

ECE58000 FunWork2

Shaunak Mukherjee

Spring 2024

Solution to problem 1

Exercise 5.6, page 65.

Given, $f(x) = \frac{x_1 x_2}{2}$, $g(s, t) = \begin{bmatrix} 4s + 3t \\ 2s + t \end{bmatrix}^T$, evaluate $\frac{d}{ds} f(g(s, t))$ and $\frac{d}{dt} f(g(s, t))$ using chain rule.

Chain rule for a multivariable composite function

$h(s, t) = (f \cdot g)(s, t) = f(g(s, t))$ is given by,

$$\frac{\partial h}{\partial s}(s, t) = \frac{\partial f}{\partial x_1}(g(s, t)) \cdot \frac{\partial g_1}{\partial s}(s, t) + \frac{\partial f}{\partial x_2}(g(s, t)) \cdot \frac{\partial g_2}{\partial s}(s, t) \quad (1)$$

and

$$\frac{\partial h}{\partial t}(s, t) = \frac{\partial f}{\partial x_1}(g(s, t)) \cdot \frac{\partial g_1}{\partial t}(s, t) + \frac{\partial f}{\partial x_2}(g(s, t)) \cdot \frac{\partial g_2}{\partial t}(s, t) \quad (2)$$

where, $\frac{\partial f}{\partial x_1}(g(s, t))$ and $\frac{\partial f}{\partial x_2}(g(s, t))$ are the partial derivatives of f with respect to its variables x_1 and x_2 , respectively, evaluated at $g(s, t)$. $\frac{\partial g_1}{\partial s}(s, t)$, $\frac{\partial g_1}{\partial t}(s, t)$, $\frac{\partial g_2}{\partial s}(s, t)$, and $\frac{\partial g_2}{\partial t}(s, t)$ are the partial derivatives of g with respect to its variables s and t , respectively.

$x_1 = 4s + 3t$, $x_2 = 2s + t$ and $\frac{\partial g_1}{\partial s}(s, t) = 4$ and $\frac{\partial g_2}{\partial s}(s, t) = 2$.

Also,

$$\frac{\partial f}{\partial x_1} = \frac{x_2}{2} \quad \text{and} \quad \frac{\partial f}{\partial x_2} = \frac{x_1}{2}$$

Substituting the values of x_1 and x_2 and adding them in (1):

$$\frac{\partial f}{\partial x_1} = \frac{2s + t}{2} \quad \text{and} \quad \frac{\partial f}{\partial x_2} = \frac{4s + 3t}{2}$$

$$\begin{aligned} \frac{d}{ds} f(g(s, t)) &= \frac{2s + t}{2} \cdot 4 + \frac{4s + 3t}{2} \cdot 2 \\ &= 4s + 2t + 4s + 3t \\ &= 8s + 5t \end{aligned}$$

Answer

To, find $\frac{d}{dt}f(g(s,t))$ using the chain rule using equation (2) and using calculations same way as before,

Here, $\frac{\partial g_1}{\partial t}(s,t) = 3$ and $\frac{\partial g_2}{\partial t}(s,t) = 1$

Following same way substituting values in (2):

$$\begin{aligned}\frac{d}{dt}f(g(s,t)) &= \frac{2s+t}{2} \cdot 3 + \frac{4s+3t}{2} \cdot 1 \\ &= 3s + \frac{3}{2}t + 2s + \frac{3}{2}t \\ &= 5s + 3t\end{aligned}$$

Answer

Solutions to problem 2

Exercise 7.2, page 105.

Given $f(x) = x^2 + 4\cos(x)$, $x \in \mathbb{R}$, to find minimizer x^* over interval $[1,2]$.

7.2 Part a)

Plot $f(x)$ vs x over interval $[1,2]$ and Matlab code use is below,

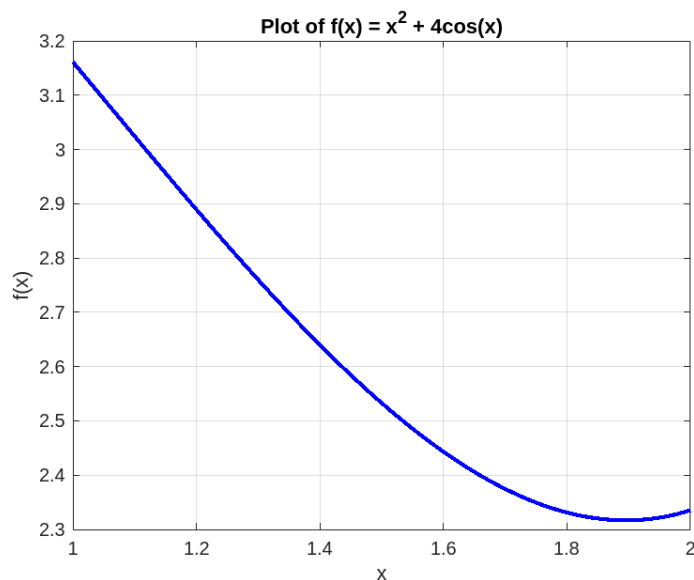


Figure 1: The plot of $f(x)$

```
% Define the function
f = @(x) x.^2 + 4*cos(x);

% Define the interval
```



```

% Loop until the interval size is within the uncertainty
while abs(b - a) > uncertainty

    k = k + 1;

    if f_a1 < f_b1
        b = b1;
        b1 = a1;
        f_b1 = f_a1;
        a1 = a + rho*(b-a);
        f_a1 = f(a1);
    else
        a = a1;
        a1 = b1;
        f_a1 = f_b1;
        b1 = a + (1 - rho)*(b-a);
        f_b1 = f(b1);
    end

fprintf('%d\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\n', k, a, b, a1,
        b1, f_a1, f_b1, abs(b - a))
end
x_min = (a + b) / 2;
f_min = f((a + b) / 2);
x_min
f_min

```

Table 1: Iterations table

N	a	b	a_k	b_k	$f(a_k)$	$f(b_k)$	$ b - a $
1	1.000000	2.000000	1.381966	1.618034	2.660671	2.429154	1.000000
2	1.381966	2.000000	1.618034	1.763932	2.429154	2.343707	0.618034
3	1.618034	2.000000	1.763932	1.854102	2.343707	2.319570	0.381966
4	1.763932	2.000000	1.854102	1.909830	2.319570	2.317147	0.236068
5	1.854102	2.000000	1.909830	1.944272	2.317147	2.320779	0.145898

Clearly, the value of x that minimizes f is located in interval $[1.854, 2.000]$ and uncertainty interval $[a_4, b_0] = [1.8541, 2.0000]$ **Answer**

7.2 Part c)

Repeat part b using Fibonacci method to locate x^* to within an uncertainty of 0.2. Display all intermediate steps using a table. Given $\epsilon = 0.05$

We will use the Matlab code below to solve it and show its range of minima

```

% Fibonacci Search Method
% Clear out text/screen

```

```
clear  
clc  
  
% Define the function f(x)  
f = @(x) x.^2 + 4 * cos(x);  
  
% Define the interval [a, b] & Fibonacci function as fib  
a = 1;  
b = 2;  
fib = @(n)fib_series(n+1);  
  
% Define the uncertainty  
uncertainty = 0.2;  
  
% Other parameters  
epsilon = 0.05;  
n = 4;  
  
% Display table header of list of parameters to list  
fprintf('k\t rho_k\t\t a\t\t b\t\t a_k\t\t b_k\t\t f(a_k)\t f(b_k)\t |b  
a|\t\t n');  
dashes = repmat('-', 1, 130); % Create a string of dashes  
disp(dashes); % Display the string of dashes  
  
% Initialization of parameters  
k = 1;  
rho = 1 - fib(n-k+1)/fib(n-k+2);  
a1 = a + rho*(b - a);  
b1 = a + (1-rho)*(b - a);  
f_a1 = f(a1);  
f_b1 = f(b1);  
  
fprintf('%d\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\n', k, rho,  
a, b, a1, b1, f_a1, f_b1, abs(b - a))  
  
% Defining loop  
while abs(b - a) > uncertainty  
  
if f_a1 < f_b1  
    b = b1;  
    b1 = a1;  
    f_b1 = f_a1;  
  
    k = k + 1;  
    rho = 1 - fib(n-k+1)/fib(n-k+2);  
    % Defining reduction limit  
    if rho == 0.5  
        rho = rho - epsilon;  
    end  
    a1 = a + rho*(b-a);
```

```

        f_a1 = f(a1);
    else
        a = a1;
        a1 = b1;
        f_a1 = f_b1;

        k = k + 1;
        rho = 1 - fib(n-k+1)/fib(n-k+2);
        % Defining reduction limit
        if rho == 0.5
            rho = rho - epsilon;
        end
        b1 = a + (1 - rho)*(b-a);
        f_b1 = f(b1);
    end

    if abs(b - a) > uncertainty
        fprintf('%d\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\n', k,
            rho, a, b, a1, b1, f_a1, f_b1, abs(b - a))
    end

end

% Define Fibonnaci Function
function fib = fib_series(n)

    if n <= 1
        fib = n;
    else
        fib = zeros(1, n+1);
        fib(1) = 0;
        fib(2) = 1;
        for i = 3:n+1
            fib(i) = fib(i-1) + fib(i-2);
        end
        fib = fib(n+1);
    end
end
end

```

Table 2: Iterations table

k	ρ_k	a	b	a_k	b_k	$f(a_k)$	$f(b_k)$	$ b - a $
1	0.375000	1.000000	2.000000	1.375000	1.625000	2.668816	2.423916	1.000000
2	0.400000	1.375000	2.000000	1.625000	1.750000	2.423916	2.349516	0.625000
3	0.333333	1.625000	2.000000	1.750000	1.875000	2.349516	2.317491	0.375000
4	0.450000	1.750000	2.000000	1.875000	1.887500	2.317491	2.316913	0.250000

Clearly, the value of x that minimizes f is located in interval $[1.750, 2.000]$ with uncertainty interval $[a_4, b_0] = [1.8750, 2.0000]$ **Answer**

7.2 Part d)

Apply Newton's Method using same number of iterations as in part b with $x^{(0)} = 1$.

We will use the Matlab code below to solve it with 4 iterations, I have assumed required accuracy of 10^{-5}

```
% Newton Method
% Clear workspace
clc
clear
% Function to minimize
syms x
f = x.^2 + 4 * cos(x);

% % Define first and second derivative of the function
fdx = diff(f);
fdx = matlabFunction(fdx);

fddx = diff(f,2);
fddx = matlabFunction(fddx);

% Display table header
fprintf('k\t xk1\t\t fxx1\t\t fdxk1\t\t fddxk1\t\t\n');
dashes = repmat('-', 1, 70); % Create a string of dashes
disp(dashes); % Display the string of dashes

% Initialization
x0 = 1; % Initial value
eps = 1e-5; % Small number to check convergency
max_iterations = 4; % Maximum number of iterations

f = matlabFunction(f);
x1 = x0 - (fdx(x0) / fddx(x0));
f1 = f(x1);

k = 1; % Counter

xk = x0;
xk1 = x1; % xk is kth and xk1 is k+1th term
fxx1 = f1;
fdxk1 = fdx(xk1);
fddxk1 = fddx(xk1);
fprintf('%d\t%.6f\t%.6f\t%.6f\t%.6f\n', k, xk1, fxx1, fdxk1, fddxk1)

while abs(xk1-xk) > eps && k <= max_iterations

    k = k + 1;
    xk = xk1;
    xk1 = xk - (fdx(xk) / fddx(xk));
```

```

    fvk1 = f(xk1);
    fdvk1 = fdx(xk1);
    fddvk1 = fddx(xk1);
    fprintf('%d\t%.6f\t%.6f\t%.6f\t%.6f\t\n', k, xk1, fvk1, fdvk1,
           fddvk1);

end

fprintf('The approximate minimum point and the value respectively are:
        %.4f and %.4f\n', xk1, fvk1);

```

Output is as follows,

Table 3: Values of k , x_k , $f(x_k)$, $f'(x_k)$, and $f''(x_k)$

k	x_k	$f(x_k)$	$f'(x_k)$	$f''(x_k)$
1	-7.472741	57.330143	-11.232667	0.511709
2	14.478521	208.288517	25.187833	3.339053
3	6.935115	51.275483	11.443344	-1.179657
4	16.635684	274.347348	36.472389	4.398638
5	8.343938	67.738946	13.158461	3.882348

The approximate minimum point x^* and the value respectively are: 8.3439 and 67.7389l. Note with Max iterations set at 100, we will get below table.
(Answer) With more iterations approximate minimum point x^* and the value

Table 4: Values of k , x_{k+1} , $f(x_{k+1})$, $f'(x_{k+1})$, and $f''(x_{k+1})$

k	x_{k+1}	$f(x_{k+1})$	$f'(x_{k+1})$	$f''(x_{k+1})$
1	-7.472741	57.330143	-11.232667	0.511709
2	14.478521	208.288517	25.187833	3.339053
3	6.935115	51.275483	11.443344	-1.179657
4	16.635684	274.347348	36.472389	4.398638
5	8.343938	67.738946	13.158461	3.882348
6	4.954633	25.507911	13.792474	1.040474
7	-8.301318	67.181618	-12.996226	3.730262
8	-4.817320	23.625525	-13.612639	1.581046
9	3.792574	11.201664	10.009020	5.181957
10	1.861061	2.318725	-0.110551	3.144823
11	1.896214	2.316809	0.002360	3.278819
12	1.895495	2.316808	0.000001	3.276091
13	1.895494	2.316808	0.000000	3.276090

respectively are: 1.8955 and 2.3168, which are close to minima as problems b.
(Answer)

Solutions to problem 3

Exercise 7.3, page 106.

Given

$$f(x) = 8e^{1-x} + 7\log(x)$$

7.3 Part a) Use Matlab to plot $f(x)$ vs x over $[1,2]$ and verify f is unimodal over the interval

```
% Clear workspace
clc
clear

% Define the function
f = @(x) 8 * exp(1 - x) + 7 * log(x);

% Define the interval
x_values = linspace(1, 2, 1000);

% Plot the function
figure;
plot(x_values, f(x_values));
xlabel('x');
ylabel('f(x)');
title('Plot of f(x)');

% Find the minimum of the function within [1, 2]
m = fminbnd(f, 1, 2);

% Create the Logic
if m > 1 && m < 2
    fprintf('The f is unimodal within the interval [1, 2].\n');
else
    fprintf('The f is not unimodal within the interval [1, 2].\n');
end
```

Below is output and the plot respectively. Output- The f is unimodal within the interval $[1, 2]$. **(Answer)**

Part 7.3 a) Plot $f(x)$ vs x over interval $[1,2]$

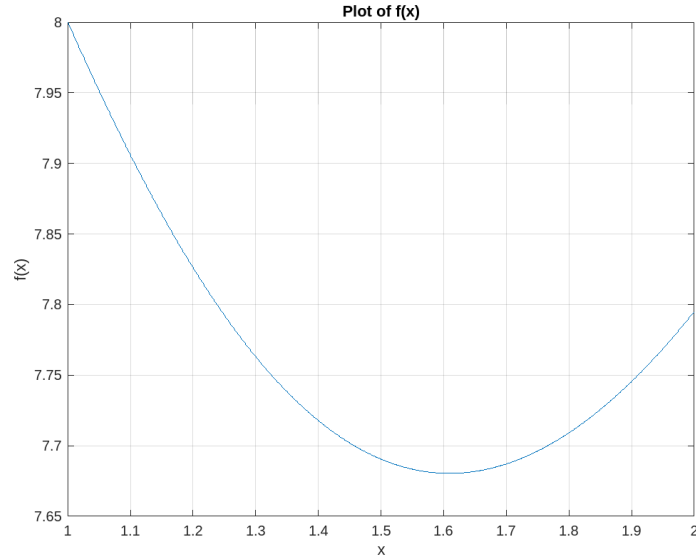


Figure 2: The plot of $f(x)$

Problem 7.3 Part b) Find minimizer of f over interval $[1,2]$, within uncertainty 0.23 and tabulate the intermediate steps (same as 7.2 (b))
We will use the following Matlab code below to estimate the location of x^*

```
clear
clc
% Define the function f(x)
f = @(x) 8*exp(1-x) + 7*log(x);
% Define the interval [a, b]
a = 1;
b = 2;
% Define the golden ratio
rho = (3 - sqrt(5)) / 2;
% Define the uncertainty
uncertainty = 0.23;
% Display table header
fprintf('k\t a\t\t b\t\t a_k\t\t b_k\t\t f(a_k)\t f(b_k)\t |b  
a|\t\n');
dashes = repmat('-', 1, 115); % Create a string of dashes
display(dashes); % Display the string of dashes
% Initialize iteration counter
k = 1;
% Initialize the variables for k=1
```

```

a1 = a + rho*(b - a);
b1 = a + (1-rho)*(b - a);
f_a1 = f(a1);
f_b1 = f(b1);
fprintf('%d\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\n', k, a, b, a1,
        b1, f_a1, f_b1, abs(b - a))
% Loop until the interval size is within the uncertainty
while abs(b - a) > uncertainty
    k = k + 1;
    if f_a1 < f_b1
        b = b1;
        b1 = a1;
        f_b1 = f_a1;
        a1 = a + rho*(b-a);
        f_a1 = f(a1);
    else
        a = a1;
        a1 = b1;
        f_a1 = f_b1;
        b1 = a + (1 - rho)*(b-a);
        f_b1 = f(b1);
    end
    fprintf('%d\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\n', k, a, b, a1,
            b1, f_a1, f_b1, abs(b - a))

end
x_min = (a+b)/2;
f_min = f(x_min);
disp(['The approx minimum point is: ', num2str(x_min)])
disp(['The approx minimum function value is: ', num2str(f_min)])

```

The output table and conclusion is as below. (Answer)

Table 5: Iterations table

k	a	b	a_k	b_k	$f(a_k)$	$f(b_k)$	$ b - a $
1	1.000000	2.000000	1.381966	1.618034	7.724696	7.680507	1.000000
2	1.381966	2.000000	1.618034	1.763932	7.680507	7.699467	0.618034
3	1.381966	1.763932	1.527864	1.618034	7.686003	7.680507	0.381966
4	1.527864	1.763932	1.618034	1.673762	7.680507	7.683814	0.236068
5	1.527864	1.673762	1.583592	1.618034	7.680996	7.680507	0.145898

clearly minimizer x^* of f lies within $[1.5279, 1.6738]$ The approx minimum point is: 1.6008 The approx minimum function value is: 7.6805 (**Answer**)

Problem 7.3 Part c) Repeat part b using fibonacci method with $\epsilon = 0.05$ and display intermediate Steps and table using Matlab (same as Prob 7.3 b)

```
% Fibonacci Search Method
% Clear out text/screen
% clear
% clc

% Define the function f(x)
f = @(x) 8*exp(1-x) + 7*log(x);

% Define the interval [a, b] & Fibonacci function as fib
a = 1;
b = 2;
fib = @(n)fib_series(n+1);

% Define the uncertainty
uncertainty = 0.23;

% Other parameters
epsilon = 0.05;
n = 100;

% Display table header of list of parameters to list
fprintf('k\t rho_k\t\t a\t\t b\t\t a_k\t\t b_k\t\t f(a_k)\t f(b_k)\t |b
      a\t\t\t');
dashes = repmat('-', 1, 130); % Create a string of dashes
disp(dashes); % Display the string of dashes

% Initialization of parameters
k = 1;
rho = 1 - fib(n-k+1)/fib(n-k+2);
a1 = a + rho*(b - a);
b1 = a + (1-rho)*(b - a);
f_a1 = f(a1);
f_b1 = f(b1);
fprintf('%d\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\n', k, rho,
      a, b, a1, b1, f_a1, f_b1, abs(b - a))

% Defining loop
while abs(b - a) > uncertainty
    if f_a1 < f_b1
        b = b1;
        b1 = a1;
        f_b1 = f_a1;
        k = k + 1;
        rho = 1 - fib(n-k+1)/fib(n-k+2);
        % Defining reduction limit
        if rho == 0.5
```

```

        rho = rho - epsilon;
    end
    a1 = a + rho*(b-a);
    f_a1 = f(a1);
else
    a = a1;
    a1 = b1;
    f_a1 = f_b1;
    k = k + 1;
    rho = 1 - fib(n-k+1)/fib(n-k+2);
    % Defining reduction limit
    if rho == 0.5
        rho = rho - epsilon;
    end
    b1 = a + (1 - rho)*(b-a);
    f_b1 = f(b1);
end

if abs(b - a) > uncertainty
    fprintf('%d\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\t%.6f\n', k,
        rho, a, b, a1, b1, f_a1, f_b1, abs(b - a))
end

end

x_min = (a+b)/2;
f_min = f(x_min);
disp(['The approx minimum point is: ', num2str(x_min)])
disp(['The approx minimum function value is: ', num2str(f_min)])

% Define Fibonnaci Function
function fib = fib_series(n)

    if n <= 1
        fib = n;
    else
        fib = zeros(1, n+1);
        fib(1) = 0;
        fib(2) = 1;
        for i = 3:n+1
            fib(i) = fib(i-1) + fib(i-2);
        end
        fib = fib(n+1);
    end
end
end

```

The output table and conclusion is as below. Using Fibonnaci, clearly minimizer x^* of f lies within $[1.5279, 1.7639]$ The approx minimum x point is: 1.6008 The approx minimum function value is: 7.6805 (**Answer**)

Table 6: Iterations table

k	ρ_k	a	b	a_k	b_k	$f(a_k)$	$f(b_k)$	$ b - a $
1	0.381966	1.000000	2.000000	1.381966	1.618034	7.724696	7.680507	1.000000
2	0.381966	1.381966	2.000000	1.618034	1.763932	7.680507	7.699467	0.618034
3	0.381966	1.381966	1.763932	1.527864	1.618034	7.686003	7.680507	0.381966
4	0.381966	1.527864	1.763932	1.618034	1.673762	7.680507	7.683814	0.236068

Solutions to problem 4

Exercise 7.12 Part a), page 107

Given, $f(x) = \frac{1}{2}x^T \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} x$ with initial guess $x^{(0)} = [0.8, -0.25]^T$. We will initialize line search using bracketing procedure from Fig 7.11 starting at $x^{(0)}$ in the direction of negative gradient and use $\epsilon = 0.075$ using following Matlab code

```
% Define the function
func = @(x) 0.5 * x' * [2, 1; 1, 2] * x;
gradient = @(x) [2, 1; 1, 2] * x; % Gradient since the matrix is
    symmetrical

% Initial guess & functional valus
x0 = [0.8; -0.25];
fk0 = func(x0);

eps = 0.075; % Step size

% Search condition & function values at brackets
xk0 = x0;
xk1 = xk0 - eps*(gradient(xk0));
eps = 2*eps; % Cadence of steps based on Fig 7.11 (page 103)
xk2 = xk1 - eps*(gradient(xk1));
fk0 = func(xk0);
fk1 = func(xk1);
fk2 = func(xk2);

% Bracketing conditions
while fk0 > fk1 && fk1 > fk2

    eps = 2*eps;
    xk_next = xk2 - eps*(gradient(xk2));
    xk0 = xk1;
    xk1 = xk2;
    xk2 = xk_next;
    % Function values
    fk0 = func(xk0);
    fk1 = func(xk1);
```

```

fk2 = func(xk2);

end
% Bracket condition
if fk1 < fk0 && fk1 < fk2
    bracket = [xk0, xk1, xk2];
    disp('Bracket containing minimum:')
    disp(bracket)
    bracket_vals = [fk0, fk1, fk2];
    disp('Function values at x0, x1, x3 are:')
    disp(bracket_vals)
else
    disp('Bracket not found!')
end

```

Bracket found and containing minimum as below

	x_{k0}	x_{k1}	x_{k2}
x_1	0.1062	0.0013	-0.1188
x_2	-0.1250	0.0475	-0.1835

Table 7: 7.12 a

Function values at x_{k0}, x_{k1}, x_{k3} : 0.0136, 0.0023, 0.0696 respectively.

Solution to problem 7.12 b)

$f(x) = \frac{1}{2}x^T \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} x$ Apply golden search method to reduce the width of uncertainty to 0.01 using below Matlab Code.

```

% Solution to Page 107 Question 7.12 b)

% Define the function
f = @(x) 0.5 * x' * [2, 1; 1, 2] * x; % Quadratic form

% Define the interval [a, b]
a = [0.1062; -0.1250]; % bracket- from problem 7.12 a)
b = [-0.1188; -0.1835]; % bracket+ from problem 7.12 a)

% Define the golden ratio
rho = (3 - sqrt(5)) / 2;

% Define the uncertainty
uncertainty = 0.01;

% Display table header

```

```

fprintf('k\t\t a\t\t b\t\t a_k\t\t b_k\t\t f(a_k)\t f(b_k)\t |b
      a|\t\n');
dashes = repmat('-', 1, 150); % Create a string of dashes
disp(dashes); % Display the string of dashes

% Initialize iteration counter
k = 1;

% Initialize the variables for k=1
a1 = a + rho*(b - a);
b1 = a + (1-rho)*(b - a);
f_a1 = f(a1);
f_b1 = f(b1);

% Output result for k=1
fprintf('%d\t[%.6f, %.6f]\t[%.6f, %.6f]\t[%.6f, %.6f]\t[%.6f,
      %.6f]\t%.6f\t%.6f\t%.6f\n', k, a(1), a(2), b(1), b(2), a1(1),
      a1(2), b1(1), b1(2), f_a1, f_b1, norm(b - a));

% Loop until the interval size is within the uncertainty
while norm(b - a) > uncertainty

    k = k + 1;

    if f_a1 < f_b1
        b = b1;
        b1 = a1;
        f_b1 = f_a1;
        a1 = a + rho*(b-a);
        f_a1 = f(a1);
    else
        a = a1;
        a1 = b1;
        f_a1 = f_b1;
        b1 = a + (1 - rho)*(b-a);
        f_b1 = f(b1);
    end

% Output results
fprintf('%d\t[%.6f, %.6f]\t[%.6f, %.6f]\t[%.6f, %.6f]\t[%.6f,
      %.6f]\t%.6f\t%.6f\t%.6f\n', k, a(1), a(2), b(1), b(2), a1(1),
      a1(2), b1(1), b1(2), f_a1, f_b1, norm(b - a));

end

% Output results
x_min = (a1+b1)/2;
f_min = f(x_min);
disp(['The approx minimum point is: (', num2str(x_min(1)), ', ',

```



```

num2str(x_min(2)), ' ']);
disp(['The approx minimum function value is: ', num2str(f_min)])

```

Table 8: Problem 7.12 b)

k	a	b	a_k	b_k	$f(a_k)$	$f(b_k)$	$ b - a $
1	[0.106200, -0.125000]	[-0.118800, -0.183500]	[0.020258, -0.147345]	[-0.032858, -0.161155]	0.019136	0.032346	0.232481
2	[0.106200, -0.125000]	[-0.032858, -0.161155]	[0.053085, -0.138810]	[0.020258, -0.147345]	0.014718	0.019136	0.143681
3	[0.106200, -0.125000]	[0.020258, -0.147345]	[0.073373, -0.133535]	[0.053085, -0.138810]	0.013417	0.014718	0.088800
4	[0.106200, -0.125000]	[0.053085, -0.138810]	[0.085912, -0.130275]	[0.073373, -0.133535]	0.013160	0.013417	0.054881
5	[0.106200, -0.125000]	[0.073373, -0.133535]	[0.093661, -0.128260]	[0.085912, -0.130275]	0.013210	0.013160	0.033918
6	[0.093661, -0.128260]	[0.073373, -0.133535]	[0.085912, -0.130275]	[0.081122, -0.131520]	0.013160	0.013209	0.020963
7	[0.093661, -0.128260]	[0.081122, -0.131520]	[0.088872, -0.129505]	[0.085912, -0.130275]	0.013160	0.013160	0.012956
8	[0.088872, -0.129505]	[0.081122, -0.131520]	[0.085912, -0.130275]	[0.084082, -0.130751]	0.013160	0.013172	0.008007

The approx minimum point is: (0.084997, -0.13051) and the approx minimum function value is: 0.013165 (Answer)

Solution to problem 7.12 c)

$f(x) = \frac{1}{2}x^T \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} x$ Use Fibonacci Search using Matlab as below

```

% An Introduction to Optimization with Applications to Machine Learning,
% E.Chong, W.S.Lu, S.H. Zak, 5e
% Solution to Page 107 Question 7.12 c)

% Fibonacci Search Method
% Clear out text/screen
clear
clc

% Define the function
f = @(x) 0.5 * x' * [2, 1; 1, 2] * x; % Quadratic form

% Define the interval [a, b] and Fibonnaci function as fib
a = [0.1062; -0.1250]; % bracket- from problem 7.12 a)
b = [-0.1188; -0.1835]; % bracket+ from problem 7.12 a)
fib = @(n)fib_series(n+1);

% Define the uncertainty
uncertainty = 0.01;

% Other parameters and Max Iterations
epsilon = 0.075;
n = 100;

% Display table header
fprintf('k\t rho_k\t\t\t a\t\t\t b\t\t\t a_k\t\t\t b_k\t\t\t f(a_k)\t\t\t f(b_k)\t\t\t |b - a|\n');
dashes = repmat('-', 1, 160); % Create a string of dashes

```

```

disp(dashes); % Display the string of dashes

% Initialization of parameters
k = 1;
rho = 1 - fib(n-k+1)/fib(n-k+2);
a1 = a + rho*(b - a);
b1 = a + (1-rho)*(b - a);
f_a1 = f(a1);
f_b1 = f(b1);

% Output result for k=1
fprintf('%d\t%.6f\t%.6f, %.6f\t%.6f, %.6f\t%.6f, %.6f\t%.6f, %.6f\t%.6f\n', k, rho, a(1), a(2), b(1), b(2), a1(1),
        a1(2), b1(1), b1(2), f_a1, f_b1, norm(b - a));

% Defining loop
while norm(b - a) > uncertainty

    if f_a1 < f_b1
        b = b1;
        b1 = a1;
        f_b1 = f_a1;

        k = k + 1;
        rho = 1 - fib(n-k+1)/fib(n-k+2);
        % Defining reduction limit
        if rho == 0.5
            rho = rho - epsilon;
        end
        a1 = a + rho*(b-a);
        f_a1 = f(a1);
    else
        a = a1;
        a1 = b1;
        f_a1 = f_b1;

        k = k + 1;
        rho = 1 - fib(n-k+1)/fib(n-k+2);
        % Defining reduction limit
        if rho == 0.5
            rho = rho - epsilon;
        end
        b1 = a + (1 - rho)*(b-a);
        f_b1 = f(b1);

    end

end

```

```

if norm(b - a) > uncertainty

    fprintf('%d\t%.6f\t[%.6f, %.6f]\t[%.6f, %.6f]\t[%.6f, %.6f]\t[%.6f, %.6f]\t[%.6f]\n', k, rho, a(1), a(2), b(1), b(2), a1(1), a1(2), b1(1), b1(2), f_a1, f_b1, norm(b - a));

end

end

x_min = (a1+b1)/2;
f_min = f(x_min);
disp(['The approx minimum point is: (', num2str(x_min(1)), ', ', num2str(x_min(2)), ')']);
disp(['The approx minimum function value is: ', num2str(f_min)])

% Define Fibonnaci Function
function fib = fib_series(n)

    if n <= 1
        fib = n;
    else
        fib = zeros(1, n+1);
        fib(1) = 0;
        fib(2) = 1;
        for i = 3:n+1
            fib(i) = fib(i-1) + fib(i-2);
        end
        fib = fib(n+1);
    end
end

```

The approx minimum point is: (0.084997, -0.13051). The approx minimum function value is: 0.013165 **Answer**

Table 9: Problem 7.12 c)

k	ρ_k	a	b	a_k	b_k	$f(a_k)$	$f(b_k)$	$ b - a $
1	0.381966	[0.106200, -0.125000]	[-0.118800, -0.183500]	[0.020258, -0.147345]	[-0.032858, -0.161155]	0.019136	0.032346	0.232481
2	0.381966	[0.106200, -0.125000]	[-0.032858, -0.161155]	[0.053085, -0.138810]	[0.020258, -0.147345]	0.014718	0.019136	0.143681
3	0.381966	[0.106200, -0.125000]	[0.020258, -0.147345]	[0.073373, -0.133535]	[0.053085, -0.138810]	0.013417	0.014718	0.088800
4	0.381966	[0.106200, -0.125000]	[0.053085, -0.138810]	[0.085912, -0.130275]	[0.073373, -0.133535]	0.013160	0.013417	0.054881
5	0.381966	[0.106200, -0.125000]	[0.073373, -0.133535]	[0.093661, -0.128260]	[0.085912, -0.130275]	0.013210	0.013160	0.033918
6	0.381966	[0.093661, -0.128260]	[0.073373, -0.133535]	[0.085912, -0.130275]	[0.081122, -0.131520]	0.013160	0.013209	0.020963
7	0.381966	[0.093661, -0.128260]	[0.081122, -0.131520]	[0.088872, -0.129505]	[0.085912, -0.130275]	0.013160	0.013160	0.012956

Solutions to problem 5 Problem 5. For the banana function of Example 5.2 on page 58 Use MATLAB's commands meshgrid and mesh to generate its 3D

plot. The ranges of x_1 and x_2 are the same and they should be equal to the ranges in Figure 5.2 and 5.3 on page 59. Set the box on. Use the command `contour` to generate contours as given in Example 5.2.

```
% % Define the function f(x)
f = @(x1, x2) 100*(x2 - x1.^2).^2 + (1 - x1).^2;

% Define the range for x1 and x2 & create meshgrid
x1_range = linspace(-2, 2, 100);
x2_range = linspace(-1, 3, 100);
[x1, x2] = meshgrid(x1_range, x2_range);

% Define Z as function of f each combination of x1 and x2
z = f(x1, x2);

% Plot the 3D surface
figure;
mesh(x1, x2, z);
xlabel('x_1');
ylabel('x_2');
zlabel('f(x)');
title('3D Plot of Rosenbrock banana function f(x)');

% Add levels
hold on;
contour3(x1, x2, z, [0.7, 7, 70, 200, 700], 'k', 'LineWidth', 1);
hold off;
legend('f(x)', 'Level Sets: 0.7, 7, 70, 200, 700', 'Location',
      'NorthEast');
colorbar

% Define the range for x1 and x2
x1_range = linspace(-2, 2, 100);
x2_range = linspace(-1, 3, 100);
% Define Z as function of f each combination of x1 and x2
z = f(x1, x2);
% Generate contours with levels
figure;
contour(x1, x2, z, [0.7, 7, 70, 200, 700], 'LineWidth', 2);
xlabel('x_1');
ylabel('x_2');
title('Contours of the Function f(x)');
legend('Level Sets: 0.7, 7, 70, 200, 700', 'Location', 'NorthEast');
colorbar
```

Using the Matlab code we get the below output in next page

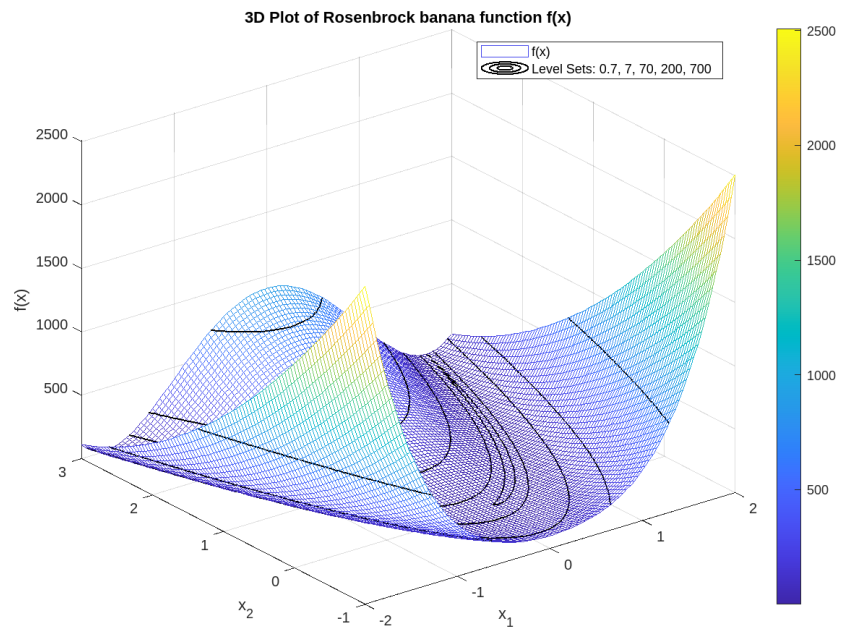


Figure 3: Banana Func 3D Plot

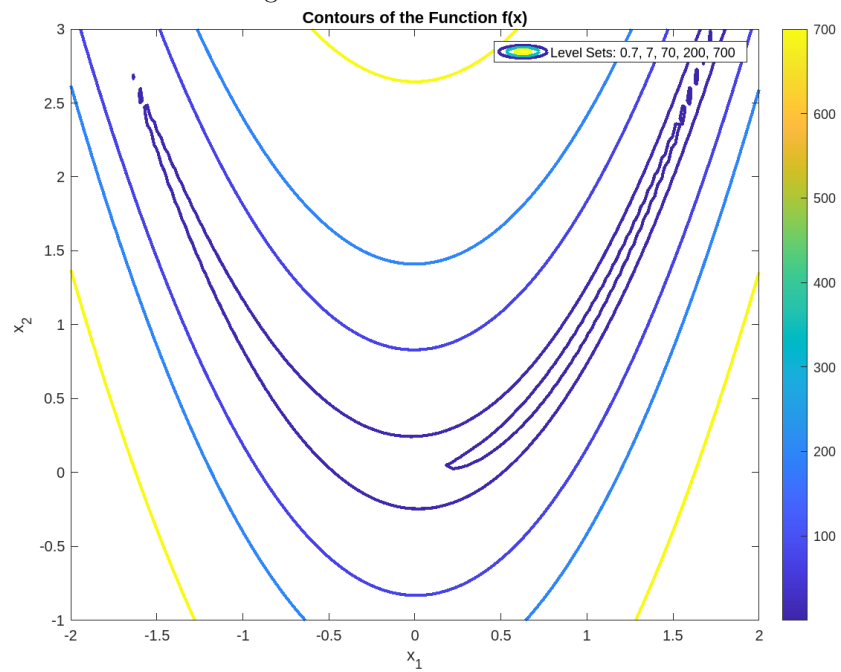


Figure 4: Banana Func 2D Contour Plot