Q1: 1)

For this decision boundary:

$$\varphi(v) = \xi$$

The induced local field should be equal to the user-defined threshold:

$$av + b = \xi$$

So, we could get one formula to describe the geometrical shape of the decision boundary:

$$v = w_1x_1 + w_2x_2 + \ldots w_nx_n + b = -\frac{\xi - b}{a}$$

So, the decision boundary is still a hyper-plane in the m-dimensional space.

2)

The induced local field should be equal to the user-defined threshold:

$$\frac{1}{1 + e^{-2v}} = \xi$$

So, we could get one formula to describe the geometrical shape of the decision boundary:

$$v = w_1x_1 + w_2x_2 + \cdots w_nx_n + b = -\frac{1}{2}\ln\left(\frac{1}{\xi} - 1\right)$$

So, the decision boundary is still a hyper-plane in the m-dimensional space.

3)

The induced local field should be equal to the user-defined threshold:

$$e^{-\frac{v^2}{2}} = \xi$$

we could get two formulas to describe the geometrical shape of the decision boundary:

$$v = w_1x_1 + w_2x_2 + \ldots w_nx_n + b = \sqrt{-2ln(\xi)}$$

Or

$$v = w_1x_1 + w_2x_2 + \ldots w_nx_n + b = -\sqrt{-2ln(\xi)}$$

So, the decision boundary is not a hyper-plane.

Q2: mathematic proof

This question could be proved by contradiction method.

If the XOR problem is linearly separable, there are only two classes: class 1 and 2.

Class 1: $y = 1$

For class 1, the induced local field could be assumed bigger (or the inverse case) than the threshold: $v = w_1x_1 + w_2x_2 + \ldots w_nx_n + b > \xi$.

Class 2: $y = 0$

For class 1, the induced local field could be assumed smaller (or the inverse case) than the threshold: $v = w_1x_1 + w_2x_2 + \ldots w_nx_n + b < \xi$.

For $x_1 = 1, x_2 = 0$, the induced local field is bigger than the threshold: $v_2 = w_1 + b > \xi \quad (1)$.

For $x_1 = 0, x_2 = 1$, the induced local field is bigger than the threshold: $v_3 = w_2 + b > \xi \quad (2)$.

For $x_1 = 1, x_2 = 1$, the induced local field is smaller than the threshold: $v_4 = w_1 + w_2 + b < \xi \quad (3)$.

If we calculate the formula (1)+(2)-(3), we could get:

$$w_1 + b + w_2 + b - (w_1 + w_2 + b) = b > \xi \quad (4)$$

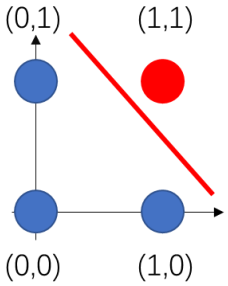
However for $x_1 = 0, x_2 = 0$, the induced local field is smaller than the threshold: $v_1 = b < \xi \quad (5)$.

So, the formula (4) is contradictory with (5) and the XOR problem is not linearly separable.

Q3:

a) AND

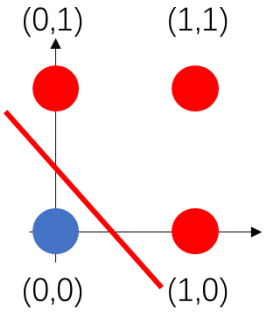
$$x_1 + x_2 - 1.5 = 0$$



$$w' = \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}$$

OR

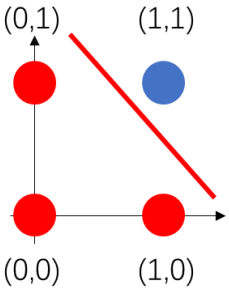
$$x_1 + x_2 - 0.5 = 0$$



$$w' = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}$$

NAND

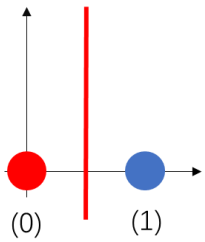
$$-x_1 - x_2 + 1.5 = 0$$



$$w' = \begin{bmatrix} 1.5 \\ -1 \\ -1 \end{bmatrix}$$

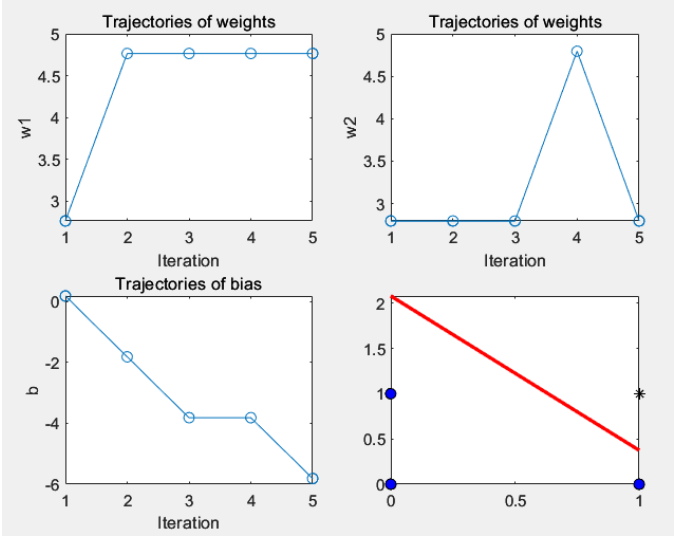
COMPLEMENT

$$-x_1 + 0.5 = 0$$



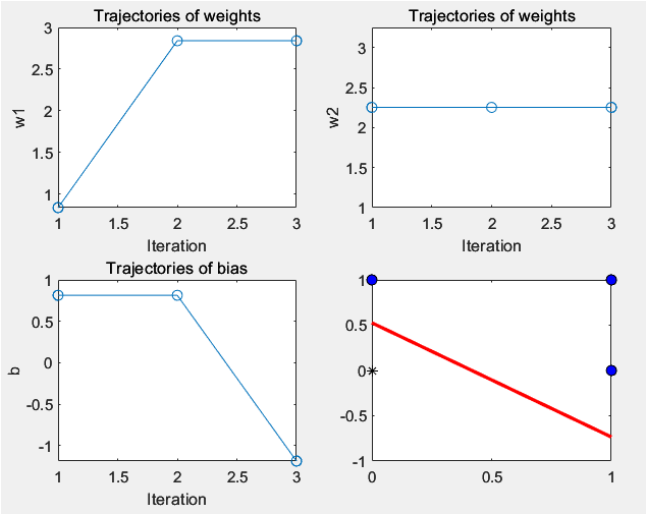
$$w' = \begin{bmatrix} 0.5 \\ -1. \end{bmatrix}$$

b) AND



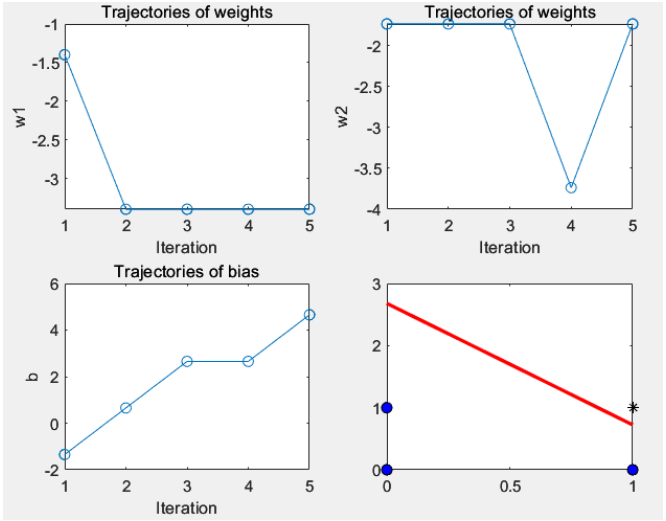
$$w' = \begin{bmatrix} -5.8131 \\ 4.7655 \\ 2.7952 \end{bmatrix}$$

OR



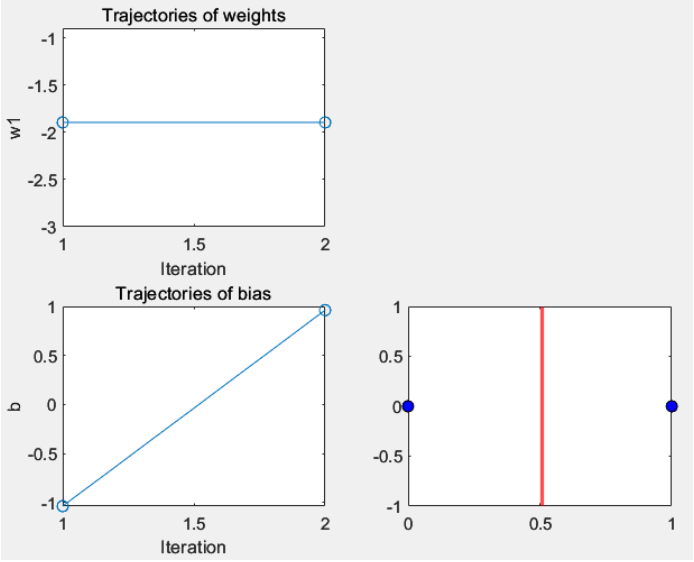
$$w' = \begin{bmatrix} -1.1857 \\ 2.8407 \\ 2.2543 \end{bmatrix}$$

NAND



$$w' = \begin{bmatrix} 4.6541 \\ -3.398 \\ -1.737 \end{bmatrix}$$

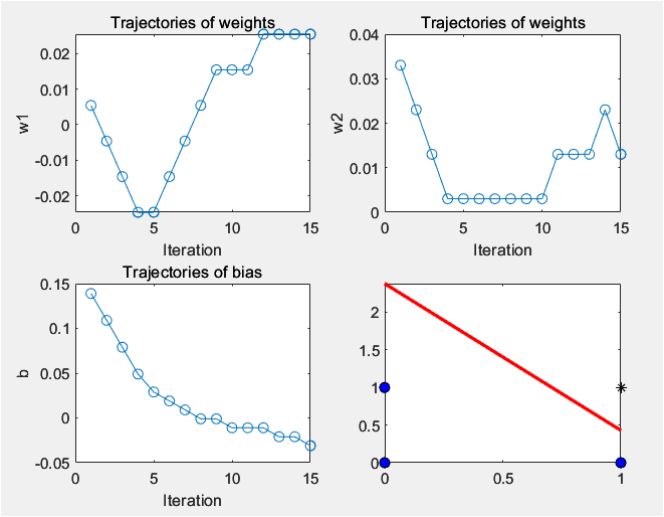
COMPLEMENT



$$w' = \begin{bmatrix} 0.9619 \\ -1.8933 \end{bmatrix}$$

Compared with the outcomes got from question a), it could be revealed that the results from learning processes could just satisfy our request as the method of off-line calculations. However, the learning results differed in each running process.

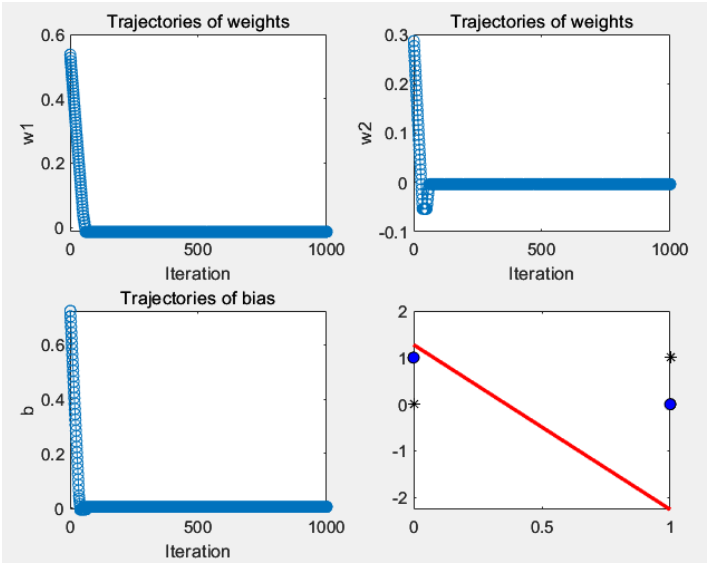
If I decreased the learning rate dramatically, iteration number generally increased. The AND gate example of the learning rate was set to be 0.005. I found that it needed 15 iterations to complete the task. And original iteration number was just 5.



The situation didn't change too much if I increased the learning rate dramatically. The required iteration number was just as small as the original one.

c)

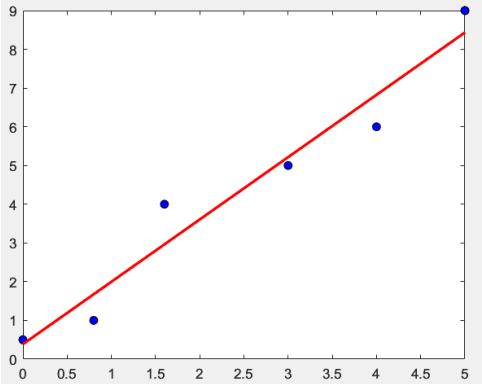
When I feed XOR problem to the code, it wouldn't give me final results. It would only be stopped by the computer for protection.



Q4:

a) using the standard linear least-squares (LLS) method:

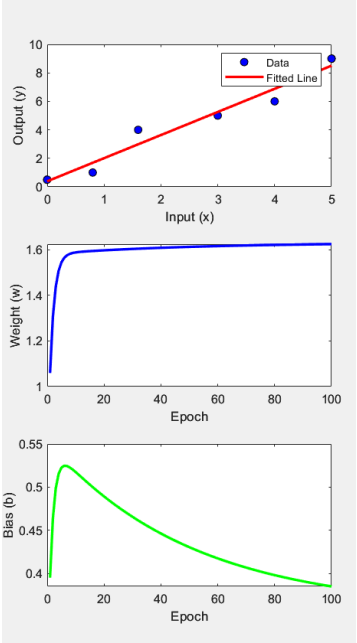
$$w = (X^T X)^{-1} X^T d$$



Intercept (b): 0.387339

Slope (w): 1.609442

b) using the least-mean-square (LMS) algorithm. The weight and bias would converge.

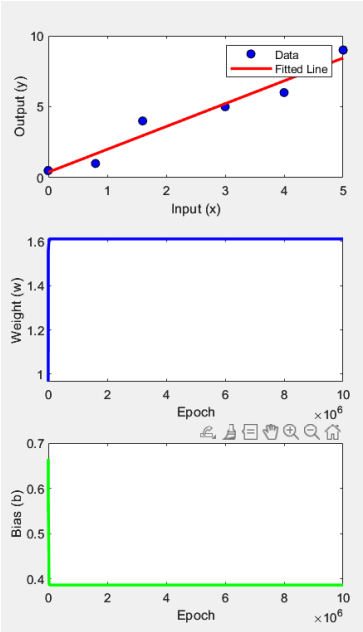


Intercept (b): 0.3852

Slope (w): 1.6236

c) The result got by the least-mean-square(LMS) was very close to the result got by the standard linear least-squares (LLS) method. LLS method could help us get the theoretical answer, but LMS would only help us get close to it. In this question, there is only one minimum value, which is the global minimum. So LMS would only converge at the value which was calculated by LLS.

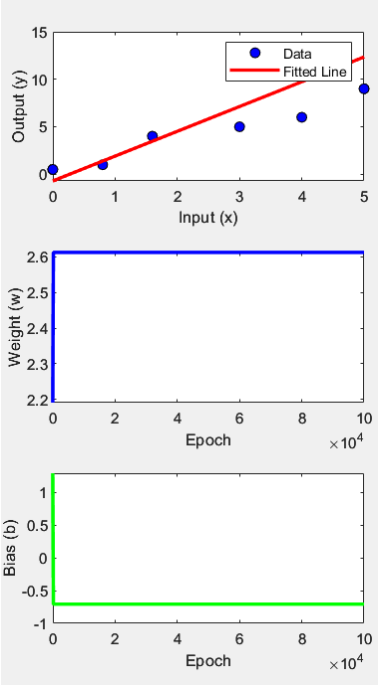
d) When we decreased the learning rate to 0.0001, I found that the LMS method would lead to the result got by LLS method after large number of epochs. The larger epochs were, the closer two results were.



Intercept (b): 0.3871

Slope (w): 1.6096

When I increased the learning rate to 0.5 from 0.01, the weight and bias would not converge. When I increased the learning rate to 0.1 from 0.01, I found the result was deviated from LLS's results even when the weights have already converged.



Intercept (b): -0.7072

Slope (w): 2.6115

Q5:

Based on the instantaneous cost function at step n:

$$J(w)_n = r(n)e(n)^2$$

Where $e(n)$ is the error signal measured at step n:

$$e(n) = d(n) - x^T(n)w(n)$$

Apply the chain rule:

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial e} * \frac{\partial e}{\partial w}$$

$$\frac{\partial J}{\partial e} = 2r(n)e(n), \frac{\partial e}{\partial w} = -x^T(n)$$

$$\frac{\partial J}{\partial w} = -2r(n)e(n)x^T(n)$$

The gradient of $J(w)$:

$$g(n) = [\frac{\partial J(n)}{\partial w(n)}]^T = -2r(n)e(n)x(n)$$

Applying steepest descent method:

$$w(n+1) = w(n) - \eta g(n) = w(n) + 2\eta r(n)e(n)x(n)$$

To calculate the optimal parameter w, two formulas must be followed:

$$w(n+1) = w(n) + 2\eta r(n)e(n)x(n)$$

$$e(n) = d(n) - x^T(n)w(n)$$

MATLAB code

Q3:

```
clear all
clc

% Input data
X = [0 0;0 1;1 0;1 1];
y = [0; 0; 0; 1];

for i = 1:4
    if y(i)== 0
        y(i)=-1;
    end
end

% Initial weights
w = rand(2, 1);
b = rand();
y_pred = rand(4,1);

% Learning rate
eta =1;

t=0;

while true
    for i = 1:4
        y_pred(i) = sign(X(i, :) * w + b);
        if (y_pred(i) ~= y(i))
            w = w + eta * (y(i) - y_pred(i)) * X(i, :)';
            b = b + eta * (y(i) - y_pred(i));
        end
    end
        if isequal(y_pred,y)
            break
        elseif t > 1000
            break
        end
    % Trajectories of weights
    t=t+1;
    w_trajectory(t, :) = w';
    b_trajectory(t, :) = b;
end
```

```
% Plot the trajectories of weights
figure;
subplot(2, 2, 1);
plot(w_trajectory(:, 1), '-o');
xlabel('Iteration');
ylabel('w1');
title('Trajectories of weights');

subplot(2, 2, 2);
plot(w_trajectory(:, 2), '-o');
xlabel('Iteration');
ylabel('w2');
title('Trajectories of weights');


subplot(2, 2, 3);
plot(b_trajectory, '-o');
xlabel('Iteration');
ylabel('b');
title('Trajectories of bias');

subplot(2,2,4);
plot(0,0,"o",1,0,"o",0,1,"o",1,1,"o");
hold on;
% xline(-b/w(1,1));
p = 0:0.05:1;
q = -(w(1,1)*p+b)/w(2,1);
plot(p,q);
```

Q4:a)

```
clear all
clc

% Define the input-output pairs
x = [0; 0.8; 1.6; 3; 4; 5];
y = [0.5; 1; 4; 5; 6; 9];

% Add a column of ones to x to represent the bias term
X = [ones(length(x),1), x];

A = inv(X' * X);

determinant = det(A);
```

```
if determinant == 0
    disp('Matrix is not invertible.');
else
    disp('Matrix is invertible.');

    % Solve for w using the LLS method

    w = inv(X' * X) * X' * y;

end

% Extract the intercept term (b)
b = w(1);

% Extract the slope (w)
k = w(2);

% Print the results
fprintf('Intercept (b): %f\n', b);
fprintf('Slope (w): %f\n', k);

figure;
for i=1:length(x)
    plot(x(i),y(i),"o");
    hold on;
end

q = x(1):0.05:x(length(x));
p = k*q+b;
plot(q,p);
hold on;
```

b)

```
clear all
clc

% Define the input-output pairs
x = [0; 0.8; 1.6; 3; 4; 5];
y = [0.5; 1; 4; 5; 6; 9];


% Initialize weights and biases
```

```
w = rand();
b = rand();

% Set learning rate and number of epochs
lr = 0.001;
epochs = 10000;

% Initialize arrays to store weights and biases
w_arr = zeros(1, epochs);
b_arr = zeros(1, epochs);

% Loop for number of epochs
for i = 1:epochs
    % Loop for each data point
    for j = 1:length(x)
        % Calculate predicted output
        y_pred = w * x(j) + b;

        % Update weights and biases
        w = w + lr * (y(j) - y_pred) * x(j);
        b = b + lr * (y(j) - y_pred);

        % Store updated weights and biases
        w_arr(i) = w;
        b_arr(i) = b;
    end
end


% Plot the fitting result
figure;
subplot(3,1,1);
plot(x, y, 'o', 'MarkerFaceColor', 'b', 'MarkerEdgeColor', 'k');
hold on;
x_fit = 0:0.1:5;
y_fit = w * x_fit + b;
plot(x_fit, y_fit, '-r', 'LineWidth', 2);
xlabel('Input (x)');
ylabel('Output (y)');
legend('Data', 'Fitted Line');

subplot(3,1,2);
plot(1:epochs, w_arr, '-b', 'LineWidth', 2);
xlabel('Epoch');
```

```
ylabel('Weight (w)');
subplot(3,1,3);
plot(1:epochs, b_arr, '-g', 'LineWidth', 2);
xlabel('Epoch');
ylabel('Bias (b)');
```