsilon = sym(epsilon);
135
136     % Compute Lambda
137     % -----
138     A_trn = fi(rTrn,epsilon);
139
140     nmA_trn = vpa(A_trn,ACC);
141 %     fprintf('\nSize A_trn \t');size(A_trn)
142
143 %     invA_trn = vpa(inv(A_trn),ACC);
144 %     fprintf('\nSize invA_trn \t');size(invA_trn)
145
146 %     fprintf('\nSize fvals_trn \t');size(fvals_trn)
147
148     lambda = nmA_trn\fvals_trn;
149     lambda = sym(lambda);
150 %     fprintf('\n Size lambda \t');size(lambda)
151
152     % Evaluate the function at Test Points
153     % -----
154     A_tst = fi(rTst,epsilon);
155 %     A_tst = sym(A_tst);
156 %     fprintf('\nSize A_tst \t');size(A_tst)
157
158     feval_tst = A_tst*lambda;
159     feval_tst = sym(feval_tst);
160 %     fprintf('\nSize feval_tst \t');size(feval_tst)
161
162     % Compute the parameters to plot.
163     % -----
164     Condition(count) = cond(nmA_trn);
165     err = vpa(feval_tst - fvals_tst , ACC);
166     error(count) = max(abs(err));
167
168     % unnecessary params
169     Lambda(count) = max(abs(vpa(lambda,ACC)));
170     Fvals(count) = max(abs(vpa(fvals_trn,ACC)));
171
172 end
173
174 profile VIEWER
175
176 count
177
178
179 fig4 = figure(4)

```

```

180 subplot(2,1,1)
181 % plot(Epsilon, Lambda)
182 plot(log10(Lambda)) % TEST
183 xlabel('Epsilon')
184 ylabel('Log Max Lambda')
185 subplot(2,1,2)
186 % plot(Epsilon, Fvals)
187 plot(Fvals) % TEST
188 xlabel('Epsilon')
189 ylabel('Fvals')
190
191 fig5 = figure(5)
192 subplot(2,1,1)
193 plot(Epsilon, log10(error))
194 % plot(error) % TEST
195 title('Error Accuracy')
196 xlabel('Epsilon')
197 ylabel('Log Error')
198 subplot(2,1,2)
199 plot(Epsilon, log10(Condition))
200 % plot(Condition) % TEST
201 title('Condiiton Number of A-Matrix')
202 xlabel('Epsilon')
203 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>@0(digits(oldd))	4000	29.184 s	0.056 s	
sym.sym>sym.sym	172000	25.955 s	9.467 s	
@(r,sp)(exp(-(sp*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

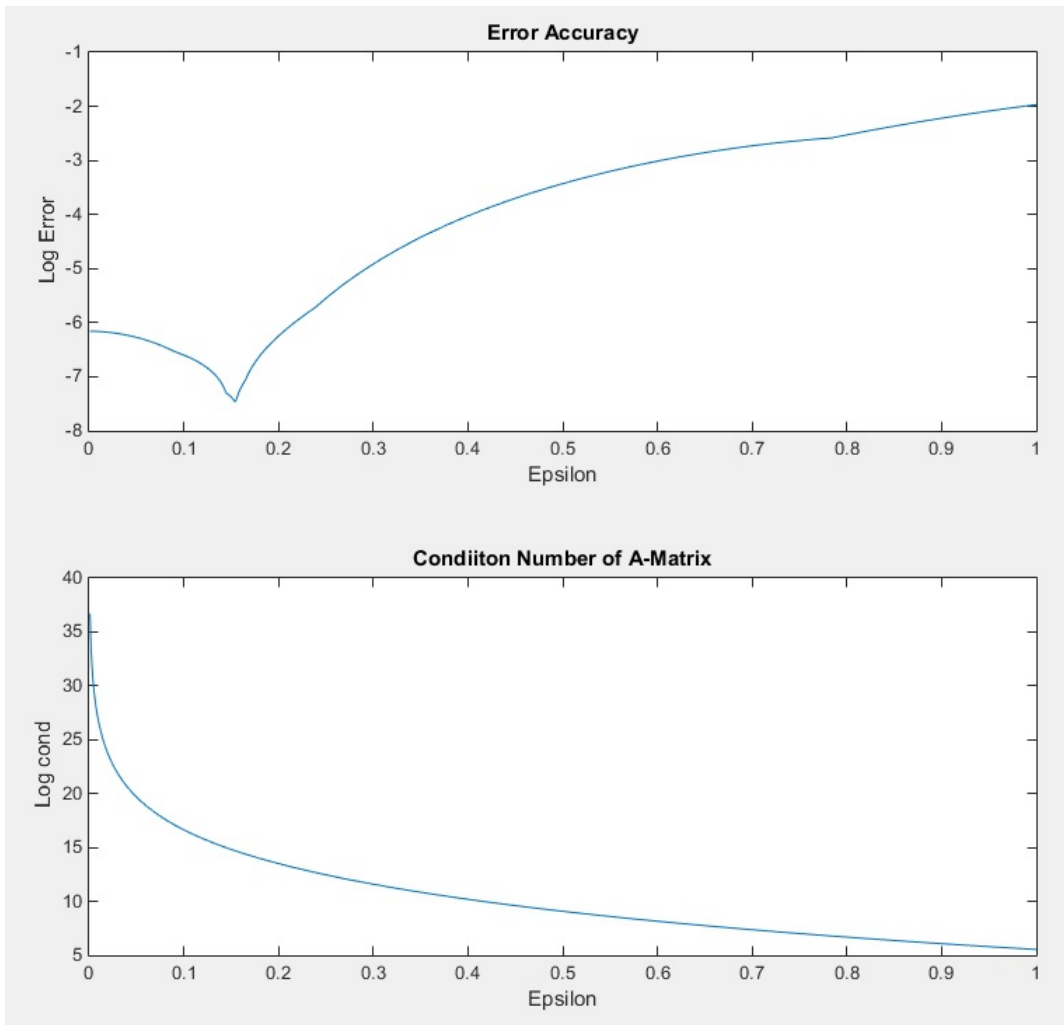


Figure 5: $\log_{10}(\text{Error})$ and Condition Number Plots