6.2 (d) Parameter estimation by EM algorithm

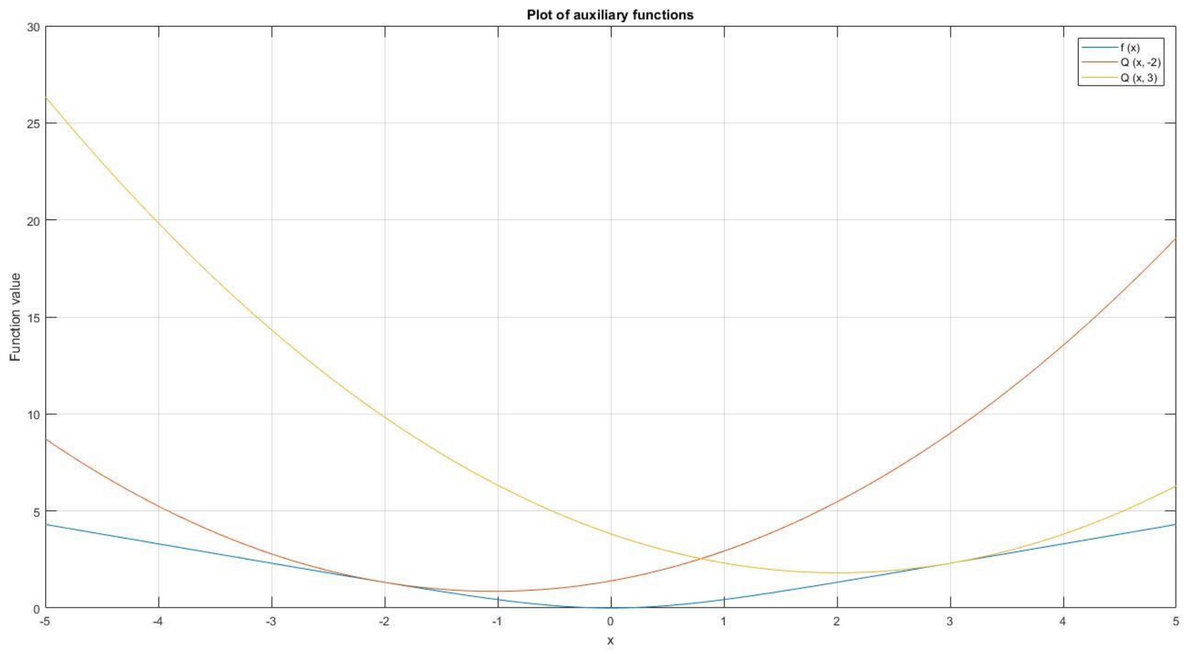
Iteration	Number of mistakes, M	Log-likelihood, L
0	175	-0.958085
1	56	-0.495916
2	43	-0.408221
4	42	-0.364615
8	44	-0.347501
16	40	-0.334617
32	37	-0.322581
64	37	-0.314827
128	36	-0.311156
256	36	-0.310161

<u>Source Code:</u>

```
% Read example datasets
n=23;
% X
tempX =
textread('data/spectX.txt','%d');
for t=1:length(tempX)/n
    X(t,:) = tempX(n*(t-1)+1:n*t);
end
% Y
y=textread('data/spectY.txt','%d');

% Number of examples
T = length(y);iter = 256;
p = 0.05*ones(1,n);

% Number of inputs with xi=1
Tk = zeros(1,n);
for i=1:n
    for t=1:T
        if(X(t,i)==1)
            Tk(i) = Tk(i) + 1;
        end
    end
end

% Iterative EM algorithm from here..
L = zeros(1,iter+1);M = zeros(1,iter+1);
for k=1:iter
        % Log-likelihood
    for t=1:T
        temp1 = 0; temp2 = 1;
        for i =1:n
            temp1 = temp1
+(X(t,i)*(log(1-p(i))));
            temp2 = temp2 *((1-
p(i))^X(t,i));
        end
        L(k) = L(k) + ((1-y(t))*temp1)
+(y(t)*(log(1-temp2)));
    end     L(k) = L(k)/T;
    % Mistakes
    pr = zeros(T,1);
    for t=1:T
        tp=1;
        for i=1:n
            tp=tp*((1-p(i))^X(t,i));
        end
        pr(t)=1-tp;
        if(((y(t)==0)&&(pr(t)>=0.5)) ||
((y(t)==1) && (pr(t)<0.5)))
            M(k) = M(k) + 1;
        end
    end
    % E-Step
    post = zeros(T,n);
    for t=1:T
        % Computing product term in
denominator tp=1;
        for j=1:n
            tp=tp*((1-p(j))^X(t,j));
        end

        % Compute posterior probability
        for i=1:n
            post(t,i) =
(y(t)*X(t,i)*p(i))/(1-tp);
        end
    end
    % M-step
    for i = 1:n
        tDen = 0;
        for t=1:T
            tDen = tDen + post(t,i);
        end
        % update pi's
        p(i) = (1/Tk(i))*(tDen);
    end
end
```

6.3 (c) Q functions vs the given functions

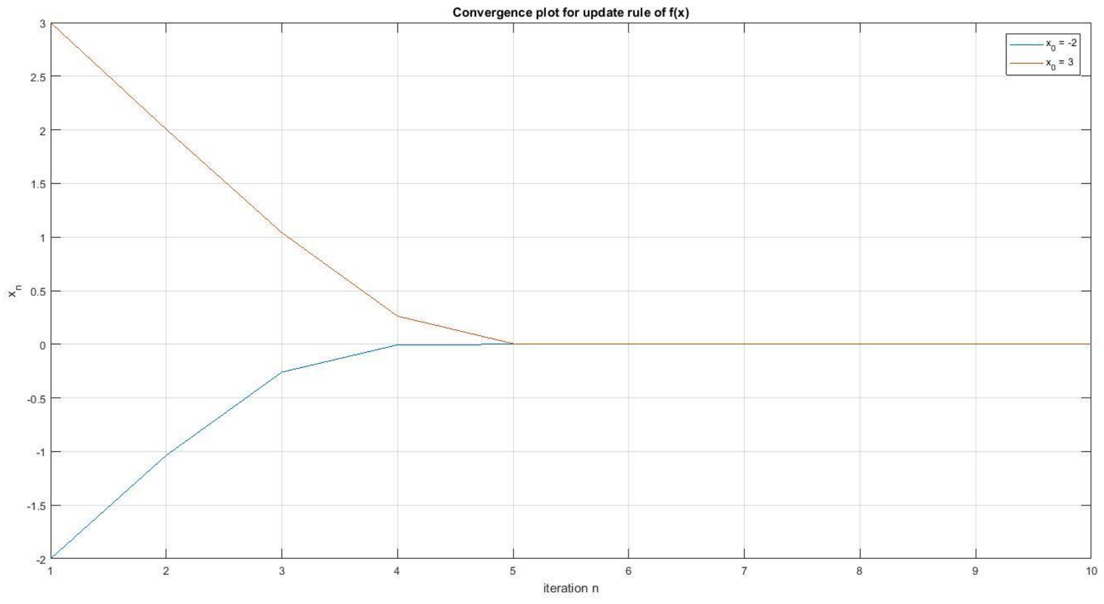
Plot of Q function at 2 different points -2,3 versus f(x)



<u>Code –</u>

```
clc;
clear all;
close all;

x=-5:0.1:5;
f1 = fx(x);
Q1 = Qxy(x,-2);
Q2 = Qxy(x,3);

% Auxiliary functions
figure;
set(gcf,'color','w');
plot(x,f1,x,Q1,x,Q2);
grid on;
title('Plot of auxiliary functions');
xlabel('x');
ylabel('Function value');
legend('f (x)','Q (x, -2)','Q (x, 3)');

%% Helper functions

% Given function
function op=fx(x)
op = log(cosh(x));
end

% Auxiliary function
function op = Qxy(x, y)
    op = fx(y) + (tanh(y)*(x-y)) + (0.5*(x-y).^2);
end
```

(f) Numerical convergence of the update rule using the Q function for different initial guesses

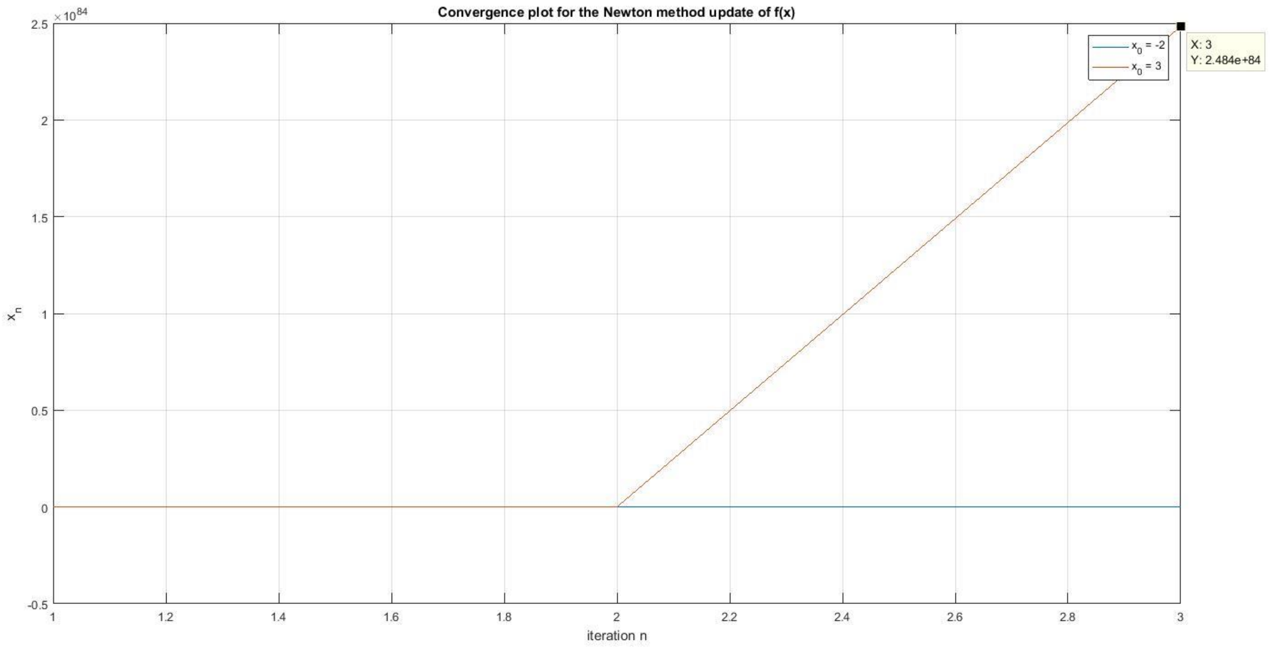


We can see that the update rule converges to x=0 for both initial guesses at x0.

<u>Code :</u>

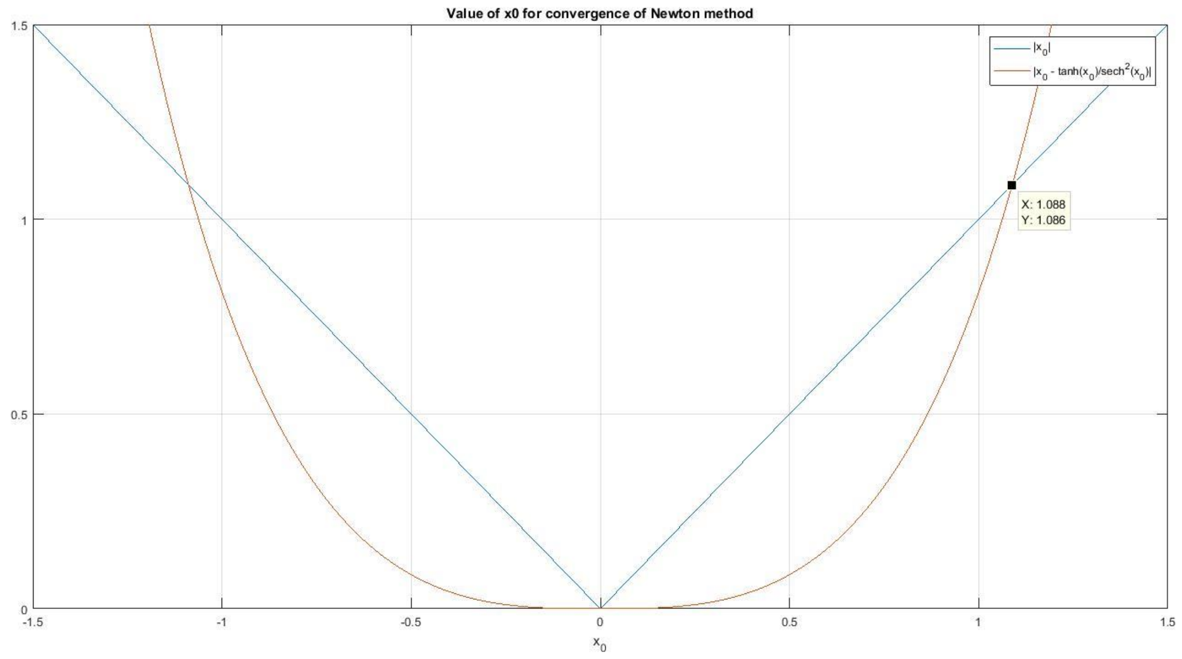
```
clc;
clear all;
close all;

iter = 5;

% With initial guess x=-2
x1(1) = -2;
for n=2:iter
    x1(n) = x1(n-1) - tanh(x1(n-1));
end

% With initial guess x=3
x2(1) = 3;
for n=2:iter
    x2(n) = x2(n-1) - tanh(x2(n-1));
end

% Convergence plot
figure;
set(gcf,'color','w');
plot(1:iter,x1,1:iter,x2);
grid on;
title('Convergence plot for update rule of f(x)');
xlabel('iteration n');
ylabel('x_{n}');
legend('x_{0} = -2', 'x_{0} = 3');
```

(g) Stability of convergence of update rule from Newton method



We can see from the plot that the update rule is highly unstable (goes to e84) for very few iterations of the algorithm. This happens because the initial guess is not close to the actual minima (x=0). Thus, newton's method has a very bad performance if the initial guess is poor.

To ensure convergence of the Newton algorithm, we need to have the right upper bound on the initial guess of x0, which we can estimate by plotting the graphs for |x1| < |x0|.

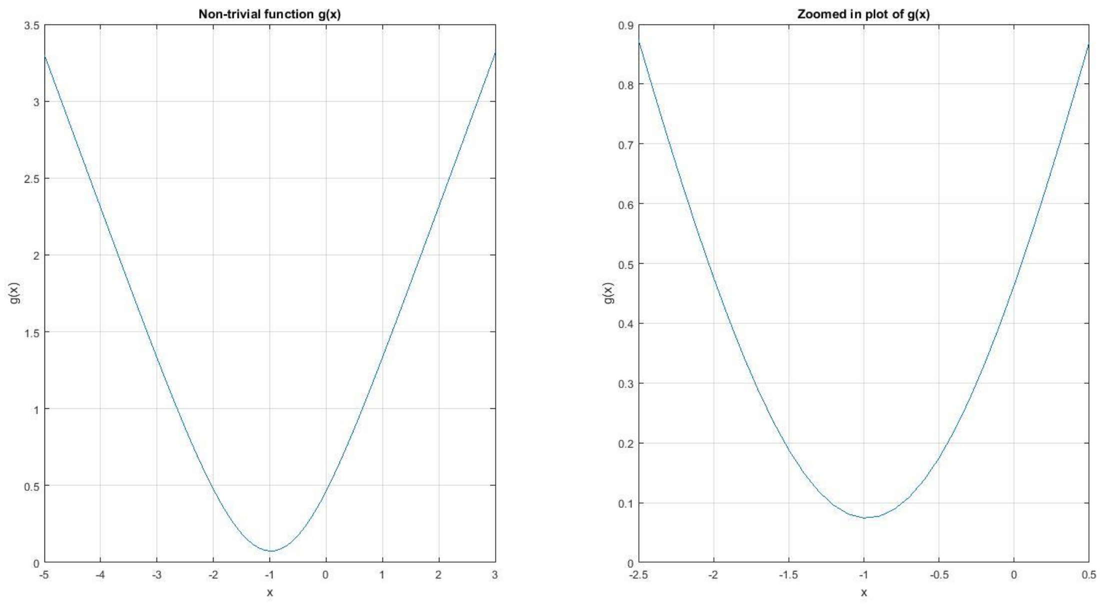


From the graph we can see that the estimate is; **|x0| = 1.088** for the Newtons algorithm to converge.

Code :

```
clc;
clear all;
close all;

% Testing the stability of the Newton update rule for different values
iter = 10;

x1(1) = -2;
for n=2:iter
    x1(n) = x1(n-1) - (tanh(x1(n-1))/(sech(x1(n-1)))^2);
end

x2(1) = 3;
for n=2:iter
    x2(n) = x2(n-1) - (tanh(x2(n-1))/(sech(x2(n-1)))^2);
end

% Convergence plot
figure;
set(gcf,'color','w');
plot(1:iter,x1);
hold on;
plot(1:3,x2(1:3));
hold off;
grid on;
title('Convergence plot for the Newton method update of f(x)');
xlabel('iteration n');
ylabel('x_{n}');
legend('x_{0} = -2', 'x_{0} = 3');

% upper boubd ob x0 for convergence of Newton method
x0 = -5:0.001:5;
f = abs(x0 - (tanh(x0)./(sech(x0)).^2));

% Estimate of x0 for convergence of Newton method
figure;
set(gcf,'color','w');
plot(x0,abs(x0),x0,f);
grid on;
ylim([0 1.5]);
xlabel('x_{0}');
legend('|x_{0}|','|x_{0} - tanh(x_{0})/sech^{2}(x_{0})|');
title('Value of x0 for convergence of Newton method');
```

(h) Plotting the new function g(x)

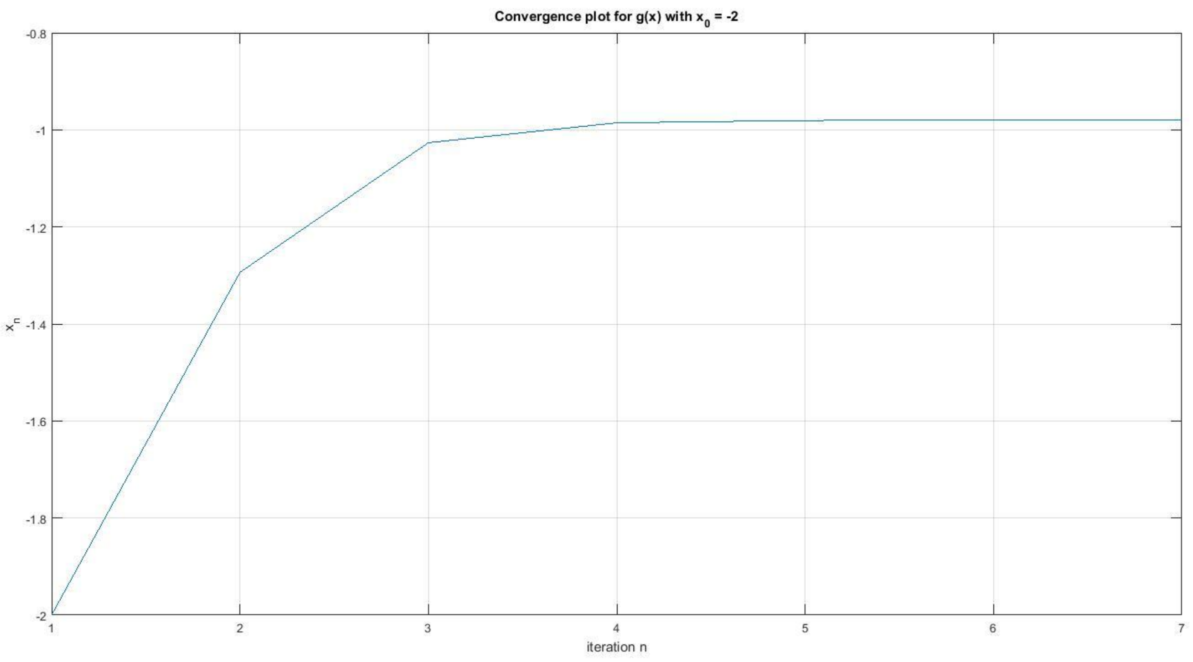


We can see from the plot that estimating the minimum of g(x) is not very simple or straight-forward

<u>Code –</u>

```
x = -5:0.1:3;
g = 0;

% Find g(x) at various points
for k = 1:10
    g = g + log(cosh(x+(2/sqrt(k))));
end
g=g/10;

% Plot g(x) vs x
figure;
subplot(1,2,1);
plot(x,g);
set(gcf,'color','w');
grid on;
xlabel('x');
ylabel('g(x)');
title('Non-trivial function g(x)');

% Zoom in the minima
subplot(1,2,2);
plot(x,g);
grid on;
xlim([-2.5 0.5]);
xlabel('x');
ylabel('g(x)');
title('Zoomed in plot of g(x)');
```

(k) Local minimum of g(x)



We can see from the graph that the update rule converges quickly. The minimum of g(x) is at :

$$x = -0.9800\ and g(x) = 0.0742$$

<u>Code :</u>

```
clc;
clear;
close all;
iter = 70;

% Convergence of update rule
x1(1) = -2;
for n=2:iter
    temp=0;
    for k=1:10
        temp=temp+tanh(x1(n-1)+(2/sqrt(k)));
    end
    x1(n) = x1(n-1) - ((1/10)*temp);
end

% convergence plot
figure;
set(gcf,'color','w');
plot(1:iter,x1);
grid on;
title('Convergence plot for g(x) with x_{0} = -2');
xlabel('iteration n');
ylabel('x_{n}');
```