Exercise 3

a) Most time it is not a convex function.
b) D = 2 * L
   We assume that there total L hidden layer of this system(M_1 , …. , M_L), and one hidden layer (M_0)
   For each layer of these system , we have known M_0 = d, and M_L = 1, and there are no bias for input layer.

   For each model parameters in Θ (i.e. W_L, $b_{L,}$. W_{L-1}, b_{L-1} ……) is a weight matrix and bias matrix depending we design.

c) If we represent the weight matrix and the bias matrix like (W1, W2, W3…… WL and b1, b2, b3, …, bL), the gradient of $\nabla E(\Theta) = \frac{\partial E}{\partial W1} + \frac{\partial E}{\partial b1} + \frac{\partial E}{\partial W2} + \frac{\partial E}{\partial b2} + \ldots . + \frac{\partial E}{\partial WL} + \frac{\partial E}{\partial bL}$.
d) For example, M = [2,5,9,5,3,1]

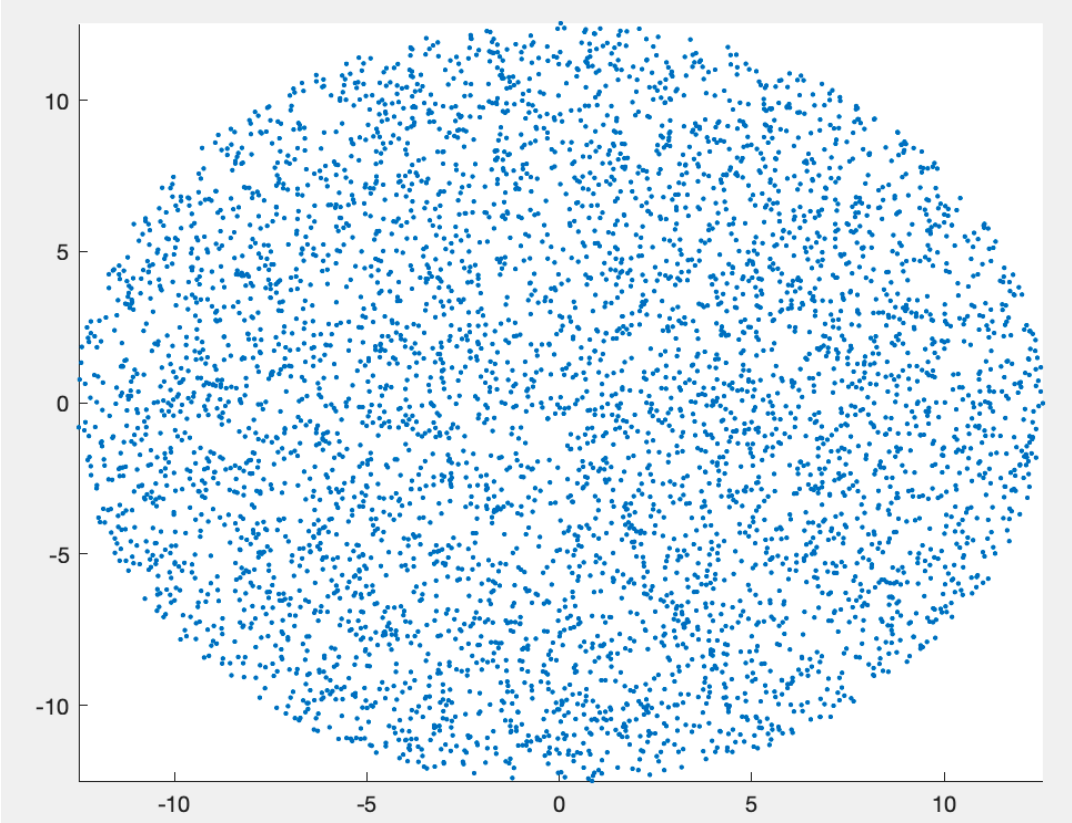
   I calculate the value of ∇E(Θ) using the chain rule, the ∇E(Θ) of M = [2,5,9,5,3,1] system the code shown in below. The Θ known is in my structure, you can find out my structure in my Matlab file.

```
function der_theta = derivative_ann(obj, x, y, output)
    error = output{10} - y;

    output_error_delta = 2 * error; % error / predict
    %output{9}

    z9_delata = output_error_delta * obj.activate_derivative(output{9}); %error / z9


    z8_error = z9_delata * transpose(obj.W5);%error / z8



    z8_b_error = z9_delata;
    z7_delata = z8_error .* obj.activate_derivative(output{7});%error / z7



    z6_error = z7_delata * transpose(obj.W4); %error / z6
    z6_b_error = z7_delata;
    z5_delata = z6_error .* obj.activate_derivative(output{5});%error / z5


    %obj.activate_derivative(output{5})

    z4_error = z5_delata * transpose(obj.W3);%error / z4
    z4_b_error = z5_delata;
    z3_delata = z4_error .* obj.activate_derivative(output{3});%error / z3

    z2_error = z3_delata * transpose(obj.W2);%error / z2
    z2_b_error = z3_delata;
    z1_delata = z2_error .* obj.activate_derivative(output{1});%error / z

    %x_error = dot(z1_delata, transpose(obj.W1));
    x_b_error = z1_delata;
```

```
    der_W1 = transpose(x) * z1_delata ;
    der_b1 = x_b_error;
    der_W2 = transpose(output{2}) * z3_delata ;
    der_b2 = z2_b_error;
    der_W3 = transpose(output{4}) * z5_delata ;
    der_b3 = z4_b_error;
    der_W4 = transpose(output{6}) * z7_delata ;
    der_b4 = z6_b_error;
    der_W5 = transpose(output{8}) * z9_delata ;
    der_b5 = z8_b_error;
    %der_b1 = x_b_error;
    der = [norm(der_W1) norm(der_b1) norm(der_W2) norm(der_b2) norm(der_W3) norm(der_b3) norm(der_W4) norm(der_b4) norm(der_W5) norm(der_b5)];
    der_theta = {der_W5, z8_b_error, der_W4, z6_b_error, der_W3, z4_b_error, der_W2, z2_b_error, der_W1, x_b_error};
    obj.der_total = norm(der);
end
```

e)

Sample scatter



For Gradient Descent Method with a fixed step (0.00001), the plot of $E(\Theta)$ and norm of $\nabla E(\Theta)$ with count number shown below.

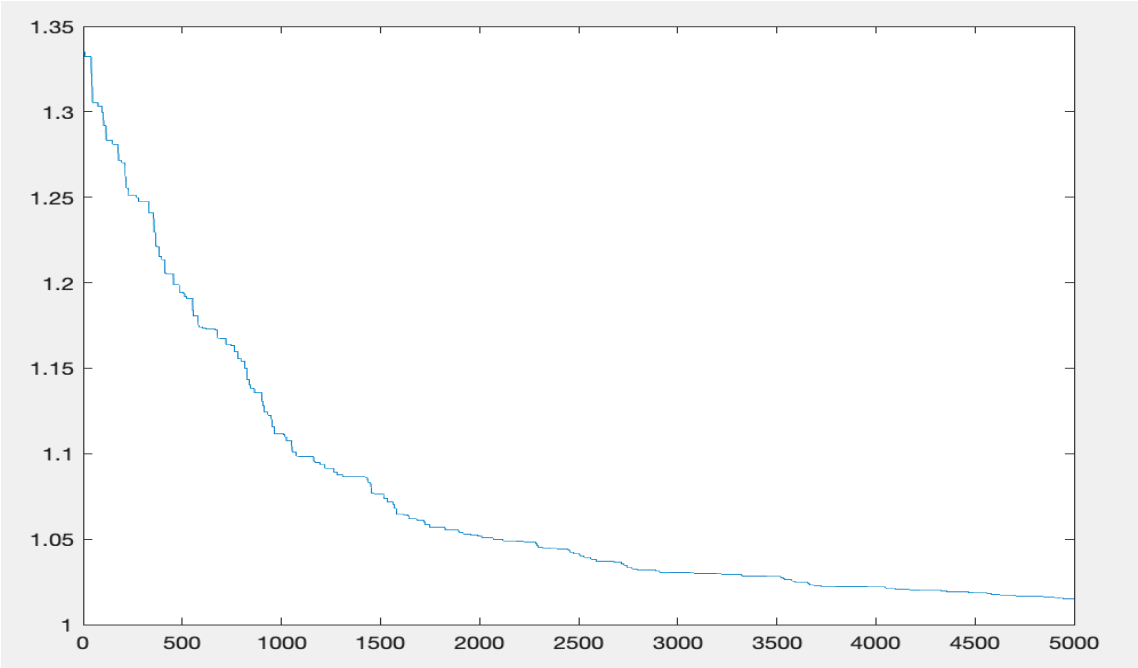


Figure 1: $E(\Theta)$ with n

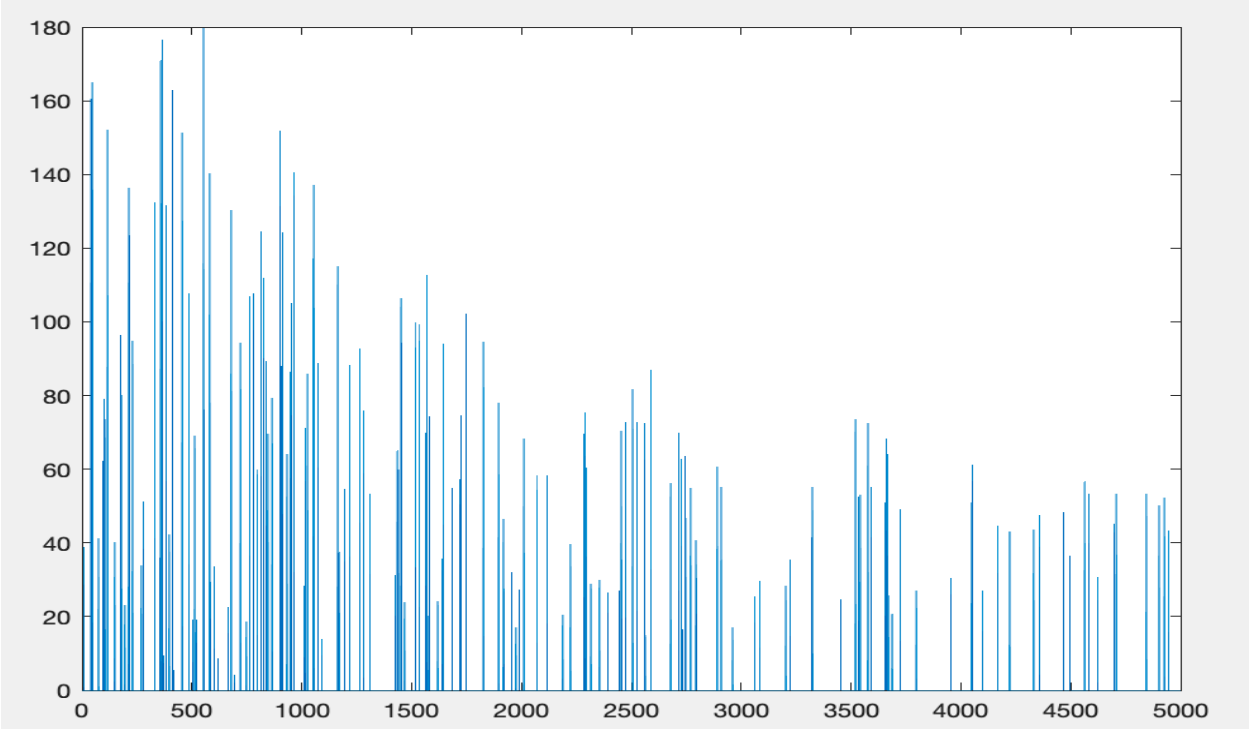


Figure 2: $\nabla E(\Theta)$ with n

For Gradient Descent Method with a backtracking, the plot of $E(\Theta)$ and norm of $\nabla E(\Theta)$ with count number shown below.

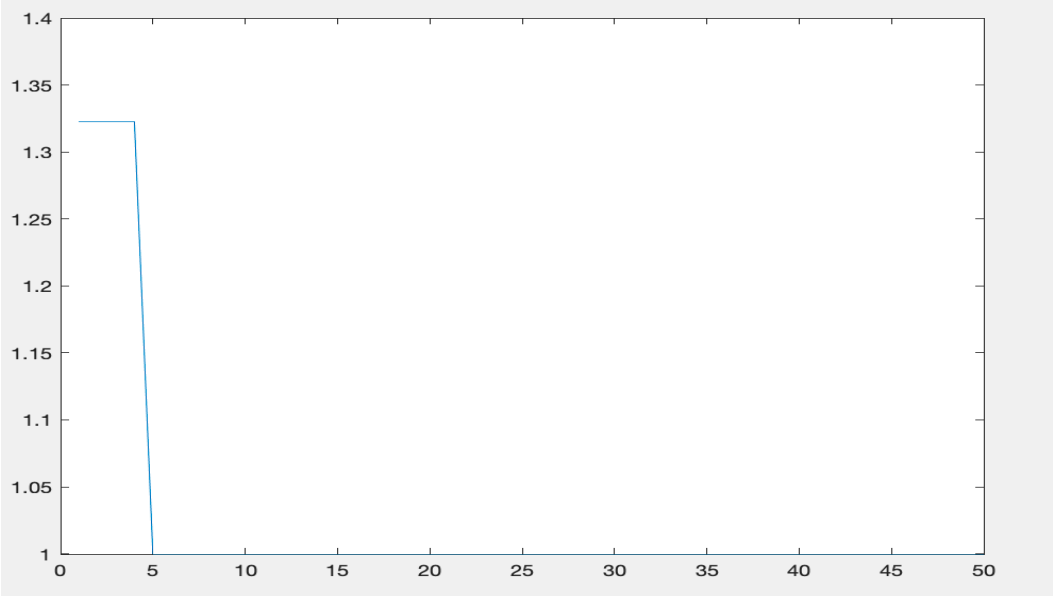


Figure 3: $E(\Theta)$ with n

Because for number 5000, I cannot get a clear picture for this, so I only train 50 times. We can see from Figure 3 that using backtracking to choose step, we can converge more quickly, which is a better result.

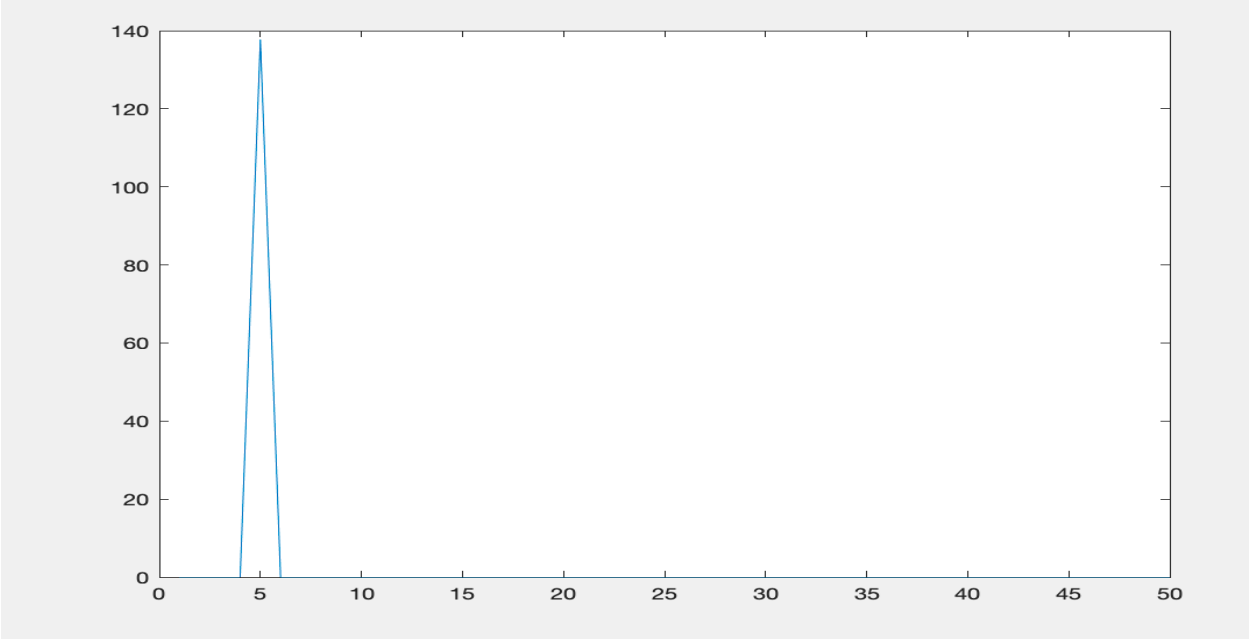


Figure 4: $\nabla E(\Theta)$ with n

For Conjugate Gradient Method with a fixed step(0.00001), the plot of $E(\Theta)$ with count number shown below.

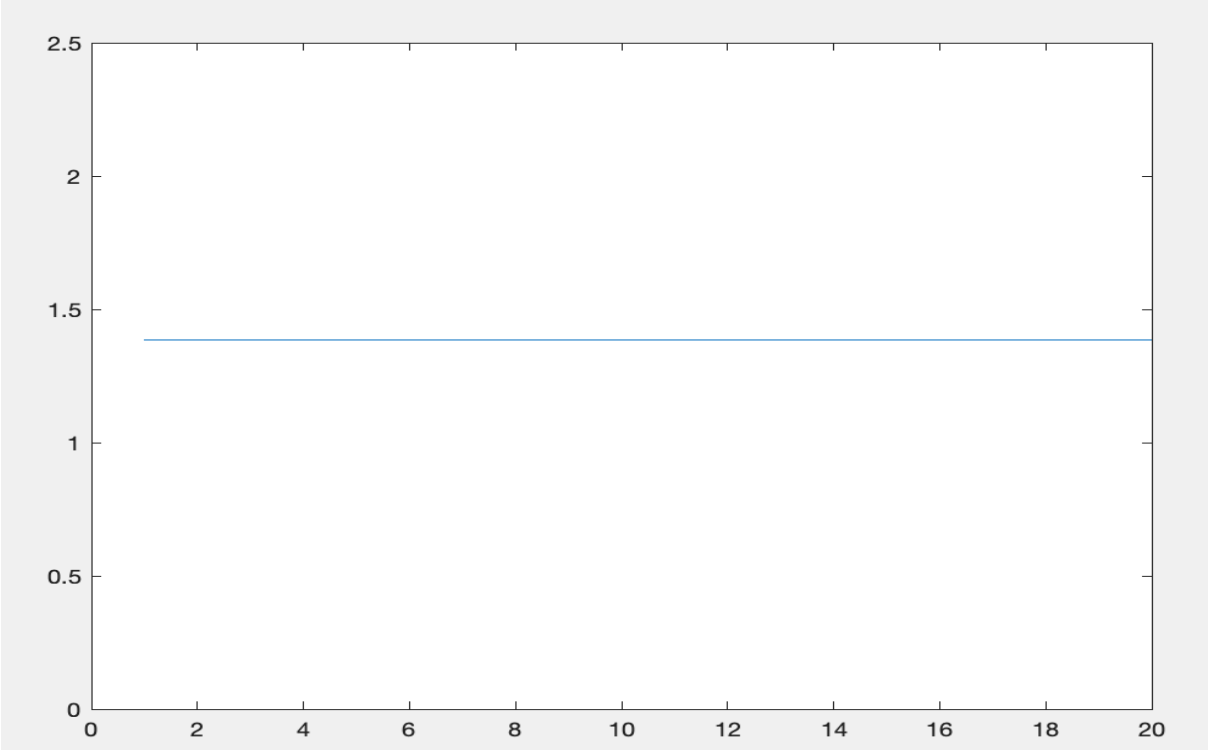


Figure 5: $E(\Theta)$ with n

From Figure 5, we can see that it converges only in one step. From lecture, we know that this method converges only in at most dimension of input. In this question, it is 2. So this result make sense.

For Conjugate Gradient Method with backtracking, the plot of E(Θ) with count number shown below.

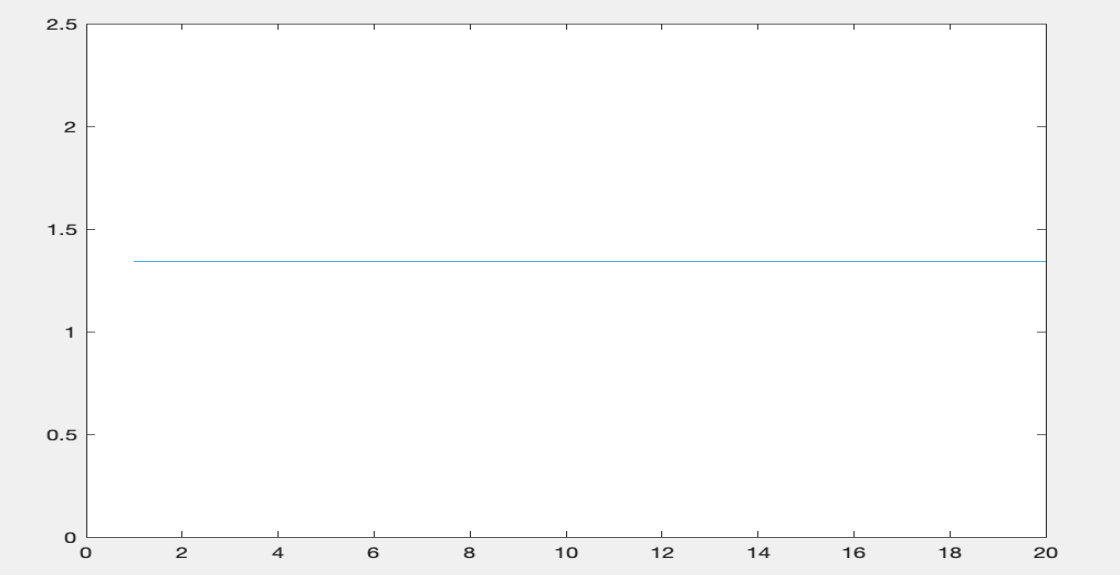


Figure 6: E(Θ) with n

From Figure 6, we can see that it converges in one step. So from figure 6, we can get that using backtracking to choose step does not improve Conjugate Gradient Method with fixed step, because the Conjugate Gradient method is fast enough.

From Figure 1 and Figure 2, we can get that E(Θ) and norm of ∇E(Θ) converges monotonously, with some fluctuation.

f)

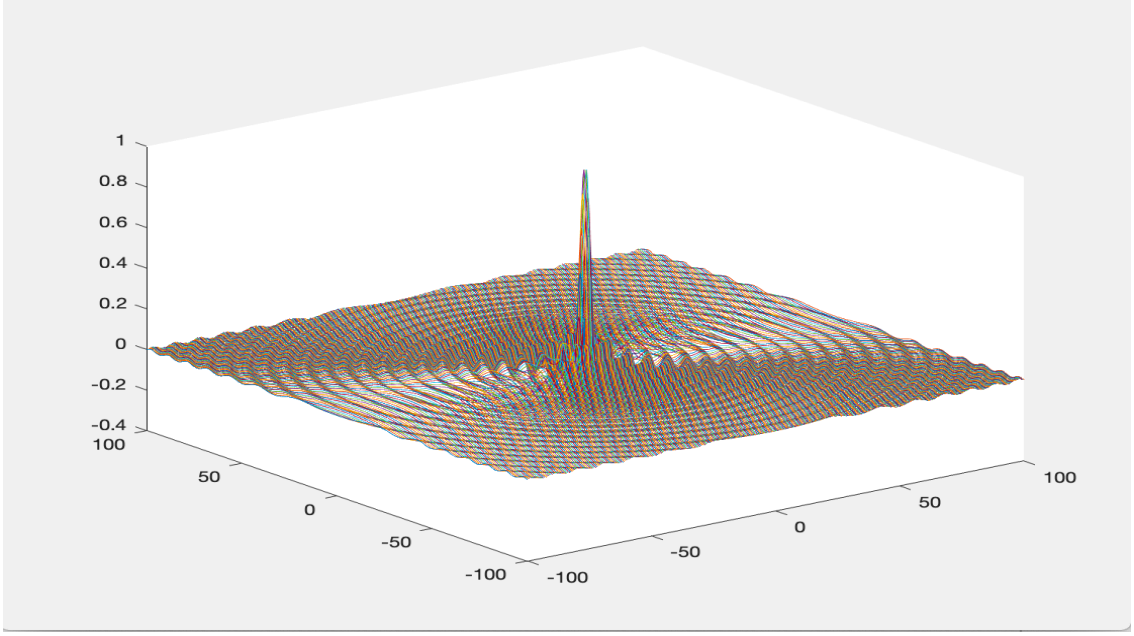


Figure 7: square grid result of f0

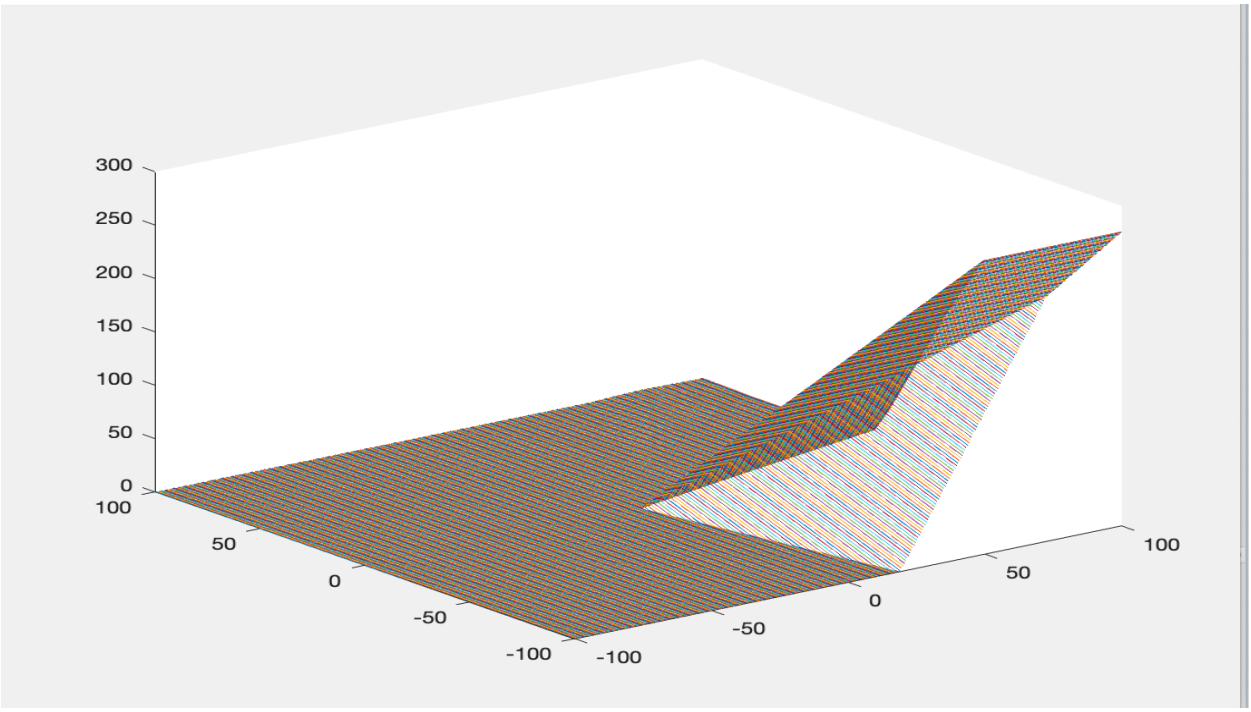


Figure 8: square grid result of ANN

From Figure 7 and Figure 8, we can see that the approximation is different. As for the reason, this is mainly because the activate function of this ANN is relu function, which can note activate the weights and bias in the system when the results of f0 are negative, means that the system cannot approximate the negative result from f0 function, thus causing this total different figure.

g) I would use another activate function, sigmoid function , which could help predict negative result.

Yes, because this norm does not produce negative result, which can be predicted by using relu activate function, thus having a better result.