Optimization Technique

BACHA Sushil

December 4,2018

$$\mathbf{f(x)} = x - 1 - ln(x) \quad (1)$$

Initially for the given Tolerance (0.001) and epsilon (0.0001) the number of iterations occurred for convergence is 13 and the precision obtained is 1.0004.

The varied tolerance range is from $10^-(7)$ *to* $10^-(4)$ and the number of iterations decreases as tolerance value increases as shown below in the figure.

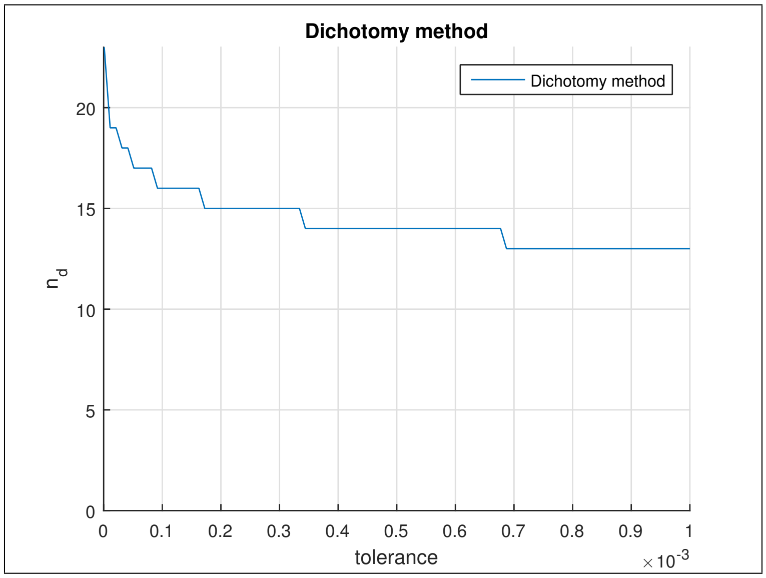
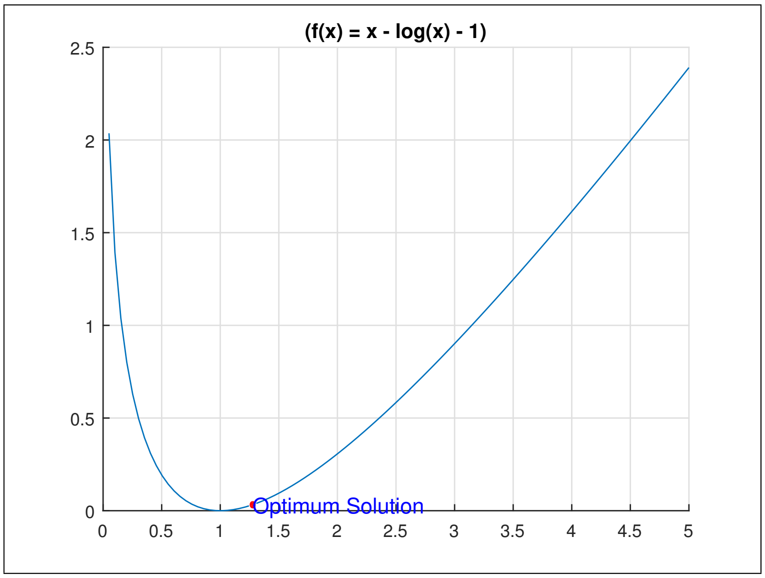


Figure 1: Tolerance vs iterations

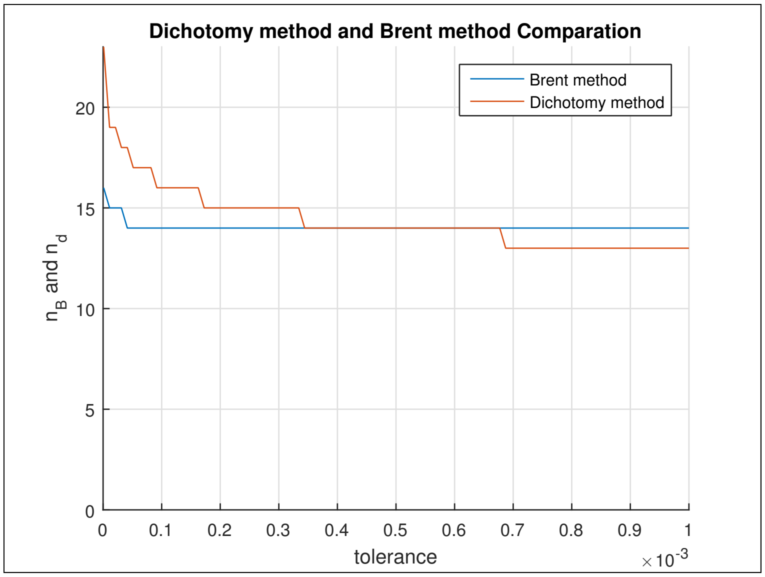
when we increase the tolerance, the soon the convergence is occurred decreasing the number of iterations but, it decreases the precision. For the smaller tolerance the minimal obtained is 1.0000 and for the larger tolerance the precision obtained is 1.0004.

By choosing the larger epsilon (0.2), we exclude the larger region of search and we might also reject the minima and precision is not obtained. For example, choosing $e = 0.2$ and by varying tolerance. For smaller and larger tolerance, the minima obtained is 1.0938 and the iterations occurred are 5.

By choosing very smaller epsilon value,the number of iterations may increase or decrease,but it might exceed the maximum number of iterations and terminates with wrong precision values.For example,by choosing epsilon as tolerance(i)/10^9.



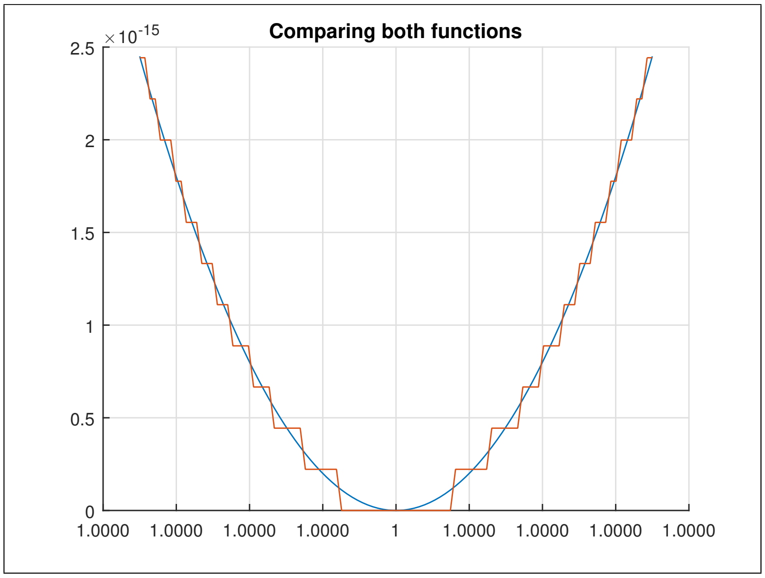
Now we compare both the methods by varying tolerance and plot tolerance vs iterations.



For the function

$$\mathbf{f(x)} = x - ln(x) - 1 \tag{2}$$

, which is same as the first function, but as matlab reads the function from left to right, it appears to be a different function.The obtained results are same with same precision.



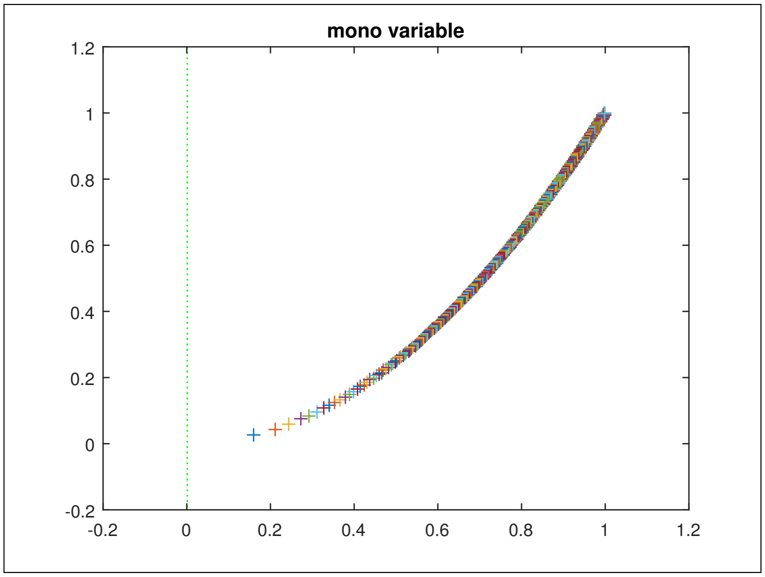
1 Rosenbrook function Minimization :

For the given function (Rosenbrook function) defined as

$$f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \tag{3}$$

1.1 Monovariable Method

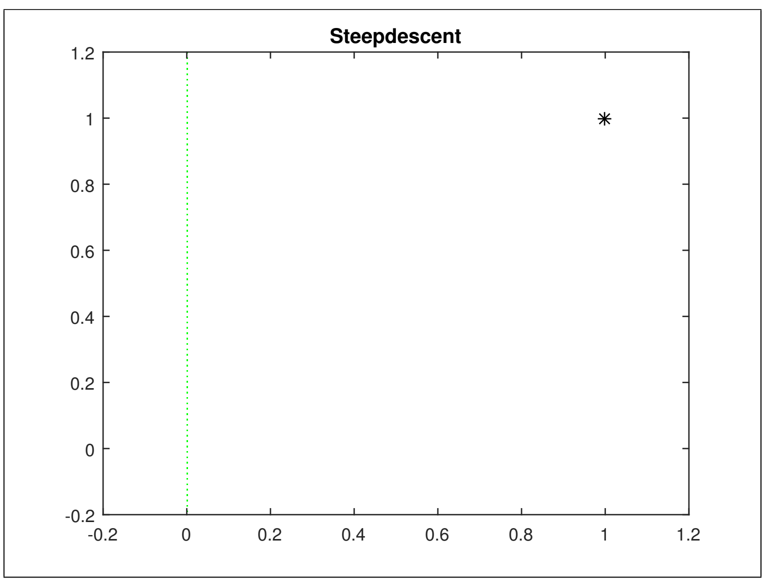
For a multivariable problem, each variable searches in all the directions, while the other variable is kept constant, similarly the second variable finds minimum in all directions, while first variable is kept constant. Successive iterations are carried out to find the minimum.



1.2 Steepest descent method

Steepest descent method uses the concept of Gradient, that is the direction towards the maximum function value, so the negative of the gradient is chosen for the minimum function value and with appropriate step length from the inital point. The step length is chosen as $\lambda_i = S_i^T{*}S_i/S_i^T{*}\mathrm{A}{*}S_i$

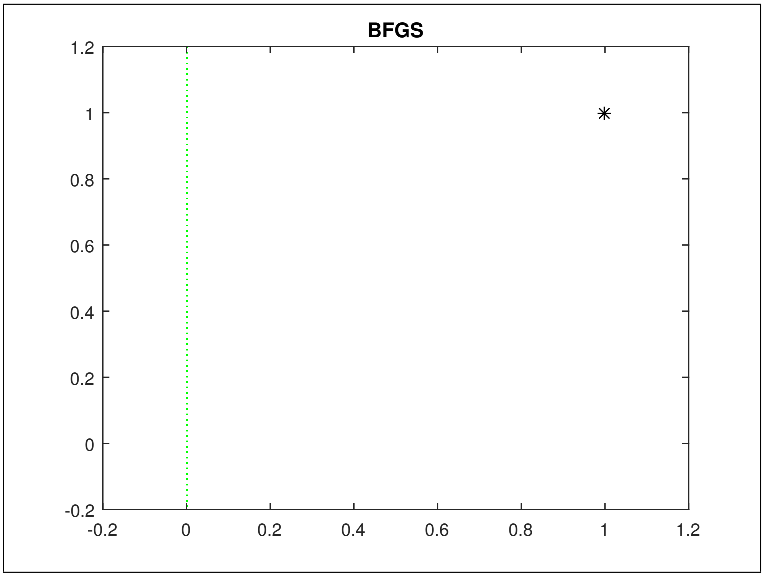
In above example we are getting the optimal point in only 1 iteration.



1.3 Quasi-Newton BFGS method

The main difference in this method with respect to other method is, in this we keep updating the Hessian matrix with B matrix,in the absence of additional information, B is chosen as identity matrix.The direction is given as $S_i = -[B_i]\nabla f_i$

In the above example the optimal step length and direction is obtained in only one iteration.



2 Estimation :

A exponential noisy series is given as

$$y_k = a_0 b_0^{(k-1)} + w_k \tag{4}$$

where w_k is a series of white Gaussian noise of variance 1

we want to estimate a_0 and b_0 by minimizing the least square criteria which is given as

$$J(a,b) = \sum_{k=1}^{40} (y_k - ab^{(k-1)})^2 \tag{5}$$

for the stimulation we take $a_0 = 10$, $b_0 = 0.9$.

2.1 Monovariable :

Rewriting the minimization criteria of J w.r.t to one of the variable.

$$\frac{\partial J}{\partial a} = 0 \tag{6}$$

if we implement this on equation 4 we get

$$J(b) = \sum_{k=1}^{40} (y_k - \frac{\sum_{i=1}^{40} b^{i-1} y_i}{\sum_{i=1}^{40} (b^{i-1})^2} (b^{(k-1)})^2 \tag{7}$$

The above equation is the the new criteria which should be used in order to minimize the given function.

2.2 Comparing :(Mono and BFGS)

By applying the above criteria's we got the following solution's :

As we use random numbers we can run the program n number of times and study/compare the methods and study the behaviour.

For the first time :

Mono-variable :

a = 10.4697
b = 0.9035
i = 14
duration = 0.9531

BFGS :

a = 10.4698
b = 0.9035
i = 22
duration = 0.8750

For the second time :

Mono-variable :

a = 10.0150
b = 0.8999
i = 14
duration = 1

BFGS :

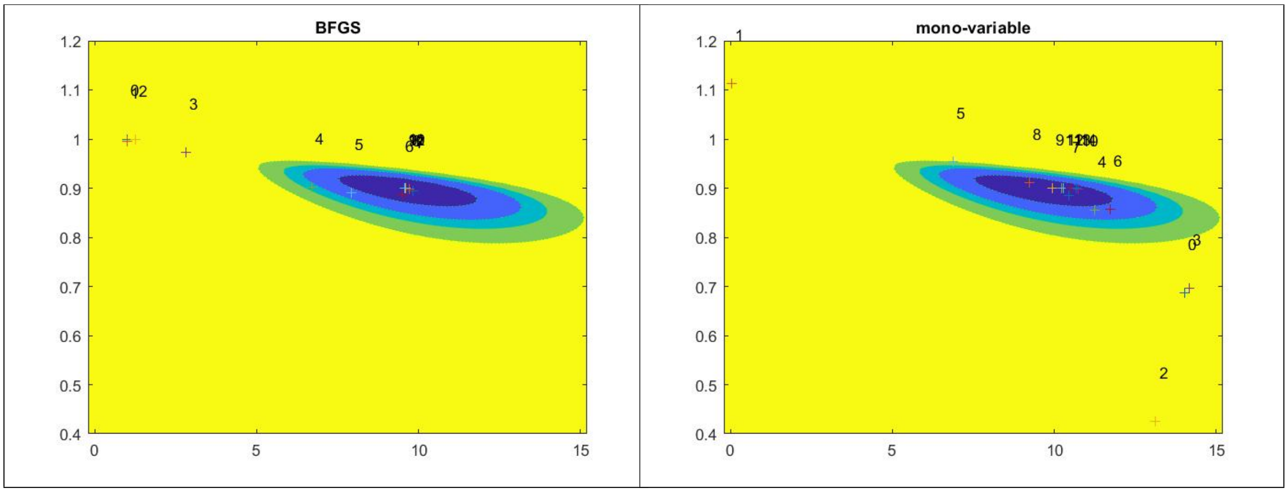
a = 10.0151
b = 0.8999
i = 12
duration = 0.8594

From the above results we can see that the time taken to calculate the minimum is best in **BFGS** i.e the result is obtained much **faster** when compared to the Mono-variable **even when the number of iterations were more in BFGS.**

2.3 Level Curves :

The above obtained results can be plotted in the form of level curves i.e the evolution of the minimum in a plane (a,b) for both the methods.

The trajectory of the minimum in both the methods is as shown below :



(a) BFGS

(b) Mono-Variable

Figure 2: Trajectory

1 MATLAB Codes

1.1 Mono-variable minimization : comparison of dichotomy and Brent's method

Listing 1: Code for setting options/Brent's method/plotting

```

close all
clear all
clc

% Setting the limits for X
xmin =  0;
xmax =  5;
xmin2 =  0;
xmax2 =  5;

n_d = [];
n_d2 = [];
n_Brent = [];

% tolerance = linspace(2e-4,1e-6,100);
tolerance = linspace(1e-6,1e-3,100);


for i = 1: length (tolerance)
```

```

options = optimset('Display', 'iter', 'TolFun', tolerance(i),.....
                'TolX', tolerance(i)/11, 'MaxFunEvals', 200 , 'MaxIter', 100);

[X1(i), X2(i), n_d(i),n_d2(i)] = dicho(xmin, xmax, xmin2, xmax2, options);

    % Brent's method

    options.TolX = tolerance(i);

    [x,fval,exitflag,output] = fminbnd(@objfun, xmin, xmax, options);

    n_Brent = [n_Brent output.funcCount];

end

figure(1)
hold on
grid on
plot(tolerance, n_Brent, tolerance, n_d)
legend('Brent method','Dichotomy method')
title('Dichotomy method and Brent method Comparation')
xlabel('tolerance')
ylabel('n_B and n_d')
ylim([0 inf])



% Plotting
x_1 = linspace(xmin,xmax);
x_2 = linspace(xmin2,xmax2);


% Objective Function
for i = 1:length(x_1)
    y_1(i) = objfun(x_1(i));
end

for i = 1:length(x_2)
    y_2(i) = objfun2(x_2(i));
end

figure(2)
hold on
grid on
plot(x_1,y_1)                              % Function Plotting
viscircles([X1(1),objfun(X1(1))], 0.01);        % Solution Plotting
title('(f(x) = x - 1 - log(x))');
txt = {'Optimum Solution'};
text(X1(1),objfun(X1(1)),txt,'Color','r','FontSize',12)

```

```
figure(3)
hold on
grid on
plot(x_2,y_2)                               % Function Plotting
viscircles([X2(1),objfun2(X2(1))], 0.01);          % Solution Plotting
title('(f(x) = x - log(x) - 1)');
txt = {'Optimum Solution'};
text(X2(1),objfun2(X2(1)),txt,'Color','b','FontSize',12)

figure(4)
hold on
grid on
len=0.00000007;
x = linspace(1-len,1+len);
plot(x,objfun(x),x,objfun2(x))
title('Comparing both functions');
```

Listing 2: Code for Dichotomy

```


function [X1, X2, n_d, n_d2] = dicho(xmin, xmax, xmin2, xmax2, options)
    iter = 0;

     while (abs(xmax-xmin)>options.TolFun)

        L = xmax + xmin;             % Interval size

        % Selecting smaller interval
        x1 = (L - options.TolX)/2;
        x2 = (L + options.TolX)/2;

        % Checking the values of objfun on both limits
        f1 = objfun(x1);
        f2 = objfun(x2);

        % Updating the interval for next iteration
        if (f1<f2)
            xmax = x2;
        else
            xmin = x1;
        end
         % Updating the State
        iter = iter+1   ;
     end

       if(f1<f2)
     X1 = x2;
     else X1 = x1;
     end
     n_d = [iter];

```

```

    iter2 = 0;

    while (abs(xmax2-xmin2)>options.TolFun)

       L2 = xmax2 + xmin2;            % Interval size

       % Selecting smaller interval
       x21 = (L2 - options.TolX)/2;
       x22 = (L2 + options.TolX)/2;

       % Checking the values of objfun on both limits
       g1 = objfun2(x21);
       g2 = objfun2(x22);

       % Updating the interval for next iteration
       if (g1<g2)
           xmax2 = x22;
       else
           xmin2 = x21;
       end
        % Updating the State
       iter2 = iter2 +1  ;
    end

    if(g1<g2)
    X2 = x22;
    else X2 = x21;
    end

    n_d2 = [iter2];
end
```

Listing 3: Code for objective function 1

```
function f = objfun(x)

    f = x - 1 - log(x) ;

end
```

Listing 4: Code for objective function 2

```
    function g = objfun2(x)

    g = x - log(x) - 1 ;

end
```

1.2 Rosenbrook function minimization

Listing 5: Monovariable/Steepest descent method/The quasi newton BFGS method

```

close all
clear all
clc


tolerance = 1e-6;
% [a,b]x[c,d]
a = -5;
b = 5;
c = -5;
d = 5;


[X,Y] = meshgrid(-0.2:0.001:1.2, -0.2:0.001:1.2);
f = 100*((X^2-Y)^2) + (1-X)^2;
v = [128,64,32,16,8,4,2,0.5,0.1,0.01];

fun = @(x) 100*((x(1)^2-x(2))^2) + (1-x(1))^2;
x2 = 0;
x1 = 0;
figure
contour(X,Y,f,v,':g');
hold on

options.TolX = tolerance;
i1=0;
i2=0;
i3=0;
n_Mon = 0;%nb of iterations using monovariable method
n_s =0;%nb of iterations using steepest method
n_B=0;%nb of iterations using BFGS method
%method monovariable
x1_prev = a;
x2_prev = c;
t1=cputime;



while (abs(x1 - x1_prev) > tolerance) & (abs(x2 - x2_prev) > tolerance)
    x1_prev = x1;
    x2_prev = x2;


    [x1,fval,exitflag,output] = fminbnd(@(x1)fun([x1;x2]), a, b, options);
    [x2,fval,exitflag,output] = fminbnd(@(x2)fun([x1;x2]), c, d, options);

    hold on
    plot (x1,x2,'+')
end
```

```

hold off
title 'mono variable'

n_Mon =output.funcCount;
i1=output.iterations;
x1_fminbnd = x1
x2_fminbnd = x2
tmon=cputime-t1;

%Stepest descent method
x0 = [x1, x2];
t2=cputime;
figure
contour(X,Y,f,v,':g');
hold on
fprintf('result-Steepdscent:');
options.HessUpdate = 'steepdesc';
x_Stepest = fminunc(fun,x0,options)
plot(x_Stepest(1),x_Stepest(2),'k*')
[x_Stepest,fval,exitflag,output]=fminunc(fun,x0,options)
 n_s =output.funcCount;
i2=output.iterations;
tsteep=cputime-t2;
hold off
title 'Steepdescent'

%Quasi - Newton BFGS method
x0 = [x1, x2];
t3=cputime;
figure
contour(X,Y,f,v,':g');
hold on
fprintf('result-BFGS:');
options.HessUpdate = 'bfgs';
x_BFGS = fminunc(fun,x0,options)
plot(x_BFGS(1),x_BFGS(2),'k*')
[x_BFGS,fval,exitflag,output]=fminunc(fun,x0,options)
n_B =output.funcCount;
i3=output.iterations;
tbfgs=cputime-t3;
title 'BFGS'
```

1.3 Estimation

Listing 6: Main Script

```
close all
clear all
clc
```

```
a0 = 10;
b0 = 0.9;
tol = power(10, -5);

y = zeros(40,1);
for i = 1:40
    y(i) = a0*power(b0,i-1)+randn(1);
end

X = 0.4:0.01:1.2;
Y = -0.2:0.01:15.2;
Z = zeros(length(X), length(Y))
for i=1:length(X)
    for j=1:length(Y)
        Z(i,j)=critere2([Y(j),X(i)],y);
    end
end

v= [0.01,0.1,1.5,10,20,40,60,75,100];


%method monovariable
t=cputime;
figure
contourf(Y,X,Z,v,':g');
hold on
options = optimset('TolX',tol,...
                   'MaxFunEvals',800,'MaxIter',200,'OutputFcn',@outfun2);

[b,fval,exitflag,output] = fminbnd(@(b) critere(b,y),0,1.8,options);
title 'mono-variable'
fprintf('result mono-variable')
b
i1 = output.iterations

a1 = 0;
a2 = 0;
for i = 1:length(y)
    a1 = a1 + power(b,i-1)*y(i);
    a2 = a2 + (power(b,i-1))^2;
end

a=a1/a2
duration1 = cputime-t

%method BFGS
t = cputime;
figure
contourf(Y,X,Z,v,':g');
options = optimset('TolX',tol,...
```

```
                    'MaxFunEvals',800,'MaxIter',200,'OutputFcn',@outfun);

[x,fval,exitflag,output] = fminunc(@(x) critere2(x,y),[1,1],options);
title 'BFGS'
fprintf('result-BFGS')
a_bfgs = x(1)
b_bfgs = x(2)
i2 = output.iterations

duration2 = cputime-t
```

Function - Critere :

```
function [J] = critere(b,y)
a1=0;
a2=0;
for i=1:length(y)
a1=a1+power(b,i-1)*y(i);
a2=a2+(power(b,i-1))^2;
end
a=a1/a2;
J=0;
for i=1:length(y)
    J=J+power(y(i)-a*power(b,i-1),2);
end
```

Function - Critere2 :

```
function [J]=critere2(x,y)
J=0;
for i=1:length(y)
    J=J+power(y(i)-x(1)*power(x(2),i-1),2);
end
```

Function - outfun :

```
function stop = outfun(x,optimValues,state)
        stop=false;
        switch state
            case 'init'
                hold on
            case 'iter'
                plot(x(1),x(2),'+');
                text(x(1)+0.1,x(2)+0.1,num2str(optimValues.iteration))
            case 'done'
                hold off
            otherwise
        end
    end
```

Function - outfun2 :

```
function stop = outfun2(b,optimValues,state)
    stop=false;
    switch state
        case 'init'
            hold on
        case 'iter'
            a0=10;
            b0=0.9;
            y=zeros(40,1);
            for i=1:40
                y(i)=a0*power(b0,i-1)+randn(1);
            end
            a1=0;
            a2=0;
            for i=1:length(y)
                a1=a1+power(b,i-1)*y(i);
                a2=a2+(power(b,i-1))^2;
            end
            a=a1/a2;
            plot(a,b,'+');
            text(a+0.1,b+0.1,num2str(optimValues.iteration));
        case 'cone'
            hold off
        otherwise
    end
end
```