

ECE 580: Homework 3

Rahul Deshmukh

March 9, 2020

For this homework, we are given the Rastrigin's function (the egg-crate function) as our objective function which is defined as:

$$f(\mathbf{x}) = 20 + \frac{x_1^2}{10} + \frac{x_2^2}{10} - 10(\cos(\frac{2\pi x_1}{10}) + \cos(\frac{2\pi x_2}{10})) \quad (1)$$

$$\nabla f = \begin{bmatrix} \frac{x_1}{5} + 2\pi \sin(\frac{2\pi x_1}{10}) & \frac{x_2}{5} + 2\pi \sin(\frac{2\pi x_2}{10}) \end{bmatrix}^T \quad (2)$$

We are also given two starting points:

$$\mathbf{x}_a = \begin{bmatrix} 7.5 \\ 9.0 \end{bmatrix} \quad \mathbf{x}_b = \begin{bmatrix} -7.0 \\ -7.5 \end{bmatrix}$$

Exercise 1

For steepest descent, we start with first finding the gradient of the function and then the negative gradient acts as our search direction. We then update our position using Eq. 3.

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \mathbf{g}(\mathbf{x}^{(k)}) \quad (3)$$

$$\text{Where } \mathbf{g}^{(k)} = \mathbf{g}(\mathbf{x}^{(k)}) = \nabla f(\mathbf{x}^{(k)})$$

For Steepest descent the step length (α_k) at each iteration (k) is computed using Fibonacci Search Method with the search direction for these methods being $\mathbf{d} = -\mathbf{g}(\mathbf{x}^{(k)})$.

We then carry out the SD updates repeatedly till a tolerance is satisfied for the solution.

For the given function in Eq. 1 along with its gradient as per Eq. 2 we carry out Steepest Descent for 2 different starting points. The details of the iterations are shown in Table 1 and Table 2. The trajectories are shown in Figure 1.

Iter(k)	$\mathbf{x}^{(k)}$	$f^{(k)}$	$\mathbf{g}^{(k)}$	α_k
1	[7.5000, 9.0000]	13.2823	[-6.1332, -3.5132]	0.3675
2	[9.7539, 10.2911]	2.2965	[-0.7725, 1.3486]	0.2532
3	[9.9496, 9.9496]	1.9899	[-0.0001, -0.0001]	0.2494

Table 1: Iteration Summary of Steepest Descent Method for \mathbf{x}_a

Iter(k)	$\mathbf{x}^{(k)}$	$f^{(k)}$	$\mathbf{g}^{(k)}$	α_k
1	[-7.0000, -7.5000]	24.1427	[5.8357, 6.1332]	0.4496
2	[-9.6237, -10.2575]	2.3871	[1.2793, -1.2172]	0.2539
3	[-9.9485, -9.9484]	1.9899	[0.0043, 0.0045]	0.2521

Table 2: Iteration Summary of Steepest Descent Method for \mathbf{x}_b

For MATLAB function for this problem refer to Listing 2 at page 11 & Listing 7 at page 17 and the call to the function can be referred at Listing 1 at page 8 with corresponding output at Listing 9 at page 21.

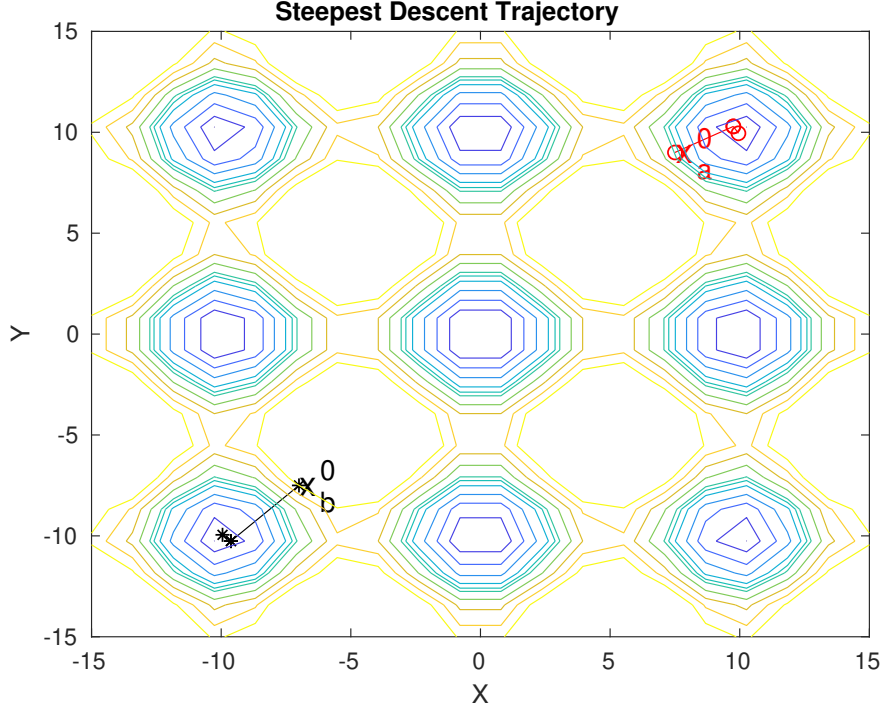


Figure 1: Steepest Descent trajectories

Exercise 2

In conjugate gradient method, we start with an initial search direction as the negative of gradient, i.e. $d^{(0)} = -g^{(0)}$. We then find the step length along this search direction using the Fibonacci search method. For future iterations we compute the conjugate directions using the following approach:

$$\begin{aligned}\beta_k &= \max(0, \frac{g^{(k+1)T}[g^{(k+1)} - g^{(k)}]}{g^{(k)T}g^{(k)}}) \\ d^{(k+1)} &= -g^{(k+1)} + \beta_k d^{(k)} \\ \alpha_k &= \underset{\alpha \geq 0}{\operatorname{argmin}} f(x^{(k)} + \alpha d^{(k)}) \\ x^{(k+1)} &= x^{(k)} + \alpha_k d^{(k)}\end{aligned}$$

We carry out the above updates repeatedly until a tolerance is achieved. Also, for non-quadratic functions we reset the search direction to the negative gradient after $(n + 1)$ steps.

For the given function in Eq. 1 along with its gradient as per Eq. 2 we carry out the Conjugate gradient method for 2 different starting points. The details of the iterations are shown in Table 3 and Table 4. The trajectories are shown in Figure 2.

For MATLAB function for this problem refer to Listing 3 at page 12 & Listing 7 at page 17 and the call to the function can be referred at Listing 1 at page 8 with corresponding output at Listing 9 at page 21.

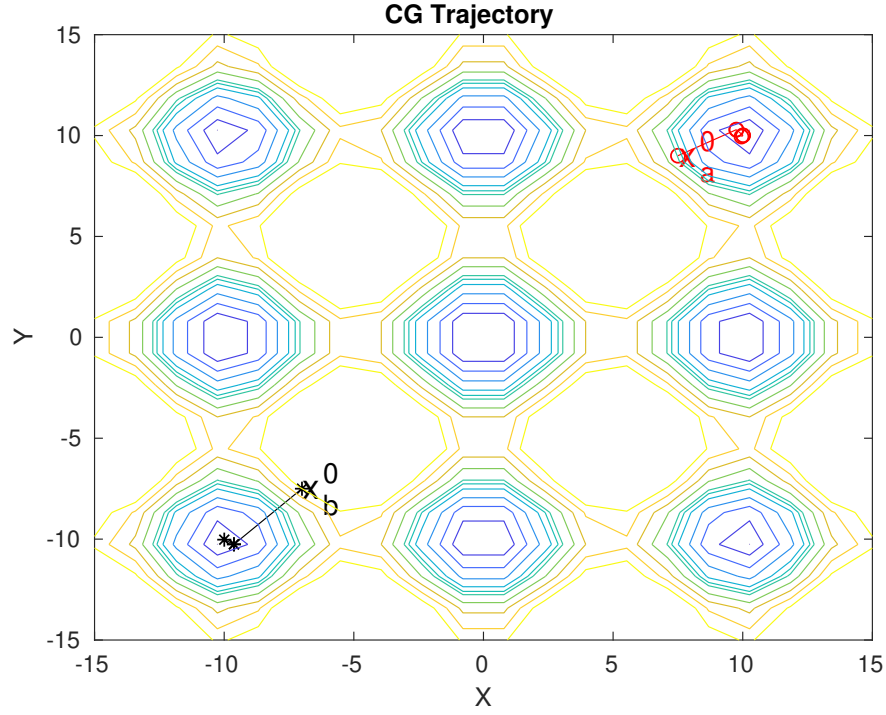


Figure 2: Conjugate Gradient trajectories

Iter(k)	$x^{(k)}$	$f^{(k)}$	$g^{(k)}$	α_k
1	[7.5000, 9.0000]	13.2823	[-6.1332, -3.5132]	0.3675
2	[9.7539, 10.2911]	2.2965	[-0.7725, 1.3486]	0.2416
3	[10.0122, 10.0063]	2.0041	[0.2483, 0.2252]	0.2521
4	[9.9496, 9.9495]	1.9899	[0.0002, -0.0002]	0.2512

Table 3: Iteration Summary of Conjugate Gradient Method for \mathbf{x}_a

Iter(k)	$x^{(k)}$	$f^{(k)}$	$g^{(k)}$	α_k
1	[-7.0000, -7.5000]	24.1427	[5.8357, 6.1332]	0.4496
2	[-9.6237, -10.2575]	2.3871	[1.2793, -1.2172]	0.2435
3	[-9.9970, -10.0261]	2.0060	[-0.1881, -0.3035]	0.2521

Table 4: Iteration Summary of Conjugate Gradient Method for \mathbf{x}_b

Exercise 3

For SRS method we start with an initial estimate of the approximation to inverse of Hessian matrix as the identity matrix. We then compute the search direction and the subsequent updates as follows:

$$\begin{aligned} d^{(k)} &= -H_k g^{(k)} \\ \alpha_k &= \underset{\alpha \geq 0}{\operatorname{argmin}} f(x^{(k)} + \alpha d^{(k)}) \\ x^{(k+1)} &= x^{(k)} + \alpha_k d^{(k)} \\ H_{k+1} &= H_k + \frac{(\Delta x^{(k)} - H_k \Delta g^{(k)})(\Delta x^{(k)} - H_k \Delta g^{(k)})^T}{\Delta g^{(k)T}(\Delta x^{(k)} - H_k \Delta g^{(k)})} \\ k &\leftarrow k + 1 \end{aligned}$$

We carry out the above iterations repeatedly till a certain tolerance is met.

For the given function in Eq. 1 along with its gradient as per Eq. 2 we carry out the SRS method for 2 different starting points. The details of the iterations are shown in Table 5 and Table 6. The trajectories are shown in Figure 3.

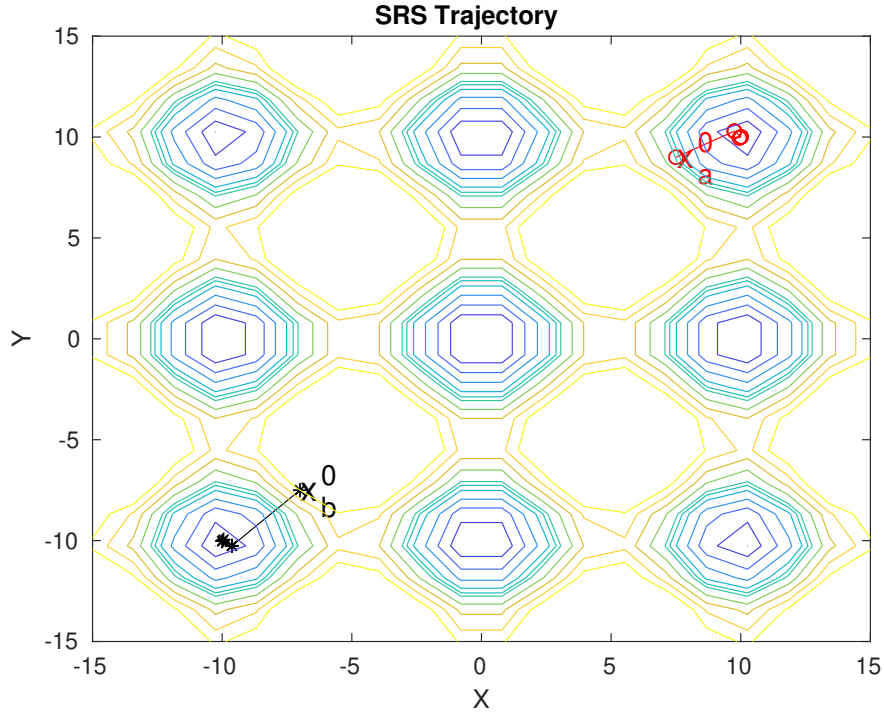


Figure 3: SRS trajectories

For MATLAB function for this problem refer to Listing 4 at page 13 & Listing 7 at page 17 and the call to the function can be referred at Listing 1 at page 8 with corresponding output at Listing 9 at page 21.

Iter(k)	$x^{(k)}$	$f^{(k)}$	$g^{(k)}$	α_k
1	[7.5000, 9.0000]	13.2823	[-6.1332, -3.5132]	0.3675
2	[9.7539, 10.2911]	2.2965	[-0.7725, 1.3486]	0.2600
3	[10.0122, 10.0063]	2.0041	[0.2483, 0.2252]	0.7361
4	[9.9496, 9.9495]	1.9899	[0.0002, -0.0002]	0.9901

Table 5: Iteration Summary of SRS for \mathbf{x}_a

Iter(k)	$x^{(k)}$	$f^{(k)}$	$g^{(k)}$	α_k
1	[-7.0000, -7.5000]	24.1427	[5.8357, 6.1332]	0.4496
2	[-9.6237, -10.2575]	2.3871	[1.2793, -1.2172]	0.2627
3	[-9.9970, -10.0261]	2.0060	[-0.1881, -0.3035]	0.6001
4	[-9.9494, -9.9497]	1.9899	[0.0006, -0.0004]	0.9905

Table 6: Iteration Summary of SRS for \mathbf{x}_b

Exercise 4

For DFP method we start with an initial estimate of the approximation to inverse of Hessian matrix as the identity matrix. We then compute the search direction and the subsequent updates as follows:

$$\begin{aligned}
d^{(k)} &= -H_k g^{(k)} \\
\alpha_k &= \underset{\alpha \geq 0}{\operatorname{argmin}} f(x^{(k)} + \alpha d^{(k)}) \\
x^{(k+1)} &= x^{(k)} + \alpha_k d^{(k)} \\
H_{k+1} &= H_k + \frac{\Delta x^{(k)} \Delta x^{(k)T}}{\Delta g^{(k)T} \Delta x^{(k)}} - \frac{(H_k \Delta g^{(k)})(H_k \Delta g^{(k)T})}{\Delta g^{(k)T} (H_k \Delta g^{(k)})} \\
k &\leftarrow k + 1
\end{aligned}$$

We carry out the above iterations repeatedly till a certain tolerance is met.

For the given function in Eq. 1 along with its gradient as per Eq. 2 we carry out the DFP method for 2 different starting points. The details of the iterations are shown in Table 7 and Table 8. The trajectories are shown in Figure 4.

Iter(k)	$x^{(k)}$	$f^{(k)}$	$g^{(k)}$	α_k
1	[7.5000, 9.0000]	13.2823	[-6.1332, -3.5132]	0.3675
2	[9.7539, 10.2911]	2.2965	[-0.7725, 1.3486]	0.0000

Table 7: Iteration Summary of DFP for \mathbf{x}_a

For MATLAB function for this problem refer to Listing 5 at page 15 & Listing 7 at page 17 and the call to the function can be referred at Listing 1 at page 8 with corresponding output at Listing 9 at page 21.

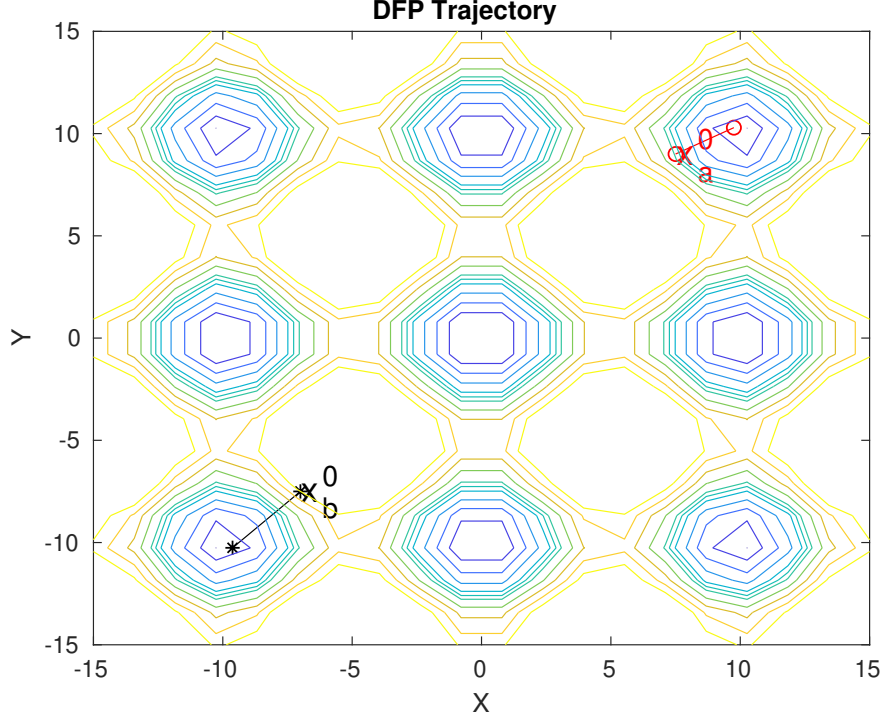


Figure 4: DFP trajectories

Iter(k)	$x^{(k)}$	$f^{(k)}$	$g^{(k)}$	α_k
1	[-7.0000, -7.5000]	24.1427	[5.8357, 6.1332]	0.4496
2	[-9.6237, -10.2575]	2.3871	[1.2793, -1.2172]	0.0000

Table 8: Iteration Summary of DFP for x_b

Exercise 5

For BFGS method we start with an initial estimate of the approximation to inverse of Hessian matrix as the identity matrix. We then compute the search direction and the subsequent updates as follows:

$$\begin{aligned}
 d^{(k)} &= -H_k g^{(k)} \\
 \alpha_k &= \underset{\alpha \geq 0}{\operatorname{argmin}} f(x^{(k)} + \alpha d^{(k)}) \\
 x^{(k+1)} &= x^{(k)} + \alpha_k d^{(k)} \\
 H_{k+1} &= H_k + \left(1 + \frac{\Delta g^{(k)T} H_k \Delta g^{(k)}}{\Delta g^{(k)T} \Delta x^{(k)}} \right) \frac{\Delta x^{(k)} \Delta x^{(k)T}}{\Delta g^{(k)T} \Delta x^{(k)}} - \left(\frac{(H_k \Delta g^{(k)} \Delta x^{(k)T}) + (H_k \Delta g^{(k)} \Delta x^{(k)T})^T}{\Delta g^{(k)T} \Delta x^{(k)}} \right) \\
 k &\leftarrow k + 1
 \end{aligned}$$

We carry out the above iterations repeatedly till a certain tolerance is met.

For the given function in Eq. 1 along with its gradient as per Eq. 2 we carry out the DFP method for 2 different starting points. The details of the iterations are shown in Table 9 and Table 10. The trajectories are shown in Figure 5.

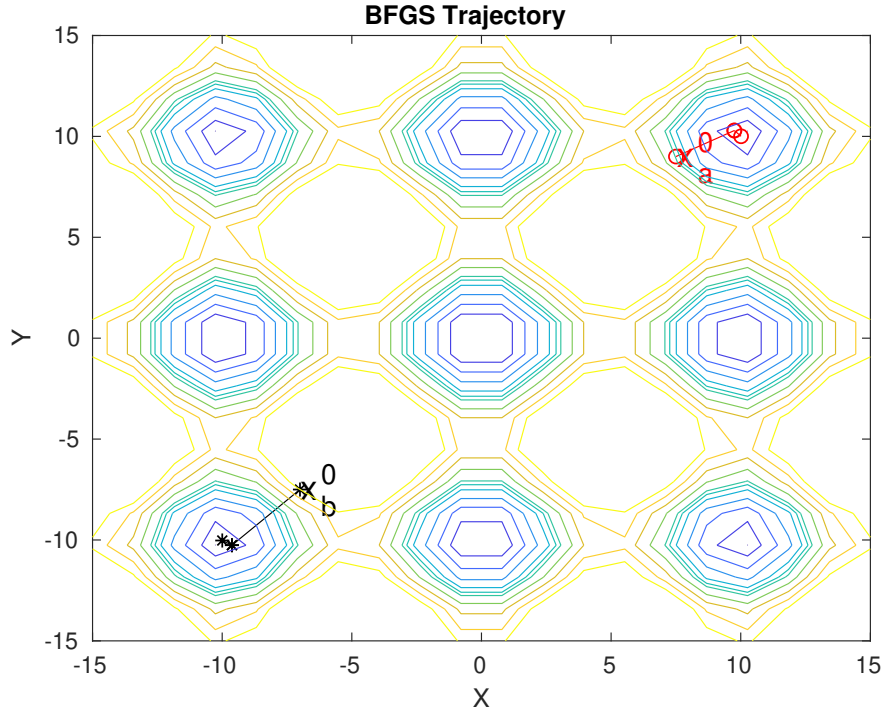


Figure 5: BFGS trajectories

Iter(k)	$x^{(k)}$	$f^{(k)}$	$g^{(k)}$	α_k
1	[7.5000, 9.0000]	13.2823	[-6.1332, -3.5132]	0.3675
2	[9.7539, 10.2911]	2.2965	[-0.7725, 1.3486]	0.2416
3	[10.0122, 10.0063]	2.0041	[0.2483, 0.2252]	0.0000

Table 9: Iteration Summary of BFGS for \mathbf{x}_a

Iter(k)	$x^{(k)}$	$f^{(k)}$	$g^{(k)}$	α_k
1	[-7.0000, -7.5000]	24.1427	[5.8357, 6.1332]	0.4496
2	[-9.6237, -10.2575]	2.3871	[1.2793, -1.2172]	0.2435
3	[-9.9970, -10.0261]	2.0060	[-0.1881, -0.3035]	0.0000

Table 10: Iteration Summary of BFGS for \mathbf{x}_b

For MATLAB function for this problem refer to Listing 6 at page 16 & Listing 7 at page 17 and the call to the function can be referred at Listing 1 at page 8 with corresponding output at Listing 9 at page 21.

MATLAB Code

Listing 1: Main Code

```
1 % ECE 580 HW3
2 % Rahul Deshmukh
3 % deshmun5@purdue.edu
4 clc; clear all; close all;
5 format short;
6 %% include paths
7 addpath(' ../OptimModule/line_search/ ');
8 addpath(' ../OptimModule/optimizers/unc/ ');
9 addpath(' ../OptimModule/optimizers/unc/QuasiNewton/ ');
10 save_dir = './pix/';
11
12 %% Rastrigin Function
13 fun2plot = @(x1,x2) 20 + (x1/10).^2 + (x2/10).^2 -10*(cos(2*pi*x1/10) + ...
    cos(2*pi*x2/10));
14 grad = @(x1,x2) [ (x1/50) +10*(2*pi/10)*(sin(2*pi*x1/10));...
    (x2/50) +10*(2*pi/10)*(sin(2*pi*x2/10))];
15
16
17 fun = @(x) 20 + (x(1)/10)^2 + (x(2)/10)^2 -10*(cos(2*pi*x(1)/10) + ...
    cos(2*pi*x(2)/10));
18 grad = @(x) [ (x(1)/50)+10*(2*pi/10)*(sin(2*pi*x(1)/10));...
    (x(2)/50)+10*(2*pi/10)*(sin(2*pi*x(2)/10))];
19
20 % initial guesses
21 xa = [7.5; 9.0]; xb = [-7.0; -7.5];
22
23 %% Problem 1: Steepest Descent
24 fprintf('\n-----SD-----\n');
25 [x_str1_sd, history1_sd] = steepest_descent(xa, fun, grad);
26 print_table(history1_sd); x_str1_sd
27 [x_str2_sd, history2_sd] = steepest_descent(xb, fun, grad);
28 print_table(history2_sd); x_str2_sd
29
30 % plotting Steepest descent paths
31 a = -15; b = 15;
32 x = linspace(a,b,20);
33 y = linspace(a,b,20);
34 [X, Y] = meshgrid(x,y);
35 Z = fun2plot(X,Y);
36
37 fig1 = figure(1);
38 hold on;
39 lvl_list1_sd = plot_traj(history1_sd, 'red', 'o', 'x^0_a');
40 lvl_list2_sd = plot_traj(history2_sd, 'black', '*', 'x^0_b');
41 lvl_list_sd = make_lvl_set(lvl_list1_sd, lvl_list2_sd);
42 contour(X,Y,Z, lvl_list_sd);
43 hold off;
44 xlabel('X'); ylabel('Y');
45 title('Steepest Descent Trajectory')
46 xlim([a,b]);
47 ylim([a,b]);
48 xticks(a:5:b);
49 yticks(a:5:b);
50 box('on');
51 saveas(fig1, strcat(save_dir, 'plot_sd'), 'epsc')
52
```



```

53 %% Problem 2: CG Powell
54 fprintf('\n-----CG-----\n');
55 [x_str1_cg, history1_cg] = CG(xa, fun, grad);
56 print_table(history1_cg); x_str1_cg
57 [x_str2_cg, history2_cg] = CG(xb, fun, grad);
58 print_table(history2_cg); x_str2_cg
59
60 fig2 = figure(2);
61 hold on;
62 lvl_list1_cg = plot_traj(history1_cg, 'red', 'o', 'x^0_a');
63 lvl_list2_cg = plot_traj(history2_cg, 'black', '*', 'x^0_b');
64 lvl_list_cg = make_lvl_set(lvl_list1_cg, lvl_list2_cg);
65 contour(X, Y, Z, lvl_list_cg);
66 hold off;
67 xlabel('X'); ylabel('Y');
68 title('CG Trajectory')
69 xlim([a, b]);
70 ylim([a, b]);
71 xticks(a:5:b);
72 yticks(a:5:b);
73 box('on');
74 saveas(fig2, strcat(save_dir, 'plot_cg'), 'epsc')
75
76 %% Problem 3: Rank one correction (SRS) Algo
77 fprintf('\n-----SRS-----\n');
78 [x_str1_srs, history1_srs] = SRS(xa, fun, grad);
79 print_table(history1_srs); x_str1_srs
80 [x_str2_srs, history2_srs] = SRS(xb, fun, grad);
81 print_table(history2_srs); x_str2_srs
82
83 fig3 = figure(3);
84 hold on;
85 lvl_list1_srs = plot_traj(history1_srs, 'red', 'o', 'x^0_a');
86 lvl_list2_srs = plot_traj(history2_srs, 'black', '*', 'x^0_b');
87 lvl_list_srs = make_lvl_set(lvl_list1_srs, lvl_list2_srs);
88 contour(X, Y, Z, lvl_list_srs);
89 hold off;
90 xlabel('X'); ylabel('Y');
91 title('SRS Trajectory')
92 xlim([a, b]);
93 ylim([a, b]);
94 xticks(a:5:b);
95 yticks(a:5:b);
96 box('on');
97 saveas(fig3, strcat(save_dir, 'plot_srs'), 'epsc')
98
99
100 %% Problem 4: DFP Algo
101 fprintf('\n-----DFP-----\n');
102 [x_str1_dfp, history1_dfp] = DFP(xa, fun, grad);
103 print_table(history1_dfp); x_str1_dfp
104 [x_str2_dfp, history2_dfp] = DFP(xb, fun, grad);
105 print_table(history2_dfp); x_str2_dfp
106
107 fig4 = figure(4);
108 hold on;
109 lvl_list1_dfp = plot_traj(history1_dfp, 'red', 'o', 'x^0_a');
110 lvl_list2_dfp = plot_traj(history2_dfp, 'black', '*', 'x^0_b');
111 lvl_list_dfp = make_lvl_set(lvl_list1_dfp, lvl_list2_dfp);

```

```

112 contour(X,Y,Z,lv1_list_dfp);
113 hold off;
114 xlabel('X'); ylabel('Y');
115 title('DFP Trajectory')
116 xlim([a,b]);
117 ylim([a,b]);
118 xticks(a:5:b);
119 yticks(a:5:b);
120 box('on');
121 saveas(fig3, strcat(save_dir, 'plot_dfp'), 'eps')
122
123
124 %% Problem 5: BFGS Algo
125 fprintf('\n-----BFGS-----\n');
126 [x_str1_bfgs, history1_bfgs] = BFGS(xa, fun, grad);
127 print_table(history1_bfgs); x_str1_bfgs
128 [x_str2_bfgs, history2_bfgs] = BFGS(xb, fun, grad);
129 print_table(history2_bfgs); x_str2_bfgs
130
131 fig5 = figure(5);
132 hold on;
133 lv1_list1_bfgs = plot_traj(history1_bfgs, 'red', 'o', 'x^0_a');
134 lv1_list2_bfgs = plot_traj(history2_bfgs, 'black', '*', 'x^0_b');
135 lv1_list_bfgs = make_lv1_set(lv1_list1_bfgs, lv1_list2_bfgs);
136 contour(X,Y,Z,lv1_list_bfgs);
137 hold off;
138 xlabel('X'); ylabel('Y');
139 title('BFGS Trajectory')
140 xlim([a,b]);
141 ylim([a,b]);
142 xticks(a:5:b);
143 yticks(a:5:b);
144 box('on');
145 saveas(fig3, strcat(save_dir, 'plot_bfgs'), 'eps')
146
147
148 %% Local plotting function
149 function lv1_list = plot_traj(history,color,marker, disp_text)
150     lv1_list = [];
151     for i=1:history.Niters-1
152         x = history.data(i).x(1);
153         y = history.data(i).x(2);
154         u = history.data(i+1).x(1);
155         v = history.data(i+1).x(2);
156         plot([x,u],[y,v], 'color', color, 'marker', marker);
157         lv1_list = [lv1_list, history.data(i).f_k];
158     end
159     lv1_list = [lv1_list, history.data(i+1).f_k];
160     text(history.data(1).x(1), history.data(1).x(2), disp_text, 'color', color, ...
161         'FontSize', 15);
162
163 end
164
165 function L = make_lv1_set(list1, list2)
166     Lmin = min(min(list1), min(list2));
167     Lmax = max(max(list1), max(list2));
168     temp = linspace(Lmin, Lmax, 10);
169     L = [list1, list2, temp];
170 end
171
172

```

```

170 function printTable(history)
171     rowvec_fmt = [' ', repmat('%0.4f ', 1, numel(history.data(1).x)-1), '%0.4f'];
172     fprintf('Iter(k)\t x_k\t f_k\t g_k\t alpha_k\n');
173
174     for k=1:history.Niters
175         fprintf('%d\t',k);
176         fprintf(rowvec_fmt,history.data(k).x); fprintf('\t');
177         fprintf('%0.4f\t',history.data(k).f_k);
178         fprintf(rowvec_fmt,history.data(k).g_k); fprintf('\t');
179         fprintf('%0.4f\t',history.data(k).alpha_k);fprintf('\n');
180     end
181 end

```

Listing 2: Steepest Descent

```

1 function [x_k, history_out] = steepest_descent(x_k,fun,grad,alpha,...
2                                     verbose, TolFun, TolX, TolGrad)
3     switch nargin
4         case 3
5             alpha_fixed = 0;
6             verbose = 0;
7             TolFun = 1e-4;
8             TolX = 1e-4;
9             TolGrad = 1e-4;
10        case 4
11            alpha_fixed = 1;
12            verbose = 0;
13            TolFun = 1e-4;
14            TolX = 1e-4;
15            TolGrad = 1e-4;
16        case 5
17            alpha_fixed = 1;
18            TolFun = 1e-4;
19            TolX = 1e-4;
20            TolGrad = 1e-4;
21        case 6
22            alpha_fixed = 1;
23            TolX = 1e-4;
24            TolGrad = 1e-4;
25        case 7
26            alpha_fixed = 1;
27            TolGrad = 1e-4;
28    end
29
30    history.parameter.TolFun = TolFun;
31    history.parameter.TolX = TolX;
32    history.parameter.TolGrad = TolGrad;
33
34    done = 0;
35    if alpha_fixed
36        history.name = 'Grad Descent (fixed alpha)';
37    else
38        history.name = 'Steepest Descent';
39    end
40    count=1;
41    history.data(count).x = x_k;
42    f_k = fun(x_k);
43    history.data(count).f_k = f_k;

```

```

44     g_k = grad(x_k);
45     history.data(count).g_k = g_k;
46     while ~done
47         d_k = -g_k;
48         if ~alpha_fixed
49             [xa,xb, initial_interval_summ]= get_search_interval(x_k, fun, d_k);
50             history.data(count).initial_search_interval_summary = ...
                initial_interval_summ;
51             [xa,~,line_search_summary] = fibonacci_method(xa, xb, fun);
52             history.data(count).line_search_summary = line_search_summary;
53             alpha_k = norm((xa-x_k),2)/norm(d_k,2);
54         end
55         history.data(count).alpha_k = alpha_k;
56         Δ_x_k = alpha_k*d_k;
57         x_k = x_k + Δ_x_k;
58         g_k = grad(x_k);
59         f_k_plus_1 = fun(x_k);
60
61         count = count+1;
62         history.data(count).x = x_k;
63         history.data(count).f_k = f_k_plus_1;
64         history.data(count).g_k = g_k;
65
66         if abs(f_k_plus_1 - f_k) < TolFun || norm(alpha_k*d_k,2) < TolX || ...
            norm(g_k) < TolGrad
67             done = 1;
68         end
69     end
70     history.Niters = count -1;
71
72     if nargin>1
73         history_out = history;
74     end
75     f_k = f_k_plus_1;
76 end

```

Listing 3: Conjugate Gradient (Powell)

```

1 function [x_k, history_out] = CG(x_k,fun,grad,...
2     verbose, TolFun, TolX, TolGrad)
3     history.name = 'Conjugate Gradient';
4     switch nargin
5         case 3
6             verbose = 0;
7             TolFun = 1e-4;
8             TolX = 1e-4;
9             TolGrad = 1e-4;
10        case 4
11            TolFun = 1e-4;
12            TolX = 1e-4;
13            TolGrad = 1e-4;
14        case 5
15            TolX = 1e-4;
16            TolGrad = 1e-4;
17        case 6
18            TolGrad = 1e-4;
19    end
20    history.parameter.TolFun = TolFun;

```

```

21     history.parameter.TolX = TolX;
22     history.parameter.TolGrad = TolGrad;
23
24     done = 0;
25     count=1;
26     f_k = fun(x_k);
27     g_k = grad(x_k);
28     history.data(count).x = x_k;
29     history.data(count).f_k = f_k;
30     history.data(count).g_k = g_k;
31     d_k = -g_k;
32     while ~done
33         % get the step length
34         % initial search interval
35         [xa,xb, initial_interval_summ]= get_search_interval(x_k, fun, d_k);
36         history.data(count).initial_search_interval_summary = ...
            initial_interval_summ;
37         % 1-d line search using fibonacci method
38         [xa,~, line_search_summary] = fibonacci_method(xa, xb, fun);
39         history.data(count).line_search_summary = line_search_summary;
40         alpha_k = norm((xa-x_k),2)/norm(d_k,2);
41         history.data(count).alpha_k = alpha_k;
42
43         Δ_x_k = alpha_k*d_k;
44         x_k = x_k + Δ_x_k;
45         Δ_g_k = grad(x_k) -g_k;
46         old_g_k_norm = g_k'*g_k;
47         g_k = g_k + Δ_g_k;
48
49         if mod(count,length(x_k)+1) ≠ 0
50             beta_k = max(0, (g_k'*(Δ_g_k))/old_g_k_norm);
51             d_k = -g_k + beta_k*d_k;
52         else
53             % reset to neg grad
54             d_k = -g_k;
55         end
56
57         count = count+1;
58         f_k_plus_1 = fun(x_k);
59         history.data(count).x = x_k;
60         history.data(count).f_k = f_k_plus_1;
61         history.data(count).g_k = g_k;
62         history.data(count).alpha_k = alpha_k;
63
64         % check if done
65         if abs(f_k_plus_1 - f_k) < TolFun || norm(Δ_x_k ,2) <TolX || ...
            norm(g_k,2)< TolGrad || count>100
66             done = 1;
67         end
68         f_k = f_k_plus_1;
69     end
70     history.Niters = count -1;
71     if nargout>1
72         history_out = history;
73     end
74 end

```

Listing 4: SRS

```

1 function [x_k, history_out] = SRS(x_k, fun, grad, ...
2                                 verbose, TolFun, TolX, TolGrad)
3     history.name = 'Quasi-Newton: SRS (Rank-1 Correction)';
4     switch nargin
5         case 3
6             verbose = 0;
7             TolFun = 1e-4;
8             TolX = 1e-4;
9             TolGrad = 1e-4;
10        case 4
11            TolFun = 1e-4;
12            TolX = 1e-4;
13            TolGrad = 1e-4;
14        case 5
15            TolX = 1e-4;
16            TolGrad = 1e-4;
17        case 6
18            TolGrad = 1e-4;
19    end
20    history.parameter.TolFun = TolFun;
21    history.parameter.TolX = TolX;
22    history.parameter.TolGrad = TolGrad;
23
24    done = 0;
25    H_k = eye(length(x_k));
26    count=1;
27    f_k = fun(x_k);
28    g_k = grad(x_k);
29    history.data(count).x = x_k;
30    history.data(count).f_k = f_k;
31    history.data(count).g_k = g_k;
32    while ~done
33        % find search direction
34        d_k = -H_k*g_k;
35
36        % get the step length
37        % initial search interval
38        [xa,xb, initial_interval_summ]= get_search_interval(x_k, fun, d_k);
39        history.data(count).initial_search_interval_summary = ...
40            initial_interval_summ;
41        % 1-d line search using fibonacci method
42        [xa,~,line_search_summary] = fibonacci_method(xa, xb, fun);
43        history.data(count).line_search_summary = line_search_summary;
44        alpha_k = norm((xa-x_k),2)/norm(d_k,2);
45        history.data(count).alpha_k = alpha_k;
46        % update H_k
47        Δ_x_k = alpha_k*d_k;
48        x_k = x_k + Δ_x_k;
49        Δ_g_k = grad(x_k) - g_k;
50        g_k = g_k + Δ_g_k;
51
52        Δ_H_k = (Δ_x_k - H_k*Δ_g_k);
53        Δ_H_k = (Δ_H_k*Δ_H_k')/(Δ_g_k'*Δ_H_k);
54        H_k = H_k + Δ_H_k;
55
56        count = count+1;
57        f_k_plus_1 = fun(x_k);
58        history.data(count).x = x_k;

```

```

58     history.data(count).f_k = f_k_plus_1;
59     history.data(count).g_k = g_k;
60     % check if converged to solution
61     if abs(f_k_plus_1 - f_k) < TolFun || norm( $\Delta$ _x_k ,2) < TolX || ...
        norm(g_k,2) < TolGrad || count > 100
62         done = 1;
63     end
64     f_k = f_k_plus_1;
65 end
66 history.Niters = count - 1;
67
68 if nargin > 1
69     history_out = history;
70 end
71 end

```

Listing 5: DFP

```

1 function [x_k, history_out] = DFP(x_k, fun, grad, ...
2     verbose, TolFun, TolX, TolGrad)
3     history.name = 'Quasi-Newton: DFP';
4     switch nargin
5         case 3
6             verbose = 0;
7             TolFun = 1e-4;
8             TolX = 1e-4;
9             TolGrad = 1e-4;
10        case 4
11            TolFun = 1e-4;
12            TolX = 1e-4;
13            TolGrad = 1e-4;
14        case 5
15            TolX = 1e-4;
16            TolGrad = 1e-4;
17        case 6
18            TolGrad = 1e-4;
19    end
20    history.parameter.TolFun = TolFun;
21    history.parameter.TolX = TolX;
22    history.parameter.TolGrad = TolGrad;
23
24    done = 0;
25    H_k = eye(length(x_k));
26    count = 1;
27    f_k = fun(x_k);
28    g_k = grad(x_k);
29    history.data(count).x = x_k;
30    history.data(count).f_k = f_k;
31    history.data(count).g_k = g_k;
32    while ~done
33        % find search direction
34        d_k = -H_k * g_k;
35
36        % get the step length
37        % initial search interval
38        [xa,xb, initial_interval_summ] = get_search_interval(x_k, fun, d_k);
39        history.data(count).initial_search_interval_summary = ...
            initial_interval_summ;

```

```

40     % 1-d line search using fibonacci method
41     [xa,~,line_search_summary] = fibonacci_method(xa, xb, fun);
42     history.data(count).line_search_summary = line_search_summary;
43     alpha_k = norm((xa-x_k),2)/norm(d_k,2);
44     history.data(count).alpha_k = alpha_k;
45     % update H_k
46     Δ_x_k = alpha_k*d_k;
47     x_k = x_k + Δ_x_k;
48     Δ_g_k = grad(x_k) - g_k;
49     g_k = g_k + Δ_g_k;
50
51     a_k = (Δ_g_k'*Δ_x_k);
52     Δ_H_k1 = (Δ_x_k*Δ_x_k')/a_k;
53     Δ_H_k2 = ((H_k*Δ_g_k)*(H_k*Δ_g_k'))/(Δ_g_k'*H_k*g_k);
54     H_k = H_k + Δ_H_k1 - Δ_H_k2;
55
56     count = count+1;
57     f_k_plus_1 = fun(x_k);
58     history.data(count).x = x_k;
59     history.data(count).f_k = f_k_plus_1;
60     history.data(count).g_k = g_k;
61
62     % check if converged to solution
63     if abs(f_k_plus_1 - f_k) < TolFun || norm(Δ_x_k,2) < TolX || ...
        norm(g_k,2) < TolGrad || count > 100
64         done = 1;
65     end
66     f_k = f_k_plus_1;
67 end
68 history.Niters = count - 1;
69
70 if nargout > 1
71     history_out = history;
72 end
73 end

```

Listing 6: BFGS

```

1 function [x_k, history_out] = BFGS(x_k, fun, grad, ...
2                                     verbose, TolFun, TolX, TolGrad)
3     history.name = 'Quasi-Newton: BFGS';
4     switch nargin
5         case 3
6             verbose = 0;
7             TolFun = 1e-4;
8             TolX = 1e-4;
9             TolGrad = 1e-4;
10        case 4
11            TolFun = 1e-4;
12            TolX = 1e-4;
13            TolGrad = 1e-4;
14        case 5
15            TolX = 1e-4;
16            TolGrad = 1e-4;
17        case 6
18            TolGrad = 1e-4;
19    end
20    history.parameter.TolFun = TolFun;

```



```

21     history.parameter.TolX = TolX;
22     history.parameter.TolGrad = TolGrad;
23
24     done = 0;
25     H_k = eye(length(x_k));
26     count=1;
27     f_k = fun(x_k);
28     g_k = grad(x_k);
29     history.data(count).x = x_k;
30     history.data(count).f_k = f_k;
31     history.data(count).g_k = g_k;
32     while ~done
33         % find search direction
34         d_k = -H_k*g_k;
35
36         % get the step length
37         % initial search interval
38         [xa,xb, initial_interval_summ]= get_search_interval(x_k, fun, d_k);
39         history.data(count).initial_search_interval_summary = ...
            initial_interval_summ;
40         % 1-d line search using fibonacci method
41         [xa,~,line_search_summary] = fibonacci_method(xa, xb, fun);
42         history.data(count).line_search_summary = line_search_summary;
43         alpha_k = norm((xa-x_k),2)/norm(d_k,2);
44         history.data(count).alpha_k = alpha_k;
45         % update H_k
46         Δ_x_k = alpha_k*d_k;
47         x_k = x_k + Δ_x_k;
48         Δ_g_k = grad(x_k) - g_k;
49         g_k = g_k + Δ_g_k;
50
51         a_k = Δ_g_k'*Δ_x_k;
52         beta_k = (Δ_g_k'*H_k*g_k)/a_k;
53         Δ_H_k1 = (1 + beta_k)*(Δ_x_k*Δ_x_k')/a_k;
54         Δ_H_k2 = ((H_k*Δ_g_k*Δ_x_k') + (H_k*Δ_g_k*Δ_x_k'))/a_k;
55         H_k = H_k + Δ_H_k1 - Δ_H_k2;
56
57         count = count+1;
58         f_k_plus_1 = fun(x_k);
59         history.data(count).x = x_k;
60         history.data(count).f_k = f_k_plus_1;
61         history.data(count).g_k = g_k;
62         % check if converged to solution
63         if abs(f_k_plus_1 - f_k) < TolFun || norm(Δ_x_k ,2) < TolX || norm(g_k, ...
            2)< TolGrad || count>100
64             done = 1;
65         end
66         f_k = f_k_plus_1;
67     end
68     history.Niters = count -1;
69
70     if nargout>1
71         history.out = history;
72     end
73 end

```

Listing 7: Fibonacci Search

```

1 function [xa, xb, history_out] = fibonacci_method(xa, xb, fun, TolX, verbose)
2     history.name = 'Fibonacci Method';
3     Δ = 0.1; % for last value of rho
4     switch nargin
5     case 3
6         TolX = 1e-6;
7         verbose = 0;
8     case 4
9         verbose = 0;
10    end
11    history.params.Δ = Δ;
12    history.params.TolX = TolX;
13    % estimate Niter
14    Niter = get_Niter_fibb(xa,xb,Δ, TolX);
15    history.Niters = Niter;
16
17    % do range reduction
18    k = 1;
19    rho_k = 1 - (fibonacci(Niter+1)/fibonacci(Niter+1+1));
20    s = xa + rho_k*(xb-xa);
21    t = xa + (1-rho_k)*(xb-xa);
22    f1 = fun(s);
23    f2 = fun(t);
24    f_xa = fun(xa);
25    f_xb = fun(xb);
26    %
27    if verbose
28        fprintf('Iter(k)\t rho_k\t ak\t bk\t f(ak)\t f(bk)\t New uncertainty ...
29                interval (a,b)\t Uncertainty width\n');
30        rowvec_fmt = ['[', repmat('%0.4f', 1, numel(xa)-1), '%0.4f']];
31    end
32    for k=1:Niter
33        xa_prev = xa;
34        xb_prev = xb;
35        f_xa_prev = f_xa;
36        f_xb_prev = f_xb;
37        rho_k = 1 - (fibonacci(Niter-(k-1)+1)/fibonacci(Niter+1 - (k-1)+1));
38        if k == Niter
39            rho_k = rho_k - Δ;
40        end
41        if f1 < f2
42            xb=t;
43            f_xb = f2;
44            t=s;
45            s= xa + rho_k*(xb-xa);
46            f2 = f1;
47            f1 = fun(s);
48        else
49            xa = s;
50            f_xa = f1;
51            s = t;
52            t = xa + (1-rho_k)*(xb-xa);
53            f1 = f2;
54            f2 = fun(t);
55        end
56        %
57        if verbose

```

```

58         fprintf('%d\t',k);
59         fprintf('%0.4f\t',rho_k);
60         fprintf(rowvec_fmt,xa_prev); fprintf('\t');
61         fprintf(rowvec_fmt,xb_prev); fprintf('\t');
62         fprintf('%0.4f\t %0.4f\t',f_xa_prev,f_xb_prev);
63         fprintf('(');
64         fprintf(rowvec_fmt,xa);
65         fprintf(' ',sqrt(sum(')
66         fprintf(rowvec_fmt,xb);
67         fprintf(')\t%0.4f\n',norm((xb-xa),2));
68     end
69     %
70     history.data(k).x_prev = [xa_prev,xb_prev];
71     history.data(k).x_new = [xa,xb];
72 end
73 if verbose
74     fprintf("** Fibonacci method took %d iters **\n** Final uncertainty ...
75         region width was %0.4f **\n",...
76         Niter,norm((xb-xa),2));
77 end
78 if nargin>2
79     history_out = history;
80 end
81
82 end
83
84 function N = get_Niter_fibb(xa,xb,delta,TolX)
85     N = 1;
86     d = norm((xb-xa),2);
87     while d*(1+2*delta)/TolX > fibonacci(N+1+1)
88         N = N + 1;
89     end
90
91 end

```

Listing 8: Initial Search Interval

```

1 function [x_prev, x_nxt, history] = get_search_interval(x0, fun, direction,...
2                                     step_size)
3 % function for identifying the interval which contains a minimizer
4 % using x0 as initial guess and evaluating the fun till the function value
5 % increases btw 2 consecutive evals
6 % Input: fun: function handle for obj fun
7 %         step_size: initial step size, user param
8 %         direction: function handle for search direction
9 switch nargin
10     case 3
11         step_size = 0.1;
12     end
13 x_prev = x0;
14 x_cur = x_prev + step_size*direction;
15 step_size = 2*step_size;
16 x_nxt = x_cur + step_size*direction;
17
18 f1 = fun(x_prev);
19 f2 = fun(x_cur);
20 f3 = fun(x_nxt);

```

```

21     f = [f1,f2,f3];
22
23     count = 1;
24     history.data(count).interval = [x_prev, x_nxt];
25     while f(2) ≥ f(3)
26         step_size = 2*step_size;
27         x_prev = x_cur;
28         x_cur = x_nxt;
29         x_nxt = x_cur + step_size*direction;
30         f = [f(2),f(3), fun(x_nxt)];
31         count = count +1;
32         history.data(count).interval = [x_prev, x_nxt];
33     end
34     if nargout>2
35         history.name = 'initial search interval';
36         history.Niters = count;
37         history.parameter.step_size = step_size;
38     end
39 end

```

Output on MATLAB command window

Listing 9: MATLAB Output

```
1  -----SD-----
2  Iter(k)  x_k  f_k  g_k  alpha_k
3  1 [7.5000, 9.0000] 13.2823 [-6.1332, -3.5132] 0.3675
4  2 [9.7539, 10.2911] 2.2965 [-0.7725, 1.3486] 0.2532
5  3 [9.9496, 9.9496] 1.9899 [-0.0001, -0.0001] 0.2494
6
7  x_str1_sd =
8
9      9.9496
10     9.9496
11
12 Iter(k)  x_k  f_k  g_k  alpha_k
13 1 [-7.0000, -7.5000] 24.1427 [5.8357, 6.1332] 0.4496
14 2 [-9.6237, -10.2575] 2.3871 [1.2793, -1.2172] 0.2539
15 3 [-9.9485, -9.9484] 1.9899 [0.0043, 0.0045] 0.2521
16
17 x_str2_sd =
18
19     -9.9496
20     -9.9496
21
22 -----CG-----
23 Iter(k)  x_k  f_k  g_k  alpha_k
24 1 [7.5000, 9.0000] 13.2823 [-6.1332, -3.5132] 0.3675
25 2 [9.7539, 10.2911] 2.2965 [-0.7725, 1.3486] 0.2416
26 3 [10.0122, 10.0063] 2.0041 [0.2483, 0.2252] 0.2521
27 4 [9.9496, 9.9495] 1.9899 [0.0002, -0.0002] 0.2512
28
29 x_str1_cg =
30
31     9.9496
32     9.9496
33
34 Iter(k)  x_k  f_k  g_k  alpha_k
35 1 [-7.0000, -7.5000] 24.1427 [5.8357, 6.1332] 0.4496
36 2 [-9.6237, -10.2575] 2.3871 [1.2793, -1.2172] 0.2435
37 3 [-9.9970, -10.0261] 2.0060 [-0.1881, -0.3035] 0.2521
38
39 x_str2_cg =
40
41     -9.9496
42     -9.9496
43
44 -----SRS-----
45 Iter(k)  x_k  f_k  g_k  alpha_k
46 1 [7.5000, 9.0000] 13.2823 [-6.1332, -3.5132] 0.3675
47 2 [9.7539, 10.2911] 2.2965 [-0.7725, 1.3486] 0.2600
48 3 [10.0122, 10.0063] 2.0041 [0.2483, 0.2252] 0.7361
49 4 [9.9496, 9.9495] 1.9899 [0.0002, -0.0002] 0.9901
50
51 x_str1_srs =
52
53     9.9496
54     9.9496
```

```

55
56 Iter(k)  x_k  f_k  g_k  alpha_k
57 1 [-7.0000, -7.5000] 24.1427 [5.8357, 6.1332] 0.4496
58 2 [-9.6237, -10.2575] 2.3871 [1.2793, -1.2172] 0.2627
59 3 [-9.9970, -10.0261] 2.0060 [-0.1881, -0.3035] 0.6001
60 4 [-9.9494, -9.9497] 1.9899 [0.0006, -0.0004] 0.9905
61
62 x_str2_srs =
63
64     -9.9496
65     -9.9496
66
67 -----DFP-----
68 Iter(k)  x_k  f_k  g_k  alpha_k
69 1 [7.5000, 9.0000] 13.2823 [-6.1332, -3.5132] 0.3675
70 2 [9.7539, 10.2911] 2.2965 [-0.7725, 1.3486] 0.0000
71
72 x_str1_dfp =
73
74     9.7539
75     10.2911
76
77 Iter(k)  x_k  f_k  g_k  alpha_k
78 1 [-7.0000, -7.5000] 24.1427 [5.8357, 6.1332] 0.4496
79 2 [-9.6237, -10.2575] 2.3871 [1.2793, -1.2172] 0.0000
80
81 x_str2_dfp =
82
83     -9.6237
84     -10.2575
85
86 -----BFGS-----
87 Iter(k)  x_k  f_k  g_k  alpha_k
88 1 [7.5000, 9.0000] 13.2823 [-6.1332, -3.5132] 0.3675
89 2 [9.7539, 10.2911] 2.2965 [-0.7725, 1.3486] 0.2416
90 3 [10.0122, 10.0063] 2.0041 [0.2483, 0.2252] 0.0000
91
92 x_str1_bfgs =
93
94     10.0122
95     10.0063
96
97 Iter(k)  x_k  f_k  g_k  alpha_k
98 1 [-7.0000, -7.5000] 24.1427 [5.8357, 6.1332] 0.4496
99 2 [-9.6237, -10.2575] 2.3871 [1.2793, -1.2172] 0.2435
100 3 [-9.9970, -10.0261] 2.0060 [-0.1881, -0.3035] 0.0000
101
102 x_str2_bfgs =
103
104     -9.9970
105     -10.0261

```