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:

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

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

We are also given two starting points:

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

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.

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

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

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 $\boldsymbol{d} = -g(\boldsymbol{x}^{(k)})$.

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)	$x^{(k)}$	$f^{(k)}$	$g^{(k)}$	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]	0.2532
3	[9.9496, 9.9496]	1.9899	[-0.0001, -0.0001]	0.2494

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

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]	0.2539
3	[-9.9485, -9.9484]	1.9899	[0.0043, 0.0045]	0.2521

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

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.

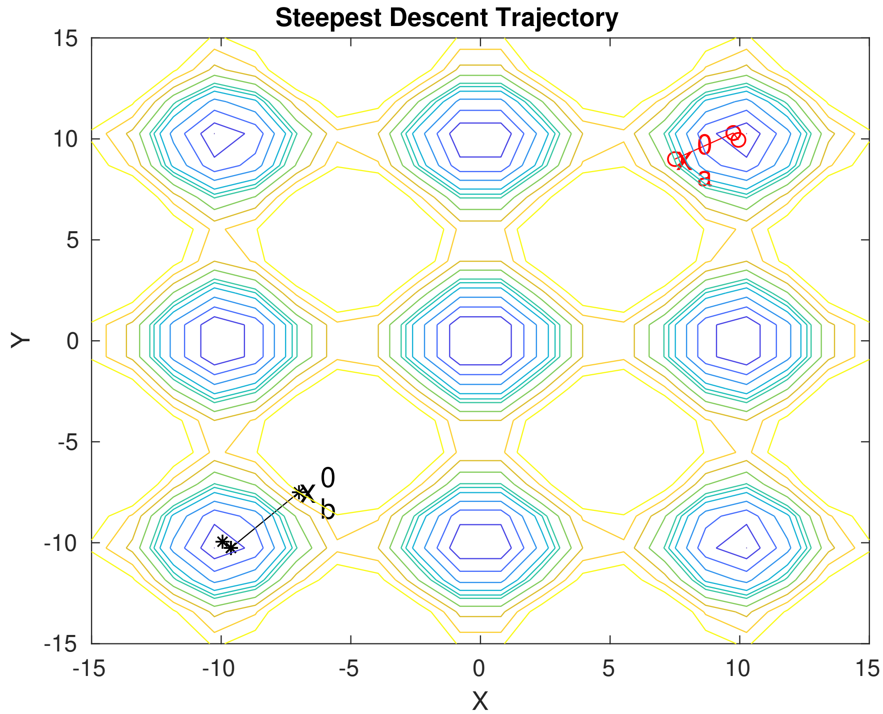

Figure 1: Steepest Descent trajectories

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.

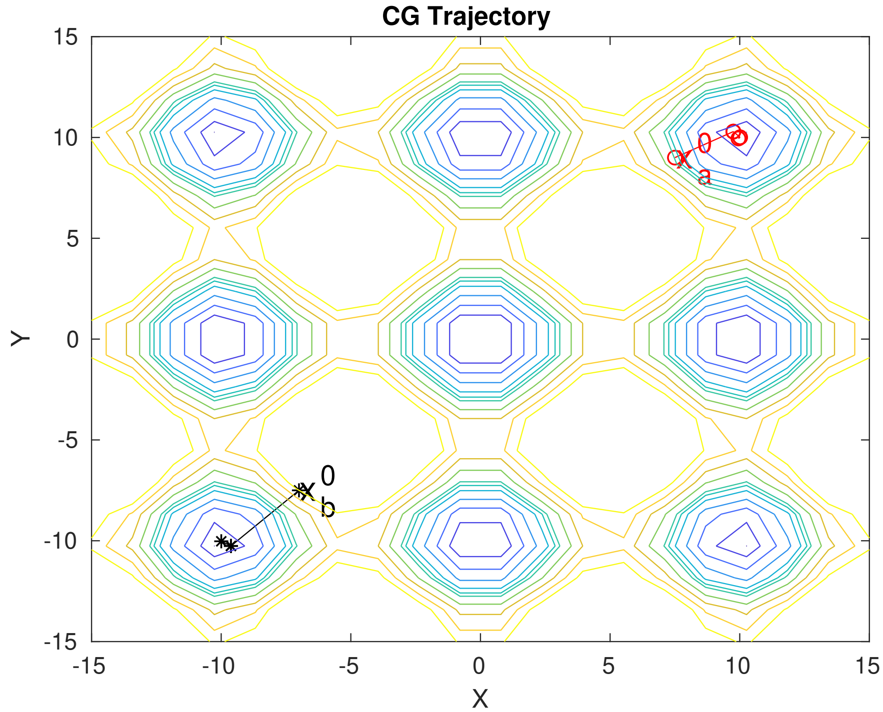

Figure 2: Conjugate Gradient trajectories

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 $\boldsymbol{x}_a$

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 $\boldsymbol{x}_b$

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:

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

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.

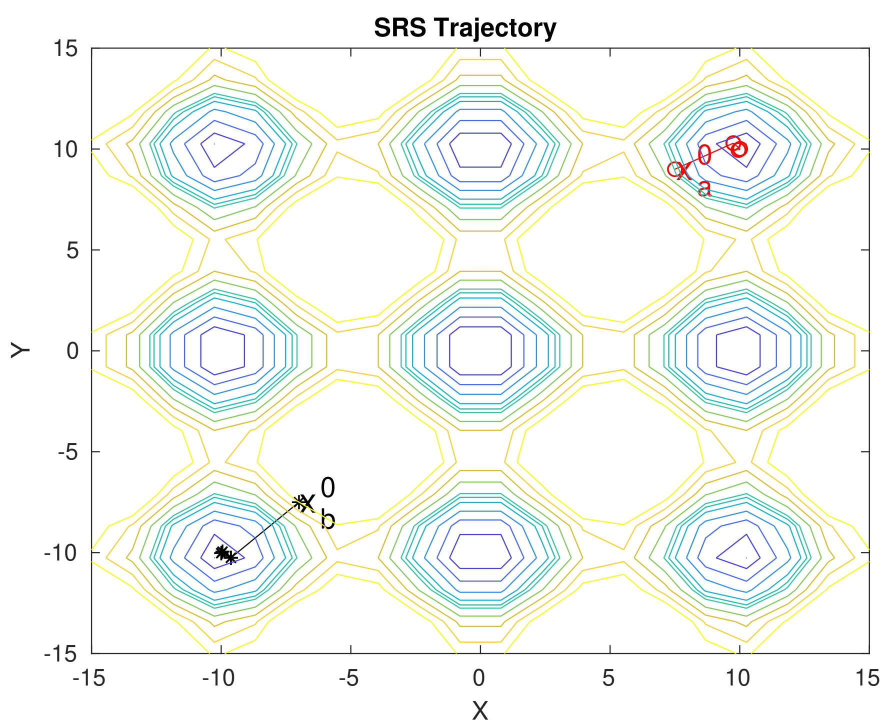

Figure 3: SRS trajectories

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 $\boldsymbol{x}_a$

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 $\boldsymbol{x}_b$

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 $\boldsymbol{x}_a$

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.

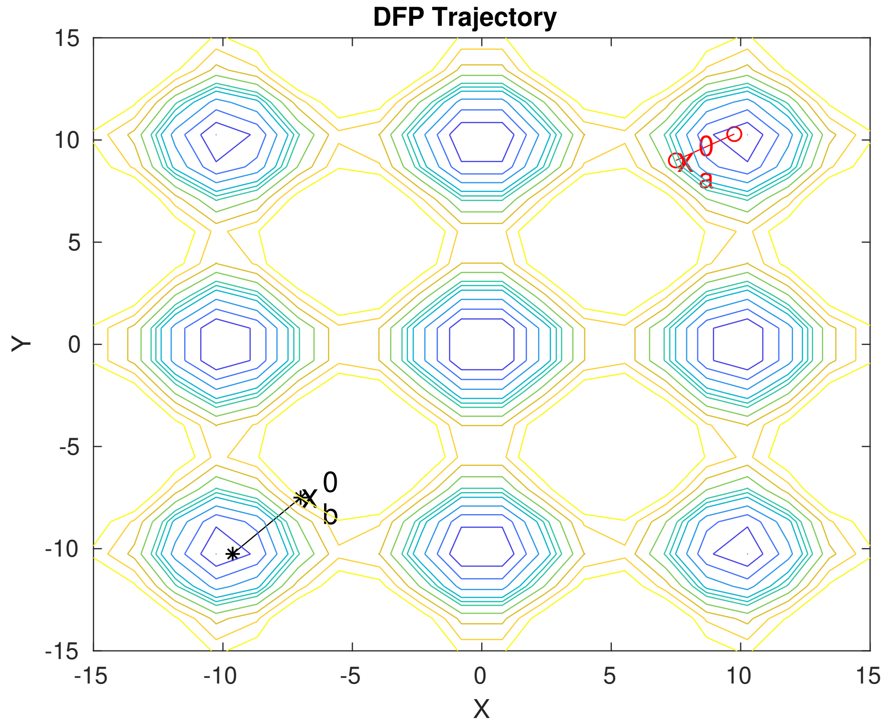

Figure 4: DFP trajectories

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 $\boldsymbol{x}_b$

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:

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

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.

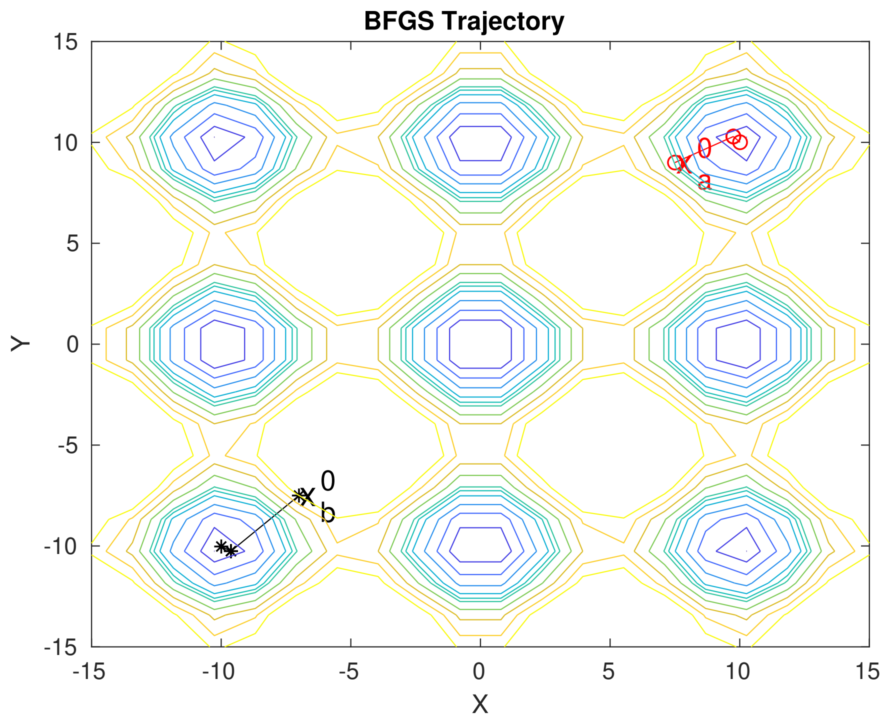

Figure 5: BFGS trajectories

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 $\boldsymbol{x}_a$

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 $\boldsymbol{x}_b$

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

```
% ECE 580 HW3
% Rahul Deshmukh
% deshmuk5@purdue.edu
clc; clear all; close all;
format short;
%% include paths
addpath('../OptimModule/line_search/');
addpath('../OptimModule/optimizers/unc/');
addpath('../OptimModule/optimizers/unc/QuasiNewton/');
save_dir = './pix/';

%% Rastrigin Function
fun2plot = @(x1,x2) 20 + (x1/10).^2 + (x2/10).^2 -10*(cos(2*pi*x1/10) + ...
    cos(2*pi*x2/10));
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))];

fun = @(x) 20 + (x(1)/10)^2 + (x(2)/10)^2 -10*(cos(2*pi*x(1)/10) + ...
    cos(2*pi*x(2)/10));
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))];
% initial guesses
xa = [7.5; 9.0]; xb = [-7.0; -7.5];

%% Problem 1: Steepest Descent
fprintf('\n--------------------SD-----------------\n');
[x_str1_sd, history1_sd] = steepest_descent(xa, fun, grad);
print_table(history1_sd);x_str1_sd
[x_str2_sd, history2_sd] = steepest_descent(xb, fun, grad);
print_table(history2_sd);x_str2_sd

% plotting Steepest descent paths
a = -15; b = 15;
x = linspace(a,b,20);
y = linspace(a,b,20);
[X, Y] = meshgrid(x,y);
Z = fun2plot(X,Y);

fig1 = figure(1);
hold on;
lvl_list1_sd = plot_traj(history1_sd,'red','o','x^0_a');
lvl_list2_sd = plot_traj(history2_sd, 'black','*','x^0_b');
lvl_list_sd = make_lvl_set(lvl_list1_sd,lvl_list2_sd);
contour(X,Y,Z,lvl_list_sd);
hold off;
xlabel('X'); ylabel('Y');
title('Steepest Descent Trajectory')
xlim([a,b]);
ylim([a,b]);
xticks(a:5:b);
yticks(a:5:b);
box('on');
saveas(fig1,strcat(save_dir,'plot_sd'),'epsc')

```

```
%% Problem 2: CG Powell
fprintf('\n--------------------CG-------------------\n');
[x_str1_cg, history1_cg] = CG(xa, fun, grad);
print_table(history1_cg);x_str1_cg
[x_str2_cg, history2_cg] = CG(xb, fun, grad);
print_table(history2_cg);x_str2_cg

fig2 = figure(2);
hold on;
lvl_list1_cg = plot_traj(history1_cg,'red','o','x^0_a');
lvl_list2_cg = plot_traj(history2_cg, 'black','*','x^0_b');
lvl_list_cg = make_lvl_set(lvl_list1_cg,lvl_list2_cg);
contour(X,Y,Z,lvl_list_cg);
hold off;
xlabel('X'); ylabel('Y');
title('CG Trajectory')
xlim([a,b]);
ylim([a,b]);
xticks(a:5:b);
yticks(a:5:b);
box('on');
saveas(fig2,strcat(save_dir,'plot_cg'),'epsc')

%% Problem 3: Rank one correction (SRS) Algo
fprintf('\n--------------------SRS-------------------\n');
[x_str1_srs, history1_srs] = SRS(xa, fun, grad);
print_table(history1_srs);x_str1_srs
[x_str2_srs, history2_srs] = SRS(xb, fun, grad);
print_table(history2_srs);x_str2_srs

fig3 = figure(3);
hold on;
lvl_list1_srs = plot_traj(history1_srs,'red','o','x^0_a');
lvl_list2_srs = plot_traj(history2_srs, 'black','*','x^0_b');
lvl_list_srs = make_lvl_set(lvl_list1_srs,lvl_list2_srs);
contour(X,Y,Z,lvl_list_srs);
hold off;
xlabel('X'); ylabel('Y');
title('SRS Trajectory')
xlim([a,b]);
ylim([a,b]);
xticks(a:5:b);
yticks(a:5:b);
box('on');
saveas(fig3,strcat(save_dir,'plot_srs'),'epsc')


%% Problem 4: DFP Algo
fprintf('\n--------------------DFP-------------------\n');
[x_str1_dfp, history1_dfp] = DFP(xa, fun, grad);
print_table(history1_dfp);x_str1_dfp
[x_str2_dfp, history2_dfp] = DFP(xb, fun, grad);
print_table(history2_dfp);x_str2_dfp

fig4 = figure(4);
hold on;
lvl_list1_dfp = plot_traj(history1_dfp,'red','o','x^0_a');
lvl_list2_dfp = plot_traj(history2_dfp, 'black','*','x^0_b');
lvl_list_dfp = make_lvl_set(lvl_list1_dfp,lvl_list2_dfp);
```

```
contour(X,Y,Z,lvl_list_dfp);
hold off;
xlabel('X'); ylabel('Y');
title('DFP Trajectory')
xlim([a,b]);
ylim([a,b]);
xticks(a:5:b);
yticks(a:5:b);
box('on');
saveas(fig3,strcat(save_dir,'plot_dfp'),'epsc')


%% Problem 5: BFGS Algo
fprintf('\n--------------------BFGS-------------------\n');
[x_str1_bfgs, history1_bfgs] = BFGS(xa, fun, grad);
print_table(history1_bfgs);x_str1_bfgs
[x_str2_bfgs, history2_bfgs] = BFGS(xb, fun, grad);
print_table(history2_bfgs);x_str2_bfgs

fig5 = figure(5);
hold on;
lvl_list1_bfgs = plot_traj(history1_bfgs,'red','o','x^0_a');
lvl_list2_bfgs = plot_traj(history2_bfgs, 'black','*','x^0_b');
lvl_list_bfgs = make_lvl_set(lvl_list1_bfgs,lvl_list2_bfgs);
contour(X,Y,Z,lvl_list_bfgs);
hold off;
xlabel('X'); ylabel('Y');
title('BFGS Trajectory')
xlim([a,b]);
ylim([a,b]);
xticks(a:5:b);
yticks(a:5:b);
box('on');
saveas(fig3,strcat(save_dir,'plot_bfgs'),'epsc')


%% Local plotting function
function lvl_list = plot_traj(history,color,marker, disp_text)
    lvl_list = [];
    for i=1:history.Niters -1
        x = history.data(i).x(1);
        y = history.data(i).x(2);
        u = history.data(i+1).x(1);
        v = history.data(i+1).x(2);
        plot([x,u],[y,v],'color',color,'marker',marker);
        lvl_list = [lvl_list, history.data(i).f_k];
    end
    lvl_list = [lvl_list, history.data(i+1).f_k];
    text(history.data(1).x(1),history.data(1).x(2),disp_text, 'color',color, ...
        'FontSize',15);
end

function L = make_lvl_set(list1, list2)
    Lmin = min(min(list1), min(list2));
    Lmax = max(max(list1), max(list2));
    temp = linspace(Lmin,Lmax,10);
    L = [list1, list2, temp];
end
```

```
function print_table(history)
    rowvec_fmt = ['[', repmat('%0.4f, ', 1, numel(history.data(1).x)-1), '%0.4f]'];
    fprintf('Iter(k)\t x_k\t f_k\t g_k\t alpha_k\n');

    for k=1:history.Niters
        fprintf('%d\t',k);
        fprintf(rowvec_fmt,history.data(k).x); fprintf('\t');
        fprintf('%0.4f\t',history.data(k).f_k);
        fprintf(rowvec_fmt,history.data(k).g_k); fprintf('\t');
        fprintf('%0.4f\t',history.data(k).alpha_k);fprintf('\n');
    end
end
```

Listing 2: Steepest Descent

```
function [x_k, history_out] = steepest_descent(x_k,fun,grad,alpha,...
                                    verbose, TolFun, TolX, TolGrad)
    switch nargin
        case 3
            alpha_fixed = 0;
            verbose = 0;
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 4
            alpha_fixed = 1;
            verbose = 0;
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 5
            alpha_fixed = 1;
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 6
            alpha_fixed = 1;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 7
            alpha_fixed = 1;
            TolGrad = 1e-4;
    end

    history.parameter.TolFun = TolFun;
    history.parameter.TolX = TolX;
    history.parameter.TolGrad = TolGrad;

    done = 0;
    if alpha_fixed
        history.name = 'Grad Descent (fixed alpha)';
    else
        history.name = 'Steepest Descent';
    end
    count=1;
    history.data(count).x = x_k;
    f_k = fun(x_k);
    history.data(count).f_k = f_k;
```

```
    g_k = grad(x_k);
    history.data(count).g_k = g_k;
    while ¬done
        d_k = -g_k;
        if ¬alpha_fixed
            [xa,xb, initial_interval_summ]= get_search_interval(x_k, fun, d_k);
            history.data(count).initial_search_interval_summary = ...
                initial_interval_summ;
            [xa,¬,line_search_summary] = fibonacci_method(xa, xb, fun);
            history.data(count).line_search_summary = line_search_summary;
            alpha_k = norm((xa-x_k),2)/norm(d_k,2);
        end
        history.data(count).alpha_k = alpha_k;
        Δ_x_k = alpha_k*d_k;
        x_k = x_k + Δ_x_k;
        g_k = grad(x_k);
        f_k_plus_1 = fun(x_k);

        count = count+1;
        history.data(count).x = x_k;
        history.data(count).f_k = f_k_plus_1;
        history.data(count).g_k = g_k;

        if abs(f_k_plus_1 -f_k) < TolFun || norm(alpha_k*d_k,2) <TolX || ...
            norm(g_k)< TolGrad
            done = 1;
        end
    end
    history.Niters = count -1;

    if nargout>1
        history_out = history;
    end
    f_k = f_k_plus_1;
end
```

Listing 3: Conjugate Gradient (Powell)

```
function [x_k, history_out] = CG(x_k,fun,grad,...
                                verbose, TolFun, TolX, TolGrad)
    history.name = 'Conjugate Gradient';
    switch nargin
        case 3
            verbose = 0;
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 4
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 5
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 6
            TolGrad = 1e-4;
    end
    history.parameter.TolFun = TolFun;
```

```
    history.parameter.TolX = TolX;
    history.parameter.TolGrad = TolGrad;

    done = 0;
    count=1;
    f_k = fun(x_k);
    g_k = grad(x_k);
    history.data(count).x = x_k;
    history.data(count).f_k = f_k;
    history.data(count).g_k = g_k;
    d_k = -g_k;
    while ¬done
        % get the step length
        % initial search interval
        [xa,xb, initial_interval_summ]= get_search_interval(x_k, fun, d_k);
        history.data(count).initial_search_interval_summary = ...
            initial_interval_summ;
        % 1-d line search using fibonacci method
        [xa,¬,line_search_summary] = fibonacci_method(xa, xb, fun);
        history.data(count).line_search_summary = line_search_summary;
        alpha_k = norm((xa-x_k),2)/norm(d_k,2);
        history.data(count).alpha_k = alpha_k;

        Δ_x_k = alpha_k*d_k;
        x_k = x_k + Δ_x_k;
        Δ_g_k = grad(x_k) -g_k;
        old_g_k_norm  = g_k'*g_k;
        g_k = g_k + Δ_g_k;

        if mod(count,length(x_k)+1) ≠ 0
            beta_k = max(0, (g_k'*(Δ_g_k))/old_g_k_norm);
            d_k = -g_k + beta_k*d_k;
        else
            % reset to neg grad
            d_k = -g_k;
        end

        count = count+1;
        f_k_plus_1 = fun(x_k);
        history.data(count).x = x_k;
        history.data(count).f_k = f_k_plus_1;
        history.data(count).g_k = g_k;
        history.data(count).alpha_k = alpha_k;

        % check if done
        if abs(f_k_plus_1 - f_k) < TolFun || norm(Δ_x_k ,2) <TolX || ...
            norm(g_k,2)< TolGrad || count>100
            done = 1;
        end
        f_k = f_k_plus_1;
    end
    history.Niters = count -1;
    if nargout>1
        history_out = history;
    end
end
```

Listing 4: SRS

```
function [x_k, history_out] = SRS(x_k,fun,grad,...
                                verbose, TolFun, TolX, TolGrad)
    history.name = 'Quasi-Newton: SRS (Rank-1 Correction)';
    switch nargin
        case 3
            verbose = 0;
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 4
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 5
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 6
            TolGrad = 1e-4;
    end
    history.parameter.TolFun = TolFun;
    history.parameter.TolX = TolX;
    history.parameter.TolGrad = TolGrad;

    done = 0;
    H_k= eye(length(x_k));
    count=1;
    f_k = fun(x_k);
    g_k = grad(x_k);
    history.data(count).x = x_k;
    history.data(count).f_k = f_k;
    history.data(count).g_k = g_k;
    while ¬done
        % find search direction
        d_k = -H_k*g_k;

        % get the step length
        % initial search interval
        [xa,xb, initial_interval_summ]= get_search_interval(x_k, fun, d_k);
        history.data(count).initial_search_interval_summary = ...
            initial_interval_summ;
        % 1-d line search using fibonacci method
        [xa,¬,line_search_summary] = fibonacci_method(xa, xb, fun);
        history.data(count).line_search_summary = line_search_summary;
        alpha_k = norm((xa-x_k),2)/norm(d_k,2);
        history.data(count).alpha_k = alpha_k;
        % update H_k
        Δ_x_k = alpha_k*d_k;
        x_k = x_k + Δ_x_k;
        Δ_g_k = grad(x_k) - g_k;
        g_k = g_k + Δ_g_k;

        Δ_H_k = (Δ_x_k - H_k*Δ_g_k);
        Δ_H_k = (Δ_H_k*Δ_H_k')/(Δ_g_k'*Δ_H_k);
        H_k = H_k + Δ_H_k;

        count = count+1;
        f_k_plus_1 = fun(x_k);
        history.data(count).x = x_k;
```

```
        history.data(count).f_k = f_k_plus_1;
        history.data(count).g_k = g_k;
        % check if converged to solution
        if abs(f_k_plus_1 - f_k) < TolFun || norm(Δ_x_k ,2) <TolX || ...
            norm(g_k,2)< TolGrad || count>100
            done = 1;
        end
        f_k = f_k_plus_1;
    end
    history.Niters = count -1;

    if nargout>1
        history_out = history;
    end
end
```

Listing 5: DFP

```
function [x_k, history_out] = DFP(x_k,fun,grad,...
                                   verbose, TolFun, TolX, TolGrad)
    history.name = 'Quasi-Newton: DFP';
    switch nargin
        case 3
            verbose = 0;
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 4
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 5
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 6
            TolGrad = 1e-4;
    end
    history.parameter.TolFun = TolFun;
    history.parameter.TolX = TolX;
    history.parameter.TolGrad = TolGrad;

    done = 0;
    H_k= eye(length(x_k));
    count=1;
    f_k = fun(x_k);
    g_k = grad(x_k);
    history.data(count).x = x_k;
    history.data(count).f_k = f_k;
    history.data(count).g_k = g_k;
    while ¬done
        % find search direction
        d_k = -H_k*g_k;

        % get the step length
        % initial search interval
        [xa,xb, initial_interval_summ]= get_search_interval(x_k, fun, d_k);
        history.data(count).initial_search_interval_summary = ...
            initial_interval_summ;
```

```
        % 1-d line search using fibonacci method
        [xa,¬,line_search_summary] = fibonacci_method(xa, xb, fun);
        history.data(count).line_search_summary = line_search_summary;
        alpha_k = norm((xa-x_k),2)/norm(d_k,2);
        history.data(count).alpha_k = alpha_k;
        % update H_k
        Δ_x_k = alpha_k*d_k;
        x_k = x_k + Δ_x_k;
        Δ_g_k = grad(x_k) - g_k;
        g_k = g_k + Δ_g_k;

        a_k = (Δ_g_k'*Δ_x_k);
        Δ_H_k1 = (Δ_x_k*Δ_x_k')/a_k;
        Δ_H_k2 = ((H_k*Δ_g_k)*(H_k*Δ_g_k)')/(Δ_g_k'*H_k*g_k);
        H_k = H_k + Δ_H_k1 - Δ_H_k2;

        count = count+1;
        f_k_plus_1 = fun(x_k);
        history.data(count).x = x_k;
        history.data(count).f_k = f_k_plus_1;
        history.data(count).g_k = g_k;

        % check if converged to solution
        if abs(f_k_plus_1- f_k) < TolFun || norm(Δ_x_k ,2) <TolX || ...
            norm(g_k,2)< TolGrad || count>100
            done = 1;
        end
        f_k = f_k_plus_1;
    end
    history.Niters = count -1;

    if nargout>1
        history_out = history;
    end
end
```

Listing 6: BFGS

```
function [x_k, history_out] = BFGS(x_k,fun,grad,...
                                   verbose, TolFun, TolX, TolGrad)
    history.name = 'Quasi-Newton: BFGS';
    switch nargin
        case 3
            verbose = 0;
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 4
            TolFun = 1e-4;
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 5
            TolX = 1e-4;
            TolGrad = 1e-4;
        case 6
            TolGrad = 1e-4;
    end
    history.parameter.TolFun = TolFun;
```

```
    history.parameter.TolX = TolX;
    history.parameter.TolGrad = TolGrad;

    done = 0;
    H_k= eye(length(x_k));
    count=1;
    f_k = fun(x_k);
    g_k = grad(x_k);
    history.data(count).x = x_k;
    history.data(count).f_k = f_k;
    history.data(count).g_k = g_k;
    while ¬done
        % find search direction
        d_k = -H_k*g_k;

        % get the step length
        % initial search interval
        [xa,xb, initial_interval_summ]= get_search_interval(x_k, fun, d_k);
        history.data(count).initial_search_interval_summary = ...
            initial_interval_summ;
        % 1-d line search using fibonacci method
        [xa,¬,line_search_summary] = fibonacci_method(xa, xb, fun);
        history.data(count).line_search_summary = line_search_summary;
        alpha_k = norm((xa-x_k),2)/norm(d_k,2);
        history.data(count).alpha_k = alpha_k;
        % update H_k
        Δ_x_k = alpha_k*d_k;
        x_k = x_k + Δ_x_k;
        Δ_g_k = grad(x_k) - g_k;
        g_k = g_k + Δ_g_k;

        a_k = Δ_g_k'*Δ_x_k;
        beta_k = (Δ_g_k'*H_k*g_k)/a_k;
        Δ_H_k1 = (1 + beta_k)*(Δ_x_k*Δ_x_k')/a_k;
        Δ_H_k2 = ((H_k*Δ_g_k*Δ_x_k') + (H_k*Δ_g_k*Δ_x_k')')/a_k;
        H_k = H_k + Δ_H_k1 - Δ_H_k2;

        count = count+1;
        f_k_plus_1 = fun(x_k);
        history.data(count).x = x_k;
        history.data(count).f_k = f_k_plus_1;
        history.data(count).g_k = g_k;
        % check if converged to solution
        if abs(f_k_plus_1 - f_k) < TolFun || norm(Δ_x_k ,2) <TolX || norm(g_k, ...
            2)< TolGrad || count>100
            done = 1;
        end
        f_k = f_k_plus_1;
    end
    history.Niters = count -1;

    if nargout>1
        history_out = history;
    end
end
```

Listing 7: Fibonacci Search

```
function [xa, xb, history_out] = fibonacci_method(xa, xb, fun, TolX, verbose)
    history.name = 'Fibonacci Method';
    Δ = 0.1; % for last value of rho
 switch nargin
  case 3
   TolX= 1e-6;
   verbose = 0;
  case 4
   verbose = 0;
    end
    history.params.Δ = Δ;
    history.params.TolX = TolX;
    % estimate Niter
    Niter = get_Niter_fibb(xa,xb,Δ, TolX);
    history.Niters = Niter;

    % do range reduction
    k = 1;
    rho_k = 1 - (fibonacci(Niter+1)/fibonacci(Niter+1+1));
    s = xa + rho_k*(xb-xa);
 t = xa + (1-rho_k)*(xb-xa);
 f1 = fun(s);
 f2 = fun(t);
    f_xa = fun(xa);
    f_xb = fun(xb);
    %
    if verbose
        fprintf('Iter(k)\t rho_k\t ak\t bk\t f(ak)\t f(bk)\t New uncertainity ...
            interval (a,b)\t Uncertainity width\n');
        rowvec_fmt = ['[', repmat('%0.4f, ', 1, numel(xa)-1), '%0.4f]'];
    end

    for k=1:Niter
        xa_prev = xa;
        xb_prev = xb;
        f_xa_prev = f_xa;
        f_xb_prev = f_xb;
        rho_k = 1 -(fibonacci(Niter-(k-1)+1)/fibonacci(Niter+1 -(k-1)+1));
        if k == Niter
           rho_k = rho_k - Δ;
        end
        if f1 < f2
   xb=t;
            f_xb = f2;
   t=s;
   s= xa + rho_k*(xb-xa);
            f2 = f1;
   f1 = fun(s);
  else
   xa = s;
            f_xa = f1;
   s = t;
   t = xa + (1-rho_k)*(xb-xa);
   f1 = f2;
   f2 = fun(t);
        end
        %
        if verbose
```

```
            fprintf('%d\t',k);
            fprintf('%0.4f\t',rho_k);
            fprintf(rowvec_fmt,xa_prev); fprintf('\t');
            fprintf(rowvec_fmt,xb_prev); fprintf('\t');
            fprintf('%0.4f\t %0.4f\t',f_xa_prev,f_xb_prev);
            fprintf('(');
            fprintf(rowvec_fmt,xa);
            fprintf(' ,sqrt(sum( ')
            fprintf(rowvec_fmt,xb);
            fprintf(')\t%0.4f\n',norm((xb-xa),2));
        end
        %
        history.data(k).x_prev = [xa_prev,xb_prev];
        history.data(k).x_new = [xa,xb];
    end
    if verbose
        fprintf("** Fibonacci method took %d iters **\n** Final uncertainity ...
            region width was %0.4f **\n",...
            Niter,norm((xb-xa),2));
    end

    if nargout>2
       history_out = history;
    end

end

function N = get_Niter_fibb(xa,xb,Δ,TolX)
    N = 1;
    d = norm((xb-xa),2);
    while d*(1+2*Δ)/TolX > fibonacci(N+1+1)
        N = N +1;
    end

end
```

Listing 8: Initial Search Interval

```
function [x_prev, x_nxt, history] = get_search_interval(x0, fun, direction,...
                                                        step_size)
% function for identifying the interval which contains a minimizer
% using x0 as initial guess and evaluating the fun till the function value
% increases btw 2 consecutive evals
% Input: fun: function handle for obj fun
%         step_size: initial step size, user param
%         direction: function handle for search direction
    switch nargin
        case 3
            step_size = 0.1;
    end
    x_prev = x0;
    x_cur = x_prev + step_size*direction;
    step_size = 2*step_size;
    x_nxt = x_cur + step_size*direction;

    f1 = fun(x_prev);
    f2 = fun(x_cur);
    f3 = fun(x_nxt);
```

```
    f = [f1,f2,f3];

    count = 1;
    history.data(count).interval = [x_prev, x_nxt];
    while f(2) ≥ f(3)
        step_size = 2*step_size;
        x_prev = x_cur;
        x_cur = x_nxt;
        x_nxt = x_cur + step_size*direction;
        f = [f(2),f(3), fun(x_nxt)];
        count = count +1;
        history.data(count).interval = [x_prev, x_nxt];
    end
    if nargout>2
        history.name = 'initial search interval';
        history.Niters = count;
        history.parameter.step_size = step_size;
    end
end
```

Output on MATLAB command window

Listing 9: MATLAB Output

```
--------------------SD------------------
Iter(k)  x_k  f_k  g_k  alpha_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

x_str1_sd =

    9.9496
    9.9496

Iter(k)  x_k  f_k  g_k  alpha_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

x_str2_sd =

   -9.9496
   -9.9496

--------------------CG-------------------
Iter(k)  x_k  f_k  g_k  alpha_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

x_str1_cg =

    9.9496
    9.9496

Iter(k)  x_k  f_k  g_k  alpha_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

x_str2_cg =

   -9.9496
   -9.9496

--------------------SRS-------------------
Iter(k)  x_k  f_k  g_k  alpha_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

x_str1_srs =

    9.9496
    9.9496
```

```

Iter(k)  x_k  f_k  g_k  alpha_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

x_str2_srs =

   -9.9496
   -9.9496

--------------------DFP-------------------
Iter(k)  x_k  f_k  g_k  alpha_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

x_str1_dfp =

    9.7539
   10.2911

Iter(k)  x_k  f_k  g_k  alpha_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

x_str2_dfp =

   -9.6237
  -10.2575

--------------------BFGS-------------------
Iter(k)  x_k  f_k  g_k  alpha_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

x_str1_bfgs =

   10.0122
   10.0063

Iter(k)  x_k  f_k  g_k  alpha_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

x_str2_bfgs =

   -9.9970
  -10.0261
```