

Homework Set 2

- a) Create a set of 2D data ($d = 2$, features: $\mathbf{x} = (x_1, x_2)$)
- b) Randomly create a set of 20 data points ($N = 20$) such that for each point $\mathbf{x} = (x_1, x_2)$, the coordinates x_1, x_2 be integers. x_1, x_2 are to be limited to the $[-30, +30]$ range and **uncorrelated**.
- c) Choose the line $x_1 + 2x_2 - 1.1 = 0$ as your target function, where the points on one side of the line map to $y = +1$ ($f = x_1 + 2x_2 - 1.1 > 0$) and the other points map to $y = -1$ ($f = x_1 + 2x_2 - 1.1 < 0$). Now, you have a set of 20 data points (\mathbf{x}, y) as your separable data points.
- d) Plot the points on the 2D plane labeling them with “+” or “-” or Red and Blue.
- e) Implement and run the simple perceptron algorithm. **You must write the perceptron code from scratch as opposed to using the code in software packages.** Make sure to include declarations next to code lines for ease of readability. How many iterations does it take to arrive at the solution boundary (estimated target function)? Plot 4 iterations including the final one showing the boundary line at each iteration. Draw function f line on the last plot and explain why they are different.

Assignment 2: Perceptron

Student: Francisco Mata

Table of Contents

Part a)	1
Part b)	1
Part c)	2
Part d)	4
Part e)	5
Output Section	9

Part a)

INSTRUCTIONS

Create a set of 2D data ($d=2$, features: $x=(x_1, x_2)$)
making some arbitrary vectors to use on the 2d

Implementation

Two row vectors were created for x_1 and x_2 with 20 columns each

format `compact`

```
%initilizing and allocating x1 and x2 vectors
disp('Initializing and allocating x1 and x2 vectors');
x1=zeros(1,20);
x2=zeros(1,20);
disp(x1);
disp(x2);
```

Initializing and allocating x1 and x2 vectors

Columns 1 through 13

```
0    0    0    0    0    0    0    0    0    0    0    0
0    0
```

Columns 14 through 20

```
0    0    0    0    0    0    0
```

Columns 1 through 13

```
0    0    0    0    0    0    0    0    0    0    0    0
0    0
```

Columns 14 through 20

```
0    0    0    0    0    0    0
```

Part b)

INSTRUCTIONS

Randomly create a set of 20 data points ($N=20$) such that
for each point $x=(x_1, x_2)$, the coordinates x_1, x_2 be integers.
 x_1, x_2 are to be limited to the $[-30, +30]$ range and uncorrelated.

Implementation

A matrix of two rows and 20 columns with random variables was created. The first row vector was set to x1 and the second row to x2.

```
X = randi([-30,30], [2,20]); % array of 2x20 random values within -30
and 30 range

x1 = X(1,:); % making the first row of X equal to x1
x2 = X(2,:); % making the first row of X equal to x1
```

Part c)

INSTRUCTIONS

Choose the line $x_1 + 2x_2 - 1.1 = 0$ as your target function, where the points on one side of the line map to $y=+1$ ($f=x_1+2x_2-1.1>0$) and the other points map to $y=-1$ ($f=x_1+2x_2-1.1<0$). Now, you have a set of 20 data points (x,y) as your separable data points.

Implementation

A for loop grabs every random point's x1 and x2 from the random vector and puts those values in the target line function. If the output is positive x1 and x2 are store in a vector varialbe called "y_plus_x1" and "y_plus_x2". If the output is negative then x1 and x2 are store in y_minus_x1 and y_minus_x2. Figure 1 shows the first plot with the target line and the 20 random points.

```
% line below is to make the target line function the same range as
% x1 maximum and minimum values
x1_sort = sort(x1, 'ascend');

% The variables and vectors declared below were used for the first
plot
% to identified the points above and below the target line
jp=1;
jm=1;
y_plus_x1=zeros(1,2);
y_plus_x2=zeros(1,2);
y_minus_x1=zeros(1,2);
y_minus_x2=zeros(1,2);
y = zeros(1,2);

% the for loop below checks for all random points (x1,x2) and
allocates
% them on 2 different vectors; one for the points above the line and
% another one for the ones below the line
for i=1:20
    f = x1(i) + 2*x2(i) - 1.1; % target function
    if f > 0
        y_plus_x1(jp) = x1(i);
        y_plus_x2(jp) = x2(i);
```

```

        y(i) = 1;
        jp= jp+1;
    else
        y_minus_x1(jm)=x1(i);
        y_minus_x2(jm)=x2(i);
        y(i)=-1;
        jm = jm+1;
    end
end
%Vlength=length(x1)

% the lines below displays the y output, -1 or 1
disp('y output of +1(above the line) or -1(below the line)');
disp(y);

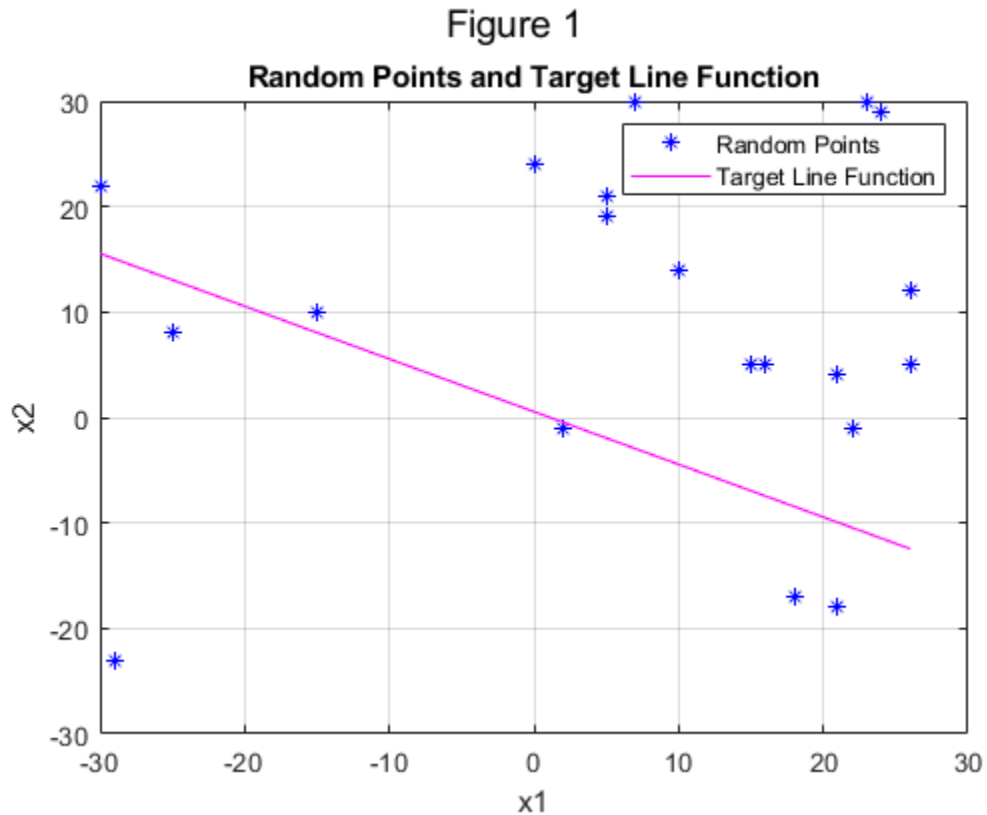
% the lines below display thr random points above the line and below
disp('Final Vector Values From Target Function');
disp('Random Points Above Line');
disp(y_plus_x1);
disp(y_plus_x2);
disp('Random Point Below Line');
disp(y_minus_x1);
disp(y_minus_x2);
%disp(jp);
%disp(jm);
x2_lf = -(x1_sort./2) +0.55; % target line function  $x_1 + 2x_2 - 1.1 = 0$ 

clf.figure(1))
figure(1)
plot(x1, x2, 'b*'); % plotting the random values of x1 and x2
title('Random Points and Target Line Function');
suptitle('Figure 1')
xlabel('x1');
ylabel('x2');
hold on
plot(x1_sort,x2_lf, 'm' ); %plotting the line function, the x-value is
the
%range of the maximun and minimum of x1 row
legend('Random Points', 'Target Line Function')
grid on
hold off

y output of +1(above the line) or -1(below the line
Columns 1 through 13
    1    1    1    1    1    1   -1   -1    1    1    1
1   -1
Columns 14 through 20
    1    1    1    1   -1    1   -1
Final Vector Values From Target Function
Random Points Above Line
Columns 1 through 13
    21    26     5    23   -30     7     0     5    15   -15    10
24    16
Columns 14 through 15

```

```
26    22
Columns 1 through 13
4    12    19    30    22    30    24    21    5    10    14
29    5
Columns 14 through 15
5    -1
Random Point Below Line
2    18    -25    -29    21
-1    -17    8    -23    -18
```



Part d)

INSTRUCTIONS

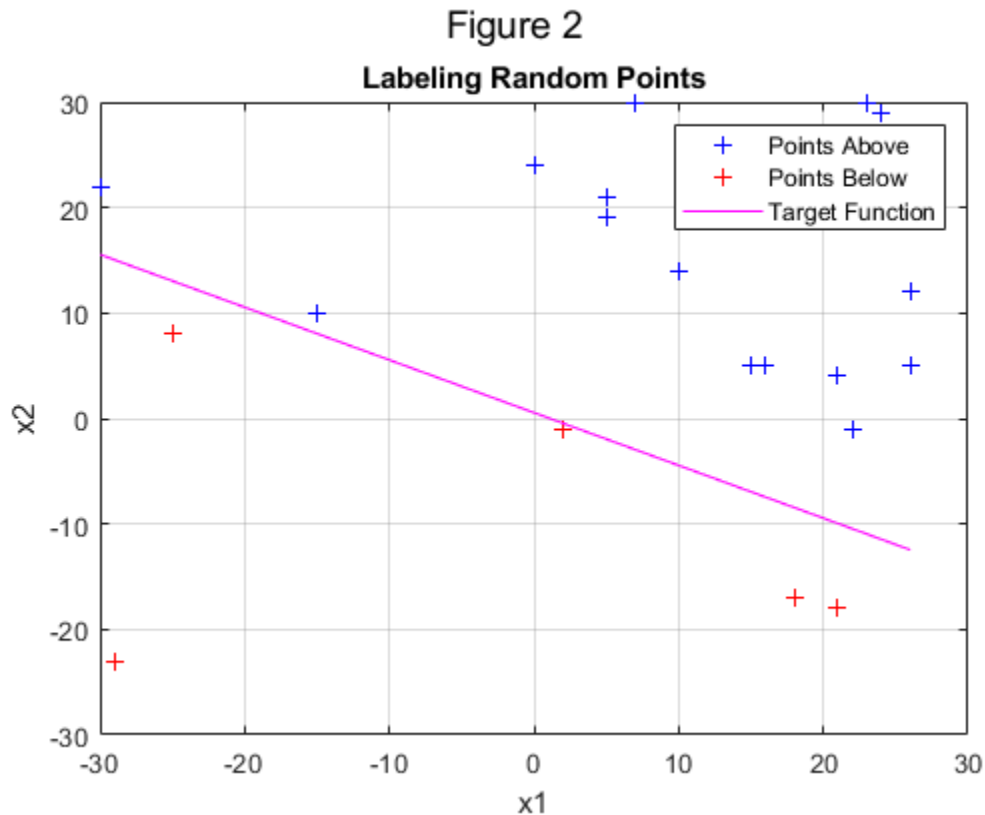
Plot the points on the 2D plane labeling them with "+" or "-" or Red and Blue.

Implementation

The figure below plots the random points above the target line with a "+" in blue for points above the target line and a "+" in red for the points below the line

```
clf(ffigure(2))
figure(2)
plot(y_plus_x1, y_plus_x2, 'b+');
title('Labeling Random Points');
suptitle('Figure 2');
```

```
xlabel('x1');  
ylabel('x2');  
hold on  
plot(y_minus_x1, y_minus_x2, 'r+');  
grid on  
plot(x1_sort, x2_lf, 'm')  
legend('Points Above', 'Points Below', 'Target Function')  
hold off
```



Part e)

INSTRUCNTIONS: Implement and run the simple perceptron algorithm. You must write the perceptron code from scratch as opposed to using the code in software packages. Make sure to include declarations next to code lines for ease of readability. How many iterations does it take to arrive at the solution boundary (estimated target function)? Plot 4 iterations including the final one showing the boundary line at each iteration. Draw function f line on the last plot and explain why they are different.

Implementation: A fixed starting line was chosen with given (w) values which are $w(x)_{old}$. Then it goes into a while loop with if statements and a couple for loops that take the first misclassified point, takes the coordinates and produces new weights (w). Once it implements the new set of weights a new function line is created, and then it goes through the loop again to recheck how many and which are the misclassified points. There is also a variable counter that counts the iterations. Figure 3 shows a plot with the path and number of iterations the line function is taking. Figure 4 shows the first 3 iterations and the last iteration along with the target line in red and the random points above and below. The last iteration is in magenta color and it should be close to the target line function. The starting and first iteration is in blue, the second in green, third in cyan and the last one in magenta.

The total number of iterations is shown in figure 3 and at the bottom of the outputs section.

The last function line and the target line are not the same for several reasons. First, there are infinite number of line functions that can separate the data linearly using 20 random points. Second, depending on how clutter the data is around the target function, the learning line will get closer towards the target line but not exact unless the number of data points keeps increasing.

```
% starting line function is  $0.5x_2 + x_1 + 0 = 0$ , which for this
% case is fixed and is the first iteration since it checks for
% misclassified points.
% below are the starting weight (w) values the x range of the
% training line and the learning rate
n = 0.005; % learning rate
w0_old = 0; % starting weight 0
w1_old = 1; % starting weight 1
w2_old = 0.5; % starting weight 2
x1_new = -30:30; % x1 range of values used on plots

%starting line function  $0.5x_2 + x_1 + 0 = 0$ 

new_y_plus_x1=zeros(1,2); % stores x1 points above the learning line
new_y_plus_x2=zeros(1,2); % stores x2 points above the learning line
new_y_minus_x1=zeros(1,2); % stores x1 points below the learning line
new_y_minus_x2=zeros(1,2); % stores x2 points below the learning line
new_y = zeros(1,20); % stores new y output, -1 or 1
misclassified_x1 = zeros(1,2); % holds x1 misclassified values
misclassified_x2 = zeros(1,2); % holds x2 misclassified values

miss = 1; % counts misclassified point each iteration
iterations = 0; %initializing iteration counter

clf(figure(3)) % clears the figure 3 each time the code is run
clf(figure(4)) % clears the figure 4 each time the code is run

while miss ~= 0 % while the counter,"miss", that holds the number of
    % misclassified points is not equal to zero

    % incrementing variables used with vectors to store x1 and x2 points
    % above and below the learning line
    new_jp=1;
    new_jm=1;

    % statement below decides if between the starting weight values
    % given above or the new weight values depending on the number
    % of iterations
    if iterations == 0
        w0 = w0_old;
        w1 = w1_old;
        w2 = w2_old;
    else
        w0 = w0_new;
        w1 = w1_new;
        w2 = w2_new;
```

```
end

% the for loop below takes the current weight (w) values and puts
% them with the line function and rechecks for new misclassified
% values
for i=1:20
    f2 = w1*x1(i) + w2*x2(i) + w0;
    if ((f2 > 0) && (new_jp < jp))
        new_y_plus_x1(new_jp) = x1(i);
        new_y_plus_x2(new_jp) = x2(i);
        new_y(i) = 1;
        %disp(new_y);
        new_jp = new_jp + 1;
    elseif ((f2 <= 0) && (new_jm < jm))
        new_y_minus_x1(new_jm)=x1(i);
        new_y_minus_x2(new_jm)=x2(i);
        new_y(i)=-1;
        %disp(new_y);
        new_jm = new_jm + 1;
    end
end

miss=0;
y_should_be = zeros(1:2); % initializing vector for the output
                        % value the misclassified point
                        % should be

% the loop below checks for the target line y output of -1 and 1
% and compares it with each line iteration to see if there is any
% misclassified points. If there are any misclassified points
% then the variable "miss" will count up, however the "miss"
% number value should decrease per iteration
for i = 1:20
    if new_y(i) ~= y(i)
        miss = miss + 1;
        misclassified_x1(miss) = x1(i);
        misclassified_x2(miss) = x2(i);
        y_should_be(miss) = y(i);
        % disp("misclassified points");
        % disp(misclassified_x1);
        % disp(misclassified_x2);
        % disp(miss);

    end
end

iterations = iterations + 1; % iteration counter

w0_new = w0 + n*y_should_be(1)*1; % creates new weight, w0
w1_new = w1 + n*y_should_be(1)*misclassified_x1(1); % ", w1
w2_new = w2 + n*y_should_be(1)*misclassified_x2(1); % ", w2
```



```
% f_line variable below receives a returning line from
% function "fun", which takes weight values and x range
f_line = fun(w0, w1, w2, x1_new);

% figure 3 below plots all the iterations
figure(3)
plot(x1_new, f_line, 'b')
title('Total Number of Iterations')
suptitle('Figure 3');
% subtitle('3');
text = ['Number of Iterations = ', num2str(iterations)];
legend(text)
hold on
xlabel('x1');
ylabel('x2');

grid on

% figure 4 below plots 4 iterations and the target line. Blue,
% green, and cyan are the first 3 iterations and magenta is
% the last iteration. The target line function is in red
figure(4)
plot(y_plus_x1, y_plus_x2, 'b+');
title('4 Iterations with Target Function Looping in Real Time')
suptitle('Figure 4');
xlabel('x1');
ylabel('x2');
hold on
plot(y_minus_x1, y_minus_x2, 'r+');
grid on
hold on
plot(x1_sort, x2_lf, 'r') % plotting target line
hold on

if iterations == 1
    first_iteration = f_line;
    plot(x1_new, f_line, 'b')
    %first_iteration = f_line;
    hold on
elseif iterations == 2
    second_iteration = f_line;
    plot(x1_new, f_line, 'g')
    hold on
elseif iterations == 3
    third_iteration = f_line;
    plot(x1_new, f_line, 'c')
    hold on

end

end

plot(x1_new, f_line, 'm') % plots the last iteration line for
hold off                % figure 4
```

```
% the lines below was used for testing the misclassified outputs -1
and 1
% disp("new_y output of +1(above the line) or -1(below the line)");
disp("New_y vector output to be verified with first set of y output");
disp(new_y);

disp("Final Vector Values from Perceptron Algorithm");
disp("Random Points Above Last Iteration Line ");
disp(new_y_plus_x1);
disp(new_y_plus_x2);
disp("Random Point Below Last Iteration Line");
disp(new_y_minus_x1);
disp(new_y_minus_x2);
disp("misclassified points");
disp(misclassified_x1);
disp(misclassified_x2);
disp(miss);
disp("The value the misclassified should be");
disp(y_should_be);

disp("Number of iterations");
disp(iterations);
%disp(first_iteration);

clf(ffigure(5))
figure(5)
plot(x1_new, first_iteration, 'b');
title('4 Iterations with Target Function');
suptitle('Figure 5');
hold on
plot(x1_new, second_iteration, 'g');
plot(x1_new, third_iteration, 'c');
plot(x1_new, f_line, 'm')
plot(x1_sort,x2_lf, 'r')
plot(y_plus_x1, y_plus_x2, 'b+');
plot(y_minus_x1, y_minus_x2, 'r+');
legend('First Iteration', 'Second Iteration', 'Third Iteration', 'Last
Iteration', 'Target Function', 'Points above', 'Points Below')
grid on
hold off
% funtion below returns a line function from a set of weights given
function f1 = fun(w0, w1, w2, x1_new)
    x2_new = ((-w1*x1_new) - w0*1)/w2 ;
    f1 = x2_new;
end
```

Output Section

```
New_y vector output to be verified with first set of y output
Columns 1 through 13
    1    1    1    1    1    1    -1    -1    1    1    1
1   -1
```

```

Columns 14 through 20
    1    1    1    1   -1    1   -1
Final Vector Values from Perceptron Algorithm
Random Points Above Last Iteration Line
Columns 1 through 13
    21    26     5    23   -30     7     0     5    15   -15    10
24     16
Columns 14 through 15
    26    22
Columns 1 through 13
     4    12    19    30    22    30    24    21     5    10    14
29      5
Columns 14 through 15
     5    -1
Random Point Below Last Iteration Line
     2    18   -25   -29    21
    -1   -17     8   -23   -18
misclassified points
     2     2    18   -15    21
    -1    -1   -17    10   -18
     0
The value the misclassified should be
     0     0
Number of iterations
    14

```

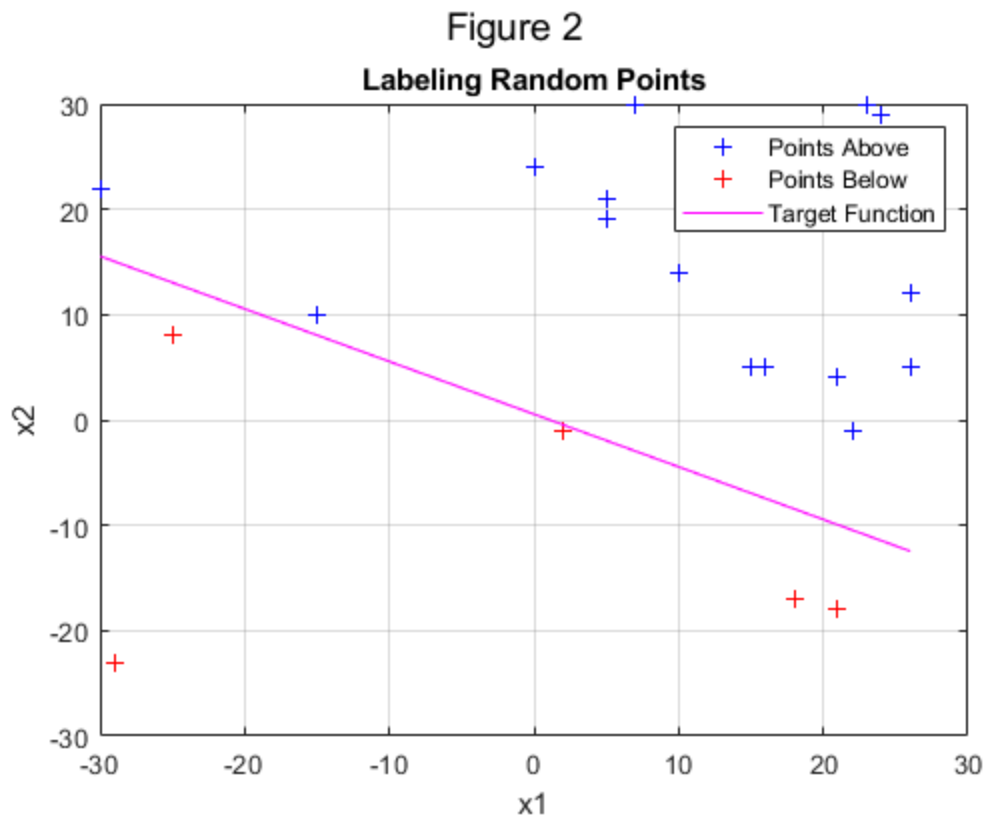


Figure 3

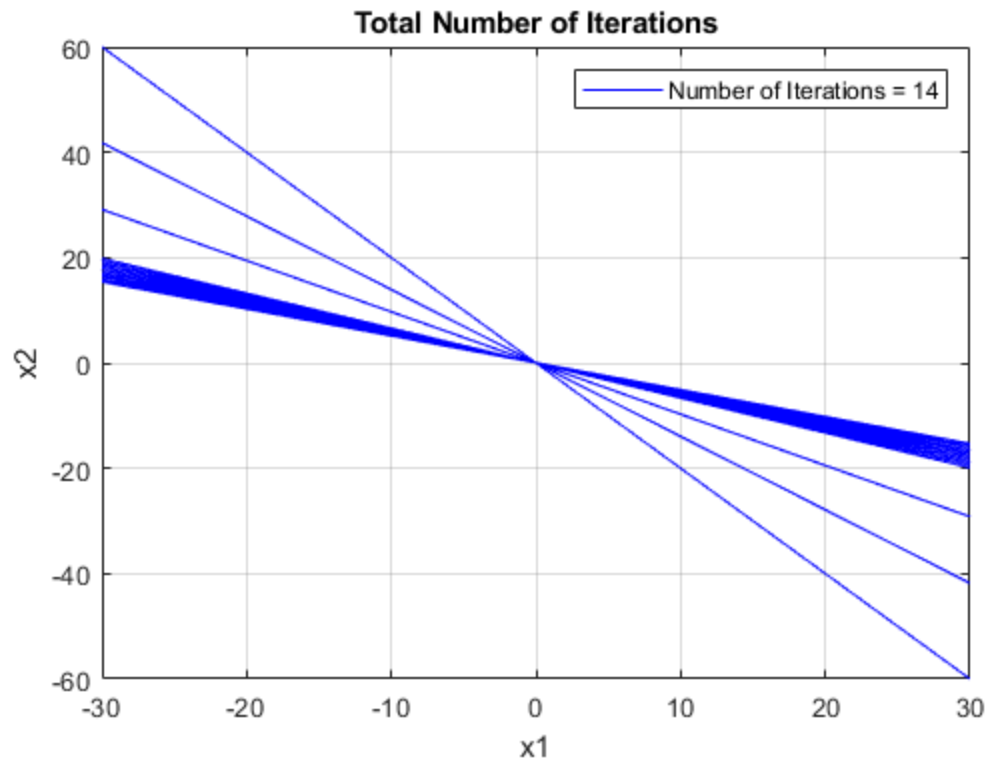
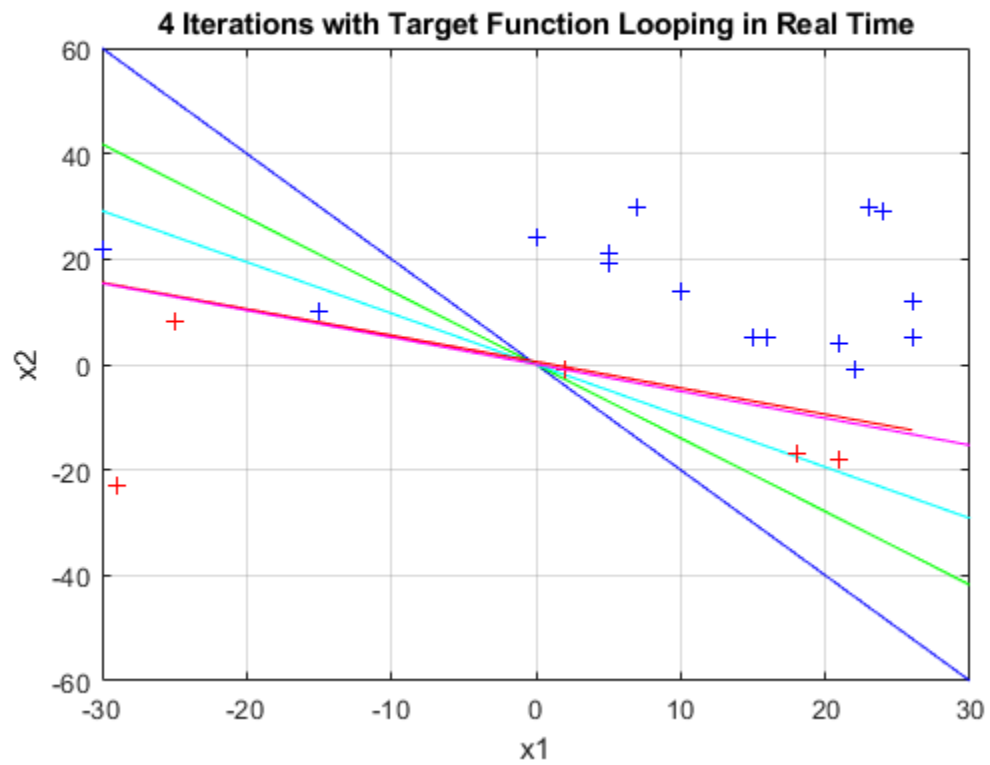
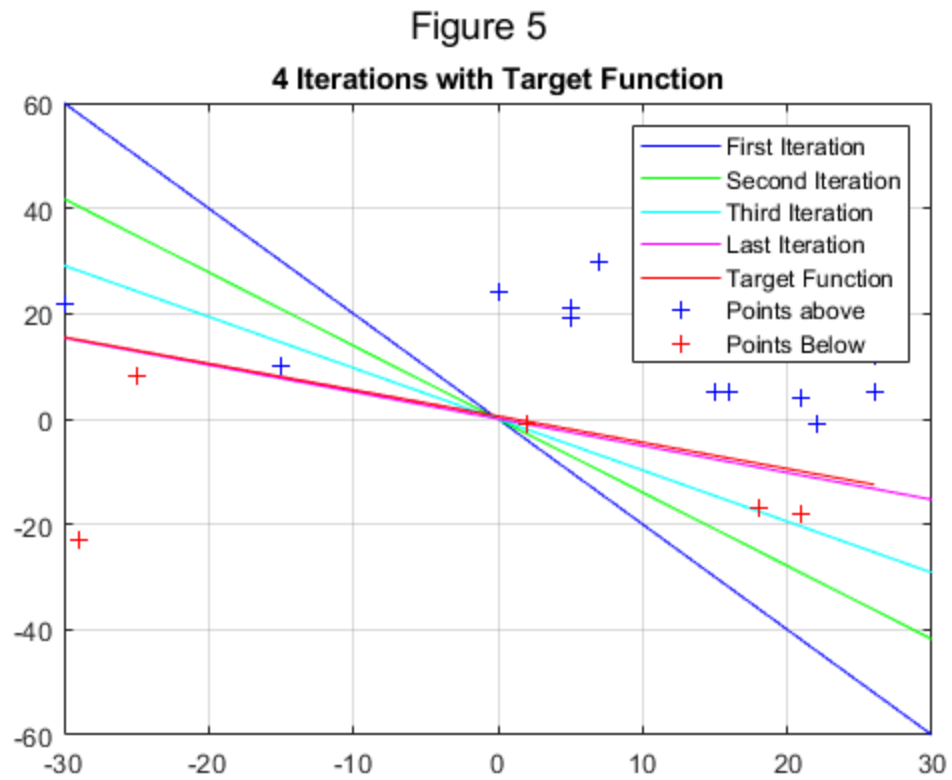


Figure 4





Published with MATLAB® R2019a