Non-Linear Programming

Project Report Spring-2018

Name: Ravi Kumar Choudhary
Roll No: 13MA20033

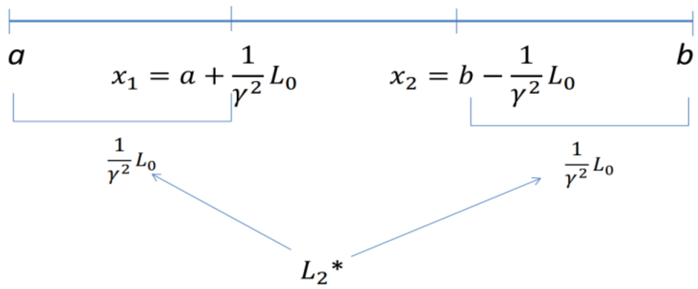
Problem 1: Unconstrained Optimization Ackley function

$$f(x, y) = -20\exp[-0.2\sqrt{0.5(3x^2 + y^2)}] - \exp[0.5(cos2\pi x + cos2\pi y)] + e + 20$$
$$Global\ minimum: f(0,0) = 0, Search\ Domain: -5 \le x, y \le 5$$

Methodology: Golden Section Method

Golden method is a one-dimensional minimization method follow the same methodology as Fibonacci method.

Step 1: Given the initial interval of uncertainty $L_0 = [a,b]$ and the number of experiments *n*.*Step 2:* To generate the first two experimental points x_1 and x_2 let us define L_2^* as:



Step 3: Discard the part of the $L_0 = [a,b]$ using unimodality assumption. Either $[a, x_2)$ or $(x_1, b]$ will be the new interval of uncertainty and thus the length of new interval of uncertainty will be

$$L_2 = L_0 - L_2^* = \frac{1}{\gamma}L_0$$

Step 4: Next experimental point x_3 is then generated in such a way that the current two experiments are located at L_3^* distance from both the ends of L_2, where $L_3^* = \frac{1}{\gamma^3}L_0$. Thus the length of new interval of uncertainty is L_3, where $L_3 = L_2 - L_3^* = \frac{1}{\gamma^2}L_0$

Next step: Repeat the above process until the new experiment x_n is being obtained, where *n* is the total number of experiments or the approximations for obtaining approximate optimal point. The whole process may be generalized with the following rules:

For obtaining j^{th} experiment generate $L_j^* = \frac{1}{\gamma^j}L_0$. Then the length of new interval of uncertainty after j^{th} experiment is $L_j = \frac{1}{\gamma^{j-1}}L_0$

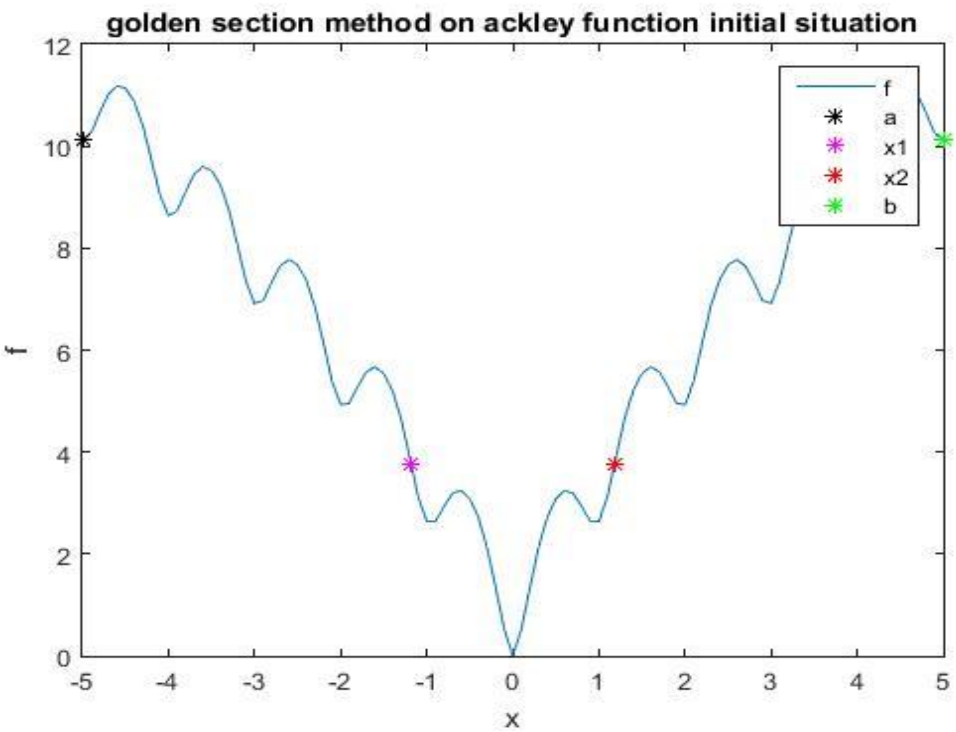
<u>Assumptions</u>: Since the function is two dimensional and Golden section in one-dimensional method. I have set y value to zero and then applied the same method.

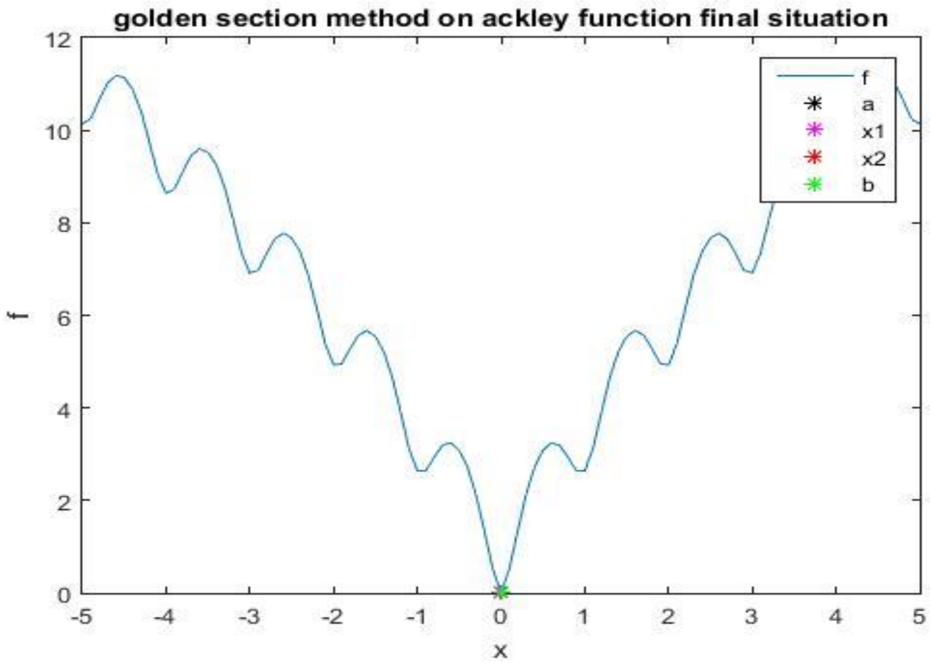
<u>Matlab Code:</u>

```
clc
clear
close all
%% declaring ackely function
ackley = @(X,Y)-20 * exp(-0.2 * sqrt(0.5*(X.^2 + Y.^2))) -
exp(0.5*(cos(2*pi*X) + cos(2*pi*Y))) + exp(1) + 20;

%% golden section implementation
% defining and declaring variables
a = -5.0;
b = 5.0;
x = a:0.1:b;
gammaInv = 0.618;
L0 = b - a;
L2_star = gammaInv.^2 * L0;
x1 = a + L2_star;
x2 = b - L2_star;
n = 10;

% ploting the function value with initial points
figure(1)
plot(x,ackley(x,0));
title('golden section method on ackley function initial situation');
xlabel('x');
ylabel('f');
hold on
plot(a,ackley(a,0),'*k');
plot(x1,ackley(x1,0),'*m');
plot(x2,ackley(x2,0),'*r');
plot(b,ackley(b,0),'*g');
legend('f','a','x1','x2','b');
hold off
figure(2)
plot(x,ackley(x,0));
title('golden section method on ackley function final situation');
xlabel('x');
ylabel('f');
hold on

for j = 1:n
   f1 = ackley(x1,0);
   f2 = ackley(x2,0);
   % checking unimodality condition and
   % assigning new a,x1,x2 and b accordingly
   if f2 > f1
       x3 = a + (x2 - x1);
       b = x2;
       x2 = x1;
       x1 = x3;
   elseif f1 > f2
       x3 = x1 + (b - x2);
       a = x1;
       x1 = x2;
       x2 = x3;
   else
```

```
        a = x1;
        b = x2;
        L0 = b - a;
        L2_star = 0.382 * L0;
        x1 = a + L2_star;
        x2 = b - L2_star;
    end
end

% plot the final point along with their function value
plot(a,ackley(a,0),'*k');
plot(x1,ackley(x1,0),'*m');
plot(x2,ackley(x2,0),'*r');
plot(b,ackley(b,0),'*g');
legend('f','a','x1','x2','b');
hold off

% display the final interval
Ln = [a b];
display(Ln);
```

<u>Generated Output:</u>



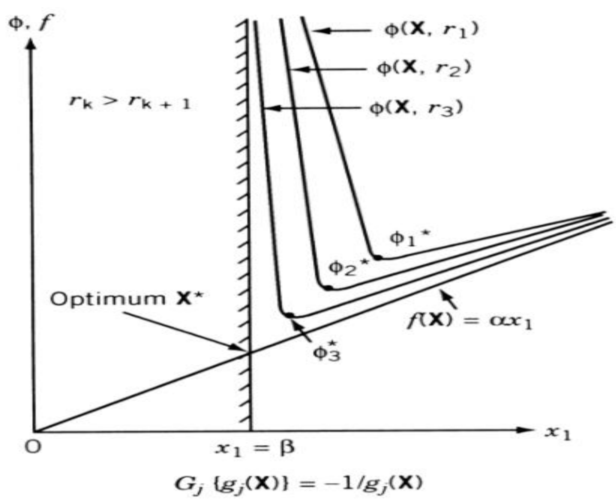


Problem 2: Constrained Optimization Three Hump Camel function

$$f(\mathrm{x},\mathrm{y}) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2$$
$$Global\ minimum: f(0,0) = 0, Search\ Domain: -5 \leq x, y \leq 5$$

Methodology: Interior Penalty Method

- In the interior penalty function method, a new function (ϕ function) is constructed by augmenting a penalty term to the objective function.
- The penalty term is chosen such that its value will be small at points away from the constraint boundaries and will tend to infinity as the constraint boundaries are approached. Hence the value of the ϕ function also blows up as the constraint boundaries are approached. This behaviour can also be seen from the figure.



- Thus once the unconstrained minimization of ϕ **(X**, *rk***)** is started from any feasible point **X1**, the subsequent points generated will always lie within the feasible domain since the constraint boundaries act as barriers during the minimization process. This is why the interior penalty function methods are also known as *barrier methods*.
- The ϕ function defined originally by Carroll is

$$\varphi(X, r_k) = f(\mathrm{X}) - r_k \sum_{j=1}^{m} \left(\frac{1}{g_j(X)}\right)$$

The iteration procedure of this method can be summarized as follows:

1. Start with an initial feasible point X1 satisfying all the constraints with strict inequality sign, that is, $g_j(X_1) < 0$ for j=1,2,...,m, and an initial value of r_1 Set k =1.
2. Minimize φ (X,r_k) by using any of the unconstrained minimization methods and obtain the solution X_k^*.
3. Test whether X_k^* is the optimum solution of the original problem. If X_k^* is found to be optimum, terminate the process. Otherwise, go to the next step
4. Find the value of the next penalty parameter r_{k+1}, as $r_{k+1} = c\, r_k$ where c < 1.
5. Set the new value of k=k+1, take the new starting point as $X_1 = X_k^*$, and go to step 2.

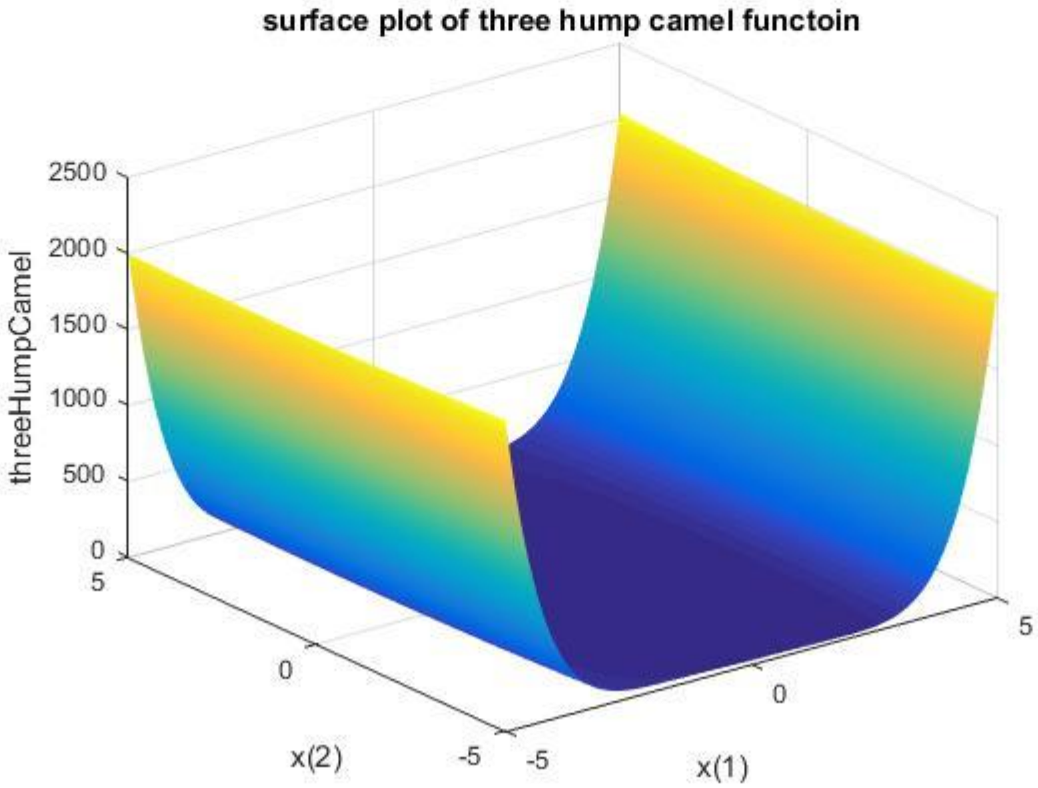
Matlab Code:

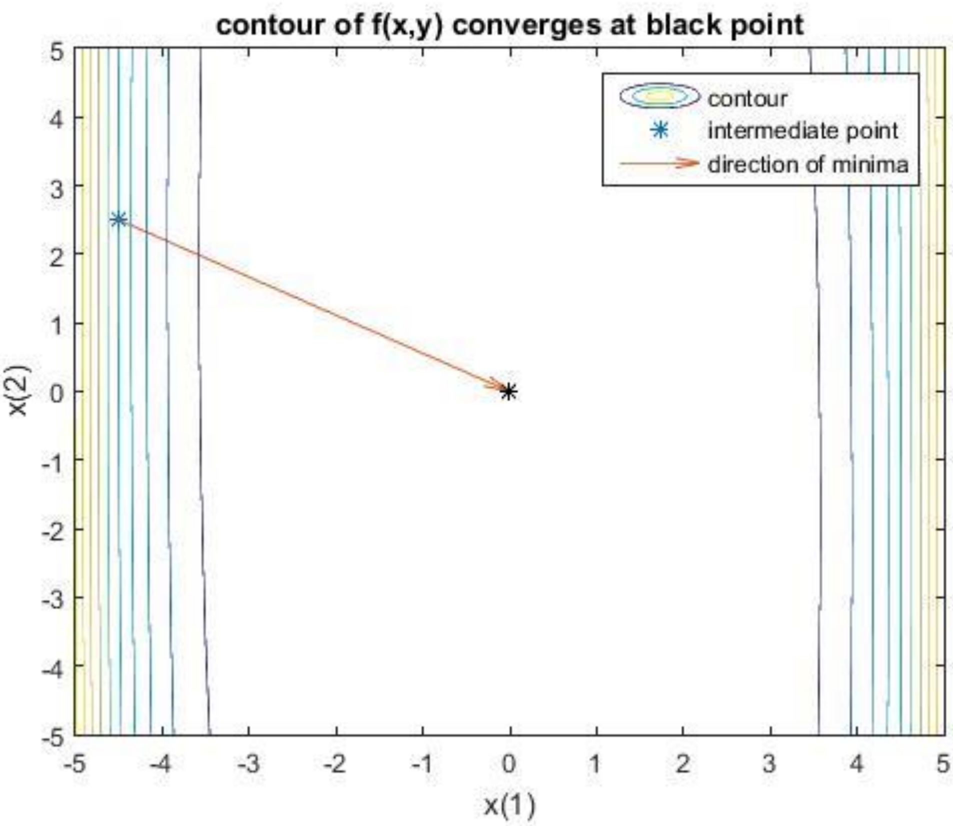
```
clc
clear
close all

%% defining three hump camel function
three_hump_camel = @(X,Y) 2*X.^2 - 1.05*X.^4 + X.^6/6 + X.*Y + Y.^2;

%% surface ploting three hump camel function
[X,Y] = meshgrid(-5:.1:5, -5:.1:5);
Z = three_hump_camel(X,Y);

% display the surface plot of the function
figure(1)
surf(X,Y,Z)
title('surface plot of three hump camel functoin');
xlabel('x(1)');
ylabel('x(2)');
zlabel('threeHumpCamel');
shading interp

%% algorithm implementation
phi_ = @(x,r) 2*x(1)^2 - 1.05*x(1)^4 + (x(1)^6)/6 + x(1)*x(2) + x(2)^2 - r
* (1/(-x(1)-5) + 1/(-x(2)-5) + 1/(x(1)-5) + 1/(x(2)-5));

% defining variables
r = 1000.0;
c = 0.01;
epsilon = 0.005;
options = optimoptions(@fminunc,'Algorithm','quasi-newton');
```

```
% display contour plot and converging points
figure(2)
contour(X,Y,Z);
title('contour plot of three hump camel func with convergence point');
xlabel('x(1)');
ylabel('x(2)');
shading interp
hold on

% declaring initial point x1
x1 = [-4.5, 2.5];
f1 = three_hump_camel(x1(1), x1(2));
scatter(x1(1),x1(2),'*');
iteration = 0;

% while function not converge keep iterating
while true
    iteration = iteration + 1;
    phi = @(x)phi_(x,r);
    [x2,opt_phi] = fminunc(phi, x1, options);
    f2 = three_hump_camel(x2(1), x2(2));
    dx = x2 - x1;
    quiver(x1(1),x1(2),dx(1),dx(2),0);

    % convergence check
    if abs((f2 - f1)/f2) <= epsilon
        min_f = f2;
        x_star = x2;
        display('function Converged!');
        fprintf('Number of Iteration taken to converge = %i\n', iteration);
        fprintf('Minimum value of function = %f\n', min_f);
        display('Optimal point is');
        display(x_star);
        scatter(x_star(1),x_star(2),'k*');
        break
    end
    scatter(x2(1),x2(2),'*');
    r = c * r;
    x1 = x2;
    f1 = f2;
end

legend('contour','intermediate point','direction of minima');
hold off
```

<u>Generated Output:</u>

```
Command Window

function Converged!
Number of Iteration taken to converge = 3
Minimum value of function = 0.000000
Optimal point is

x_star =

   1.0e-07 *

   -0.5559    0.4801

>>
```


surface plot of three hump camel functoin
threeHumpCamel
2500
2000
1500
1000
500
0
5
0
-5
x(2)
-5
0
5
x(1)


contour of f(x,y) converges at black point
contour
intermediate point
direction of minima
x(2)
x(1)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-5
-4
-3
-2
-1
0
1
2
3
4
5