دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکده مهندسی برق-گروه مهندسی کنترل

تمرین درس محاسبات نرم
در رشتهٔ مهندسی برق گرایش مهندسی کنترل

عنوان

تمرین چهارم: شبکه‌های عصبی

نگارش

محمدجواد احمدی

استاد درس

دکتر مهدی علیاری شوره‌دلی

بهمن‌ماه ۱۴۰۱

فهرست مطالب

فهرست شکل‌ها

پاسخ سوالات

پاسخ سوال ۲.۶

راه‌حل اول

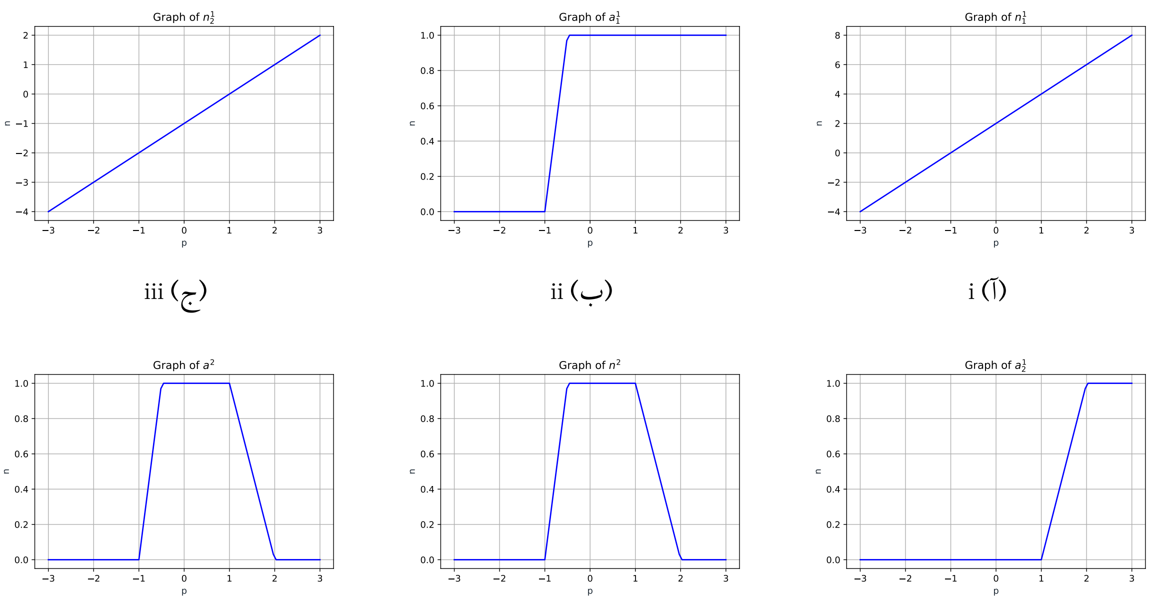
برای این سوال دستورات پایتون زیر را نوشته و استفاده می‌کنیم:

```python
import numpy as np
import matplotlib.pyplot as plt

def hardlim(x):
    return np.where(x >= 0, 1, 0)
def hardlims(x):
    return np.where(x >= 0, 1, -1)
def purelin(x):
  return x
def satlin(x):
    return np.maximum(0, np.minimum(x, 1))
def logsig(x):
    return 1/(1+np.exp(-x))
def htansig(x):
    return (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
def poslin(x):
    return np.maximum(0, x)
def compet(x):
    return (x==np.max(x)).astype(int)

```

```
def plotfunc(k) :
  plt.plot(p, k, '-b')
  plt.xlabel('p', color = '#1C2833' )
  plt.ylabel('n', color = '#1C2833' )
  plt.grid()

weights = [2, 1, 1, -1]
bias = [2, -1, 0]
p = np.linspace(-3, 3, 100) #Plot indicated variable versus p for -3<p<3

#i.
n_1 = p*weights[0] + bias[0]
#ii.
a_1 = satlin(n_1)
#iii.
n_2 = p*weights[1] + bias[1]
#iv.
a_2 = satlin(n_2)
# v.
n_n = a_1*weights[2] + a_2*weights[3] + bias[2]
#vi.
a_a = purelin(n_n)

# plot i .
plt.title('Graph of $n_1^1$')
plotfunc(n_1)
plt.savefig('020601.pdf')
plt.show()
# plot ii.
plt.title('Graph of $a_1^1$')
plotfunc(a_1)
plt.savefig('020602.pdf')
plt.show()
# plot iii.
plt.title('Graph of $n_2^1$')
```

```
plotfunc(n_2)
plt.savefig('020603.pdf')
plt.show()
# plot iv.
plt.title('Graph of $a_2^1$')
plotfunc(a_2)
plt.savefig('020604.pdf')
plt.show()
# plo t v.
plt.title('Graph of $n^2$')
plotfunc(n_n)
plt.savefig('020605.pdf')
plt.show()
# plot vi.
plt.title('Graph of $a^2$')
plotfunc(a_a)
plt.savefig('020606.pdf')
plt.show()
```

نتایج به صورتی است که در شکل ۱ نشان داده شده است.



(آ) i

(ب) ii

(ج) iii

(د) iv

(ه) v

(و) vi

شکل ۱: نتایج سوال ۲.۶.

توضیح پوشهٔ کدها

در پوشهٔ اصلی این سوال (`Q1__02_06`)، فایل‌های پایتون مربوط به این سوال قرار گرفته‌اند. راه‌حل دومی هم برای سوال درنظرگرفته شده است که در همان پوشه موجود است. هم‌چنین فایل اجرای برخط کدها از طریق پیوند گوگل‌کولب در دسترس است.

پاسخ سوال ۳.۶

برای حل این سوال از توابع و دستورات آورده‌شده در کدهای `transfer_functions.py`، `Perceptron.py`، `HammingNetwork.py` و `HopfieldNetwork.py` استفاده می‌کنیم که به شرح زیر تعریف می‌گردند:

```python
## transfer_functions.py ##

import numpy as np

# Transfer functions as defined by Neural Network Toolbox for MATLAB

def hardlim(n):
    """
    Hard Limit
    """
    if n < 0:
        return 0
    else:
        return 1

def hardlims(n):
    """
    Symmetrical Hard Limit
    """
    if n < 0:
        return -1
    else:
        return 1
```

```python

def purelin(n):
    """
    Linear
    """
    return n

def satlin(n):
    """
    Saturing Linear
    """
    if n < 0:
        return 0
    elif n > 1:
        return 1
    else:
        return n

def satlins(n):
    """
    Symmetric Saturating Linear
    """
    if n < -1:
        return -1
    elif n > 1:
        return 1
    else:
        return n

def logsig(n):
    """
    Log-Sigmoid
    """
    return 1/(1+np.exp(-1*n))

```

```python
def tansig(n):
    """
    Hyperbolic Tangent Sigmoid
    """
    return (np.exp(n) - np.exp(-1*n))/(np.exp(n) + np.exp(-1*n))

def poslin(n):
    """
    Positive Linear
    """
    if n < 0:
        return 0
    else:
        return n

def compet(n):
    """
    Competitive
    """
    return (n > 0).astype(np.int)

## Perceptron.py ##

import numpy as np

from transfer_functions import *

class Perceptron(object):

    def __init__(self, W, b, transfer_function=hardlims):
        self.Weights = W
        self.bias = b
        self.transfer_function = np.vectorize(transfer_function)
    def classify(self, prototype):
        net_input = self.Weights.dot(prototype) + self.bias
```

```python
        return self.transfer_function(net_input)


if __name__ == "__main__":
    # prototype = [shape, texture, weight] as a column vector
    orange_prototype = np.array([1, -1, -1]).reshape((3, 1))
    apple_prototype = np.array([1, 1, -1]).reshape((3, 1))

    # Weight matrix and bias determined by decision boundary.
    decision_boundary = (orange_prototype != apple_prototype).astype(np.int).reshape((1, len(orange_prototype)))
    print(decision_boundary)
    bias = 0

    fruit_perceptron = Perceptron(W=decision_boundary, b=bias)

    test_prototype = np.array([-1, -1, -1]).reshape((3, 1))
    print(fruit_perceptron.classify(test_prototype))

## HammingNetwork.py ##

import numpy as np

from transfer_functions import *

class HammingNetwork(object):

    def __init__(self, prototypes):
        self.feedForwardLayer = self.FeedFowardLayer(W=prototypes)
        self.recurrentLayer = self.RecurrentLayer()

    def classify(self, obj):
        a1 = self.feedForwardLayer.propagate(obj=obj)
        recurrent_result = self.recurrentLayer.propagate(initial_a=a1)
        return compet(recurrent_result)
```

```python

class FeedFowardLayer(object):
    def __init__(self, W, transfer_function = purelin):
        self.Weights = W
        self.bias = np.array(self.Weights.shape[0]).repeat(self.Weights.
shape[0], axis=0).reshape((self.Weights.shape[0], 1))
        self.transfer_function = np.vectorize(transfer_function, otypes=[np
.float])
    def propagate(self, obj):
        return self.transfer_function(self.Weights.dot(obj) + self.bias)

class RecurrentLayer(object):
    def __init__(self, W = None, transfer_function = poslin):
        if W is None:
            self.Weights = W
        else:
            self.Weights = None
        self.transfer_function = np.vectorize(transfer_function, otypes=[np
.float])

    def propagate(self, initial_a):
        if self.Weights is None:
            s = initial_a.shape[0]
            epsilon = 1 / (s - 1)
            epsilon -= 0.01
            epsilon *= -1
            self.Weights = np.ones((s, s))
            for i in range(s):
                for j in range(s):
                    if i != j:
                        self.Weights[i][j] = epsilon

        a2 = self.transfer_function(self.Weights.dot(initial_a))

        while True:
```

```
                a3 = self.transfer_function(self.Weights.dot(a2))
                if a2.all() != a3.all():
                    a2 = a3
                else:
                    return a3


if __name__ == "__main__":
    # prototype = [shape, texture, weight] as a column vector
    orange_prototype = np.array([1, -1, -1]).reshape((3, 1))
    apple_prototype = np.array([1, 1, -1]).reshape((3, 1))
    prototypes = np.array([orange_prototype.T[0], apple_prototype.T[0]])

    test_fruit = np.array([-1, -1, -1]).reshape((3, 1))

    hammingFruitClassifier = HammingNetwork(prototypes=prototypes)
    print(hammingFruitClassifier.classify(obj=test_fruit))

## HopfieldNetwork.py ##

import numpy as np

from transfer_functions import *

class HopfieldNetwork(object):

    def __init__(self, weights = np.array([[0.2, 0, 0], [0, 1.2, 0], [0, 0,
    0.2]]), transfer_function = satlins, bias=np.array([0.9, 0, -0.9])):
        self.Weights = weights
        self.bias = bias.reshape((weights.shape[0], 1))
        self.transfer_function = np.vectorize(transfer_function, otypes=[np.
    float])
        self.activations = list([])

    def classify(self, a0):
```

```python
        self.activations.append(a0)
        a1 = self.transfer_function(self.Weights.dot(a0) + self.bias)
        self.activations.append(a1)
        if np.array_equal(a0, a1):
            return a1
        else:
            while True:
                an = self.transfer_function(self.Weights.dot(a1) + self.bias)
                self.activations.append(an)
                if not np.array_equal(an, a1):
                    a1 = an
                else:
                    return an

if __name__ == "__main__":
    test_obj = np.array([-1, -1, -1]).reshape((3, 1))

    test_hopfield = HopfieldNetwork()
    print(test_hopfield.classify(test_obj))
```

در ادامه دستوراتی مطابق زیر را اجرا می‌کنیم:

```python
import numpy as np
import matplotlib.pyplot as plt

from Perceptron import Perceptron
from HammingNetwork import HammingNetwork
from HopfieldNetwork import HopfieldNetwork

if __name__ == "__main__":
    print("We have the following two prototype vectors:",
          "\n\tp1 = [-1, 1].T    p2 = [1,1].T")
    p1 = np.array([-1, 1]).reshape((2, 1))
    p2 = np.array([1, 1]).reshape((2, 1))
    prototypes = np.array([p1.T[0], p2.T[0]])

```

```
x = np.linspace(0, 0, 100)
y = np.linspace(0, 2, 100)

print("\ni.    Find and sketch a decision boundary for a perceptron network that will recognize these two vectors.")
plt.scatter(prototypes[0], prototypes[1])
plt.plot(x, y, color='black')
plt.grid(True)
plt.show()

print("\nii.   Find weights and bias that will produce the decision boundary you found in part i.")
Weights = np.array([1, 0])
bias = 0

e3_6_perceptron = Perceptron(W = Weights, b = bias)

print("p1", p1, "classification:", e3_6_perceptron.classify(p1))
print("p2", p2, "classification:", e3_6_perceptron.classify(p2))

print("\niv.   For the vector given below, calculate the net input, n, and the network output, a, for the network you have designed.  Does the network produce a good output? Explain.",
      "\n\tp_test = [0.5, -0.5].T")

p_test = np.array([0.5, -0.5]).reshape((2,1))
print("p_test Classification = ", e3_6_perceptron.classify(p_test))
print("This is a good classification because the test array lands on the right side of the purposed decision boundary and gets classified as such.")

print("\nv.    Design a Hamming network to recognize the two vectors used in part i.")
e3_6_hamming = HammingNetwork(prototypes=prototypes)
print("p1", p1, "classification:", e3_6_hamming.classify(p1))
print("p2", p2, "classification:", e3_6_hamming.classify(p2))
```

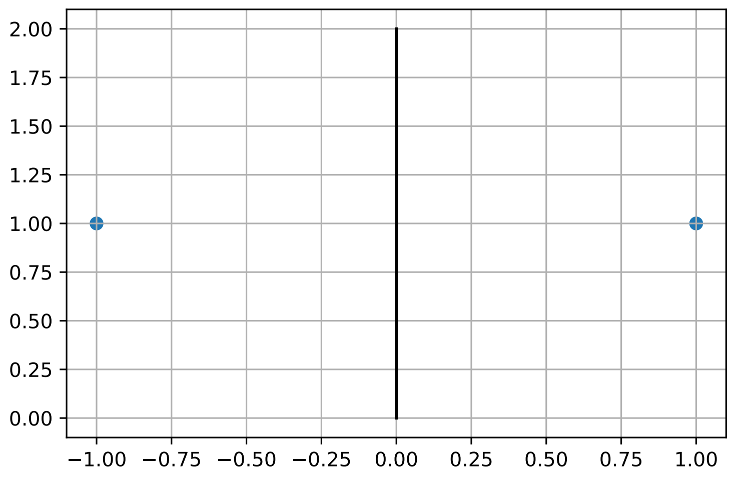
```python

print("\nvi.  Calculate the network output for the Hamming network for the input vector given in part iv.  Does the network produce a good output? Explain.")
print("p_test Classification = ", e3_6_hamming.classify(p_test))
print("This is a correct classification but only by chance as the Hamming Network is only designed to classify prototypes that contain two possible values only.")

print("\nvii.  Design a Hopfield network to recognize the two vectors used in part i.")
Weights = np.array([[1.2, 0], [0, 0.2]])
bias = np.array([0, 0.9])

e3_6_hopfield = HopfieldNetwork(weights=Weights, bias = bias)
print("p1", p1, "classification:", e3_6_hopfield.classify(p1))
print("p2", p2, "classification:", e3_6_hopfield.classify(p2))

print("\nviii. Calculate the network output for the Hopfield network for the input vector given in part iv.  Does the network produce a good output? Explain.")
print("p_test Classification = ", e3_6_hopfield.classify(p_test))
print("Yes, this is the same classification outcome as expected and previously calculated from the other two networks.")
```

نتایج قسمت‌های مختلف سوال به صورتی خواهد بود که در زیر آورده شده است:

```
We have the following two prototype vectors:
  p1 = [-1, 1].T    p2 = [1,1].T

i.    Find and sketch a decision boundary for a perceptron network that will recognize these two vectors.
In-Text...

ii.   Find weights and bias that will produce the decision boundary you found in part i.
```

```
p1 [[-1]
 [ 1]] classification: [-1]
p2 [[1]
 [1]] classification: [1]

iv.   For the vector given below, calculate the net input, n, and the network output, a, for the network you have designed.  Does the network produce a good output? Explain.
  p_test = [0.5, -0.5].T
p_test Classification =  [1]
This is a good classification because the test array lands on the right side of the purposed decision boundary and gets classified as such.

v.    Design a Hamming network to recognize the two vectors used in part i.
p1 [[-1]
 [ 1]] classification: [[1]
 [0]]
p2 [[1]
 [1]] classification: [[0]
 [1]]

vi.   Calculate the network output for the Hamming network for the input vector given in part iv.  Does the network produce a good output? Explain.
p_test Classification =  [[0]
 [1]]
This is a correct classification but only by chance as the Hamming Network is only designed to classify prototypes that contain two possible values only.

vii.  Design a Hopfield network to recognize the two vectors used in part i.
p1 [[-1]
 [ 1]] classification: [[-1.]
 [ 1.]]
p2 [[1]
 [1]] classification: [[1.]
 [1.]]
```

```

viii. Calculate the network output for the Hopfield network for the input
    vector given in part iv.  Does the network produce a good output? Explain.
p_test Classification =  [[1.]
 [1.]]
Yes, this is the same classification outcome as expected and previously
    calculated from the other two networks.
```



شکل ۲: نمودار مربوط به پاسخ قسمت اول سوال ۳.۶.

توضیح پوشهٔ کدها

در پوشهٔ اصلی این سوال (Q2__03_06_07)، فایل‌های پایتون مربوط به این سوال قرار گرفته‌اند. هم‌چنین فایل اجرای برخط کدها از طریق پیوند گوگل‌کولب در دسترس است.

پاسخ سوال ۳.۷

با توجه به توضیحات ارائه‌شده در پاسخ سوال ۳.۶، از دستورات زیر برای پاسخ به تمامی قسمت‌های سوال ۳.۷ استفاده می‌کنیم:

```python
import numpy as np
import matplotlib.pyplot as plt
```

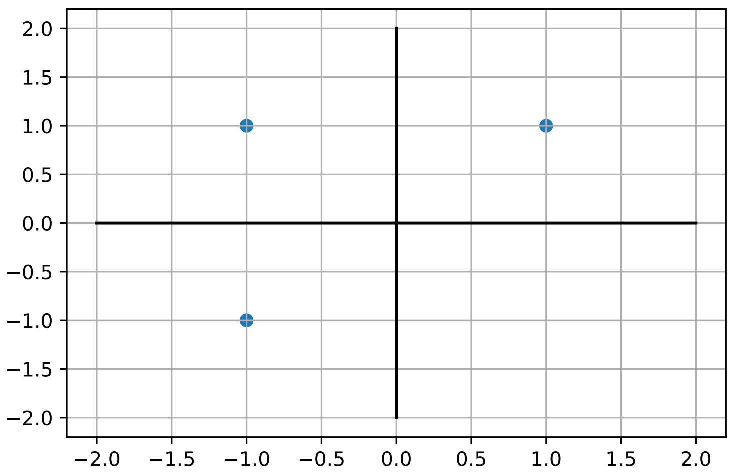
```python
from HammingNetwork import HammingNetwork

if __name__ == "__main__":
    print("We want to design a Hamming network to recognize the following prototype vectors:",
          "\n\tp1 = [1, 1].T", "p2 = [-1, -1].T", "p3 = [-1, 1].T")
    p1 = np.array([1, 1]).reshape((2, 1))
    p2 = np.array([-1,-1]).reshape((2, 1))
    p3 = np.array([-1, 1]).reshape((2, 1))

    prototypes = np.array([p1.T, p2.T, p3.T]).reshape((3, 2))
    print("prototypes", prototypes)

    e3_7_hamming = HammingNetwork(prototypes=prototypes)

    print("p1", p1, "classification:", e3_7_hamming.classify(p1))
    print("p2", p2, "classification:", e3_7_hamming.classify(p2))
    print("p3", p3, "classification:", e3_7_hamming.classify(p3))

    print("\ni.   Find the weight matrices and bias vectors for the Hamming network.",
          "\n\tFeedForwardLayer", "Weights", e3_7_hamming.feedForwardLayer.Weights, "bias", e3_7_hamming.feedForwardLayer.bias,
          "\n\tRecurrentLayer", "Weights", e3_7_hamming.recurrentLayer.Weights, "bias", "There are no biases in the recurrrent layer.")

    print("\niii. Apply the following input vector and calculate the total network response (iterating the second layer to convergence). Explain the meaning of the final network output.",
          "\n\tp_test = [1, 0].T")
    p_test = np.array([1, 0]).reshape((2, 1))
    print("\tClassification:", e3_7_hamming.classify(p_test))

    print("\niv.  Sketch the decision boundaries for this network. Explain how you determined the boundaries.")
```

```
    x = np.linspace(-2, 2, 100)
    y = np.linspace(0, 0, 100)

    x2 = np.linspace(0, 0, 100)
    y2 = np.linspace(-2, 2, 100)

    plt.scatter(prototypes[:, 0], prototypes[:, 1])
    plt.plot(x, y, color='black')
    plt.plot(x2, y2, color='black')
    plt.grid(True)
    plt.savefig('030701.pdf')
    plt.show()
```

نتایج قسمت‌های مختلف سوال به صورتی خواهد بود که در زیر آورده شده است:

```
We want to design a Hamming network to recognize the following prototype
    vectors:
  p1 = [1, 1].T p2 = [-1, -1].T p3 = [-1, 1].T
prototypes [[ 1  1]
 [-1 -1]
 [-1  1]]
p1 [[1]
 [1]] classification: [[1]
 [0]
 [0]]
p2 [[-1]
 [-1]] classification: [[0]
 [1]
 [0]]
p3 [[-1]
 [ 1]] classification: [[0]
 [0]
 [1]]

i.   Find the weight matrices and bias vectors for the Hamming network.
  FeedForwardLayer Weights [[ 1  1]
```

```
[-1 -1]
[-1  1]] bias [[3]
[3]
[3]]
 RecurrentLayer Weights [[ 1.   -0.49 -0.49]
[-0.49  1.   -0.49]
[-0.49 -0.49  1.  ]] bias There are no biases in the recurrrent layer.

iii. Apply the following input vector and calculate the total network response (iterating the second layer to convergence). Explain the meaning of the final network output.
 p_test = [1, 0].T
 Classification: [[1]
[0]
[0]]

iv.  Sketch the decision boundaries for this network. Explain how you determined the boundaries.
In-Text...
```



شکل ۳: نمودار مربوط به پاسخ قسمت آخر سوال ۳.۷.

توضیح پوشهٔ کدها

در پوشهٔ اصلی این سوال (Q2__03_06_07)، فایل‌های پایتون مربوط به این سوال قرار گرفته‌اند.
هم‌چنین فایل اجرای برخط کدها از طریق پیوند گوگل‌کولب در دسترس است.

پاسخ سوال ۴.۱۱

پاسخ سوال ۴.۱۲

برای حل این سوال تابع زیر را تعریف می‌کنیم:

```
function E412(p, t)
error = 1;
W = [rand rand; rand rand];
b = [rand rand]';
count = 0;
while (error ~= 0 & count<=50)
   count = count + 1;
      error = 0;
   for i = 1 : 8
      a = hardlim(W * p(:, i) + b);
      e = t(:, i) - a;
      newW = W + e * p(:, i)';
      W = newW;
      b = b + e;
      if (~isequal(e, [0, 0]'))
         error = error + 1;
      end
   end
end
if error ~=0
      fprintf('Program terminated with no satisfying weight or bias after 50
    iterations.\n');
```

```
        return;
end

W
b

figure
hold on

plot(p(1,1), p(2,1), 'r*');
plot(p(1,2), p(2,2), 'r*');
plot(p(1,3), p(2,3), 'g*');
plot(p(1,4), p(2,4), 'g*');
plot(p(1,5), p(2,5), 'b*');
plot(p(1,6), p(2,6), 'b*');
plot(p(1,7), p(2,7), 'c*');
plot(p(1,8), p(2,8), 'c*');

x = -3:1:3;
y = -(W(1,1) * x + b(1)) / W(1,2);
plot(x, y);
x = -3:1:3;
y = -(W(2,1) * x + b(2)) / W(2,2);
plot(x, y);
```

و در ادامه دستورات زیر را جهت پاسخ به قسمت‌های سوال می‌نویسیم:

```
clc;
clear all;
close all;
disp("4 - 12 - i:")
disp("_____________________________")
p = [1 1 2 2 -1 -2 -1 -2; 1 2 2 0 2 1 -1 -2];
t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
hold on
plot(p(1,1), p(2,1), 'R*');
```

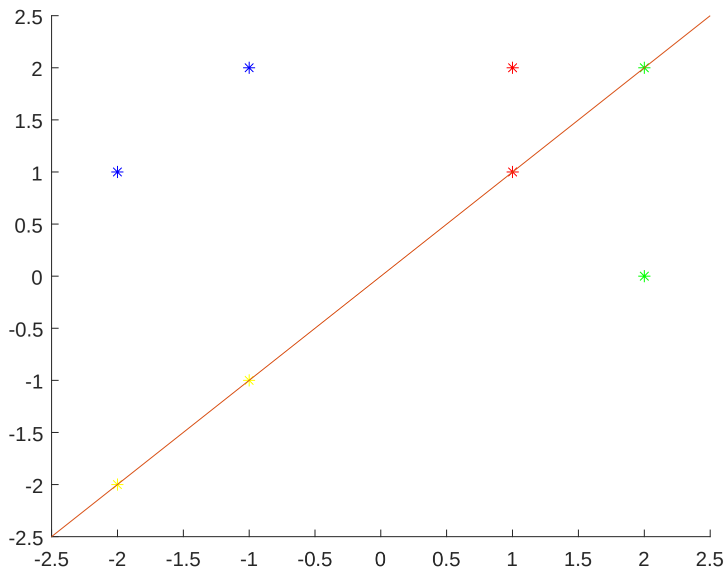
```
plot(p(1,2), p(2,2), 'R*');
plot(p(1,3), p(2,3), 'G*');
plot(p(1,4), p(2,4), 'G*');
plot(p(1,5), p(2,5), 'B*');
plot(p(1,6), p(2,6), 'B*');
plot(p(1,7), p(2,7), 'Y*');
plot(p(1,8), p(2,8), 'Y*');
x = -2.5:1:2.5;
y = x;
plot(x, y)
disp("4 - 12 - ii:")
disp("_____________________________")
p = [1 1 2 2 -1 -2 -1 -2; 1 2 2 0 2 1 -1 -2];
t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
E412(p, t)
disp("4 - 12 - iii & iv:")
disp("_____________________________")
p = [1 1 2 2 -1 -2 -1 -2; 1 2 1.5 0 2 1 -1 -2];
t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
E412(p, t)
```

i

اگر بردار ورودی p_3 را به $\mathbf{p}_3 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ تغییر دهیم مسأله با دو شبکهٔ دونورونه به صورت خطی تفکیک‌پذیر نخواهد بود. این به آن دلیل است که نمی‌توانیم مرز تصمیم‌گیری‌ای که بتواند این نقاط را طبقه‌بندی کند پیدا کنیم. همان‌طور که از شکل ۴ برآمده از دستورات زیر مشاهده می‌کنیم، نقاطی با کلاس‌های مختلف روی خط $y = x$ قرار می‌گبرند که هرچقدر هم مرز و خط را بچرخانیم امکان طبقه‌بندی صحیح نقاط را نخواهیم داشت.

```
p = [1 1 2 2 -1 -2 -1 -2; 1 2 2 0 2 1 -1 -2];
t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
hold on
plot(p(1,1), p(2,1), 'R*');
```

```
plot(p(1,2), p(2,2), 'R*');
plot(p(1,3), p(2,3), 'G*');
plot(p(1,4), p(2,4), 'G*');
plot(p(1,5), p(2,5), 'B*');
plot(p(1,6), p(2,6), 'B*');
plot(p(1,7), p(2,7), 'Y*');
plot(p(1,8), p(2,8), 'Y*');
x = -2.5:1:2.5;
y = x;
plot(x, y)
```



شکل ۴: نمودار قسمت اول سوال ۴.۱۲.

ii

با فراخوانی تابع تعریف‌شده به صورت زیر، نتیجهٔ نشان داده‌شده به دست آورده می‌شود که این نشان می‌دهد که شبکهٔ پرسپترون نمی‌تواند مرزهای تصمیم‌گیری را برای تفکیک بردارهای ورودی به شکل صحیح پیدا کند.

```
p = [1 1 2 2 -1 -2 -1 -2; 1 2 1.5 0 2 1 -1 -2];
t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
E412(p, t)
```

```
------------------------------------------------

Program terminated with no satisfying weight or bias after 50 iterations.
```

iii

با تغییر نقطهٔ $p_۳$ به $\mathbf{p}_۳ = \begin{bmatrix} ۲ \\ ۱/۵ \end{bmatrix}$، مسأله به‌صورت خطی تفکیک‌پذیر است.

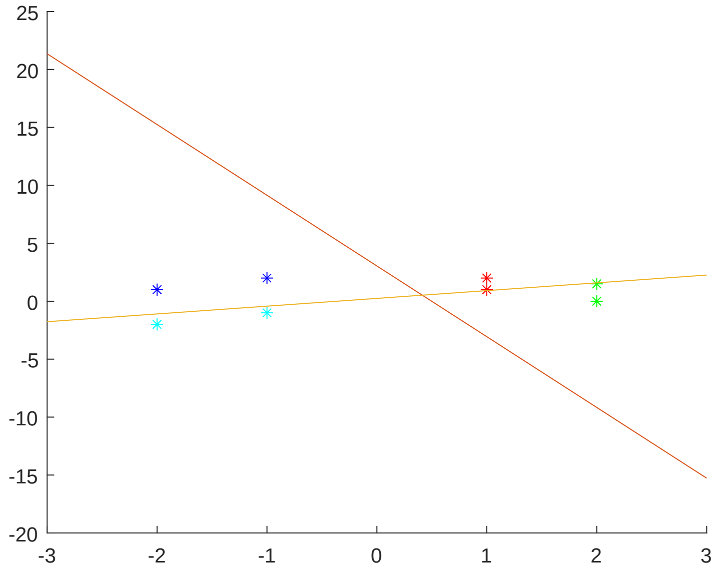
iv

با فراخوانی تابع تعریف‌شده به صورت زیر، نتیجهٔ نشان داده‌شده در زیر و در شکل ۵ به دست آورده می‌شود

```
p = [1 1 2 2 -1 -2 -1 -2; 1 2 1.5 0 2 1 -1 -2];
t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
E412(p, t)
------------------------------------------------

W =
   -3.6404    1.0268
    1.5004   -1.6730

b =
    0.2590
    0.0459
```

توضیح پوشهٔ کدها

در پوشهٔ اصلی این سوال (Q4__04_12)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند.



شکل ۵: نمودار قسمت چهارم سوال ۴.۱۲.

پاسخ سوال ۷.۱۰

i

ورودی و اهداف را به صورت زیر در نظر می‌گیریم:

$$\left\{\mathbf{p}_۱ = \begin{bmatrix} ۱ \\ ۱ \end{bmatrix}, t_۱ = [۱]\right\} \quad \left\{\mathbf{p}_۲ = \begin{bmatrix} ۱ \\ -۱ \end{bmatrix}, t_۲ = [-۱]\right\} \tag{۱}$$

برای محاسبات خواسته شده از دستورات زیر استفاده می‌کنیم. نتایج در ادامهٔ کد آورده شده است.

```
clc
clear all
close all

P = [1 1; 1 -1];
p1 = [1; 1];
p2 = [1; -1];
T = [1 -1];

```

```
Wh = T * P'
PPseudoinverse = inv(P'*P)*P'
W = T * pinv(P)

a1h = Wh*p1
a1p = W*p1
-----------------------------

Wh =
     0     2

PPseudoinverse =
    0.5000    0.5000
    0.5000   -0.5000

W =
   -0.0000    1.0000

a1h =
     2

a1p =
    1.0000
```

درواقع:

$$p = \begin{bmatrix} ۱ & ۱ \\ ۱ & -۱ \end{bmatrix}, \quad T = \begin{bmatrix} ۱ & -۱ \end{bmatrix} \tag{۲}$$

$$w^h = TP^T = \begin{bmatrix} ۱ & -۱ \end{bmatrix} \begin{bmatrix} ۱ & ۱ \\ ۱ & -۱ \end{bmatrix} = \begin{bmatrix} ۰ & ۲ \end{bmatrix} \tag{۳}$$

هم‌چنین همین دستورات را در محیط پایتون به‌صورت زیر نوشته‌ایم و نتایج را در ادامهٔ کد گزارش می‌کنیم:

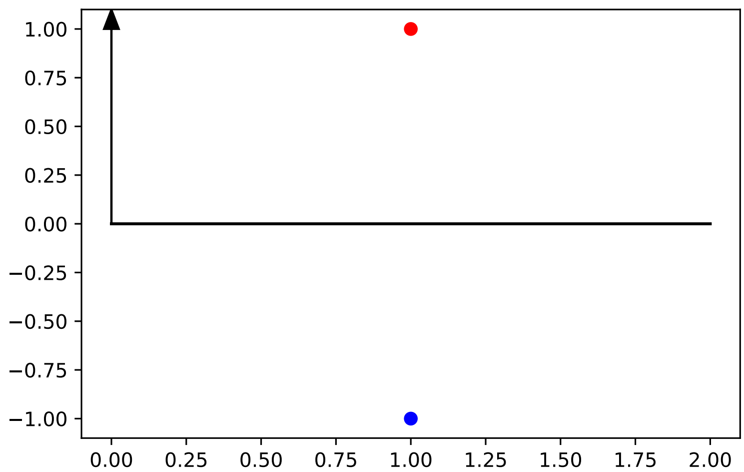
```
import numpy as np

P = np.array([[1, 1], [1, -1]])
p1 = np.array([[1], [1]])
p2 = np.array([[1], [-1]])
T = np.array([1, -1])

Wh = np.dot(T, P.T)
PPseudoinverse = np.dot(np.linalg.inv(np.dot(P.T, P)), P.T)
W = np.dot(T, np.linalg.pinv(P))

a1h = np.dot(Wh, p1)
a1p = np.dot(W, p1)

print("Wh = ", Wh)
print("PPseudoinverse = ", PPseudoinverse)
print("W = ", W)
print("a1h = ", a1h)
print("a1p = ", a1p)
------------------------------------------

Wh =  [0 2]
PPseudoinverse =  [[ 0.5  0.5]
 [ 0.5 -0.5]]
W =  [-2.22044605e-16  1.00000000e+00]
a1h =  [2]
a1p =  [1.]
```

ii

با استفاده از دستور زیر مرز ناحیهٔ تصمیم‌گیری را رسم می‌کنیم و نتیجه به صورتی خواهد بود که در شکل ۶ نشان داده شده است.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
p1 = np.array([1,1])
p2 = np.array([1,-1])
plt.plot(p1[0], p1[1], 'ro')
plt.plot(p2[0], p2[1], 'bo')

plt.plot([0,2],[0,0],'k-')
plt.arrow(0,0,0,1,head_width=0.05, head_length=0.1, fc='k', ec='k')

plt.savefig('E710.pdf')
plt.show()
```



شکل ۶: مرز سوال پنجم.

iii

برای محاسبات خواسته شده از دستورات زیر استفاده می‌کنیم. نتایج در ادامهٔ کد آورده شده است.

```
clc
clear all
close all

P = [1 1; 1 -1];
```

```
p1 = [1; 1];
p2 = [1; -1];
T = [1 -1];

Wh = T * P'
PPseudoinverse = inv(P'*P)*P'
W = T * pinv(P)

a1h = Wh*p1
a1p = W*p1
-----------------------------

Wh =
     0     2

PPseudoinverse =
    0.5000    0.5000
    0.5000   -0.5000

W =
   -0.0000    1.0000

a1h =
     2

a1p =
    1.0000
```

درواقع:

$$w^p = TP^+ = T\left(P^T P\right)^{-1} P^T = \begin{bmatrix} 1 & -1 \end{bmatrix} \left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \left(\left(\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \right) \right)^{-1} \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{۴}$$

iv

برای محاسبات خواسته شده از دستورات زیر استفاده می‌کنیم. نتایج در ادامهٔ کد آورده شده است.

```
clc
clear all
close all

P = [1 1; 1 -1];
p1 = [1; 1];
p2 = [1; -1];
T = [1 -1];

Wh = T * P'
PPseudoinverse = inv(P'*P)*P'
W = T * pinv(P)

a1h = Wh*p1
a1p = W*p1
-----------------------------

Wh =
     0     2

PPseudoinverse =
    0.5000    0.5000
    0.5000   -0.5000

W =
   -0.0000    1.0000

a1h =
     2

a1p =
    1.0000
```

همان‌طور که مشاهده می‌شود:

$$w^h p_1 = \begin{bmatrix} 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2 \neq t_1 \tag{5}$$

$$w^p p_1 = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 = t_1 \tag{6}$$

که این ناشان می‌دهد $w^h p_1$ به‌دلیل نرمال‌نبودن $((p_1^T p_1) \neq 1)$ برابر با هدف نشده؛ اما در روش شبه‌معکوس علاوه بر تعامد با هدف هم هم‌سنگ شده‌ایم.

توضیح پوشهٔ کدها

در پوشهٔ اصلی این سوال (Q5__07_10)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند. هم‌چنین فایل اجرای برخط کدها از طریق پیوند گوگل‌کولب در دسترس است.

پاسخ سوال ۷.۱۱

برای حل این سوال از دستورات زیر که در پوشهٔ مربوط به کدها نیز آمده است استفاده می‌کنیم:

```
% Autoassociative network for digit recognition using Hebbian learning
clc;
clear all;
close all;

%% load training images(5x7 px)
M=7; N=5;
P=zeros(M*N,10);
for n=0:9
    s=num2str(n);
    RGB=imread(s,'png');
    m=n+1;
```

```
    P(:,m)=reshape(rgb2gray(RGB),[M*N,1]); %column prototype vector
end
P = P/255*2-1; %normalizes data to be either -1 or 1

%% compute weight matrix using Hebb rule
W=zeros(M*N,M*N);
for n=1:10
    W=W+P(:,n)*P(:,n)';
end

% simple Hebb rule did not make it recognize images well
% try again using pseudoinverse
T=P;
W1=T*pinv(P);

%% test out network with training images
for n = 10:-1:1
    a=hardlims(W*P(:,n));
    outputImg=reshape(a,[M,N]);
    figure;
    imshow(outputImg,'InitialMagnification','Fit')
end

%% test out network with noisy inputs
P_noisy = P + randi([-1,1],M*N,10);

for n = 10:-1:1
    outputImg1=reshape(P_noisy(:,n),[M,N]);
    figure;
    subplot(1,2,1)
    imshow(outputImg1,'InitialMagnification','Fit')
    title('noisy image (Hebb)')

    subplot(1,2,2)
    a=hardlims(W*P_noisy(:,n));
```

```
    outputImg2=reshape(a,[M,N]);
    imshow(outputImg2,'InitialMagnification','Fit')
    title('reconstructed image (Hebb)')
end

for n = 10:-1:1
    outputImg1=reshape(P_noisy(:,n),[M,N]);
    figure;
    subplot(1,2,1)
    imshow(outputImg1,'InitialMagnification','Fit')
    title('noisy image (Pseudoinverse)')

    subplot(1,2,2)
    a=hardlims(W1*P_noisy(:,n));
    outputImg2=reshape(a,[M,N]);
    imshow(outputImg2,'InitialMagnification','Fit')
    title('reconstructed image (Pseudoinverse)')
end

%% test out network with parts of image missing
P_partial=P;
P_partial(8:17,:)=1;

for n = 10:-1:1
    outputImg1=reshape(P_partial(:,n),[M,N]);
    figure;
    subplot(1,2,1)
    imshow(outputImg1,'InitialMagnification','Fit')
    title('partial image (Hebb)')

    subplot(1,2,2)
    a=hardlims(W*P_partial(:,n));
    outputImg2=reshape(a,[M,N]);
    imshow(outputImg2,'InitialMagnification','Fit')
    title('reconstructed image (Hebb)')
```

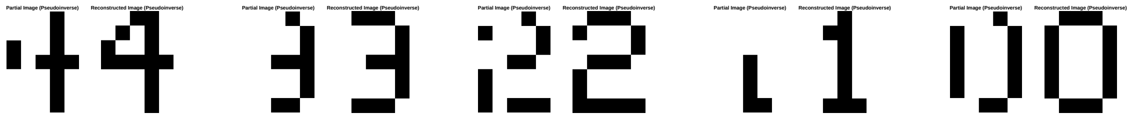
```
end

P_partial=P;
P_partial(8:17,:)=1;

for n = 10:-1:1
    outputImg1=reshape(P_partial(:,n),[M,N]);
    figure;
    subplot(1,2,1)
    imshow(outputImg1,'InitialMagnification','Fit')
    title('partial image (Pseudoinverse)')

    subplot(1,2,2)
    a=hardlims(W1*P_partial(:,n));
    outputImg2=reshape(a,[M,N]);
    imshow(outputImg2,'InitialMagnification','Fit')
    title('reconstructed image (Pseudoinverse)')
end
```

تابع hardlims هم که در کد اصلی بالا تعریف شده است به‌صورت زیر تعریف می‌گردد:

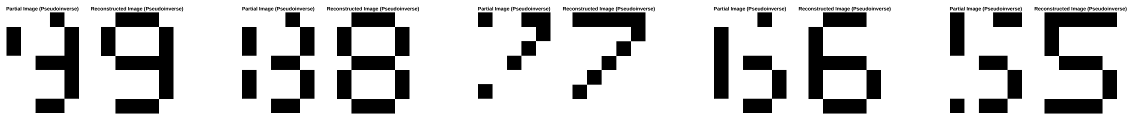
```
function [ a ] = hardlims( n )
%   hardlims Symmetric Hard Limit transfer function
%   Accepts column vector. Returns column vector.
%   behaves like unit step of amplitude 2 and vertical offset of -1
%   a = -1 for n < 0
%   a =  1 for n >= 0
    dims = size(n);
    a = zeros(dims);
    for i = 1:dims(1)
        for j = 1:dims(2)
            if n(i,j) < 0
                a(i,j) = -1;
            else
                a(i,j) = 1;
            end
```

```
        end
    end
end
```

با اجرای این دستورات نتایج به صورتی که در ادامه آورده شده است خواهد بود:

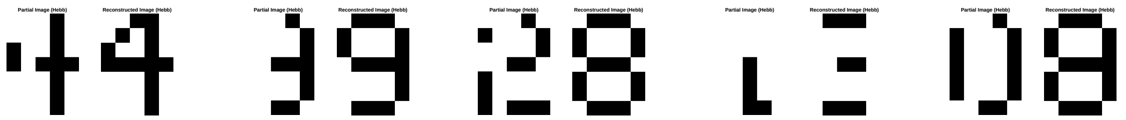


(آ) ۰ (ب) ۱ (ج) ۲ (د) ۳ (ه) ۴

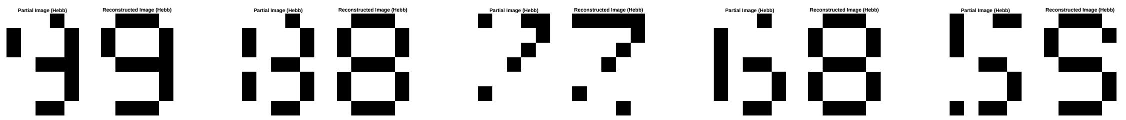


(و) ۵ (ز) ۶ (ح) ۷ (ط) ۸ (ی) ۹

شکل ۷: بازسازی اعداد از روی قسمتی از آن‌ها (Pseudoinverse).



(آ) ۰ (ب) ۱ (ج) ۲ (د) ۳ (ه) ۴

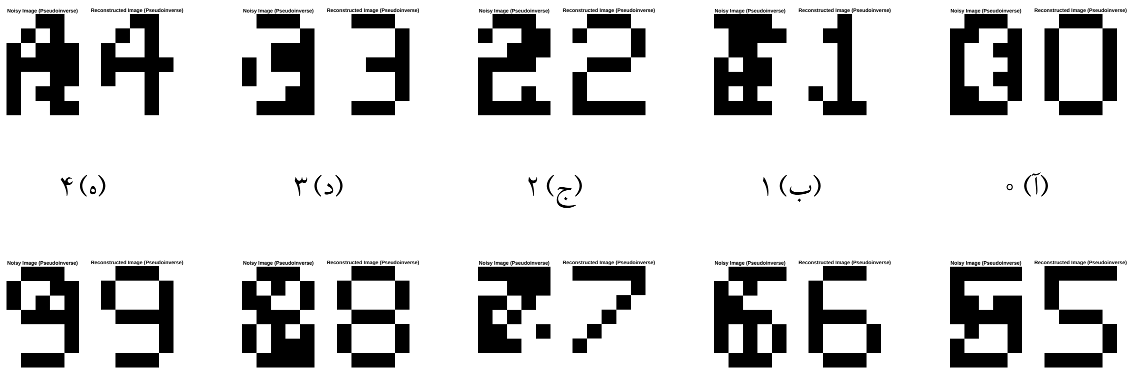


(و) ۵ (ز) ۶ (ح) ۷ (ط) ۸ (ی) ۹

شکل ۸: بازسازی اعداد از روی قسمتی از آن‌ها (Hebb).

توضیح پوشهٔ کدها

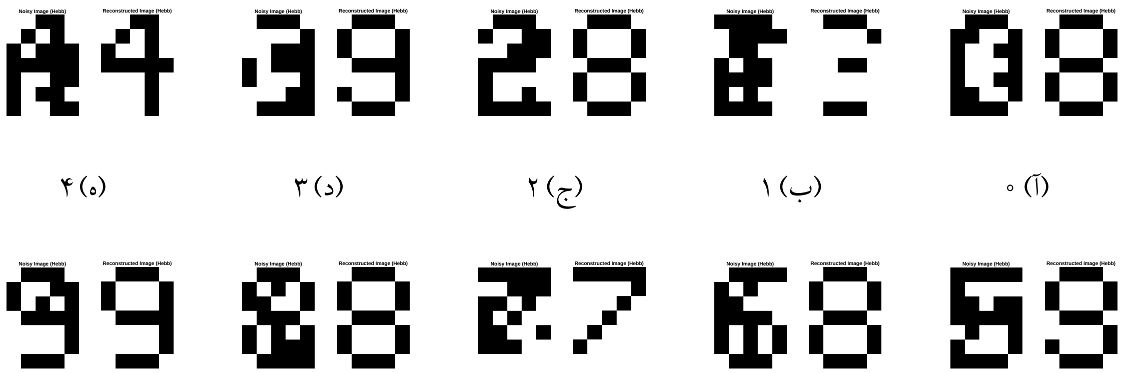
در پوشهٔ اصلی این سوال (`Q6__07_11`)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند.



(آ) ۰ (ب) ۱ (ج) ۲ (د) ۳ (ه) ۴

(و) ۵ (ز) ۶ (ح) ۷ (ط) ۸ (ی) ۹

شکل ۹: بازسازی اعداد از روی نویزی‌شدهٔ آن‌ها (Pseudoinverse).



(آ) ۰ (ب) ۱ (ج) ۲ (د) ۳ (ه) ۴

(و) ۵ (ز) ۶ (ح) ۷ (ط) ۸ (ی) ۹

شکل ۱۰: بازسازی اعداد از روی نویزی‌شدهٔ آن‌ها (Hebb).

پاسخ سوال ۹.۱۰

راه‌حل اول

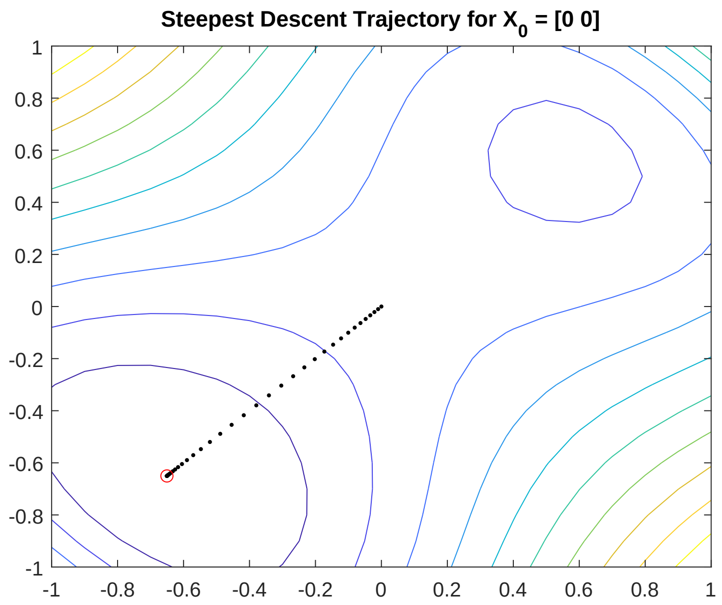
برای روش نزولی و مقدار اولیهٔ $x_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}'$ دستورات زیر را می‌نویسیم و نتیجه به صورتی خواهد بود که در شکل ۱۲ آورده شده است.

```
clear
%plot the function contour lines
[X,Y] = meshgrid(-1 : .1 : 1);
Z = (X+Y).^4 - 12*X.*Y + X + Y + 1;
N = 10;
```

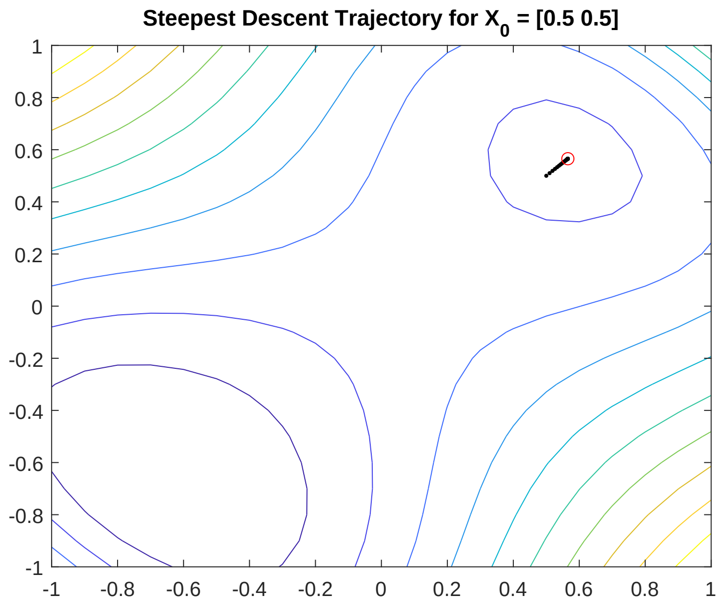
```
figure;
contour(X, Y, Z, N), title('Steepest Descent Trajectory for X_0 = [0 0]');
hold on;

x = [0 0]';   %Initialize x
G = zeros(2,1);
G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1;
G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
alfa = 0.01;
dx = [1e4 1e4]';
small = [1.0e-5,1.0e-5]';

%Find the stationary point
while (abs(dx(1)) >= small(1) | abs(dx(2))>= small(2))
  plot(x(1), x(2),'k.')
  old = x;
  x = x - alfa * G;
  dx = x - old;
  G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1;  G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
end
x
plot(x(1), x(2),'ro')
---------------------

x =
   -0.6504
   -0.6504
```

برای مقدار اولیهٔ $x_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}'$ نتیجه به صورتی خواهد بود که در شکل ۱۲ آورده شده است. برای روش نیوتن و مقدار اولیهٔ $x_0 = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}'$ دستورات زیر را می‌نویسیم و نتیجه به صورتی خواهد بود که در شکل ۱۳ آورده شده است.

```
clc
clear all
close all

```



شکل ۱۱: پاسخ سوال ۹.۱۰.



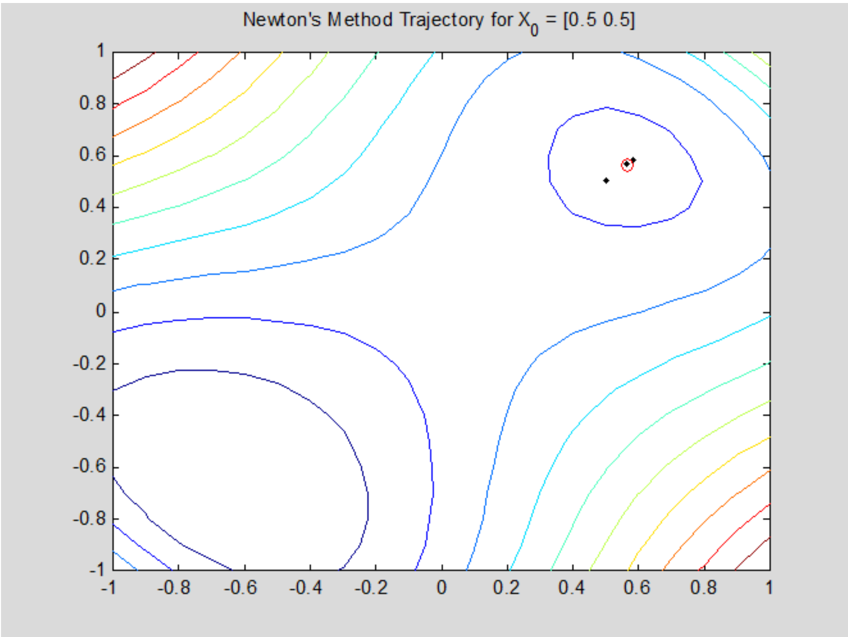
شکل ۱۲: پاسخ سوال ۹.۱۰.

```
%plot the function contour lines
[X,Y] = meshgrid(-1 : .1 : 1);
Z = (X+Y).^4 - 12*X.*Y + X + Y + 1;
N = 10;
figure;
contour(X, Y, Z, N), title('Newton''s Method Trajectory for X_0 = [0 0]');
```

```
hold on;

G = zeros(2,1);
A = zeros(2);
x = [0.5 0.5]';   %Initialize x
% G = grad(x);
G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1;
G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
% G = jacobian(Z,x)
% G = [-12*Y + 4*(X + Y)^3 + 1; -12*X + 4*(X + Y)^3 + 1]
% A = hessian(x);
A(1,1) = 12*(x(1) + x(2))^2; A(1,2) = 12*((x(1) + x(2))^2 - 12);
A(2,1) = 12*((x(1) + x(2))^2 - 12); A(2,2) = 12*(x(1) + x(2))^2;
% A = jacobian(G,x);
% A = [12*(X + Y)^2  12*((X + Y)^2 - 1); 12*((X + Y)^2 - 1), 12*(X + Y)^2]
dx = [1e2 1e2]';
small = [1.0e-3,1.0e-3]';

%Find the stationary point
while (abs(dx(1)) >= small(1) | abs(dx(2))>= small(2))
   plot(x(1), x(2),'k.')
   old = x;
   x = x - inv(A).*G;
   dx = x - old;
%    G = grad(x);
   G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1;  G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
   %G = [-12*Y + 4*(X + Y)^3 + 1; -12*X + 4*(X + Y)^3 + 1]
%    A = hessian(x);
   A(1,1) = 12*(x(1) + x(2))^2; A(1,2) = 12*((x(1) + x(2))^2 - 1);
   A(2,1) = 12*((x(1) + x(2))^2 - 1); A(2,2) = 12*(x(1) + x(2))^2;
   %A = [12*(X + Y)^2  12*((X + Y)^2 - 1); 12*((X + Y)^2 - 1), 12*(X + Y)^2]
end
x
plot(x(1), x(2),'ro')
---------------------
```

```
clc
clear all
close all

%plot the function contour lines
[X,Y] = meshgrid(-1 : .1 : 1);
Z = (X+Y).^4 - 12*X.*Y + X + Y + 1;
N = 10;
figure;
contour(X, Y, Z, N), title('Newton''s Method Trajectory for X_0 = [0 0]');
hold on;

G = zeros(2,1);
A = zeros(2);
x = [0.5 0.5]';    %Initialize x
% G = grad(x);
G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1;
G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
% G = jacobian(Z,x)
% G = [-12*Y + 4*(X + Y)^3 + 1; -12*X + 4*(X + Y)^3 + 1]
% A = hessian(x);
A(1,1) = 12*(x(1) + x(2))^2; A(1,2) = 12*((x(1) + x(2))^2 - 12);
A(2,1) = 12*((x(1) + x(2))^2 - 12); A(2,2) = 12*(x(1) + x(2))^2;
% A = jacobian(G,x);
% A = [12*(X + Y)^2  12*((X + Y)^2 - 1); 12*((X + Y)^2 - 1), 12*(X + Y)^2]
dx = [1e2 1e2]';
small = [1.0e-3,1.0e-3]';

%Find the stationary point
while (abs(dx(1)) >= small(1) | abs(dx(2))>= small(2))
   plot(x(1), x(2),'k.')
   old = x;
   x = x - inv(A).*G;
   dx = x - old;
```

```
%    G = grad(x);
   G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1;  G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
   %G = [-12*Y + 4*(X + Y)^3 + 1; -12*X + 4*(X + Y)^3 + 1]
%    A = hessian(x);
   A(1,1) = 12*(x(1) + x(2))^2; A(1,2) = 12*((x(1) + x(2))^2 - 1);
   A(2,1) = 12*((x(1) + x(2))^2 - 1); A(2,2) = 12*(x(1) + x(2))^2;
   %A = [12*(X + Y)^2  12*((X + Y)^2 - 1); 12*((X + Y)^2 - 1), 12*(X + Y)^2]
end
x
plot(x(1), x(2),'ro')
```



شکل ۱۳: پاسخ سوال ۹.۱۰.

راه‌حل دوم

برای تابعِ

$$F(X) = (x_1 + x_2)^4 - 12x_1x_2 + x_1 + x_2 - 1 \qquad (7)$$

گرادیان را تشکیل می‌دهیم:

$$\nabla F(X)=\begin{bmatrix}\frac{\partial}{\partial x_۱}F(x)\\ \frac{\partial}{\partial x_۲}F(x)\end{bmatrix}=\begin{bmatrix}۴\left(x_۱+x_۲\right)^۳-۱۲x_۲+۱\\ ۴\left(x_۱+x_۲\right)^۳-۱۲x_۱+۱\end{bmatrix} \tag{۸}$$

بنابراین نقاط ایستا به‌صورت زیر به دست می‌آید:

$$x^۱=\begin{bmatrix}-۰٫۶۵۰۴\\ -۰٫۶۵۰۴\end{bmatrix},x^۲=\begin{bmatrix}۰٫۰۸۵\\ ۰٫۰۸۵\end{bmatrix}x^۳=\begin{bmatrix}۰٫۵۶۵۵\\ ۰٫۵۶۵۵\end{bmatrix} \tag{۹}$$

حال ماتریس هسیان را تشکیل می‌دهیم:

$$\nabla^۲F(X)=\begin{bmatrix}۱۲\left(x_۱+x_۲\right)^۲ & ۱۲\left(x_۱+x_۲\right)^۲-۱۲\\ ۱۲\left(x_۱+x_۲\right)^۲-۱۲ & ۱۲\left(x_۱+x_۲\right)^۲\end{bmatrix} \tag{۱۰}$$

ادامهٔ حل این راه‌حل به صورتی که در سه دستور پایتون زیر نوشته شده پیگیری می‌شود. نتایج هر کد هم در ذیل همان کد آورده شده است. برای اختصار از آوردن کدهای نوشته‌شده و نتایج دیگر صرف‌نظر می‌شود. همگی از طریق پوشهٔ کدها در دسترس هستند.

```
# Steepest Descent
#
# Requires an initial point x - matrix nx1
# Requires a function parameters to the gradient and hessian matrices of the
    function
# Requires a maximum number of iterations, used to stop if algorithm diverges
# Requires a tolerance for exiting condition
# Requires a mask boolean for whether or not to print out values inbetween
    iterations

steepest_descent = function(init_point, gradient, hessian, maxIter, tol, mask)
    {

  x = init_point

```

```
for(i in 1:maxIter) {

  if(!mask) { # if the user does not want print out statements
    print(sprintf("      Iteration: %d", i))
    print("x value:")
    print(x)
  }

  g = gradient(x) # find gradient at x

  if(!mask) { # if the user does not want print out statements
    print("gradient value:")
    print(g)
  }

  # calculate the frobenius norm of the gradient vector evaluated at x
  # remember, stationary points occur when gradient is zero, so we take
  # the square root of the sum of squares of the gradient values and
  # compare to the tolerance
  if(norm(g, "F") < tol) { # convert gradient into single value
    # through frobenius norm
    print("Stationary Point Found at point:")
    print(x)
    print(sprintf("Iterations taken: %d", i))
    return(x)
  }

  # creating the search direction vectors
  p = -g

  # Here we have a dynamic learning rate that is minimized against
  # the line of the next iteration
  alpha = -(t(g)%*%p) / (t(p)%*%hessian(x)%*%p)

  x = x - alpha[1]*g # find next x
```

```



  }

  print("MAXIMUM ITERATIONS REACHED")
  print("Current point is: ")
  print(x)
  print("Error: ")
  print(abs(norm(g, "F")-tol)) # computing the error
  return(x)
}


# example function:
# (x+y)^4-12xy+x+y-1
# Has three stationary points
# 1) [-0.65, -0.65]
# 2) [0.5654, 0.564]
# 3) [0.0849, 0.0849]

gradient = function(X) {
  x = X[1,1]
  y = X[2,1]
  g1 = 4*(x+y)^3-12*y+1
  g2 = 4*(x+y)^3-12*x+1
  matrix(c(g1, g2), ncol=1)
}

hessian = function(X) {
  x = X[1,1]
  y = X[2,1]
  fxx = 12*(x + y)^2
  fxy = 12*(x + y)^2 - 12
  matrix(c(fxx, fxy, fxy, fxx), ncol=2)
}
```

```

# initial points
p1 = matrix(c(-1, -1), ncol=1)

# Function call
min = steepest_descent(p1,gradient, hessian, 1000, 1e-10, 1)
------------------------------------------------------------

[1] "Stationary Point Found at point:"
           [,1]
[1,] -0.6504198
[2,] -0.6504198
[1] "Iterations taken: 7"
```

```

# Newton's Method
#
# Requires an initial point x - matrix nx1
# Requires a function parameters to the gradient and hessian matrices of the
   function
# Requires a maximum number of iterations, used to stop if algorithm diverges
# Requires a tolerance for exiting condition
# Requires a mask boolean for whether or not to print out values inbetween
   iterations

newtons_method = function(init_point, gradient, hessian, maxIter, tol, mask) {


  x = init_point


  for(i in 1:maxIter) {

    if(!mask) { # if the user does not want print out statements
      print(sprintf("      Iteration: %d", i))
```

```
      print("x value:")
      print(x)
    }

    A = hessian(x)
    g = gradient(x)
    x = x - solve(A)%*%g # solve means inverse, A^-1

    if(!mask) { # if the user does not want print out statements
      print("gradient value:")
      print(g)
    }

    # calculate the frobenius norm of the gradient vector evaluated at x
    # remember, stationary points occur when gradient is zero, so we take
    # the square root of the sum of squares of the gradient values and
    # compare to the tolerance
    if(norm(g, "F") < tol) { # convert gradient into single value
      # through frobenius norm
      print("Stationary Point Found at point:")
      print(x)
      print(sprintf("Iterations taken: %d", i))
      return(x)
    }
  }

  print("MAXIMUM ITERATIONS REACHED")
  print("Current point is: ")
  print(x)
  print("Error: ")
  print(abs(norm(g, "F")-tol)) # computing the error
  return(x)
}
```

```r
# example function:
# (x+y)^4-12xy+x+y-1
# Has three stationary points
# 1) [-0.65, -0.65]
# 2) [0.5654, 0.564]
# 3) [0.0849, 0.0849]

gradient = function(X) {
  x = X[1,1]
  y = X[2,1]
  g1 = 4*(x+y)^3-12*y+1
  g2 = 4*(x+y)^3-12*x+1
  matrix(c(g1, g2), ncol=1)
}

hessian = function(X) {
  x = X[1,1]
  y = X[2,1]
  fxx = 12*(x + y)^2
  fxy = 12*(x + y)^2 - 12
  matrix(c(fxx, fxy, fxy, fxx), ncol=2)
}

# initial points
p1 = matrix(c(-1, -1), ncol=1)

# Function call
min = newtons_method(p1,gradient, hessian, 1000, 1e-10, 1)
------------------------------------------------------------

[1] "Stationary Point Found at point:"
           [,1]
[1,] -0.6504198
[2,] -0.6504198
[1] "Iterations taken: 7"
```

```

# Conjugate Gradient method
#
# Requires an initial point x - matrix nx1
# Requires a function parameters to the gradient and hessian matrices of the function
# Requires a maximum number of iterations, used to stop if algorithm diverges
# Requires a tolerance for exiting condition
# Requires a mask boolean for whether or not to print out values inbetween iterations
# Requires a value ranging from 1,2,3 for the type of beta method

conjuage_gradient = function(init_point, gradient, hessian, maxIter, tol, mask, betaMethod) {

  x = init_point

  for(i in 1:maxIter) {

    if(!mask) { # if the user does not want print out statements
      print(sprintf("      Iteration: %d", i))
      print("x value:")
      print(x)
    }

    g = gradient(x) # find gradient at x

    if(!mask) { # if the user does not want print out statements
      print("gradient value:")
      print(g)
    }

    # calculate the frobenius norm of the gradient vector evaluated at x
    # remember, stationary points occur when gradient is zero, so we take
    # the square root of the sum of squares of the gradient values and
```

```
  # compare to the tolerance
  if(norm(g, "F") < tol) { # convert gradient into single value
    # through frobenius norm
    print("Stationary Point Found at point:")
    print(x)
    print(sprintf("Iterations taken: %d", i))
    return(x)
  }

  # creating the search direction vectors
  if(i==1) { # first search direction
    p = -g
  }
  else { # use beta
    if(betaMethod == 1) { # Common choice
      beta = (t(g-g0)%*%g)/(t(g0)%*%p)
    }
    else if (betaMethod == 2) { # hestens and stiefel
      beta = (t(g)%*%g)/(t(g0)%*%g)
    }
    else { # Fletcher and Reeves
      beta = (t(g-g0)%*%g)/(t(g0)%*%g0)
    }

    if(!mask) { # if the user does not want print out statements
      print("beta value:")
      print(beta)
    }

    p = -g+beta[1]*p
  }


  alpha = -(t(g)%*%p) / (t(p)%*%hessian(x)%*%p) # learning rate
  x = x - alpha[1]*g # find next x
```

```
    g0 = g # update previous gradient to current
  }

  print("MAXIMUM ITERATIONS REACHED")
  print("Current point is: ")
  print(x)
  print("Error: ")
  print(abs(norm(g, "F")-tol)) # computing the error
  return(x)
}

# example function:
# (x+y)^4-12xy+x+y-1
# Has three stationary points
# 1) [-0.65, -0.65]
# 2) [0.5654, 0.564]
# 3) [0.0849, 0.0849]

gradient = function(X) {
  x = X[1,1]
  y = X[2,1]
  g1 = 4*(x+y)^3-12*y+1
  g2 = 4*(x+y)^3-12*x+1
  matrix(c(g1, g2), ncol=1)
}

hessian = function(X) {
  x = X[1,1]
  y = X[2,1]
  fxx = 12*(x + y)^2
  fxy = 12*(x + y)^2 - 12
  matrix(c(fxx, fxy, fxy, fxx), ncol=2)
}
```

```
# initial points
p1 = matrix(c(-1, -1), ncol=1)

# Function call
min = conjuage_gradient(p1,gradient, hessian, 1000, 1e-10, 1,1)
------------------------------------------------------------

[1] "Stationary Point Found at point:"
          [,1]
[1,] -0.6504198
[2,] -0.6504198
[1] "Iterations taken: 29"
```

توضیح پوشهٔ کدها

در پوشهٔ اصلی این سوال (Q7__09_10)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند. هم‌چنین فایل اجرای برخط کدهای جامعی از چندین مثال مرتبط با فصول هشتم و نهم کتاب هگان از طریق پیوند گوگل کولب در دسترس است.

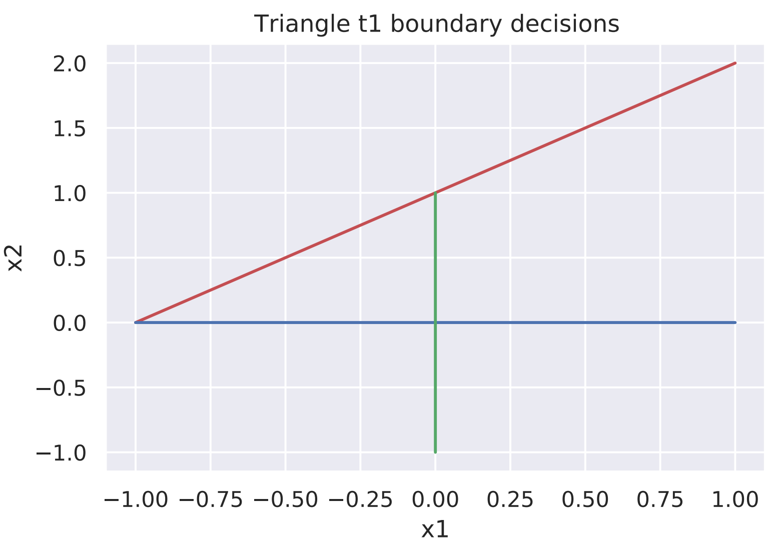
پاسخ سوال ۱۱.۱

برای هر الگو، ابتدا مرزهای تصمیم را ایجاد می‌کنیم، ماتریس وزن را تشکیل می‌دهیم و سپس شبکه‌ای را ایجاد می‌کنیم که AND و OR مرزهای مناسب را ایجاد می‌کند. برای تمام قسمت‌های این سوال، از ساختار شبکه مشابه شکل P11.6 کتاب استفاده می‌کنیم که شامل یک لایهٔ تصمیم‌گیری اولیه برای ایجاد مرز برای دسته‌ها، و یک لایهٔ AND برای تثبیت آن لایه‌ها، و یک لایه OR برای ترکیب دسته‌ها خواهد بود.

i

در این قسمت دو مثلث با نقاط انتهایی $(۰,۱),(۰,۰),(-۱,۰)$ و $(-۱,۰),(-۱,-۱),(۰,-۱)$ مشاهده می‌شود. مثلث نخستکه نام $t_۱$ را برای آن برمی‌گزینیم، می‌تواند با خطوط $y_۱ = x_۱ + ۱, y_۲ = ۰, x_۳ = ۰$ تعیین شود. این ناحیهٔ مثلثی‌شکل با بهره‌گیری از دستورات زیر در شکل ۱۴ نشان داده شده است.

```python
import matplotlib.pyplot as plt
import numpy as np

# Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
x1 = np.linspace(-1, 1, 100)
y1 = x1 + 1
y2 = np.zeros(100)
x3 = np.zeros(100)

plt.plot(x1, y1, 'r', x1, y2, 'b', x3, x1, 'g')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Triangle t1 boundary decisions')
plt.savefig('E111_1.pdf')
plt.show()
```



شکل ۱۴: مرزهای تصمیم‌گیری قسمت اول سوال ۱-۱۱ (t_1).

این سه معادلهٔ خط به اوزان زیر می‌انجامد:

- برای خط $y_1 = x_1 + 1$، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $p_1 - p_2 + 1$.
- برای خط $x_2 = 0$ می‌خواهیم هر مقداری در سمت چپ خط عمودی یک شود؛ چراکه، می‌خواهیم

مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $-p_1$.

- برای خط $y_2 = 0$ می‌خواهیم هر مقداری در بالای خط افقی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم p_2.

بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی مثلث به صورت زیر خواهد بود:

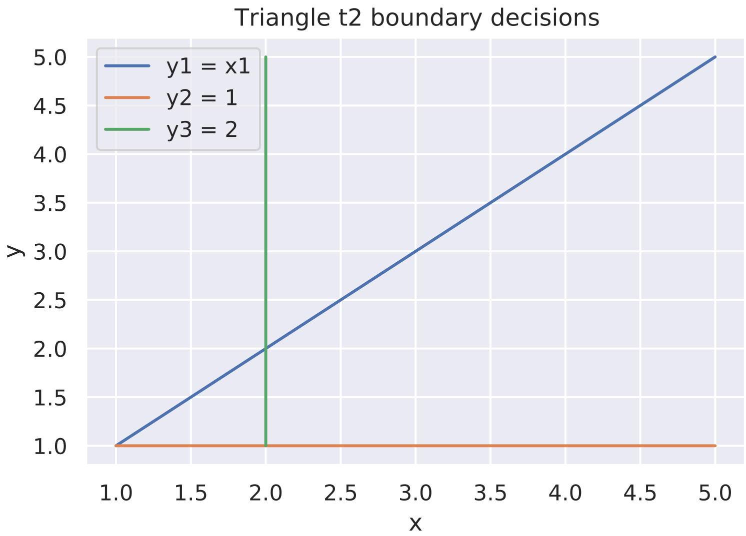
$$w^T = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, b^T = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{11}$$

از طرف دیگر مثلث دوم با نام t_2 با خطوط $y_1 = x_1, y_2 = 1, x_3 = 2$ ساخته می‌شود. این ناحیه با بهره‌گیری از دستورات زیر در شکل ۱۵ نشان داده شده است.

```python
import matplotlib.pyplot as plt
import numpy as np

# Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
x1 = np.linspace(-1, 1, 100)
y1 = x1 + 1
y2 = np.zeros(100)
x3 = np.zeros(100)

plt.plot(x1, y1, 'r', x1, y2, 'b', x3, x1, 'g')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Triangle t1 boundary decisions')
plt.savefig('E111_1.pdf')
plt.show()
```

این سه معادلهٔ خط به اوزان زیر می‌انجامد:

- برای خط $y_1 = x_1$، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $p_1 - p_2$.
- برای خط $x_3 = 2$ می‌خواهیم هر مقداری در سمت چپ خط عمودی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $-p_1 + 2$.



شکل ۱۵: مرزهای تصمیم‌گیری قسمت اول سوال ۱-۱۱ (t_2).

- برای خط $y_2 = 1$ می‌خواهیم هر مقداری در بالای خط افقی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $p_2 - 1$.

بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی مثلث به صورت زیر خواهد بود:

$$w^T = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, b^T = \begin{bmatrix} 0 & 2 & -1 \end{bmatrix} \tag{12}$$

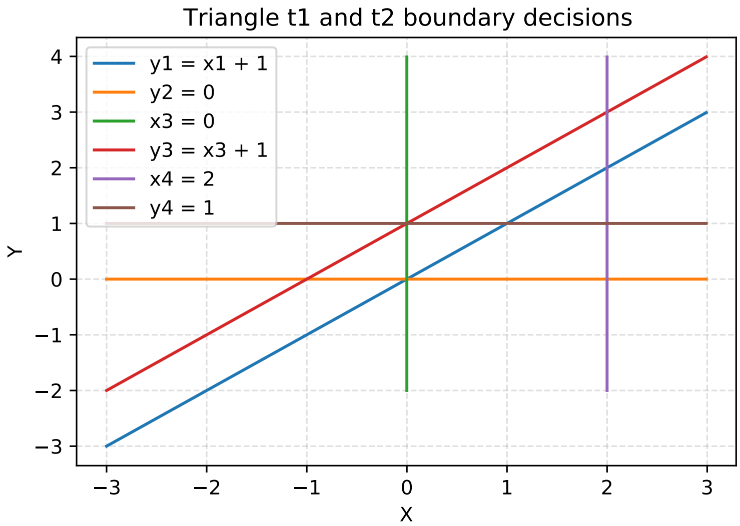
حال می‌توانیم این مرزهای تصمیم‌گیری را در ماتریس‌های واحد ادغام کنیم (شکل ۱۶).

$$w^T = \begin{bmatrix} 1 & -1 & 0 & 1 & -1 & 0 \\ -1 & 0 & 1 & -1 & 0 & 1 \end{bmatrix}, b^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 & -1 \end{bmatrix} \tag{13}$$

```python
import matplotlib.pyplot as plt
import numpy as np

# Create the vectors X and Y
x = np.arange(-3, 3, 0.01)
y1 = x
y2 = np.zeros(600)
x3 = np.zeros(600)
```

```python
y3 = x + 1
x4 = 2*np.ones(600)
y4 = 1*np.ones(600)

# Create the plot
plt.plot(x, y1, label='y1 = x1 + 1')
plt.plot(x, y2, label='y2 = 0')
plt.plot(x3, y3, label='x3 = 0')
plt.plot(x, y3, label='y3 = x3 + 1')
plt.plot(x4, y3, label='x4 = 2')
plt.plot(x, y4, label='y4 = 1')

# Add a title
plt.title('Triangle t1 and t2 boundary decisions')

# Add X and y Label
plt.xlabel('X')
plt.ylabel('Y')

# Add a grid
plt.grid(alpha=.4,linestyle='--')

# Add a Legend
plt.legend()

# Save the plot
plt.savefig('E1113.pdf')

# Show the plot
plt.show()
```

در ادامه به ماتریس وزنی نیاز داریم که مرزهای تصمیم‌گیری را به‌گونه‌ای که به فرم مثلثی دربیایند ترکیب کند (مرزهای از حالت بی سر و ته بودن درآیند). برای مثلث اول، سه ستون اول ماتریس وزن اول مرزهای را می‌سازند و درنتیجه باید این سه قسمت در نظر گرفته شوند و بقیه نادیده گرفته شوند. این کار می‌تواند با یک سطر به‌صورت $[1, 1, 1, 0, 0, 0]$انجام گیرد. همین کار را برای مثلث دوم تکرار می‌کنیم. بنابراین، این



شکل ۱۶: مرزهای تصمیم‌گیری قسمت اول سوال ۱-۱۱.

کار را با استفاده از ماتریس وزن لایه‌ی دوم انجام می‌دهیم:

$$w_{۲} = \begin{bmatrix} ۱ & ۱ & ۱ & ۰ & ۰ & ۰ \\ ۰ & ۰ & ۰ & ۱ & ۱ & ۱ \end{bmatrix} \tag{۱۴}$$

برای تشکیل ماتریس بایاس به این نکته توجه می‌کنیم که نتیجه باید برابر با یک شود؛ اما در هر سطر از ماتریس وزن لایهٔ دوم سه مقدار یک داریم. بنابراین می‌نویسیم:

$$b_{۲} = \begin{bmatrix} -۲ & -۲ \end{bmatrix} \tag{۱۵}$$

حال که به مرزهای تصمیم‌گیری خود فرم دادیم و آن‌ها را تثبیت کردیم، نیاز به یک مکانیزم OR داریم که در صورتی که ورودی در نواحی مثلث‌های اول و دوم افتاد دستهٔ صحیحی تولید کند. ما به این لایه احتیاج داریم تا اگر خروجی پرسپترون AND یک را بازگرداند، یک بازگردد و اگر ۱− را بازگرداندند، ۱− بازگردد. این کار می‌تواند با ترکیب دو مقدار ورودی و جمعشان با یک انجام شود؛ در نتیجه داریم:

$$w_{۳} = \begin{bmatrix} ۱ & ۱ \end{bmatrix}, b_{۳} = [۱] \tag{۱۶}$$

به عنوان جمع‌بندی، در لایهٔ اول شبکه که تصمیم‌گیری اولیه نام دارد، اوزان و بایاس زیر را در نظر گرفتیم:

$$w^T = \begin{bmatrix} ۱ & -۱ & ۰ & ۱ & -۱ & ۰ \\ -۱ & ۰ & ۱ & -۱ & ۰ & ۱ \end{bmatrix}, b^T = \begin{bmatrix} ۱ & ۰ & ۰ & ۰ & ۲ & -۱ \end{bmatrix} \tag{۱۷}$$

در ادامه، در لایهٔ پنهان برای عملیات AND داریم:

$$w_۲ = \begin{bmatrix} ۱ & ۱ & ۱ & ۰ & ۰ & ۰ \\ ۰ & ۰ & ۰ & ۱ & ۱ & ۱ \end{bmatrix}, b_۲ = \begin{bmatrix} -۲ & -۲ \end{bmatrix} \tag{۱۸}$$

درنهایت، در لایهٔ انتهایی برای عملیات OR داریم:

$$w_۳ = \begin{bmatrix} ۱ & ۱ \end{bmatrix}, b_۳ = [۱] \tag{۱۹}$$

از نمودار می‌فهمیم که نقطهٔ $(۰/۲۵, -۰/۵)$ باید در مثلث اول، و در نتیجه دستهٔ یک، و نقطهٔ $(۰/۵, ۱)$ باید خارج از هر دو مثلث یعنی دستهٔ دو طبقه‌بندی شود. حال نقاط را با استفاده از دستورات زیر تست می‌کنیم. نتایج ذیل هربخش آورده شده است.

```
import numpy as np

def hardlims(x):
    result = np.zeros((x.shape[0], x.shape[1]))
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if x[i,j] < 0:
                result[i,j] = -1
            else:
                result[i,j] = 1
    return result

w1 = np.array([[1, -1], [-1, 0], [0, 1], [1, -1], [-1, 0], [0, 1]])
b1 = np.array([[1], [0], [0], [0], [2], [-1]])
w2 = np.array([[1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1]])
b2 = np.array([[-2], [-2]])
```

```
w3 = np.array([[1, 1]])
b3 = 1

p1 = np.array([[-0.5], [0.25]])
p2 = np.array([[0.5], [1]])

# Test
a11 = hardlims(np.dot(w1, p1) + b1)
print(a11)

[[ 1.]
 [ 1.]
 [ 1.]
 [-1.]
 [ 1.]
 [-1.]]

a21 = hardlims(np.dot(w1, p2) + b1)
print(a21)

[[ 1.]
 [-1.]
 [ 1.]
 [-1.]
 [ 1.]
 [ 1.]]

w2

array([[1, 1, 1, 0, 0, 0],
       [0, 0, 0, 1, 1, 1]])

a12 = hardlims(np.dot(w2, a11) + b2)
print(a12)

```

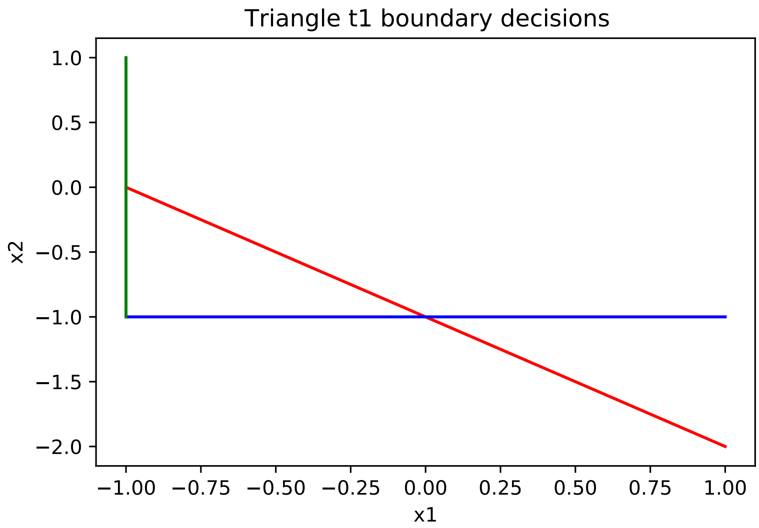
```
[[ 1.]
 [-1.]]

a22 = hardlims(np.dot(w2, a21) + b2)
print(a22)

[[-1.]
 [-1.]]

a13 = hardlims(np.dot(w3, a12) + b3)
print(a13)

[[1.]]

a23 = hardlims(np.dot(w3, a22) + b3)
print(a23)

[[-1.]]
```

ii

در این قسمت دو مثلث با نقاط انتهایی $(-1,0),(-1,-1),(0,-1)$ و $(0,0),(1,0),(1,1)$ مشاهده می‌شود. مثلث نخست که نام t_1 را برای آن برمی‌گزینیم، می‌تواند با خطوط $y_1 = -x_1 - 1, y_2 = -1, x_3 = -1$ تعیین شود. این ناحیهٔ مثلثی‌شکل با بهره‌گیری از دستورات زیر در شکل ۱۷ نشان داده شده است.

```
import matplotlib.pyplot as plt
import numpy as np

# Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
x1 = np.linspace(-1, 1, 100)
y1 = -x1 - 1
y2 = -1*np.ones(100)
x3 = -1*np.ones(100)

```

```python
plt.plot(x1, y1, 'r', x1, y2, 'b', x3, x1, 'g')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Triangle t1 boundary decisions')
plt.savefig('E1121.pdf')
plt.show()
```



شکل ۱۷: مرزهای تصمیم‌گیری قسمت دوم سوال ۱-۱۱ (t_1).

این سه معادلهٔ خط به اوزان زیر می‌انجامد:

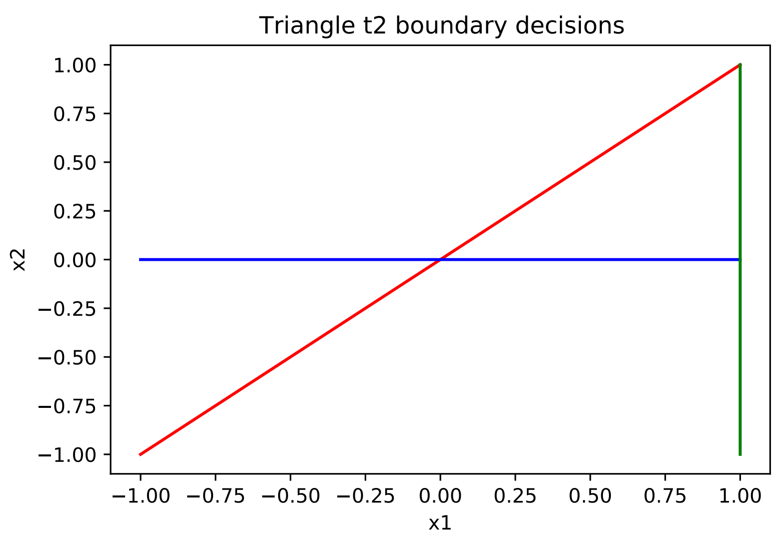
- برای خط $y_1 = -x_1 - 1$، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $-p_1 - p_2 - 1$.
- برای خط $x_2 = -1$ می‌خواهیم هر مقداری در سمت راست خط عمودی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $p_1 + 1$.
- برای خط $y_2 = -1$ می‌خواهیم هر مقداری در بالای خط افقی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $p_2 + 1$.

بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی مثلث به صورت زیر خواهد بود:

$$w^T = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, b^T = \begin{bmatrix} -1 & 1 & 1 \end{bmatrix} \tag{20}$$

از طرف دیگر مثلث دوم با نام t_2 با خطوط $y_1 = x_1, x_2 = 1, y_2 = 0$ ساخته می‌شود. این ناحیه با بهره‌گیری از دستورات زیر در شکل ۱۸ نشان داده شده است.

```python
import matplotlib.pyplot as plt
import numpy as np

# Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
x1 = np.linspace(-1, 1, 100)
y1 = x1
y2 = np.zeros(100)
x2 = 1*np.ones(100)

plt.plot(x1, y1, 'r', x1, y2, 'b', x2, x1, 'g')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Triangle t2 boundary decisions')
plt.savefig('E1122.pdf')
plt.show()
```



شکل ۱۸: مرزهای تصمیم‌گیری قسمت دوم سوال ۱-۱۱ (t_2).

این سه معادلهٔ خط به اوزان زیر می‌انجامد:

- برای خط $y_1 = x_1$، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $p_1 - p_2$.

- برای خط $x_۲ = ۱$ می‌خواهیم هر مقداری در سمت چپ خط عمودی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $-p_۱ + ۱$.
- برای خط $y_۲ = ۰$ می‌خواهیم هر مقداری در بالای خط افقی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $p_۲$.

بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی مثلث به صورت زیر خواهد بود:

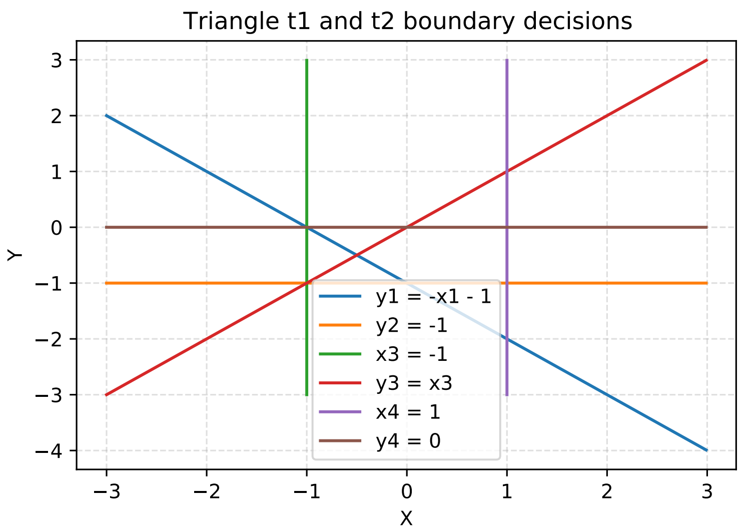
$$w^T = \begin{bmatrix} ۱ & -۱ & ۰ \\ -۱ & ۰ & ۱ \end{bmatrix}, b^T = \begin{bmatrix} ۰ & ۱ & ۰ \end{bmatrix} \tag{۲۱}$$

حال می‌توانیم این مرزهای تصمیم‌گیری را در ماتریس‌های واحد ادغام کنیم (شکل ۱۹).

$$w^T = \begin{bmatrix} -۱ & ۱ & ۰ & ۱ & -۱ & ۰ \\ -۱ & ۰ & ۱ & -۱ & ۰ & ۱ \end{bmatrix}, b^T = \begin{bmatrix} -۱ & ۱ & ۱ & ۰ & ۱ & ۰ \end{bmatrix} \tag{۲۲}$$

```python
import matplotlib.pyplot as plt
import numpy as np

# Create the vectors X and Y
x = np.arange(-3, 3, 0.01)
y1 = -x-1
y2 = -1*np.ones(600)
x3 = -1*np.ones(600)
y3 = x
x4 = 1*np.ones(600)
y4 = np.zeros(600)

# Create the plot
plt.plot(x, y1, label='y1 = -x1 - 1')
plt.plot(x, y2, label='y2 = -1')
plt.plot(x3, y3, label='x3 = -1')
plt.plot(x, y3, label='y3 = x3')
plt.plot(x4, y3, label='x4 = 1')
```

```
plt.plot(x, y4, label='y4 = 0')

# Add a title
plt.title('Triangle t1 and t2 boundary decisions')

# Add X and y Label
plt.xlabel('X')
plt.ylabel('Y')

# Add a grid
plt.grid(alpha=.4,linestyle='--')

# Add a Legend
plt.legend()

# Save the plot
plt.savefig('E1123.pdf')

# Show the plot
plt.show()
```



شکل ۱۹: مرزهای تصمیم‌گیری قسمت دوم سوال ۱-۱۱.

در ادامه به ماتریس وزنی نیاز داریم که مرزهای تصمیم‌گیری را به‌گونه‌ای که به فرم مثلثی دربیایند ترکیب

کند (مرزهای از حالت بی سر و ته بودن درآیند). برای مثلث اول، سه ستون اول ماتریس وزن اول مرزهای را می‌سازند و درنتیجه باید این سه قسمت در نظر گرفته شوند و بقیه نادیده گرفته شوند. این کار می‌تواند با یک سطر به‌صورت $[۱, ۱, ۱, ۰, ۰, ۰]$ انجام گیرد. همین کار را برای مثلث دوم تکرار می‌کنیم. بنابراین، این کار را با استفاده از ماتریس وزن لایه‌ی دوم انجام می‌دهیم:

$$w_{۲} = \begin{bmatrix} ۱ & ۱ & ۱ & ۰ & ۰ & ۰ \\ ۰ & ۰ & ۰ & ۱ & ۱ & ۱ \end{bmatrix} \tag{۲۳}$$

برای تشکیل ماتریس بایاس به این نکته توجه می‌کنیم که نتیجه باید برابر با یک شود؛ اما در هر سطر از ماتریس وزن لایهٔ دوم سه مقدار یک داریم. بنابراین می‌نویسیم:

$$b_{۲} = \begin{bmatrix} -۲ & -۲ \end{bmatrix} \tag{۲۴}$$

حال که به مرزهای تصمیم‌گیری خود فرم دادیم و آن‌ها را تثبیت کردیم، نیاز به یک مکانیزم OR داریم که در صورتی که ورودی در نواحی مثلث‌های اول و دوم افتاد دستهٔ صحیحی تولید کند. ما به این لایه احتیاج داریم تا اگر خروجی پرسپترون AND یک را بازگرداند، یک بازگردد و اگر ۱− را بازگرداند، ۱− بازگردد. این کار می‌تواند با ترکیب دو مقدار ورودی و جمعشان با یک انچام شود؛ در نتیجه داریم:

$$w_{۳} = \begin{bmatrix} ۱ & ۱ \end{bmatrix}, b_{۳} = [۱] \tag{۲۵}$$

به عنوان جمع‌بندی، در لایهٔ اول شبکه که تصمیم‌گیری اولیه نام دارد، اوزان و بایاس زیر را در نظر گرفتیم:

$$w^T = \begin{bmatrix} -۱ & ۱ & ۰ & ۱ & -۱ & ۰ \\ -۱ & ۰ & ۱ & -۱ & ۰ & ۱ \end{bmatrix}, b^T = \begin{bmatrix} -۱ & ۱ & ۱ & ۰ & ۱ & ۰ \end{bmatrix} \tag{۲۶}$$

در ادامه، در لایهٔ پنهان برای عملیات AND داریم:

$$w_{۲} = \begin{bmatrix} ۱ & ۱ & ۱ & ۰ & ۰ & ۰ \\ ۰ & ۰ & ۰ & ۱ & ۱ & ۱ \end{bmatrix}, b_{۲} = \begin{bmatrix} -۲ & -۲ \end{bmatrix} \tag{۲۷}$$

درنهایت، در لایهٔ انتهایی برای عملیات OR داریم:

$$w_۳ = \begin{bmatrix} ۱ & ۱ \end{bmatrix}, b_۳ = [۱] \tag{۲۸}$$

از نمودار می‌فهمیم که نقطهٔ $(-۰/۵, -۰/۵)$ باید در مثلث اول، و در نتیجه دستهٔ یک، و نقطهٔ $(۰/۵, ۱)$ باید خارج از هر دو مثلث یعنی دستهٔ دو طبقه‌بندی شود. حال نقاط را با استفاده از دستورات زیر تست می‌کنیم. نتایج ذیل هربخش آورده شده است. مشاهده می‌شود که هر دو نقطه در دسته‌های درست طبقه‌بندی شده‌اند.

```python
import numpy as np

def hardlims(x):
    result = np.zeros((x.shape[0], x.shape[1]))
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if x[i,j] < 0:
                result[i,j] = -1
            else:
                result[i,j] = 1
    return result

w1 = np.array([[-1, -1], [1, 0], [0, 1], [1, -1], [-1, 0], [0, 1]])
b1 = np.array([[-1], [1], [1], [0], [1], [0]])
w2 = np.array([[1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1]])
b2 = np.array([[-2], [-2]])
w3 = np.array([[1, 1]])
b3 = 1

p1 = np.array([[-0.5], [-0.5]])
p2 = np.array([[0.5], [1]])

# Test
a11 = hardlims(np.dot(w1, p1) + b1)
print(a11)

```

```
[[ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [-1.]]

a21 = hardlims(np.dot(w1, p2) + b1)
print(a21)

[[-1.]
 [ 1.]
 [ 1.]
 [-1.]
 [ 1.]
 [ 1.]]

w2

array([[1, 1, 1, 0, 0, 0],
       [0, 0, 0, 1, 1, 1]])

a12 = hardlims(np.dot(w2, a11) + b2)
print(a12)

[[ 1.]
 [-1.]]

a22 = hardlims(np.dot(w2, a21) + b2)
print(a22)

[[-1.]
 [-1.]]

a13 = hardlims(np.dot(w3, a12) + b3)
```

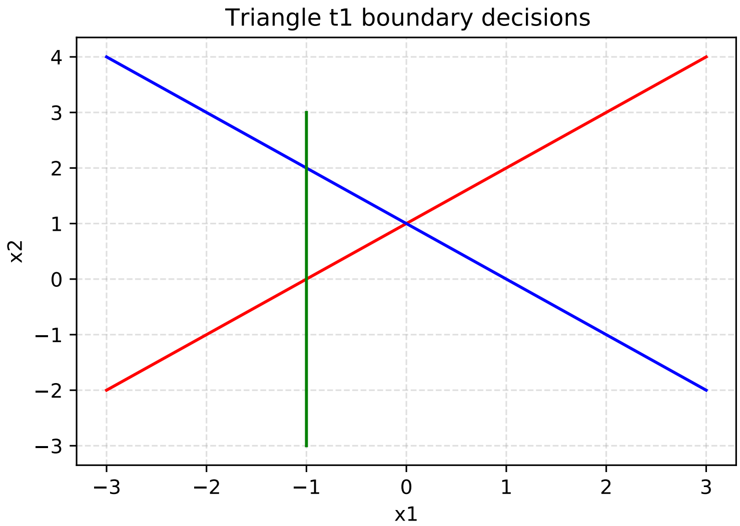
```
print(a13)

[[1.]]

a23 = hardlims(np.dot(w3, a22) + b3)
print(a23)

[[-1.]]
```

iii

در این قسمت سه شکل (شامل دو شکل مثلثی و یک شکل مربعی) با نقاط انتهایی $(۰,۱)$, $(−۱,۲)$, $(−۱,۰)$، $(۱,−۲)$, $(−۱,−۲)$, $(۰,−۱)$ و $(۰,۲)$, $(−۱,−۱)$, $(۱,۱)$, $(۰,۰)$ مشاهده می‌شود. شکل نخست که نام $t_۱$ را برای آن برمی‌گزینیم، می‌تواند با خطوط $y_۱ = x_۱ + ۱, y_۲ = -x_۲ + ۱, x_۳ = -۱$ تعیین شود. این ناحیهٔ مثلثی‌شکل با بهره‌گیری از دستورات زیر در شکل ۲۰ نشان داده شده است.

```
import matplotlib.pyplot as plt
import numpy as np

# Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
x1 = np.linspace(-3, 3, 600)
y1 = x1 + 1
y2 = -x1 + 1
x3 = -1*np.ones(600)

plt.plot(x1, y1, 'r', x1, y2, 'b', x3, x1, 'g')
plt.xlabel('x1')
plt.ylabel('x2')
# Add a grid
plt.grid(alpha=.4,linestyle='--')
plt.title('Triangle t1 boundary decisions')
plt.savefig('E1131.pdf')
plt.show()
```

این سه معادلهٔ خط به اوزان زیر می‌انجامد:



شکل ۲۰: مرزهای تصمیم‌گیری قسمت سوم سوال ۱-۱۱ ($t_۱$).

- برای خط $y_۱ = x_۱ + ۱$، می‌خواهیم تمام نقاط بالای خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $-p_۱ + p_۲ - ۱$.
- برای خط $y_۲ = -x_۲ + ۱$، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $-p_۱ - p_۲ + ۱$.
- برای خط $x_۳ = -۱$ می‌خواهیم هر مقداری در سمت راست خط عمودی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $p_۱ + ۱$.

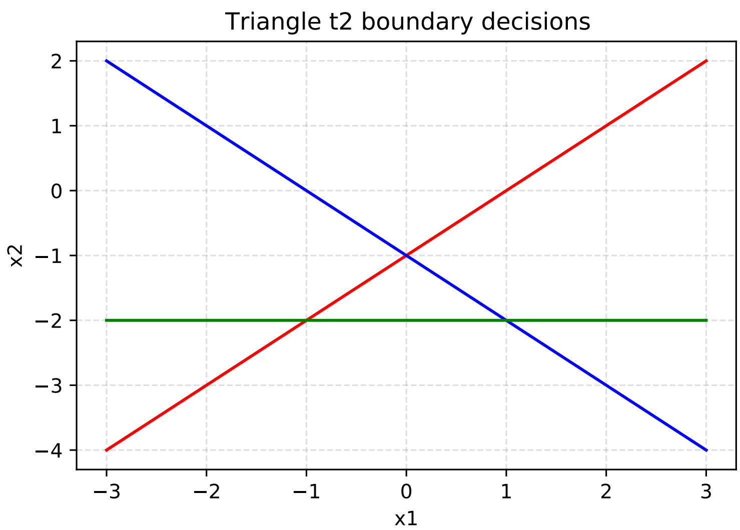
بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی مثلث به صورت زیر خواهد بود:

$$w^T = \begin{bmatrix} -۱ & -۱ & ۱ \\ ۱ & -۱ & ۰ \end{bmatrix}, b^T = \begin{bmatrix} -۱ & ۱ & ۱ \end{bmatrix} \tag{۲۹}$$

از طرف دیگر مثلث دوم با نام $t_۲$ با خطوط $y_۱ = x_۱ - ۱, y_۲ = -x_۲ - ۱, y_۳ = -۲$ ساخته می‌شود. این ناحیه با بهره‌گیری از دستورات زیر در شکل ۲۲ نشان داده شده است.

```
import matplotlib.pyplot as plt
import numpy as np

# Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
x1 = np.linspace(-3, 3, 600)
```

```python
y1 = x1 - 1
y2 = -x1 -1
y3 = -2*np.ones(600)

plt.plot(x1, y1, 'r', x1, y2, 'b', x1, y3, 'g')
plt.xlabel('x1')
plt.ylabel('x2')
# Add a grid
plt.grid(alpha=.4,linestyle='--')
plt.title('Triangle t2 boundary decisions')
plt.savefig('E1132.pdf')
plt.show()
```



شکل ۲۱: مرزهای تصمیم‌گیری قسمت سوم سوال ۱-۱۱ (t_2).

این سه معادلهٔ خط به اوزان زیر می‌انجامد:

- برای خط $y_1 = x_1 - 1$، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $p_1 - p_2 - 1$.
- برای خط $y_1 = -x_2 - 1$، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $-p_1 - p_2 - 1$.
- برای خط $y_3 = -2$ می‌خواهیم هر مقداری در بالای خط افقی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $p_2 + 2$.

بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی مثلث به صورت زیر خواهد بود:

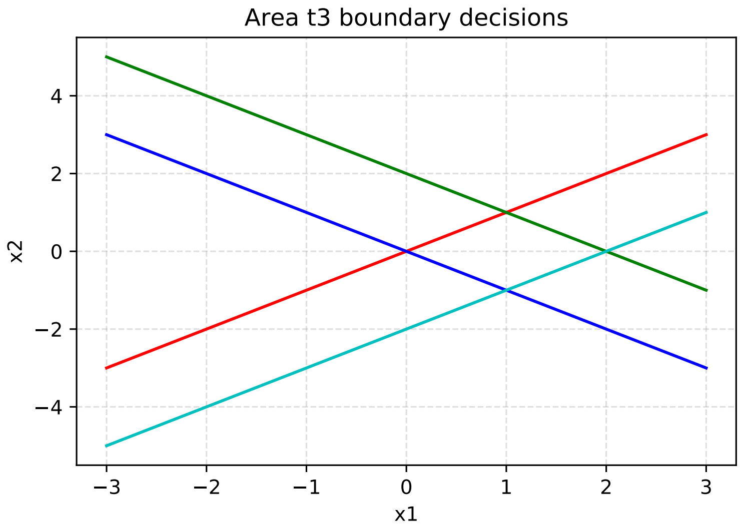
$$w^T = \begin{bmatrix} 1 & -1 & 0 \\ -1 & -1 & 1 \end{bmatrix}, b^T = \begin{bmatrix} -1 & -1 & 2 \end{bmatrix} \tag{30}$$

ناحیهٔ سوم با نام t_3 با خطوط $y_1 = x_1, y_2 = -x_2 y_3 = -x_3 + 2, y_4 = x_4 - 2$ ساخته می‌شود. این ناحیه با بهره‌گیری از دستورات زیر در ؟؟ نشان داده شده است.

```python
import matplotlib.pyplot as plt
import numpy as np

# Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
x1 = np.linspace(-3, 3, 600)
y1 = x1
y2 = -x1
y3 = -x1 + 2
y4 = x1 -2

plt.plot(x1, y1, 'r', x1, y2, 'b', x1, y3, 'g', x1, y4, 'c')
plt.xlabel('x1')
plt.ylabel('x2')
# Add a grid
plt.grid(alpha=.4,linestyle='--')
plt.title('Area t3 boundary decisions')
plt.savefig('E1133.pdf')
plt.show()
```

این معادلات به اوزان زیر می‌انجامد:

- برای خط $y_1 = x_1$، می‌خواهیم تمام نقاط زیر (راست) خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $p_1 - p_2$.
- برای خط $y_1 = -x_2$، می‌خواهیم تمام نقاط بالای خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $-p_1 + p_2$.
- برای خط $y = -x + 2$، می‌خواهیم تمام نقاط بالای خط یک شوند؛ چراکه، می‌خواهیم مساحت



شکل ۲۲: مرزهای تصمیم‌گیری قسمت سوم سوال ۱-۱۱ ($t_۳$).

مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $-p_۱ - p_۲ + ۲$.

- برای خط $y = x - ۲$، می‌خواهیم تمام نقاط بالای خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $-p_۱ = +p_۲ + ۲$.

بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی ناحیه به صورت زیر خواهد بود:

$$w^T = \begin{bmatrix} ۱ & ۱ & -۱ & -۱ \\ -۱ & ۱ & -۱ & ۱ \end{bmatrix}, b^T = \begin{bmatrix} ۰ & ۰ & ۲ & ۲ \end{bmatrix} \tag{۳۱}$$

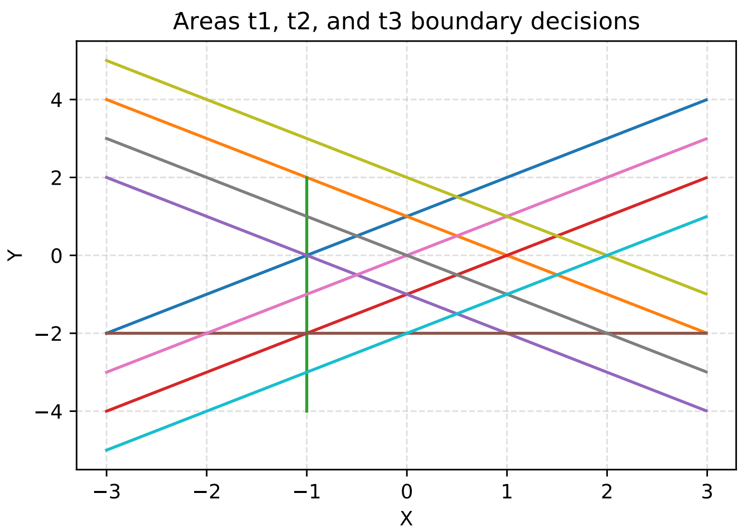
حال می‌توانیم این مرزهای تصمیم‌گیری را در ماتریس‌های واحد ادغام کنیم (شکل ۲۳).

$$w^T = \begin{bmatrix} -۱ & -۱ & ۱ & ۱ & -۱ & ۰ & ۱ & ۱ & -۱ & -۱ \\ ۱ & -۱ & ۰ & -۱ & -۱ & ۱ & -۱ & ۱ & -۱ & ۱ \end{bmatrix}, b^T = \begin{bmatrix} -۱ & ۱ & ۱ & -۱ & -۱ & ۲ & ۰ & ۰ & ۲ & ۲ \end{bmatrix} \tag{۳۲}$$

```python
import matplotlib.pyplot as plt
import numpy as np

# Create the vectors X and Y
x = np.arange(-3, 3, 0.01)
```

```python
y1 = x + 1
y2 = -x + 1
x3 = -1*np.ones(600)
y3 = x - 1
y4 = -x -1
y5 = -2*np.ones(600)
y6 = x
y7 = -x
y8 = -x + 2
y9 = x -2

# Create the plot
plt.plot(x, y1, label='y1 = -x1 - 1')
plt.plot(x, y2, label='y2 = -1')
plt.plot(x3, y3, label='x3 = -1')
plt.plot(x, y3, label='y3 = x3')
plt.plot(x4, y3, label='x4 = 1')
plt.plot(x, y4, label='y4 = 0')
plt.plot(x, y5, label='y1 = -x1 - 1')
plt.plot(x, y6, label='y2 = -1')
plt.plot(x, y7, label='x3 = -1')
plt.plot(x, y8, label='y3 = x3')
plt.plot(x, y9, label='x4 = 1')
# plt.plot(x, y4, label='y4 = 0')

# Add a title
plt.title('Triangle t1 and t2 boundary decisions')

# Add X and y Label
plt.xlabel('X')
plt.ylabel('Y')

# Add a grid
plt.grid(alpha=.4,linestyle='--')

```

```python
# Add a Legend
# plt.legend()

# Save the plot
plt.savefig('E1134.pdf')

# Show the plot
plt.show()
```



شکل ۲۳: مرزهای تصمیم‌گیری قسمت سوم سوال ۱-۱۱.

در ادامه به ماتریس وزنی نیاز داریم که مرزهای تصمیم‌گیری را به‌گونه‌ای که به فرم مثلثی دربیایند ترکیب کند (مرزهای از حالت بی سر و ته بودن درآیند). برای مثلث اول، سه ستون اول ماتریس وزن اول مرزهای را می‌سازند و درنتیجه باید این سه قسمت در نظر گرفته شوند و بقیه نادیده گرفته شوند. این کار می‌تواند با یک سطر به‌صورت $[1, 1, 1, 0, 0, 0]$انجام گیرد. همین کار را برای مثلث دوم تکرار می‌کنیم. بنابراین، این کار را با استفاده از ماتریس وزن لایه‌ی دوم انجام می‌دهیم:

$$w_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad (33)$$

برای تشکیل ماتریس بایاس به این نکته توجه می‌کنیم که نتیجه باید برابر با یک شود؛ اما در هر سطر از

ماتریس وزن لایهٔ دوم سه یا چهار مقدار یک داریم. بنابراین می‌نویسیم:

$$b_۲ = \begin{bmatrix} -۲ & -۲ & -۳ \end{bmatrix} \tag{۳۴}$$

حال که به مرزهای تصمیم‌گیری خود فرم دادیم و آن‌ها را تثبیت کردیم، نیاز به یک مکانیزم OR داریم که در صورتی که ورودی در نواحی مثلث‌های اول و دوم افتاد دستهٔ صحیحی تولید کند. ما به این لایه احتیاج داریم تا اگر خروجی پرسپترون AND یک را بازگرداند، یک بازگردد و اگر -۱ را بازگرداند، -۱ بازگردد. این کار می‌تواند با ترکیب دو مقدار ورودی و جمعشان با یک انچام شود؛ در نتیجه داریم:

$$w_۳ = \begin{bmatrix} ۱ & ۱ & ۱ \end{bmatrix}, b_۳ = [۲] \tag{۳۵}$$

به عنوان جمع‌بندی، در لایهٔ اول شبکه که تصمیم‌گیری اولیه نام دارد، اوزان و بایاس زیر را در نظر گرفتیم:

$$w^T = \begin{bmatrix} -۱ & -۱ & ۱ & ۱ & -۱ & ۰ & ۱ & ۱ & -۱ & -۱ \\ ۱ & -۱ & ۰ & -۱ & -۱ & ۱ & -۱ & ۱ & -۱ & ۱ \end{bmatrix}, b^T = \begin{bmatrix} -۱ & ۱ & ۱ & -۱ & -۱ & ۲ & ۰ & ۰ & ۲ & ۲ \end{bmatrix} \tag{۳۶}$$

در ادامه، در لایهٔ پنهان برای عملیات AND داریم:

$$w_۲ = \begin{bmatrix} ۱ & ۱ & ۱ & ۰ & ۰ & ۰ & ۰ & ۰ & ۰ & ۰ \\ ۰ & ۰ & ۰ & ۱ & ۱ & ۱ & ۰ & ۰ & ۰ & ۰ \\ ۰ & ۰ & ۰ & ۰ & ۰ & ۰ & ۱ & ۱ & ۱ & ۱ \end{bmatrix}, b_۲ = \begin{bmatrix} -۲ & -۲ & -۳ \end{bmatrix} \tag{۳۷}$$

درنهایت، در لایهٔ انتهایی برای عملیات OR داریم:

$$w_۳ = \begin{bmatrix} ۱ & ۱ & ۱ \end{bmatrix}, b_۳ = [۲] \tag{۳۸}$$

از نمودار می‌فهمیم که نقطهٔ $(۰/۵, ۰/۵-)$ باید در مثلث اول، و در نتیجه دستهٔ یک، و نقطهٔ $(۱-, ۲-)$ باید خارج از هر دو مثلث یعنی دستهٔ دو طبقه‌بندی شود. حال نقاط را با استفاده از دستورات زیر تست می‌کنیم. نتایج ذیل هربخش آورده شده است. همان‌طور که مشاهده می‌شود نقطهٔ اول یک و درون نواحی و نقطهٔ دوم -۱ و خارج از نواحی تشخیص داده شده است.

```python
import numpy as np

def hardlims(x):
    result = np.zeros((x.shape[0], x.shape[1]))
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if x[i,j] < 0:
                result[i,j] = -1
            else:
                result[i,j] = 1
    return result

w1 = np.array([[-1, 1], [-1, -1], [1, 0], [1, -1], [-1, -1], [0, 1], [1, -1],
    [1, 1], [-1, -1], [-1, 1]])
b1 = np.array([[-1], [1], [1], [-1], [-1], [2], [0], [0], [2], [2]])
w2 = np.array([[1, 1, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]])
b2 = np.array([[-2], [-2], [-3]])
w3 = np.array([[1, 1, 1]])
b3 = 2

p1 = np.array([[-0.5], [0.5]])
p2 = np.array([[-2], [-1]])

# Test
a11 = hardlims(np.dot(w1, p1) + b1)
print(a11)

[[ 1.]
 [ 1.]
 [ 1.]
 [-1.]
 [-1.]
 [ 1.]
 [-1.]
```

```
 [ 1.]
 [ 1.]
 [ 1.]]

a21 = hardlims(np.dot(w1, p2) + b1)
print(a21)

[[ 1.]
 [ 1.]
 [-1.]
 [-1.]
 [ 1.]
 [ 1.]
 [-1.]
 [-1.]
 [ 1.]
 [ 1.]]

w2

array([[1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]])

a12 = hardlims(np.dot(w2, a11) + b2)
print(a12)

[[ 1.]
 [-1.]
 [-1.]]

a22 = hardlims(np.dot(w2, a21) + b2)
print(a22)

[[-1.]
```

```
 [-1.]
 [-1.]]

a13 = hardlims(np.dot(w3, a12) + b3)
print(a13)

[[1.]]

a23 = hardlims(np.dot(w3, a22) + b3)
print(a23)

[[-1.]]
```

iv

در این قسمت دو مثلث با نقاط انتهایی $(0,0),(-1,1),(-2,1)$ و $(2,1),(1,-1),(1,0)$ مشاهده می‌شود. مثلث نخست که نام t_1 را برای آن برمی‌گزینیم، می‌تواند با خطوط $y=-2x, y=1, y=-x$ تعیین شود. این سه معادلهٔ خط به اوزان زیر می‌انجامد:

- برای خط $y=-x$، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $-p_1-p_2$.
- برای خط $y=-0/5x$، می‌خواهیم تمام نقاط بالای خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $0/5p_1+p_2$.
- برای خط $y=1$ می‌خواهیم هر مقداری در زیر خط افقی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $1-p_2$.

بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی مثلث به صورت زیر خواهد بود:

$$w^T=\begin{bmatrix}-1 & 0/5 & 0\\ -1 & 1 & -1\end{bmatrix}, b^T=\begin{bmatrix}0 & 0 & 1\end{bmatrix} \tag{39}$$

از طرف دیگر مثلث دوم با نام t_2 با خطوط $x = 1, y = \frac{2}{3}x - \frac{1}{3}, y = x - 1$ ساخته می‌شود. این سه معادلهٔ خط به اوزان زیر می‌انجامد:

- برای خط $y = x - 1$، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $p_1 - p_2 - 1$.
- برای خط $y = \frac{2}{3}x - \frac{1}{3}$، می‌خواهیم تمام نقاط بالای خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم: $p_2 - \frac{2}{3}p_1 + \frac{1}{3}$.
- برای خط $x = 1$ می‌خواهیم هر مقداری در سمت راست خط عمودی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم $p_1 - 1$.

بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی مثلث به صورت زیر خواهد بود:

$$w^T = \begin{bmatrix} 1 & -\frac{2}{3} & 1 \\ -1 & 1 & 0 \end{bmatrix}, b^T = \begin{bmatrix} -1 & \frac{1}{3} & -1 \end{bmatrix} \tag{40}$$

حال می‌توانیم این مرزهای تصمیم‌گیری را در ماتریس‌های واحد ادغام کنیم.

$$w^T = \begin{bmatrix} -1 & 0/5 & 0 & 1 & -\frac{2}{3} & 1 \\ 0 & 0 & 1 & -1 & 1 & 0 \end{bmatrix}, b^T = \begin{bmatrix} 0 & 0 & 1 & -1 & \frac{1}{3} & -1 \end{bmatrix} \tag{41}$$

در ادامه به ماتریس وزنی نیاز داریم که مرزهای تصمیم‌گیری را به‌گونه‌ای که به فرم مثلثی دربیایند ترکیب کند (مرزهای از حالت بی سر و ته بودن درآیند). برای مثلث اول، سه ستون اول ماتریس وزن اول مرزهای را می‌سازند و درنتیجه باید این سه قسمت در نظر گرفته شوند و بقیه نادیده گرفته شوند. این کار می‌تواند با یک سطر به‌صورت $[1, 1, 1, 0, 0, 0]$انجام گیرد. همین کار را برای مثلث دوم تکرار می‌کنیم. بنابراین، این کار را با استفاده از ماتریس وزن لایه‌ی دوم انجام می‌دهیم:

$$w_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{42}$$

برای تشکیل ماتریس بایاس به این نکته توجه می‌کنیم که نتیجه باید برابر با یک شود؛ اما در هر سطر از ماتریس وزن لایهٔ دوم سه مقدار یک داریم. بنابراین می‌نویسیم:

$$b_۲ = \begin{bmatrix} -۲ & -۲ \end{bmatrix} \tag{۴۳}$$

حال که به مرزهای تصمیم‌گیری خود فرم دادیم و آن‌ها را تثبیت کردیم، نیاز به یک مکانیزم OR داریم که در صورتی که ورودی در نواحی مثلث‌های اول و دوم افتاد دستهٔ صحیحی تولید کند. ما به این لایه احتیاج داریم تا اگر خروجی پرسپترون AND یک را بازگرداند، یک بازگردد و اگر −۱ را بازگرداند، −۱ بازگردد. این کار می‌تواند با ترکیب دو مقدار ورودی و جمعشان با یک انچام شود؛ در نتیجه داریم:

$$w_۳ = \begin{bmatrix} ۱ & ۱ \end{bmatrix}, b_۳ = [۱] \tag{۴۴}$$

به عنوان جمع‌بندی، در لایهٔ اول شبکه که تصمیم‌گیری اولیه نام دارد، اوزان و بایاس زیر را در نظر گرفتیم:

$$w^T = \begin{bmatrix} -۱ & ۲ & ۰ & ۱ & -\frac{۲}{۳} & ۱ \\ ۰ & ۰ & ۱ & -۱ & ۱ & ۰ \end{bmatrix}, b^T = \begin{bmatrix} ۰ & ۰ & ۱ & -۱ & \frac{۱}{۳} & -۱ \end{bmatrix} \tag{۴۵}$$

در ادامه، در لایهٔ پنهان برای عملیات AND داریم:

$$w_۲ = \begin{bmatrix} ۱ & ۱ & ۱ & ۰ & ۰ & ۰ \\ ۰ & ۰ & ۰ & ۱ & ۱ & ۱ \end{bmatrix}, b_۲ = \begin{bmatrix} -۲ & -۲ \end{bmatrix} \tag{۴۶}$$

درنهایت، در لایهٔ انتهایی برای عملیات OR داریم:

$$w_۳ = \begin{bmatrix} ۱ & ۱ \end{bmatrix}, b_۳ = [۱] \tag{۴۷}$$

از نمودار می‌فهمیم که نقطهٔ (−۱, ۰٫۵) باید در مثلث اول، و در نتیجه دستهٔ یک، و نقطهٔ (۰, −۱) باید خارج از هر دو مثلث یعنی دستهٔ دو طبقه‌بندی شود. حال نقاط را با استفاده از دستورات زیر تست می‌کنیم. نتایج ذیل هربخش آورده شده است. مشاهده می‌شود که هر دو نقطه در دسته‌های درست طبقه‌بندی شده‌اند.

```
import numpy as np

```

```python
def hardlims(x):
    result = np.zeros((x.shape[0], x.shape[1]))
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if x[i,j] < 0:
                result[i,j] = -1
            else:
                result[i,j] = 1
    return result

w1 = np.array([[-1, 0], [0.5, 0], [0, 1], [1, -1], [-2/3, 1], [1, 0]])
b1 = np.array([[0], [0], [1], [-1], [1/3], [-1]])
w2 = np.array([[1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1]])
b2 = np.array([[-2], [-2]])
w3 = np.array([[1, 1]])
b3 = 1

p1 = np.array([[-1], [0.5]])
p2 = np.array([[0], [-1]])

# Test
a11 = hardlims(np.dot(w1, p1) + b1)
print(a11)

a21 = hardlims(np.dot(w1, p2) + b1)
print(a21)

w2

a12 = hardlims(np.dot(w2, a11) + b2)
print(a12)

a22 = hardlims(np.dot(w2, a21) + b2)
print(a22)

```

```python
a13 = hardlims(np.dot(w3, a12) + b3)
print(a13)

a23 = hardlims(np.dot(w3, a22) + b3)
print(a23)
```

توضیح پوشهٔ کدها

در پوشهٔ اصلی این سوال (Q8__11_01)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند. هم‌چنین فایل اجرای برخط کدها از طریق پیوند گوگل‌کولب در دسترس است.

پاسخ سوال ۱۱.۲

i

برای این قسمت از دستورات زیر استفاده می‌کنیم و نتایج مطابق شکل ۲۴ و شکل ۲۵ خواهد بود که نشان می‌دهد هدف سوال به خوبی به سرانجام رسیده است.

```matlab
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
% Created 18-Jan-2023 11:57:05
%
% This script assumes these variables are defined:
%
%   TestInputs - input data.
%   TestTargets - target data.
clc
close all
clear all

%% Load Data

x = [-2 -1.5 -1 0.5 0 0.5 1 1.5 2];
t = [-2 -2 -1 0 0 0 1 2 2];
```

```
% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize = 4;
net = fitnet(hiddenLayerSize,trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 10/100;
net.divideParam.testRatio = 10/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean Squared Error

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit'};

% Train the Network
```

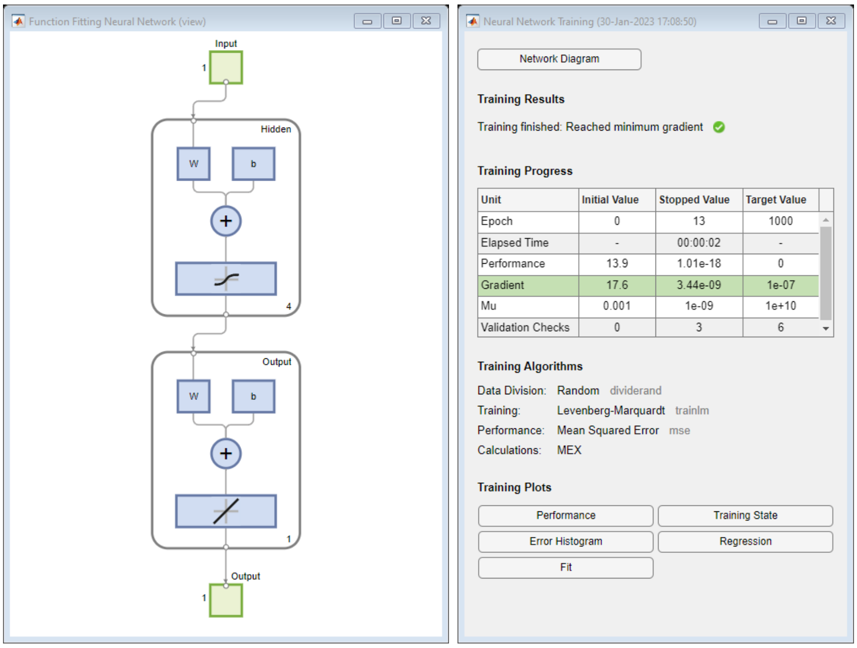
```
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
```

```
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end

---------------------------------------------------------------------

performance =
    0.0016

trainPerformance =
   2.3843e-05

valPerformance =
    0.0124

testPerformance =
    0.0014
```

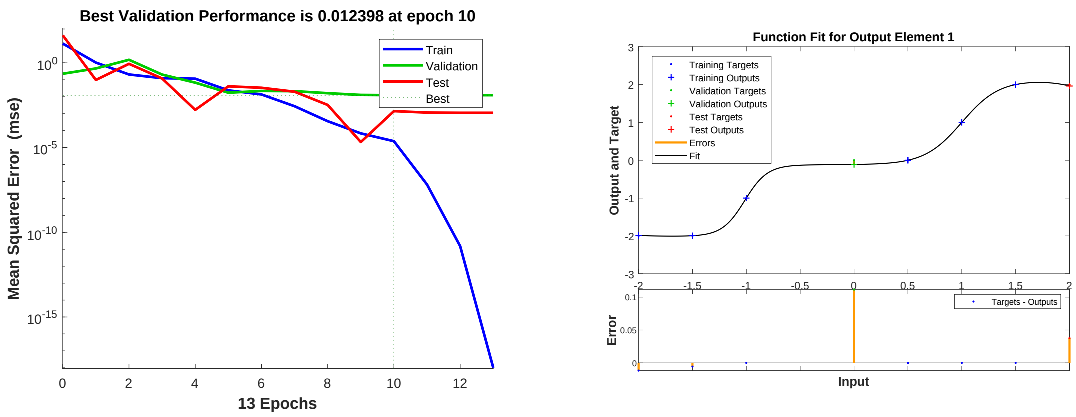
ii

برای این قسمت از دستورات زیر استفاده می‌کنیم و نتایج مطابق شکل ۲۶ و شکل ۲۷ خواهد بود که نشان می‌دهد هدف سوال به خوبی به سرانجام رسیده است.

```
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
```



شکل ۲۴: نتیجهٔ سوال ۱-۲-۱۱.



(ب) (آ)

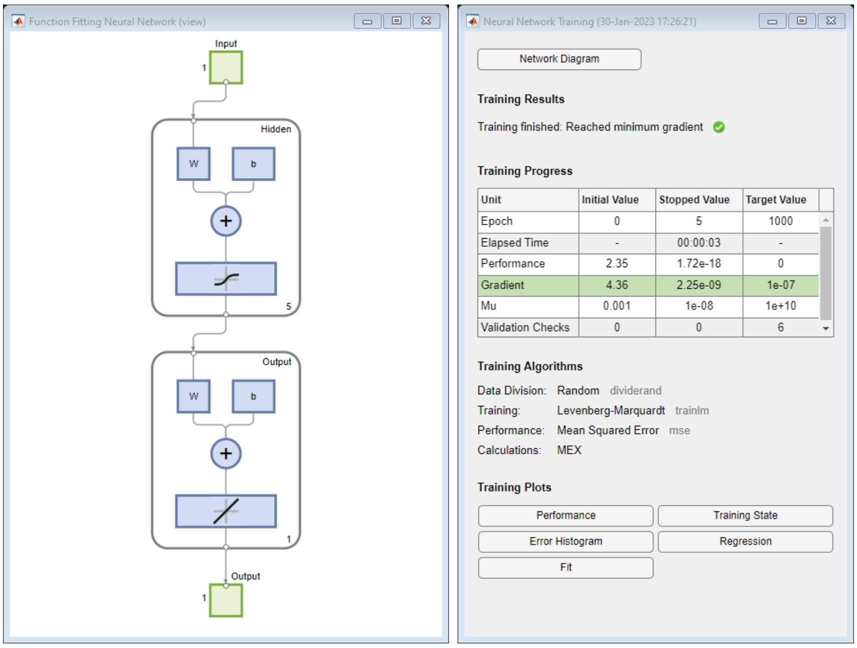
شکل ۲۵: نتیجهٔ سوال ۱-۲-۱۱.

```
% Created 18-Jan-2023 11:57:05
%
% This script assumes these variables are defined:
%
%   TestInputs - input data.
%   TestTargets - target data.
clc
close all
```

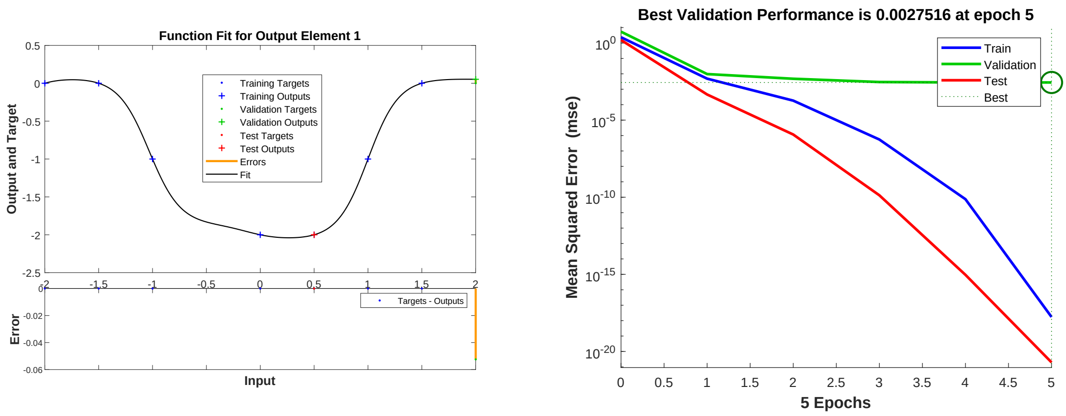
```
clear all

%% Load Data

x = [-2 -1.5 -1 0.5 0 0.5 1 1.5 2];
t = [0 0 -1 -2 -2 -2 -1 0 0];

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize =5;
net = fitnet(hiddenLayerSize,trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 10/100;
net.divideParam.testRatio = 10/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean Squared Error

```

```
% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
```

```
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end
-------------------------------------------------------------------

performance =
   3.0573e-04

trainPerformance =
   1.7216e-18

valPerformance =
    0.0028

testPerformance =
   1.9680e-21
```



شکل ۲۶: نتیجهٔ سوال ۲-۲-۱۱.



(ب) (آ)

شکل ۲۷: نتیجهٔ سوال ۲-۲-۱۱.

iii

برای این قسمت از دستورات زیر استفاده می‌کنیم و نتایج مطابق شکل ۲۸ و شکل ۲۹ خواهد بود که نشان می‌دهد هدف سوال به خوبی به سرانجام رسیده است.

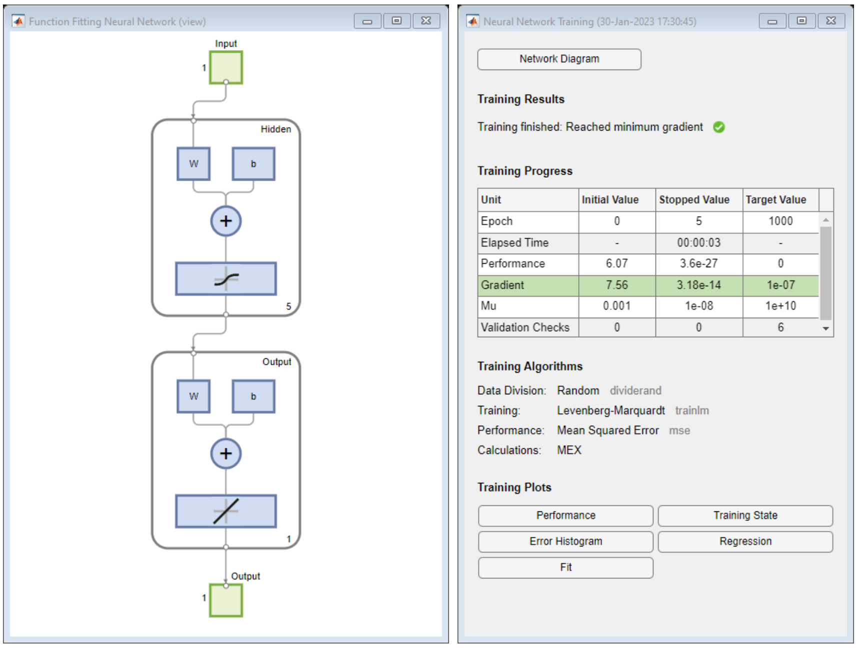
```
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
% Created 18-Jan-2023 11:57:05
%
% This script assumes these variables are defined:
```

```
%
%   TestInputs - input data.
%   TestTargets - target data.
clc
close all
clear all

%% Load Data

x = [-2 -1.5 -1 0.5 0 0.5 1 1.5 2];
t = [0 0 1 2 2 2 1  0 0];

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize = 5;
net = fitnet(hiddenLayerSize,trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 10/100;
net.divideParam.testRatio = 10/100;
```

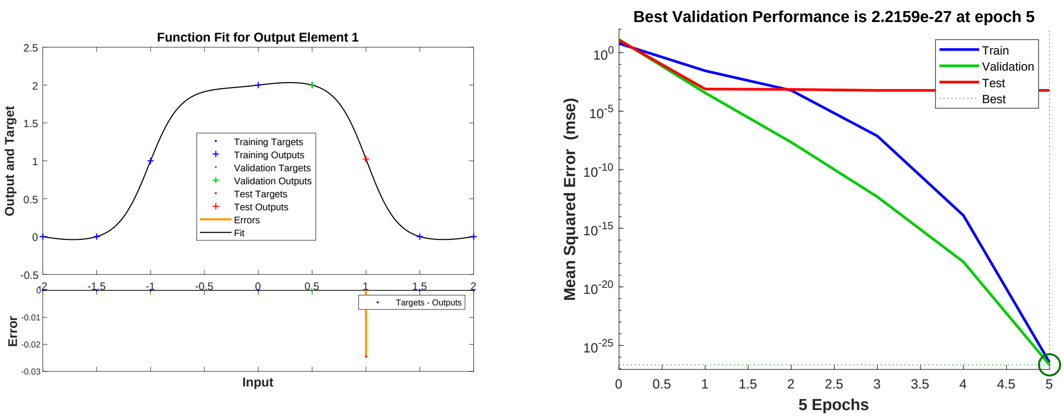
```
% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean Squared Error

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
```

```
%figure, plotfit(net,x,t)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end
--------------------------------------------------------------------

performance =
   6.6775e-05

trainPerformance =
   3.6007e-27

valPerformance =
   2.2159e-27

```

```
testPerformance =
    6.0097e-04
```



شکل ۲۸: نتیجهٔ سوال ۳-۲-۱۱.



(ب) (آ)

شکل ۲۹: نتیجهٔ سوال ۳-۲-۱۱.

iv

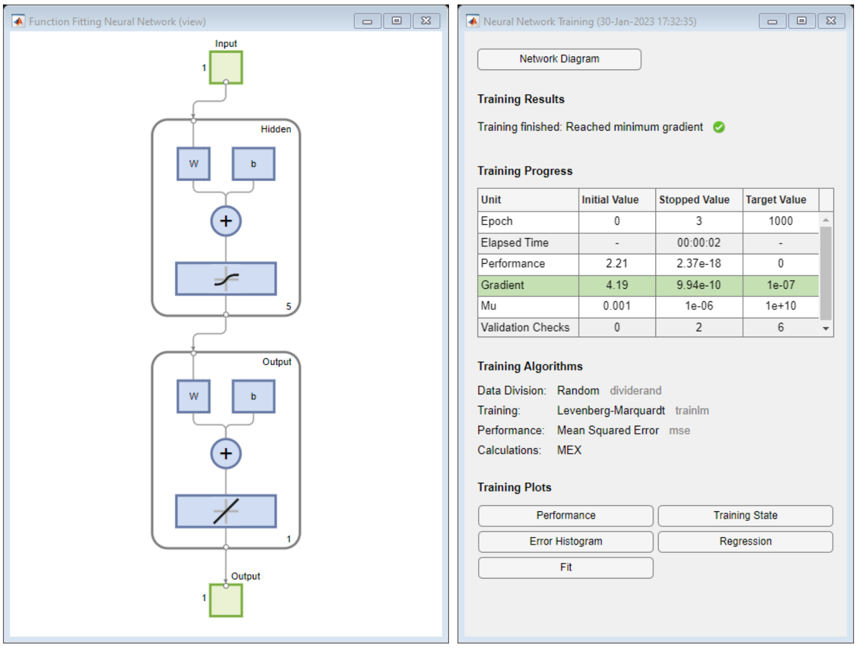
برای این قسمت از دستورات زیر استفاده می‌کنیم و نتایج مطابق شکل ۳۰ و شکل ۳۱ خواهد بود که نشان می‌دهد هدف سوال به خوبی به سرانجام رسیده است.

```
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
% Created 18-Jan-2023 11:57:05
%
% This script assumes these variables are defined:
%
%   TestInputs - input data.
%   TestTargets - target data.
clc
close all
clear all

%% Load Data

x = [-2 -1.5 -1 0.5 0 0.5 1 1.5 2];
t = [0 0 1 2 2 2 1  0 0];

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize = 5;
net = fitnet(hiddenLayerSize,trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
```

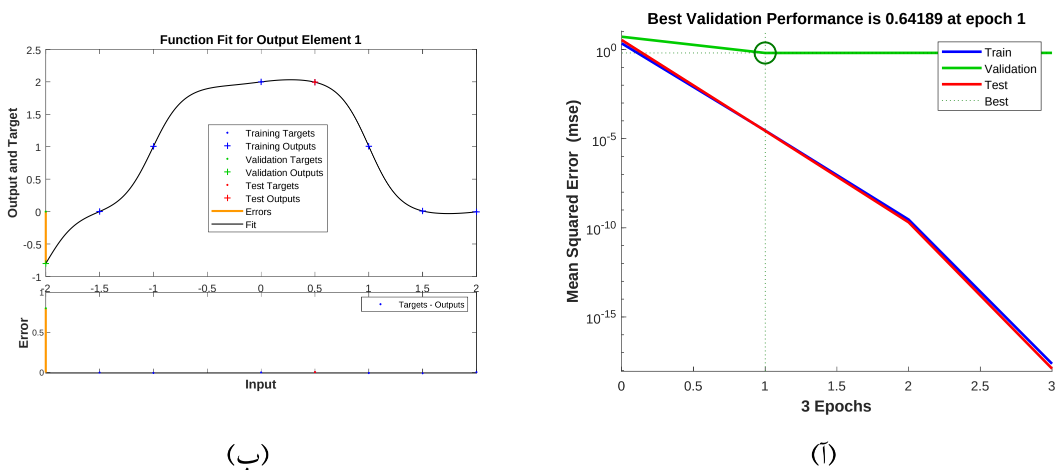
```
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 10/100;
net.divideParam.testRatio = 10/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean Squared Error

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
view(net)

% Plots
```

```
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end
-------------------------------------------------------------------

performance =
    0.0713

trainPerformance =
```

```
    2.8563e-05

valPerformance =

     0.6419

testPerformance =

    2.7548e-05
```



شکل ۳۰: نتیجهٔ سوال ۴-۲-۱۱.



شکل ۳۱: نتیجهٔ سوال ۴-۲-۱۱.

توضیح پوشهٔ کدها

در پوشهٔ اصلی این سوال (`Q9__11_02`)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند.

پاسخ سوال ۱۱.۱۴

با درنظرگرفتن توابع تبدیل هر لایه مشتقات آن‌ها را محاسبه می‌کنیم:

$$f^{1}(n) = \frac{1}{1+e^{-n}} \Longrightarrow \dot{f}^{1}(n) = \frac{1}{1+e^{n}}$$
$$f^{2}(n) = n \Longrightarrow \dot{f}^{2}(n) = 1 \tag{48}$$

ورودی به‌صورت $p = 1$ داده شده و بایاس‌ها و اوزان به‌صورت رابطهٔ ۴۹ تعریف می‌شوند.

$$w^{1}(0) = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad b^{1}(0) = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad w^{2}(0) = \begin{bmatrix} 2 & 3 \end{bmatrix} \quad b^{2}(0) = [1] \tag{49}$$

i

برای مسیر پیش‌رو می‌نویسیم:

$$a^{0} = 1$$
$$a^{1} = \log\operatorname{sig}\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) = \begin{bmatrix} 0.7311 \\ 0.8808 \end{bmatrix} \tag{50}$$
$$a_{2} = \operatorname{pureline}\left(n^{2}\right) = n^{2} \qquad \Rightarrow \frac{\partial a^{2}}{\partial n^{2}} = 1$$

$$\frac{\partial a^{2}}{\partial n^{1}_{1}} = \frac{\partial a^{2}}{\partial n^{2}} \times \frac{\partial n^{2}}{\partial n^{1}_{1}} = 1 \times \left(w^{2}_{1} \dot{f}^{1}\left(n^{1}\right)\right) = 2 \times \begin{bmatrix} 0 & 0.1192 \end{bmatrix} = \begin{bmatrix} 0 & 0.2389 \end{bmatrix}$$
$$\frac{\partial a^{2}}{\partial n^{1}_{2}} = \frac{\partial a^{2}}{\partial n^{2}} \times \frac{\partial n^{2}}{\partial n^{1}_{2}} = 1 \times \left(w^{2}_{2} \dot{f}^{1}\left(n^{1}_{2}\right)\right) = 3 \times \begin{bmatrix} 0 & 0.1192 \end{bmatrix} = \begin{bmatrix} 0 & 0.7152 \end{bmatrix} \tag{51}$$

ii

می‌نویسیم:

$$\frac{\partial a^{۲}}{\partial p}=\frac{\partial a^{۲}}{\partial n_{۱}^{۱}}\times\frac{\partial n_{۱}^{۱}}{\partial p}+\frac{\partial a^{۲}}{\partial n_{۲}^{۱}}\times\frac{\partial n_{۲}^{۱}}{\partial p}=\begin{bmatrix} ۰ & ۰٫۲۳۸۴ \end{bmatrix}\times ۲+\begin{bmatrix} ۰ & ۰٫۷۱۵۲ \end{bmatrix}\times ۱=\begin{bmatrix} ۰ & ۱٫۱۹۲ \end{bmatrix} \quad (۵۲)$$

پاسخ سوال ۱۱.۲۵

راه‌حل اول

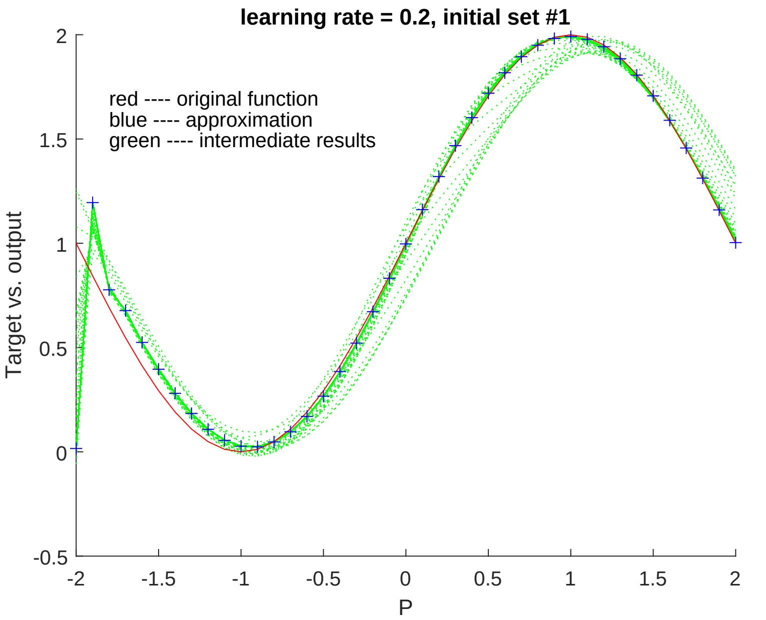
برای حل این سوال از دستورات زیر استفاده می‌کنیم و نتیجه به صورتی خواهد بود که در ؟؟ آمده است.

```
clc
close all
clear all

tic

%Initialize data
W1 = rand(2,1) - 0.5;
W2 = rand(1,2) - 0.5;
b1 = rand(2,1) - 0.5;
b2 = rand - 0.5;
a1 = zeros(2,1);

%Output the initial set
W1_0 = W1
b1_0 = b1
W2_0 = W2
b2_0 = b2

alfa = 0.2;     %learning rate
tol = 0.001;    %tol: tolerance
mse = 1;        %mse: mean square error
```

```
iter = 0;

figure;
while (mse > tol && iter<=1500)
   mse = 0;
   i = 0;
   iter = iter + 1;
   for P = -2 : .1 :2
      i = i + 1;
      T = 1 + sin(pi*P/2);
      a1 = logsig(W1*P + b1);
    a2 = purelin(W2*a1 + b2);
      mse = mse + (T - a2)^2;
      A(i) = a2;

    dlogsig = [(1 - a1(1))* a1(1) 0;0 (1 - a1(2))* a1(2)];
    s2 = -2 * (T - a2);
    s1 = dlogsig * W2' * s2;

    W2 = W2 - alfa * s2 * a1';
    W1 = W1 - alfa * s1 * P;
    b2 = b2 - alfa * s2;
      b1 = b1 - alfa * s1;
   end
   P = -2 : .1 : 2;
   if (mod(iter,10) == 0)
      plot(P,A,'g:')
   end
   hold on;
end

%Display in graph
P = -2 : .1 : 2;
T = 1 + sin(pi*P/2);
%figure;
```

```
plot(P,T,'r-',P,A,'b+')
title('Fig6.1 learning rate = 0.2, initial set #1');
text(-1.8,1.7,'red ---- original function');
text(-1.8,1.6,'blue ---- approximation');
text(-1.8,1.5,'green ---- intermediate results');
xlabel('P'), ylabel('Target vs. output');
W1
b1
W2
b2
iter

toc
----------------------------------------------------

W1_0 =
    0.1892
    0.2482

b1_0 =
   -0.2710
    0.4133

W2_0 =
   -0.0495   -0.4162

b2_0 =
   -0.3476

W1 =
   -1.7262
   -2.6171

b1 =
    3.3501
```

```
    0.5607

W2 =
    3.6560   -2.9336

b2 =
   -0.7111

iter =
        1501

Elapsed time is 1.047967 seconds.
```



شکل ۳۲: نتیجهٔ سوال ۲۵-۱۱.

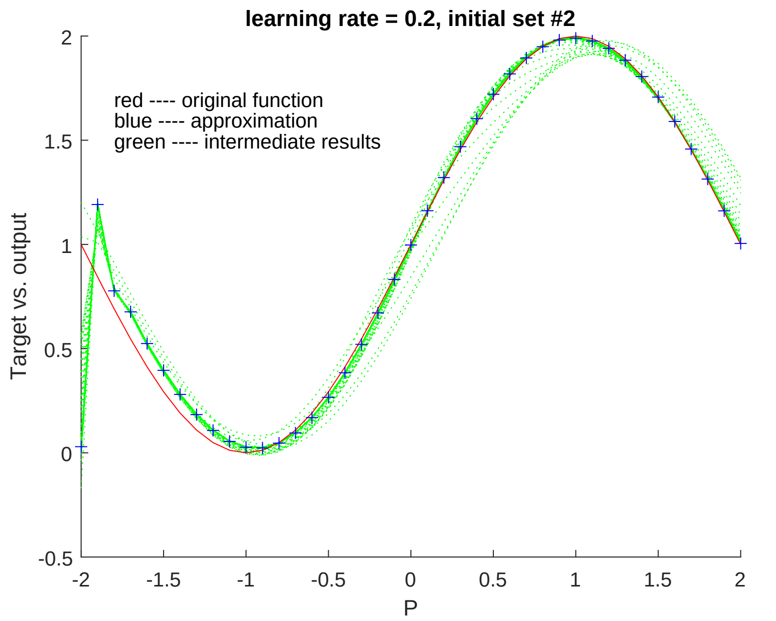
با تغییر مقادیر اولیه داریم:

```
W1_0 =
    0.1477
   -0.0491

b1_0 =
    0.2447
```

```
   -0.3110

W2_0 =
    0.0470   -0.2037

b2_0 =
    0.1868

W1 =
   -1.5950
   -2.5929

b1 =
    3.0224
    0.5975

W2 =
    3.8997   -3.1101

b2 =
   -0.7619

iter =
        1501

Elapsed time is 1.023033 seconds.
```

حال با افزایش مقدار نرخ یادگیری به ۰/۴ داریم:

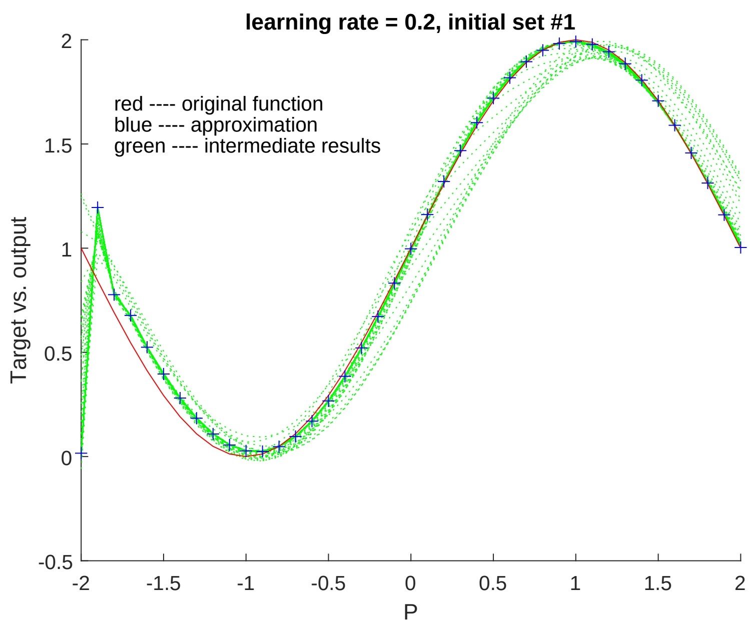
```

W1_0 =
    0.2948
    0.1443

b1_0 =
    0.0328
```



شکل ۳۳: نتیجهٔ سوال ۲۵-۱۱.

```
   -0.1493

W2_0 =
   -0.1214    0.3116

b2_0 =
    0.4390

W1 =
   -2.7608
    2.1593

b1 =
    0.3814
   -4.3332

W2 =
   -2.5486   -3.0327

b2 =
```

```
    2.5222

iter =
        1501

Elapsed time is 1.087420 seconds.
```

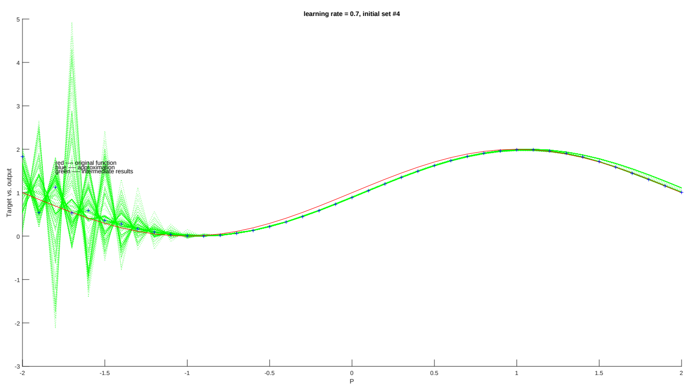


شکل ۳۴: نتیجهٔ سوال ۲۵-۱۱.

با افزایش مقدار نرخ یادگیری به ۰٫۸ داریم:

```
W1_0 =
   -0.2695
    0.3443

b1_0 =
   -0.3293
   -0.2723

W2_0 =
   -0.3052   -0.2741

b2_0 =
```

```
   -0.0643

W1 =

    4.6886
   -5.8325

b1 =

   -8.5763
  -11.8243

W2 =

   -1.4307   -0.8085

b2 =

    1.9758

iter =

        1501

Elapsed time is 1.120159 seconds.
```



شکل ۳۵: نتیجهٔ سوال ۲۵-۱۱.

همان‌طور که مشاهده می‌شود با زیادشدن نرخ یادگیری، سرعت گام‌ها بیش‌تر میشود و اگر این نرخ از عددی مانند ۰/۵۵ بیش‌تر شود پاسخ ما به سمت واگرایی می‌رود. هم‌چنین مقادیر اولیهٔ اوزان و بایاس‌ها در پاسخ نهایی تأثیر به‌سزایی دارد.

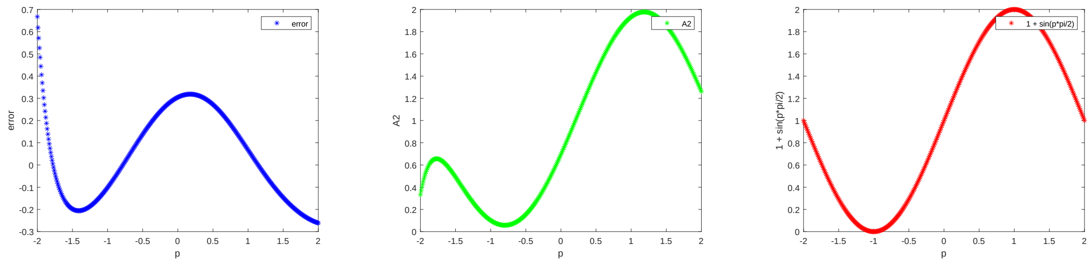
راه‌حل دوم

در راه‌حل دوم از دستورات زیر استفاده می‌کنیم و نتایج به ازای نرخ یادگیری ۰/۳ به صورتی است که در شکل ۳۶ آورده شده است.

```
clf; clc; clear all; close all;

% Init min/max
xMax = 0.5;
xMin = -xMax;
difference = xMax-xMin;

% Init w1, b1, w2, b2

W1(:, 1) = xMin + rand(1, 10)*difference; % For the first layer (logsig)
B1(:, 1) = xMin + rand(1, 10)*difference; % The new values of the weights, biases will be added to each column

W2(1, :) =(xMin+rand(1,10)*difference)'; % For the second layer (purelin)
B2(1) = xMin + rand(1, 1)*(difference); % The new values of the weights, biases will be added to each row

i = 1;
learningRate = 0.01;

while (1)
for p = -2:0.01:2

% FEED-FORWARD PHASE

product = (W1(:, i).* p + B1(:, i)); % Compute activation for 1st layer
A1(:, i) = 1./(1+exp(-product));
A2(i) = W2(i, :)*A1(:, i) + B2(1); % Compute activation for 2nd layer

% COMPUTE ERROR THROUGH THE COST FUNCTION

```

```
error = 1 + sin(p * (pi/2)) - A2(i);

% BACKPROPAGATION PREPARATION

% Compute derivative for 1st activation, compute sensitivities
derivative =[];
for row = 1:10
element = (1-A1(row,i))*A1(row,i);
derivative = [derivative element];
end
R = diag(derivative);

s2 = -2 * 1 * error;

s1 = R * W2(i, :)' *s2;

% UPDATE WEIGHTS AND BIASES

W1(:, i+1) = W1(:, i) - learningRate * s1 * p;
B1(:, i+1) = B1(:, i) - learningRate * s1;

W2(i+1, :) = W2(i, :) - learningRate * s2 * A1(:,i)';
B2(i+1) = B2(i) - learningRate * s2;

figure(1);
plot(p, error,'*b');
hold on;
xlabel('p');
ylabel('error');
legend('error');

figure(2);
plot(p, A2(i),'*g');
hold on;
xlabel('p');
```

```
ylabel('A2');
legend('A2');

figure(3)
plot(p,1 + sin(p*(pi/2)),'*r');
hold on;
xlabel('p');
ylabel('1 + sin(p*pi/2)');
legend('1 + sin(p*pi/2)');

i = i + 1;

end

if (abs(error) < 0.01 || i<1500)
return % End if absolute error is smaller than the predefined threshold
end

end
```



(آ) اصلی (ب) خروجی (ج) خطا

شکل ۳۶: نتیجهٔ سوال ۲۵-۱۱.

توضیح پوشهٔ کدها

در پوشهٔ اصلی این سوال (Q11__11_25)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند.

پاسخ سوال ۱۲.۱۴

گام اول انتشار ورودی از شبکه و محاسبهٔ خطا است:

$$al^{۰} = p_{۱} = [۱] \tag{۵۳}$$

از صفحهٔ پانزدهم فصل یازدهم کتاب مرجع داریم:

$$\begin{aligned} n_{۱}^{۱} &= [-۰/۷۵; -۰/۵۴] \\ a_{۱}^{۱} &= [۰/۳۲۱; ۰/۳۶۸] \\ nl^{۲} &= [۰/۰۹ - ۰/۱۷] * [۰/۳۲۱; ۰/۳۶۸] + [۰/۴۸] = [۰/۴۴۶۳] \\ a_{۱}^{۲} &= [۰/۴۴۶۳]; \end{aligned} \tag{۵۴}$$

خطا برابر است با:

$$e_{l} = t - a = (۱ + \sin(۱ * pi/۴)) - a^{۲} = ۱/۲۶۱ \tag{۵۵}$$

می‌نویسیم:

$$\begin{aligned} a_{۲}{}^{۰} &= p_{۲} = [۰] \\ n_{۲} &== [-۰/۲۷; -۰/۴۱] * [۰] + [-۰/۴۸; -۰/۱۳] = [-۰/۴۸; ۰/۱۳] \\ a_{۲}{}^{۱} &= \log sig([-۰/۴۸; ۰/۱۳]) = [۰/۳۸۲۳; ۰/۵۳۲۵]; \\ n_{۲}^{۲} &= [۰/۰۹ - ۰/۱۷] * [۰/۳۸۲۳; ۰/۵۳۲۵] + [۰/۴۸] = [۰/۴۲۳۹] \\ a_{۲}^{۲} &= [۰/۴۲۳۹]; \end{aligned} \tag{۵۶}$$

خطا برابر است با:

$$e_{l} = t - a = (۱ + \sin(۰ {}^{*} pi/۴)) - a_{۲}{}^{۲} = ۰/۵۷۶۱ \tag{۵۷}$$

گام بعدی آغازگری پس‌انتشار یا حساسیت مارکوات و استفاده از رابطهٔ ۵۸ و رابطهٔ ۵۹ است.

$$\tilde{\mathbf{S}}_q^M = -\dot{\mathbf{F}}^M\left(\mathbf{n}_q^M\right) \tag{۵۸}$$

$$\tilde{\mathbf{S}}_q^m = \mathbf{F}^m\left(\mathbf{n}_q^m\right)\left(\mathbf{W}^{m+۱}\right)^T\tilde{\mathbf{S}}_q^{m+۱} \tag{۵۹}$$

درنتیجه می‌نویسیم:

$$\begin{aligned}
\tilde{S}۱^{۲} &= -\dot{F}^{۲}\left(\mathrm{n}_۱{}^{۲}\right) = -[۱]\\
\tilde{S}۱^{۱} &= \dot{F}^{۱}\left(\mathrm{n}_۱{}^{۱}\right)\left(\mathrm{w}^{۲}\right)^{\mathrm{T}}\tilde{S}۱^{۲۲}\\
&= \left[\left(۱-\mathrm{a}_{۱,۱}\right)^{۱}\right)^{*}\mathrm{a}_{۱,۱}{}^{۱}, ۰; ۰, \left(۱-\mathrm{a}_{۱,۱}\right)^{*}\Big)^{*}\mathrm{a}_{۱,۱}\Big]^{*}[۰/۰۹, -۰/۱۷]'^{*}[-۱]\\
&= [(۱-۰/۳۲۱)*۰/۳۲۱, ۰; ۰, (۱-۰/۳۶۸)^{*}۰/۳۶۸]*[۰/۰۹, -۰/۱۷]^{*}*[-۱]\\
&= [-۰/۰۱۹۶; ۰/۰۳۹۵];
\end{aligned} \tag{۶۰}$$

$$\begin{aligned}
\tilde{S}۲^{۲} &= -\dot{F}^{۲}\left(\mathrm{n}_۱{}^{۲}\right) = -[۱]\\
\tilde{S}۲^{۱} &= \dot{F}^{۱}\left(\mathrm{n}_۲{}^{۱}\right)\left(\mathrm{w}^{۲}\right)^{\mathrm{T}}\tilde{S}^{۲۲}\\
&= \left[\left(۱-\mathrm{a}_{۱,۲}\right)^{۱}\mathrm{a}_{۱,۲}{}^{۱}, ۰; ۰, \left(۱-\mathrm{a}_{۱,۲}\right)^{۱}\mathrm{a}_{۱,۲}{}^{۱}\right]^{*}[۰/۰۹, -۰/۱۷]'*[-۱]\\
&= [(۱-۰/۳۸۲۳)*۰/۳۸۲۳, ۰; ۰, (۱-۰/۵۳۲۵)^{*}۰/۵۳۲۵]*[۰/۰۹, -۰/۱۷]'*[-۱]\\
&= [-۰/۰۲۱۳; ۰/۰۴۲۳];
\end{aligned} \tag{۶۱}$$

$$\tilde{S}^{۱} = \left[\tilde{S}۱^{۱} \mid \tilde{S}۲^{۱}\right] = [-۰/۰۱۹۶-۰/۰۲۱۳; ۰/۰۳۹۵۰/۰۴۲۳] \tag{۶۲}$$

حال ماتریس ژاکوبی را محاسبه می‌کنیم:

$$\mathbf{J}(\mathrm{x}) = \begin{bmatrix} \frac{\partial v۱}{\partial x۱} \frac{\partial v۱}{\partial x۲} \frac{\partial v۱}{\partial x۳} \frac{\partial v۱}{\partial x۴} \\ \frac{\partial v۲}{\partial x۱} \frac{\partial v۲}{\partial x۲} \frac{\partial v۲}{\partial x۳} \frac{\partial v۲}{\partial x۴} \end{bmatrix} = \begin{bmatrix} \frac{\partial e۱,۱}{\partial w۱۱۱} \frac{\partial e۱,۱}{\partial w۲۱۱} \frac{\partial e۱,۱}{\partial b۱۱۱} \frac{\partial e۱,۱}{\partial b۲۱۱} \frac{\partial e۱,۱}{\partial w۱۱۲} \frac{\partial e۱,۱}{\partial w۱۲۲} \frac{\partial e۱,۱}{\partial b۱۲} \\ \frac{\partial e۱,۲}{\partial w۱۱۱} \frac{\partial e۱,۲}{\partial w۲۱۱} \frac{\partial e۱,۲}{\partial b۱۱} \frac{\partial e۱,۲}{\partial b۲۱۱} \frac{\partial e۱,۲}{\partial w۱۱۲} \frac{\partial e۱,۲}{\partial w۱۲۲} \frac{\partial e۱,۲}{\partial b۱۲} \end{bmatrix} \tag{۶۳}$$

می‌نویسیم:

$$\begin{aligned}
&[\mathbf{J}]_{۱,۱} = S^{۱}{}_{۱,۱} * \mathrm{a}_{۱,۱۰} = [-۰٫۰۱۹۶] * ۱ = [-۰٫۰۱۹۶] \text{ ؛} \\
&[\mathbf{J}]_{۱,۲} = S^{۱}۲,۱ * \mathrm{a}_{۱,۱}{}^{\circ} = [۰٫۰۳۹۵] * ۱ = [۰٫۰۳۹۵]; \\
&[\mathbf{J}]_{۱,۳} = S^{۱۱,۱} = [-۰٫۰۱۹۶] = [-۰٫۰۱۹۶]; \\
&[\mathbf{J}]_{۱,۴} = S^{۱}{}_{۲,۱} = [۰٫۰۳۹۵] = [۰٫۰۳۹۵]; \\
&[\mathbf{J}]_{۱,۵} = S^{۲}۱,۱ * \mathrm{a}_{۱,۱}{}^{۱} = [-۱] * [۰٫۳۲۱] = [-۰٫۳۲۱]; \\
&[\mathbf{J}]_{۱,۶} = S^{۲}{}_{۱,۱} * \mathrm{a}_{۲,۱}{}^{۱} = [-۱] * [۰٫۳۶۸] = [-۰٫۳۶۸] \text{ ؛} \\
&[\mathbf{J}]_{۱,۷} = S^{۲}{}_{۱,۱} = [-۱];
\end{aligned} \tag{۶۴}$$

$$\begin{aligned}
&[\mathbf{J}]_{۲,۱} = \boldsymbol{S}^{۱}۱,۲ * \mathrm{a}_{۱,۲}{}^{\circ} = [-۰٫۰۲۱۳] * ۰ = [۰] \\
&[\mathbf{J}]_{۲,۲} = \boldsymbol{S}^{۱۲,۲} * \mathrm{a}_{۱,۲۰} = [۰٫۰۴۲۳] * ۰ = [۰]; \\
&[\mathbf{J}]_{۲,۳} = S^{۱۱,۲} = [-۰٫۰۲۱۳]; \\
&[\mathbf{J}]_{۲,۴} = S^{۱}۲,۲ = [۰٫۰۴۲۳] \text{ ؛} \\
&[\mathbf{J}]_{۲,۵} = \boldsymbol{S}^{۲}{}_{۱,۱} * \mathrm{a}_{۱,۲}{}^{۱} = [-۱] * [-۰٫۳۸۲۳] = [۰٫۳۸۲۳]; \\
&[\mathbf{J}]_{۲,۶} = S^{۲}۱,۱ * \mathrm{a}_{۲,۲}{}^{۱} = [-۱] * [۰٫۵۳۲۵] = [-۰٫۵۳۲۵]; \\
&[\mathbf{J}]_{۲,۷} = S^{۲}۱,۱ = [-۱];
\end{aligned} \tag{۶۵}$$

بنابراین ماتریس ژاکوبی به صورت زیر خواهد بود:

$$\mathbf{J}(\mathrm{x}) = \begin{bmatrix} -۰٫۰۱۹۶ & ۰٫۰۳۹۵ & -۰٫۰۱۹۶ & ۰٫۰۳۹۵ & -۰٫۳۲۱ & -۰٫۳۶۸ & -۱ \\ ۰ & ۰ & -۰٫۰۲۱۳ & ۰٫۰۴۲۳ & ۰٫۳۸۲۳ & -۰٫۵۳۲۵ & -۱ \end{bmatrix} \qquad (۶۶)$$

تخمین توابع مقالهٔ ژنگ با MLP

i

برای این مثال از دستورات زیر را در محیط متلب استفاده می‌کنیم و نتایج مطابق شکل ۳۷ و شکل ۴۶ خواهد بود که نشان می‌دهد هدف سوال به خوبی به سرانجام رسیده است.

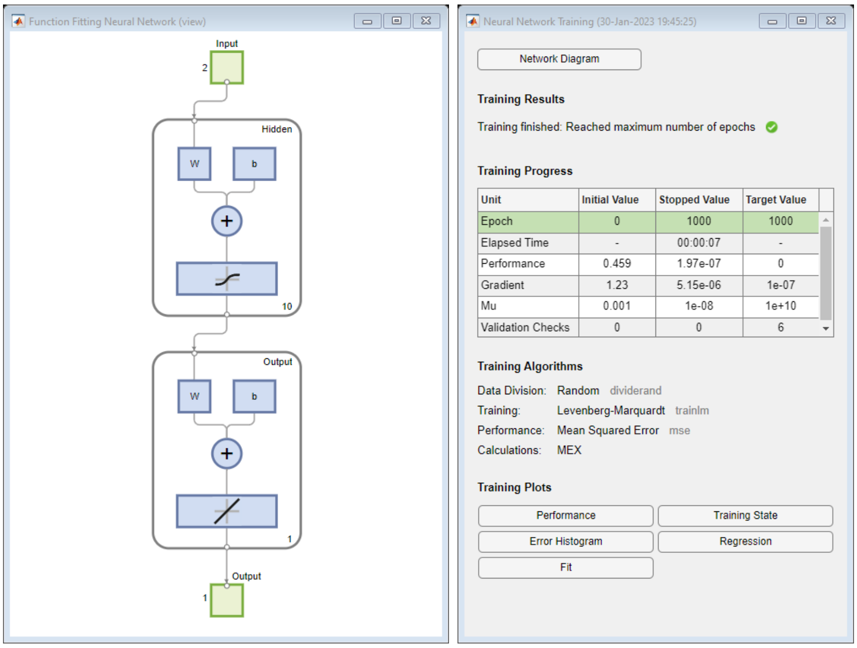
```
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
% Created 18-Jan-2023 11:57:05
%
% This script assumes these variables are defined:
%
%   TestInputs - input data.
%   TestTargets - target data.
clc
close all
clear all

%% Load Data
f=@(x,y) sin(x)./x .* sin(y)./y;

xmin=1;
xmax=6;
ymin=1;
ymax=6;

x = xmin + (xmax-xmin)*rand(26,1);
y = ymin + (ymax-ymin)*rand(26,1);
```

```
F=f(x,y);

TrainInputs=[x,y];
TrainTargets=F;
TrainData=[TrainInputs TrainTargets];

xx = xmin + (xmax-xmin)*rand(1000,1);
yy = ymin + (ymax-ymin)*rand(1000,1);

FF=f(xx,yy);

TestInputs=[xx,yy];
TestTargets=FF;
TestData=[TestInputs TestTargets];


x = TestInputs';
t = TestTargets';

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize,trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};
```

```
% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 10/100;
net.divideParam.testRatio = 10/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean Squared Error

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
```

```
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end
--------------------------------------------------------------------

performance =
```

```
    1.9310e-07

trainPerformance =
    1.9719e-07

valPerformance =
    1.8551e-07

testPerformance =
    1.6801e-07
```

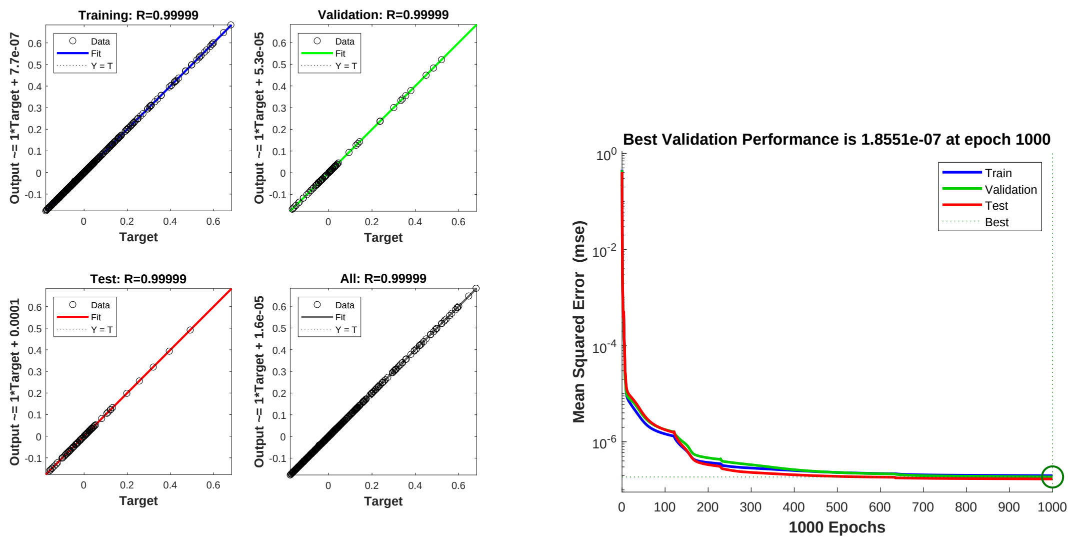


شکل ۳۷: نتیجهٔ قسمت اول سوال اول.

ii

برای این مثال از دستورات زیر را در محیط متلب استفاده می‌کنیم و نتایج مطابق شکل ۳۹ و شکل ۴۰ خواهد بود که نشان می‌دهد هدف سوال به خوبی به سرانجام رسیده است.

```
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
% Created 18-Jan-2023 11:57:05
%
```



(ب) (آ)

شکل ۳۸: نتیجهٔ قسمت اول سوال اول.

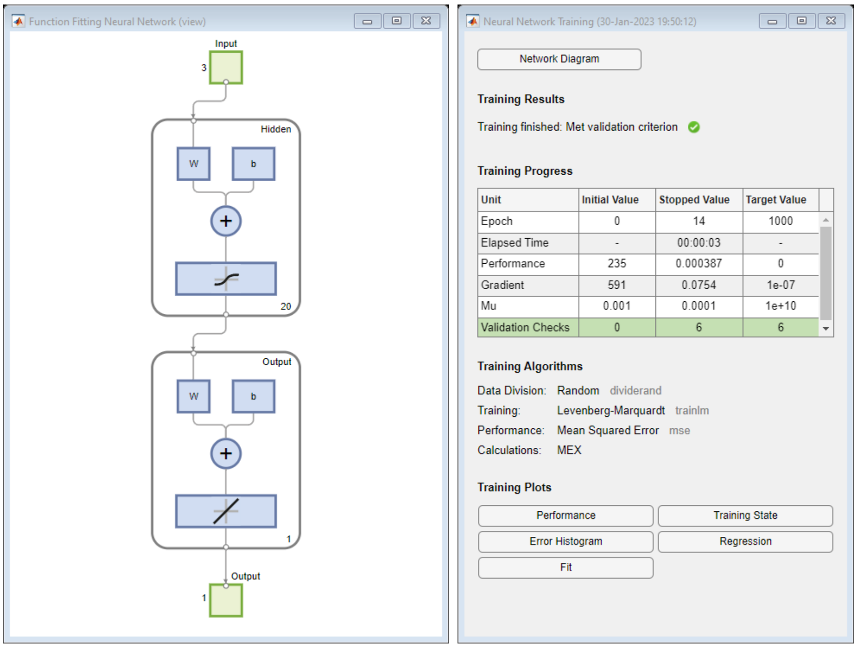
```
% This script assumes these variables are defined:
%
%   TestInputs - input data.
%   TestTargets - target data.
clc
close all
clear all

%% Load Data
f=@(x,y,z) (1+x.^0.5+y.^-1+z.^(-1.5)).^2;

xmin=1;
xmax=6;
ymin=1;
ymax=6;
zmin=1;
zmax=6;
x = xmin + (xmax-xmin)*rand(26,1);
y = ymin + (ymax-ymin)*rand(26,1);
z = zmin + (zmax-zmin)*rand(26,1);
```

```
F=f(x,y,z);

TrainInputs=[x,y,z];
TrainTargets=F;
TrainData=[TrainInputs TrainTargets];

xx = xmin + (xmax-xmin)*rand(101,1);
yy = ymin + (ymax-ymin)*rand(101,1);
zz = zmin + (zmax-zmin)*rand(101,1);
FF=f(xx,yy,zz);

TestInputs=[xx,yy,zz];
TestTargets=FF;
TestData=[TestInputs TestTargets];


x = TestInputs';
t = TestTargets';

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize = 20;
net = fitnet(hiddenLayerSize,trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};

```

```
% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 10/100;
net.divideParam.testRatio = 10/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean Squared Error

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
view(net)
```

```
% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end
--------------------------------------------------------------------

performance =
    0.0529
```

```

trainPerformance =

    0.0630

valPerformance =

    0.0079

testPerformance =

    0.0168
```

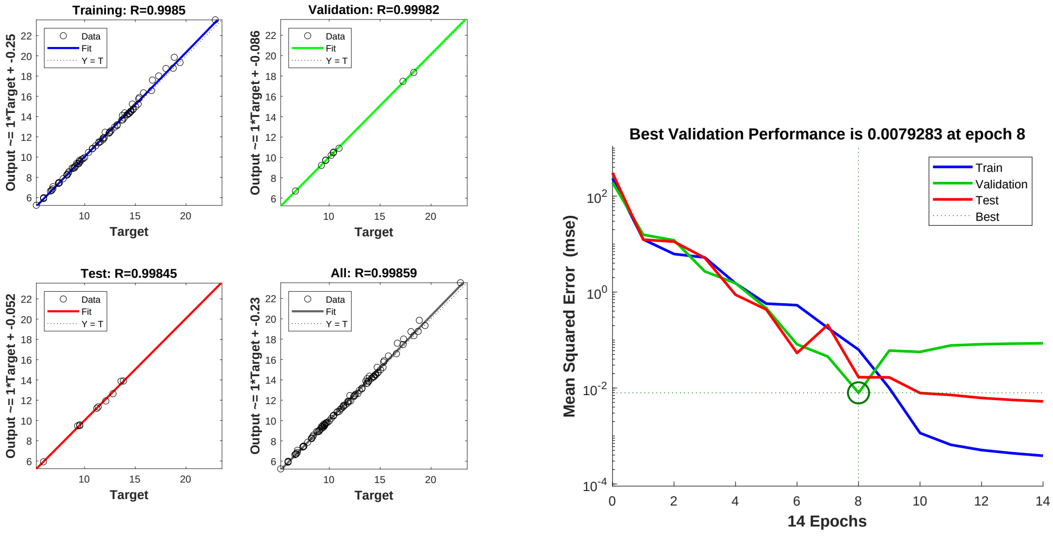


شکل ۳۹: نتیجهٔ قسمت دوم سوال اول.

iii

برای این مثال از محیط سیمولینک آورده‌شده در شکل ۴۱ و دستورات زیر استفاده می‌کنیم.

```
clc;
clear all;
close all;

for k=1:700
    if k<=500
```



(ب) (آ)

شکل ۴۰: نتیجهٔ قسمت دوم سوال اول.

```
        o=@(x) sin((2*pi*x)/250);
    else
        o=@(x) 0.5*sin((2*pi*x)/250) + 0.5*sin((2*pi*x)/25);
    end
    u(k)=o(k);
end

k=1:700;

figure
plot(k,u,LineWidth=2)
xlabel("time")

o=@(y) 0.6*sin(pi*y)+0.3*sin(3*pi*y)+0.1*sin(5*pi*y);
f=o(u);

figure
plot(k,f,LineWidth=2)
xlabel("time")

```

```
train_data=k';
train_target=f';
net = feedforwardnet(30);

[trainedNet,tr] = train(net,train_data',train_target')

kk=k(1:500);

test_data=kk';

for i=1:500
NNsOutput(i)=sim(trainedNet,test_data(i,:)');
end

ff=f(1:500);
plot(1:500,ff,'r',1:500,NNsOutput,'b',LineWidth=2)
legend('Data','MLP Output')


time=kk;
F=NNsOutput;

out=sim('anf33.slx');

yk=out.Output;
yk=yk.Data;

yhk=out.Output1;
yhk=yhk.Data;

figure
plot(yhk,'b',LineWidth=2)
hold on
plot(yk,'r--',LineWidth=2)
legend("yhk","yk")
```

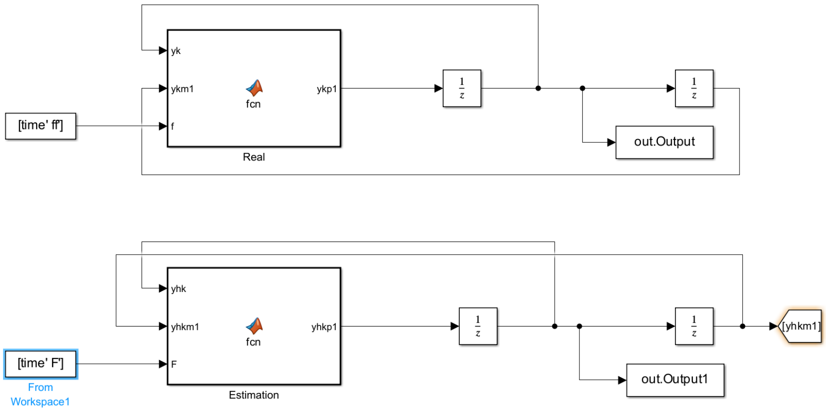
```
axis([0 500 -6 6])clc;
clear all;
close all;

for k=1:700
    if k<=500
        o=@(x) sin((2*pi*x)/250);
    else
        o=@(x) 0.5*sin((2*pi*x)/250) + 0.5*sin((2*pi*x)/25);
    end
    u(k)=o(k);
end

k=1:700;

figure
plot(k,u,LineWidth=2)
xlabel("time")

o=@(y) 0.6*sin(pi*y)+0.3*sin(3*pi*y)+0.1*sin(5*pi*y);
f=o(u);

figure
plot(k,f,LineWidth=2)
xlabel("time")

train_data=k';
train_target=f';
net = feedforwardnet(30);

[trainedNet,tr] = train(net,train_data',train_target')

kk=k(1:500);

test_data=kk';
```

```
for i=1:500
NNsOutput(i)=sim(trainedNet,test_data(i,:)');
end

ff=f(1:500);
plot(1:500,ff,'r',1:500,NNsOutput,'b',LineWidth=2)
legend('Data','MLP Output')


time=kk;
F=NNsOutput;

out=sim('anf33.slx');

yk=out.Output;
yk=yk.Data;

yhk=out.Output1;
yhk=yhk.Data;

figure
plot(yhk,'b',LineWidth=2)
hold on
plot(yk,'r--',LineWidth=2)
legend("yhk","yk")
axis([0 500 -6 6])
```

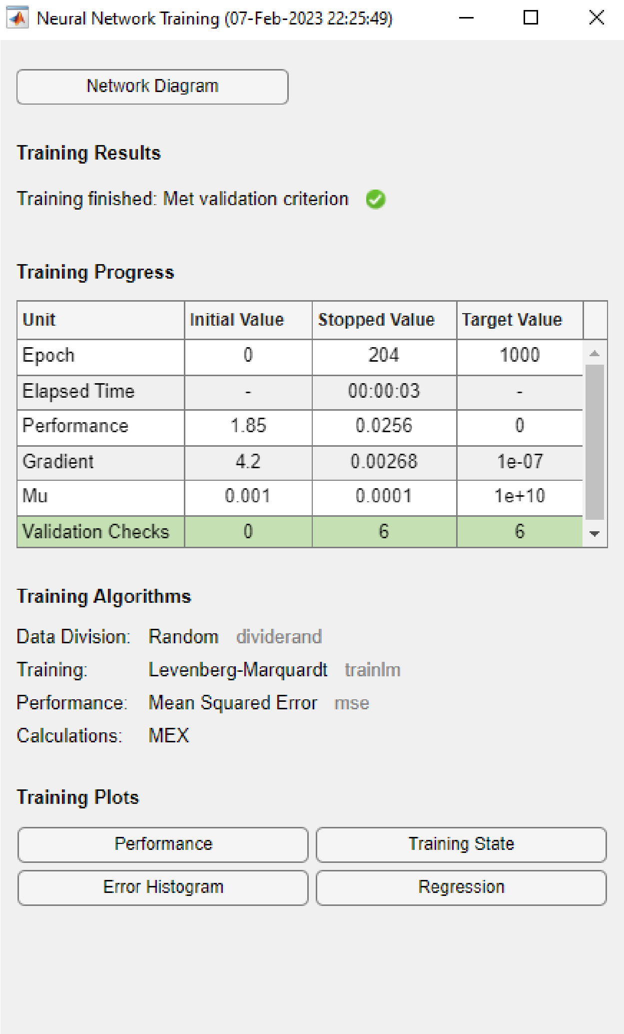
iv

برای این مثال از محیط سیمولینک آورده‌شده در شکل ۴۴ و دستورات زیر استفاده می‌کنیم.

```
clc;
clear all;
close all;
```

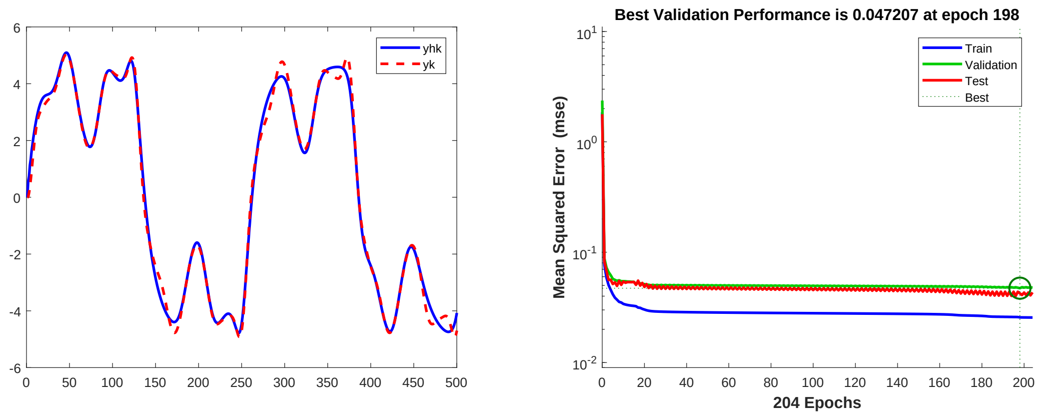


شکل ۴۱: سیمولینک.



شکل ۴۲: نتیجهٔ قسمت سوم سوال اول.

```

out=sim('MLPQ4.slx');

```

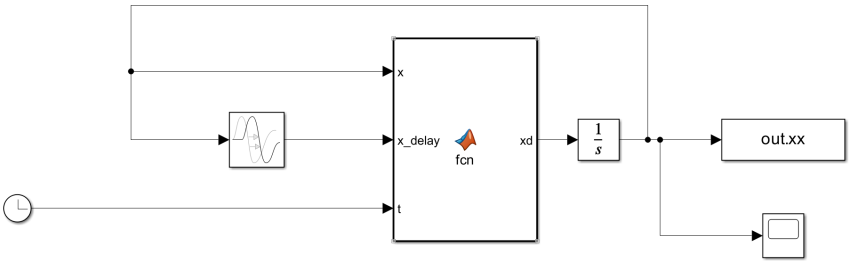


(ب) (آ)

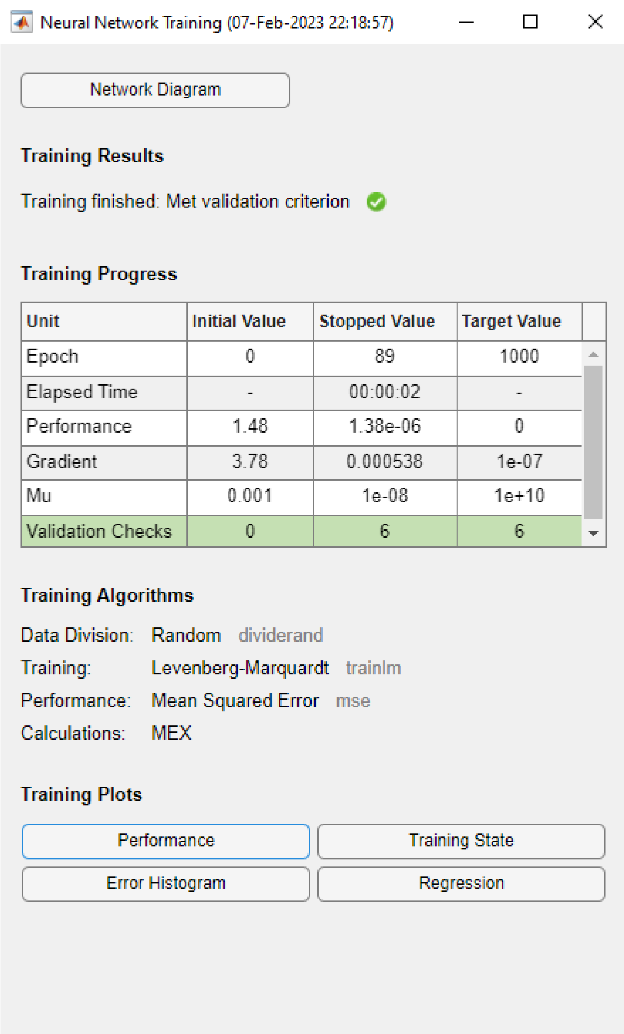
شکل ۴۳: نتیجهٔ قسمت سوم سوال اول.

```
x=out.xx;
x=x.Data;

x_start = 117 ;
x_end = 1118 ;

x=[x(x_start-18:x_end-18,1),x(x_start-12:x_end-12,1),x(x_start-6:x_end-6,1),x(x_start:x_end,1),x(x_start+6:x_end+6,1)];

x_train=x(1:500,1:4);
y_train=x(1:500,5);

net = feedforwardnet(30);

[trainedNet,tr] = train(net,x_train',y_train')


x_test=x(501:1000,1:4);
y_test=x(501:1000,5);


for i=1:500
NNsOutput(i)=sim(trainedNet,x_test(i,:)');
```

```
end

figure
plot(1:500,NNsOutput,'k',LineWidth=1.2)
hold on
plot(1:500,y_test,'r--',LineWidth=2)
legend('Data','MLP Output')
```



شکل ۴۴: سیمولینک.



Neural Network Training (07-Feb-2023 22:18:57)

Network Diagram

Training Results

Training finished: Met validation criterion

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	89	1000
Elapsed Time	-	00:00:02	-
Performance	1.48	1.38e-06	0
Gradient	3.78	0.000538	1e-07
Mu	0.001	1e-08	1e+10
Validation Checks	0	6	6

Training Algorithms

Data Division: Random dividerand

Training: Levenberg-Marquardt trainlm

Performance: Mean Squared Error mse

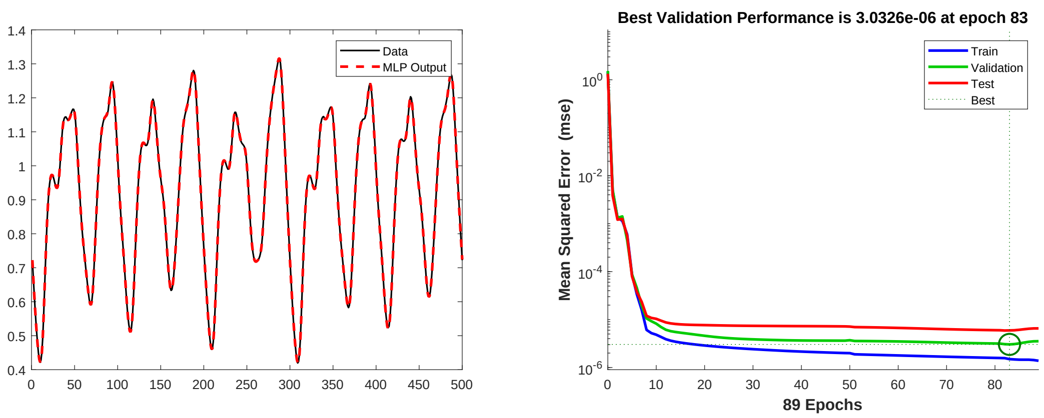
Calculations: MEX

Training Plots

Performance | Training State

Error Histogram | Regression

شکل ۴۵: نتیجهٔ قسمت چهارم سوال اول.



(ب) (آ)

شکل ۴۶: نتیجهٔ قسمت چهارم سوال اول.

جدول ۱: پارامترهای سیستم

۲ kg	M
۰٫۸ kg	m
۰٫۰۰۵ kg/s	$\mathrm{b_x}$
۰٫۰۰۰۵kgm۲/s	θ_x
۰٫۲۵ m	l

طراحی کنترل کننده Fuzzy PID

راه‌حل اول

از پارامترهای آورده‌شده در جدول ۱ برای سیستم خود استفاده می‌کنیم. معادلهٔ لاگرانژ به‌صورت زیر خواهد بود:

$$\begin{aligned} L &= T - V \\ &= \frac{1}{2}Mv_1^2 + \frac{1}{2}mv_2^2 - mgl\cos\theta \end{aligned} \tag{۶۷}$$

که در آن:

$$\begin{aligned} v_1^2 &= \dot{x}^2 \\ v_2^2 &= \left(\frac{d}{dt}(x - l\sin\theta)\right)^2 + \left(\frac{d}{dt}(l\cos\theta)\right)^2 \\ &= \dot{x}^2 - 2l\dot{x}\dot{\theta}\cos\theta + l^2\dot{\theta}^2 \end{aligned} \tag{۶۸}$$

حال می‌توانیم بنویسیم:

$$L = \frac{1}{2}(M+m)\dot{x}^2 - ml\dot{x}\dot{\theta}\cos\theta + \frac{1}{2}ml^2\dot{\theta}^2 - mgl\cos\theta \tag{۶۹}$$

از تعریف معادلات لاگرانژ داریم:

$$\begin{aligned} &\frac{d}{dt}\left(\frac{dL}{d\dot{x}}\right) - \frac{dL}{dx} = F - b_x\dot{\theta} \quad \to (۱) \\ &\frac{d}{dt}\left(\frac{dL}{d\dot{\theta}}\right) - \frac{dL}{d\theta} = b_\theta\dot{\theta} \quad \to (۲) \end{aligned} \tag{۷۰}$$

که در آن b_x و b_θ ضرایب اصطکاک کارت و پاندول هستند. رابطهٔ ۷۰ را می‌توان به‌صورت زیر بازنویسی کرد:

$$\begin{aligned} &(M+m)\ddot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^{۲}\sin\theta = F - b_x\dot{x} \quad \to (۳) \\ &-\ddot{x}\cos\theta + l\ddot{\theta} - g\sin\theta = -b_\theta\dot{\theta} \quad \to (۴) \end{aligned} \tag{۷۱}$$

معادلات آورده‌شده در رابطهٔ ۷۱ می‌تواند با $\ddot{x}$ و $\ddot{\theta}$ بیان شود تا در حالت ODE45 مورد استفاده قرار گیرد. از معادلهٔ (۴) در رابطهٔ ۷۱ می‌توانیم بنویسیم: $\ddot{x} = \frac{۱}{\cos\theta}\left(l\ddot{\theta} - g\sin\theta + b_\theta\dot{\theta}\right)$ و اگر آن را جابگزین معادلهٔ (۳) در رابطهٔ ۷۱ کنیم می‌توانیم بنویسیم:

$$\ddot{\theta} = \frac{۱}{\frac{(M+m)}{\cos\theta}l - ml\cos\theta}\left\{(M+m)g\tan\theta - \frac{(M+m)}{\cos\theta}b_\theta\dot{\theta} - ml\dot{\theta}^{۲}\sin\theta + F - b_x\dot{x}\right\} \tag{۷۲}$$

در ادامه، از معادلهٔ (۴) در رابطهٔ ۷۱ می‌نویسیم: $\ddot{\theta} = \frac{۱}{l}\left(\ddot{x}\cos\theta + g\sin\theta - b_\theta\dot{\theta}\right)$ و اگر آن را جابگزین معادلهٔ (۳) در رابطهٔ ۷۱ کنیم می‌توانیم بنویسیم:

$$\ddot{x} = \frac{۱}{M + m - m\cos^{۲}\theta}\left(mg\sin\theta\cos\theta - mb_\theta\dot{\theta}\cos\theta - ml\dot{\theta}^{۲}\sin\theta + F - b_x\dot{x}\right) \tag{۷۳}$$

بنابراین حالات در ODE45 به‌صورت زیر بیان می‌شوند:

$$\begin{aligned} &\begin{bmatrix} x_۱ & x_۲ & x_۳ & x_۴ \end{bmatrix}^T = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T \\ &\dot{x}_۱ = x_۲ \\ &\dot{x}_۲ = \frac{۱}{M + m - m\cos^{۲} x_۳}\left(mg\sin x_۳\cos x_۳ - mb_\theta\cos x_۳\cdot x_۴ - mlx_۴^{۲}\sin x_۳ + F - b_x x_۲\right) \\ &\dot{x}_۳ = x_۴ \\ &\dot{x}_۴ = \frac{۱}{\frac{M+m}{\cos x_۳}l - ml\cos x_۳}\left\{(M+m)g\tan x_۳ - \frac{M+m}{\cos x_۳}b_\theta x_۴ - mlx_۴^{۲}\sin x_۳ + F - b_x x_۲\right\} \end{aligned} \tag{۷۴}$$

در ادامه توابع زیر را تعریف می‌کنیم تا ساختار فازی را ایجادکرده باشیم:

```
function dX = diff_pendulum(~, X)
global M m bx bq l g J F
dX = zeros(4,1);

% Lagrangian equation
dX(1) = X(2);
dX(2) = 1/(M+m-m*cos(X(3))^2)*(m*g*sin(X(3))*cos(X(3)) - m*bq*cos(X(3))*X(4) -
    m*l*X(4)^2*sin(X(3)) + F - bx*X(2));
dX(3) = X(4);
dX(4) = 1/((M+m)*l/cos(X(3)) - m*l*cos(X(3)))*((M+m)*g*tan(X(3)) - (M+m)/cos(X
    (3))*bq*X(4) - m*l*X(4)^2*sin(X(3)) + F - bx*X(2));

% Text book equation : Not good
% dX(1) = X(2);
% dX(2) = ((J+m*l^2)/((M+m)*(J+m*l^2)-(m*l*cos(X(3)))^2)*(F + m^2*l^2*g*sin(X
    (3))*cos(X(3))/(J+m*l^2)) - m*l*cos(X(3))*bq*X(4)/(J+m*l^2) + m*l*X(4)^2*
    sin(X(3)) - bx*X(2));
% dX(3) = X(4);
% dX(4) = ( (m*l*cos(X(3)))/((m*l*cos(X(3)))^2-(M+m)*(J+m*l^2)) )*( F - (M+m)*g
    *tan(X(3)) + (M+m)*bq/(m*l*cos(X(3)))*X(4) + m*l*X(4)^2*sin(X(3)) - bx*X(2)
     );

end
-------------------------------
function fuzzy = fuzzification(e, de)
% Get the normalized fuzzy output value from error, error_dot
% input(sigma) : e(error), de(delta error)
% output(fuzzy) : mu(e), mu(de) : normalized fuzzy value
% Fuzzy layer configuration : NB, NS, ZO, PS, PB
% 2017.12.07 Hyosung Hong

fuzzy = zeros(5,2); % Column: [NB, NS, ZO, PS, PB]' , Row: [mu(e), mu(de)]

for i=1:2
```

```
    if i==1
        sigma = e;
    else
        sigma = de;
    end

    if sigma < -2/3
        fuzzy(1,i) = 1;
    elseif sigma >= -2/3 && sigma < -1/3
        fuzzy(1,i) = -3*sigma - 1;
        fuzzy(2,i) = 3*sigma + 2;
    elseif sigma >= -1/3 && sigma < 0
        fuzzy(2,i) = -3*sigma;
        fuzzy(3,i) = 3*sigma + 1;
    elseif sigma >= 0 && sigma < 1/3
        fuzzy(3,i) = -3*sigma + 1;
        fuzzy(4,i) = 3*sigma;
    elseif sigma >= 1/3 && sigma < 2/3
        fuzzy(4,i) = -3*sigma + 2;
        fuzzy(5,i) = 3*sigma - 1;
    else
        fuzzy(5,i) = 1;
    end
end

end
-------------------------------

function u_tilde = defuzzification(u_indx)
% defuzzification to generate output u_tilde(normalized value) from fuzzy inference output u_indx
% input(u_indx) : fuzzy set output
% output(fuzzy) : normalized u_tilde
% Fuzzy layer configuration : NB(1), NS(2), ZO(3), PS(4), PB(5)

```

```
u_range = linspace(-1, 1, 100);
if isempty(u_indx) ~= 1      % if u_indx contains some data
    u_candidate = zeros(size(u_indx,1), size(u_range,2));

    for i=1:size(u_indx, 1)
        switch u_indx(i,1)
            case 1
                for j=1:size(u_range,2)
                    if u_range(j) < -2/3
                        u_candidate(i,j) = min(1, u_indx(i,2));
                    elseif u_range(j) > -2/3 && u_range(j) < -1/3
                        u_candidate(i,j) = min(-3*u_range(j) - 1, u_indx(i,2));
                    end
                end

            case 2
                for j=1:size(u_range,2)
                    if u_range(j) >= -2/3 && u_range(j) < -1/3
                        u_candidate(i,j) = min( 3*u_range(j) + 2, u_indx(i,2));
                    elseif u_range(j) > -1/3 && u_range(j) < 0
                        u_candidate(i,j) = min(-3*u_range(j)    , u_indx(i,2));
                    end
                end

            case 3
                for j=1:size(u_range,2)
                    if u_range(j) >= -1/3 && u_range(j) < 0
                        u_candidate(i,j) = min( 3*u_range(j) + 1, u_indx(i,2));
                    elseif u_range(j) > 0 && u_range(j) < 1/3
                        u_candidate(i,j) = min(-3*u_range(j) + 1, u_indx(i,2));
                    end
                end

            case 4
                for j=1:size(u_range,2)
```

```
                    if u_range(j) >= 0 && u_range(j) < 1/3
                        u_candidate(i,j) = min( 3*u_range(j)     , u_indx(i,2));
                    elseif u_range(j) > 1/3 && u_range(j) < 2/3
                        u_candidate(i,j) = min(-3*u_range(j) + 2, u_indx(i,2));
                    end
                end

            case 5
                for j=1:size(u_range,2)
                    if u_range(j) >= 1/3 && u_range(j) < 2/3
                        u_candidate(i,j) = min( 3*u_range(j) - 1, u_indx(i,2));
                    elseif u_range(j) > 2/3
                        u_candidate(i,j) = min(1, u_indx(i,2));
                    end
                end

            otherwise
                u_candidate(i,:) = 0;
        end
    end
else
    u_candidate = zeros(1,size(u_range,2));
end

u_union = max(u_candidate, [], 1);

u_tilde = sum(u_range*u_union')/sum(u_union);

end
-------------------------------

function u_indx = fuzzy_inference(fuzzy)
% fuzzy inference to generate fuzzy set and corresponding output u set
% input(fuzzy) : normalized fuzzy value mu(e), mu(de)
% output(u_indx) : fuzzy set output
```

```
% Fuzzy layer configuration : NB(1), NS(2), ZO(3), PS(4), PB(5)

fuzzy_rule =  [1 1 1 2 1;
                1 1 2 5 4;
                1 1 3 5 5;
                4 1 4 5 5;
                5 4 5 5 5];
fuzzy_indx = [];

for i=1:5
    if fuzzy(i,1) > 0
        for j=1:5
            if fuzzy(j,2) > 0
                if fuzzy_rule(i,j) > 0     % only use the defined fuzzy rules
                    fuzzy_indx = [fuzzy_indx; [i,j]];    % example: fuzzy_indx = [NS NS; ZO NB; ZO NS] to determine u_indx
                end
            end
        end
    end
end

if isempty(fuzzy_indx) == 1
    u_indx = [];
else
    u_indx = zeros(size(fuzzy_indx));

    for i=1:size(u_indx,1)
        u_indx(i,:) = [fuzzy_rule(fuzzy_indx(i,1), fuzzy_indx(i,2)), min(fuzzy(fuzzy_indx(i,1), 1), fuzzy(fuzzy_indx(i,2), 2))];
        % example: u_indx = [NS, 0.2;  ZO, 0.2;  ZO, 0.7];
    end
end

end
```

برای اجرا هم از دستورات زیر استفاده می‌کنیم:

```
% Cart and Pole Control
% Fuzzy control

close all; clear;
addpath ./sub_functions
global M m bx bq l g J F
%% pendulum parameters
M    = 2;          % [kg]
m    = 0.8;        % [kg]
bx   = 0.005;      % [kg/s]
bq   = 0.0005;     % [kg m^2/s]
l    = 0.25;       % [m]
g    = 9.81;       % [m/s^2]
J    = 0.0326;     % [kg m]
F    = 0;          % [N]
T_final = 25;      % [s]
Ts = 0.01;         % control step time [s]
video_flag = 1; % choose 1 if you want to save a video file of plot animation

%% initial condition
X    = [-0.5; 0; 15*pi/180; 0];
xd       = 0;
xd_dot   = 0;
qd       = 0;
qd_dot   = 0;

%% Fuzzy Control
% Fuzzy data normalization
x_normal = 12;        % [m]
dx_normal = 1.5;       % [m/s]
q_normal = 360*pi/180;       % [rad]
dq_normal = 180*pi/180; % [rad/s]
u_normal = 1000;      % [N]

```

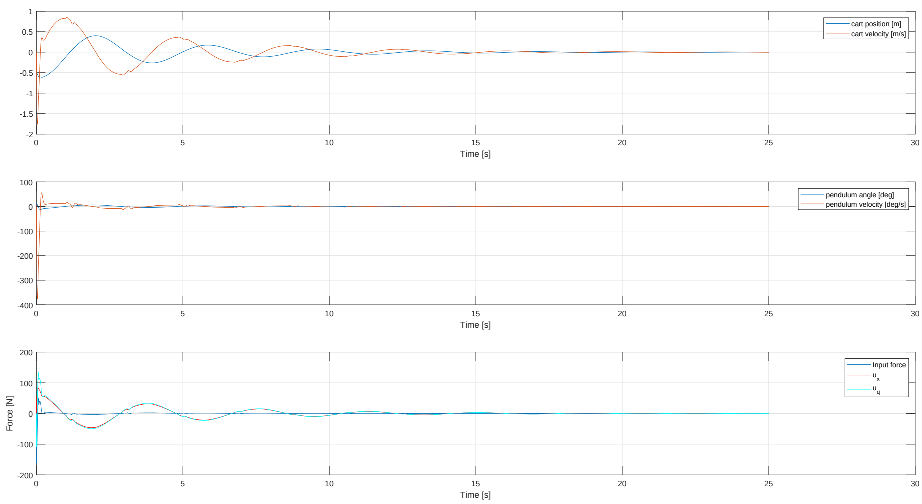
```
% data save
X_Fuzzy = zeros(T_final/Ts+1,4);
time_Fuzzy = zeros(T_final/Ts+1,1);
F_save = zeros(T_final/Ts+1,1);
u_save = zeros(T_final/Ts+1,2);
X_Fuzzy(1,:) = X';
count = 2;
x_prev = X(1);
q_prev = X(3);
for time=Ts:Ts:T_final

    ex  = xd - X(1);          % error of cart position
    dex = x_prev - X(1);      % error difference of cart position

    eq  = qd - X(3);          % error of pendulum angle
    deq = q_prev - X(3);      % error difference of pendulum angle

    % Fuzzy control (u_x: cart position, u_q: pendulum angle)
    u_x = defuzzification(fuzzy_inference(fuzzification(ex/x_normal, dex/
    dx_normal))) * u_normal*1;
    u_q = defuzzification(fuzzy_inference(fuzzification(eq/q_normal, deq/
    dq_normal))) * u_normal*2;
    u_save(count,:) = [u_x, u_q];

    F = -u_x + u_q;

    x_prev = X(1);
    q_prev = X(3);

    [T, X_next] = ode45(@diff_pendulum, [0, Ts], X);
    X = X_next(end,:)';
    X_Fuzzy(count,:) = X';
    time_Fuzzy(count) = time;
    F_save(count) = F;
    count = count + 1;
```

```
end


%% Free fall (No control)
%[T, Z_ode] = ode45(@diff_pendulum, [0, T_final], X);

%% Make video
if video_flag == 1
    % Video file open
    makeVideo = VideoWriter('Fuzzy');
    % Frame Rate - Frames per second
    makeVideo.FrameRate = 100;
    % Quality - ¿ë·®°ú °ü·Ã µÊ (0 ~ 100)
    makeVideo.Quality = 80;
    open(makeVideo);
end

%% Plot
X_result = X_Fuzzy;
Time_result = time_Fuzzy;

figure(1)
axis_limit = 1;
for i=1:size(X_result,1)
    cart_position_x = X_result(i,1);
    pend_position_x = X_result(i,1) - l*sin(X_result(i,3));
    pend_position_y = l*cos(X_result(i,3));

    hold off
    plot(pend_position_x, pend_position_y, 'ok', 'MarkerSize', 25, 'MarkerFaceColor',[0.2 0.9 0.2])   % pendulum
    hold on
    plot(cart_position_x, 0, 'sk', 'MarkerSize', 50, 'MarkerFaceColor',[0.8 0.8 0.8])    % cart
    hold on
```

```
    if cart_position_x > pend_position_x
        plot(linspace(cart_position_x,pend_position_x,2), linspace(0, pend_position_y,2), 'k', 'LineWidth', 3)   % rod
    else
        plot(linspace(cart_position_x,pend_position_x,2), linspace(0, pend_position_y,2), 'r', 'LineWidth', 3)   % rod
    end
    grid on
    xlabel('[m]')
    ylabel('[m]')
    axis([-axis_limit axis_limit -0.5 0.5])
    sim_status = sprintf('Simulation time: %5.2f s',Time_result(i));
    title(sim_status,'FontSize',13)

    if video_flag == 1 && rem(Time_result(i),0.01) == 0
        frame = getframe(gcf);
        writeVideo(makeVideo,frame);
    end

    pause(Ts);
end

if video_flag == 1
    close(makeVideo);
end

figure(2)
subplot(3,1,1)
plot(Time_result, X_result(:,1:2))
grid on
xlabel('Time [s]')
legend('cart position [m]', 'cart velocity [m/s]')

subplot(3,1,2)
plot(Time_result, X_result(:,3:4)*180/pi)
```

```
grid on
xlabel('Time [s]')
legend('pendulum angle [deg]', 'pendulum velocity [deg/s]')

subplot(3,1,3)
plot(Time_result, F_save)
hold on
plot(Time_result, u_save(:,1),'r')
plot(Time_result, u_save(:,2),'c')
grid on
xlabel('Time [s]')
ylabel('Force [N]')
legend('Input force', 'u_x', 'u_q')
```

نتیجه به صورتی خواهد بود که در شکل ۴۷ و این ویدیو آورده شده است.



شکل ۴۷: نتیجهٔ راه‌حل اول سوال طراحی.

راه‌حل دوم

هدف این سوال آن است که از یک سیستم فازی استفاده کنیم و طراحی PID و ضرایب آن را به‌صورت متغیر با زمان به آن بسپاریم. کنترلر PID در حالت کلی به‌صورت زیر تعریف می‌شود که شامل ضرایبی برای

تعیین مشخصه‌های کنترلی سیستم است.

$$G_c(s) = k_p + \frac{k_i}{s} + k_d \qquad (۷۵)$$

در نوع دیگری از نمایش فرکانسی-زمانی و بدون بُعد رابطهٔ ۷۵ به صورت زیر نمایش داده می‌شود:

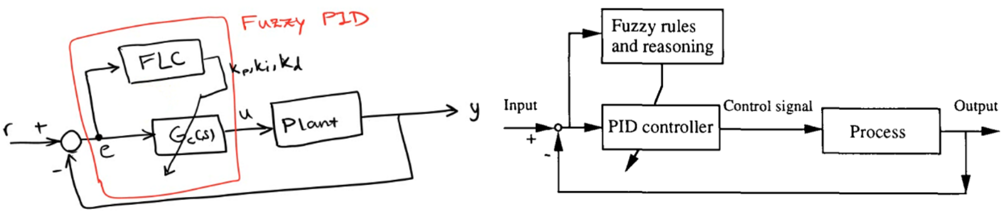
$$\begin{gathered} K_p\left(۱ + \frac{۱}{T_{is}} + T_{dS}\right) \\ k_i = \frac{K_p}{T_i}, \qquad k_d = K_p T_d \end{gathered} \qquad (۷۶)$$

تنظیم ضرایب کنترلر PID راه‌حل قطعی ندارد و کار ساده‌ای نیست. هرچند برخی قواعد برای ساده‌ترشدن انتخاب ضرایب وجود دارند که از جملهٔ آن‌ها می‌توان به قواعد زیگلر-نیکولز اشاره کرد (رابطهٔ ۷۷) که در آن K_p, T_u با ایجاد حالت نوسانی در سیستم و به‌صورت بهره‌ی سیستم نوسانی و دوره‌ی زمانی نوسان تعیین می‌شوند (در مرز پایداری و ناپایداری).

Type Control	K_p	K_i	K_d
P	$۰/۵K_u$	–	–
PI	$۰/۴۵K_u$	$۱/۲K_p/T_u$	–
PD	$۰/۸K_u$	–	$K_pT_u/۸$
PID classic	$۰/۶۰K_u$	$۲K_p/T_u$	$K_pT_u/۸$

(۷۷)

طرح کلی کنترل مدنظر ما به صورتی است که در شکل ۴۸ نشان داده شده است. همان‌طور که مشاهده می‌شود، خطا علاوه بر کنترلر ورد یک سیستم فازی هم می‌شود که این سیستم با درنظرگفتن خطا و مشتق آن ضرایب کنترلر را تنظیم می‌کند. قبل از آغاز طراحی، خروجی‌ها و ضرایب را به صورتی که در رابطهٔ ۷۸

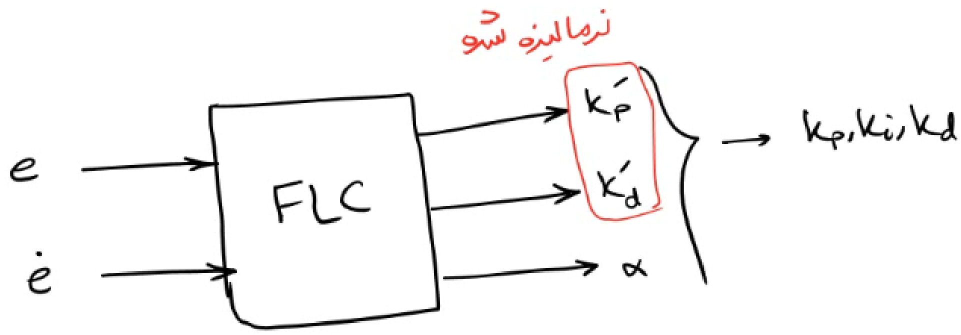


شکل ۴۸: طرح کلی کنترل مدنظر ما.

آورده شده است در نظر می‌گیریم.

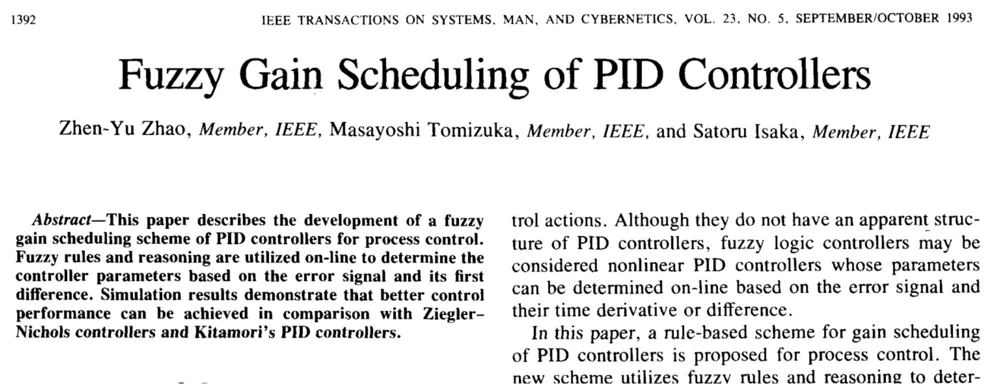
$$k_p, k_i, k_d \rightsquigarrow k_p, k_d, \alpha \qquad \alpha = \frac{T_i}{T_d}, k_i = \frac{k_p^2}{\alpha k_d} \tag{۷۸}$$

حال ضرایب را به‌صورتی که در تصویر نشان داده شده نرمال‌سازی می‌کنیم (مستقل از سیستم و بهرهٔ آن). هم‌چنین می‌دانیم α عددی بین ۲ تا ۵ است که در روش زیگلر-نیکولز همواره برابر با ۴ است. اما ما در روش خود می‌خواهیم آن را محدود به یک نقطهٔ خاص نکنیم و آن را متغیر در نظر بگیریم (زیگلر-نیکولز تطبیقی). بنابراین هستهٔ کلی سیستم مدنظر ما به صورتی خواهد بود که در شکل ۴۹ آورده شده است. حال



شکل ۴۹: هستهٔ کلی سیستم مدنظر ما.

باید دید که قواعد و قوانین را به چه صورت تنظیم کرد که اتفاق مدنظر ما به خوبی بیافتد. برای این منظور از مقاله‌ی آورده‌شده در شکل ۵۰ استفاده می‌کنیم. شکل کلی قواعد بر اساس خطا و تغییرات آن به صورت



1392 IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. 23, NO. 5, SEPTEMBER/OCTOBER 1993

Fuzzy Gain Scheduling of PID Controllers

Zhen-Yu Zhao, *Member, IEEE,* Masayoshi Tomizuka, *Member, IEEE,* and Satoru Isaka, *Member, IEEE*

***Abstract*—This paper describes the development of a fuzzy gain scheduling scheme of PID controllers for process control. Fuzzy rules and reasoning are utilized on-line to determine the controller parameters based on the error signal and its first difference. Simulation results demonstrate that better control performance can be achieved in comparison with Ziegler–Nichols controllers and Kitamori's PID controllers.**

trol actions. Although they do not have an apparent structure of PID controllers, fuzzy logic controllers may be considered nonlinear PID controllers whose parameters can be determined on-line based on the error signal and their time derivative or difference.

In this paper, a rule-based scheme for gain scheduling of PID controllers is proposed for process control. The new scheme utilizes fuzzy rules and reasoning to deter-

شکل ۵۰: مقالهٔ مورد توجه برای ایدهٔ تنظیم قواعد.

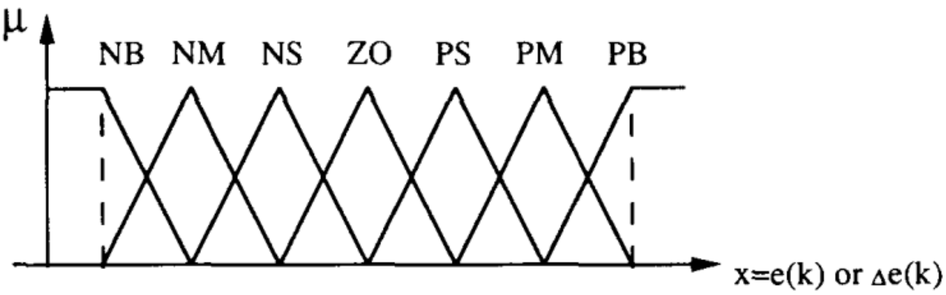
زیر تعیین می‌شود.

$$\text{Rule i: If } e \text{ is } A_i \text{ and } \dot{e} \text{ is } B_i, \text{ Then } k'_p \text{ is } \left\{ \begin{array}{c} Big \\ \text{Small} \end{array} \right\}$$

$$k'_d \text{ is } \left\{ \begin{array}{c} \text{Big} \\ \text{Small} \end{array} \right\}$$

$$\alpha = 2, 3, 4, 5$$

شکل کلی و عمومی تقسیمات فازی خطا و مشتق آن به صورتی است که در شکل ۵۱ نشان داده شده است. در تمام فرآیند شبیه‌سازی با توجه به انجام نرمال‌سازی باید به درنظرگرفتن برخی توابع مربوط به آن توجه کرد. تعریف بزرگ و کوچک و توابع عضویت مربوط به آن‌ها از نظر گرافیکی به صورتی است که در

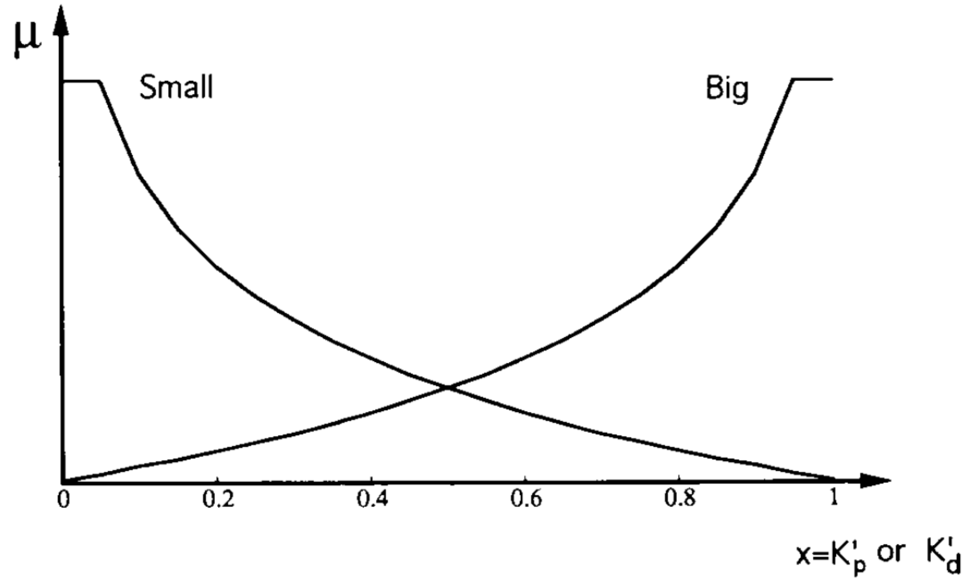


شکل ۵۱: تقسیمات کلی فازی خطا و مشتق آن.

رابطهٔ ۷۹ و شکل ۵۲ آورده شده است. البته مقالهٔ آورده‌شده در شکل ۵۰ به این نکته توجه نکرده که در جایی که x صفر شود مقدار تابع عضویت برابر با بی‌نهایت می‌شود و در واقع رابطهٔ درست و صحیح به صورتی است که در صفحه ۱۴۳ نوشته شده. اعداد ۲ تا ۵ هم به‌صورت سینگلتون فازی اضافه می‌شوند.

$$\begin{aligned} \mu_{\text{Small}}(x) &= \min\left(-\frac{۱}{۴}\ln x, ۱\right) \\ \mu_{\text{Big}}(x) &= \min\left(-\frac{۱}{۴}\ln(۱-x), ۱\right) \end{aligned} \tag{۷۹}$$

می‌توانیم نشان دهیم که سیستم فازی با چند قاعده یک سطح پیوسته را خروجی خواهد داد:



شکل ۵۲: توابع عضویت بزرگ و کوچک.

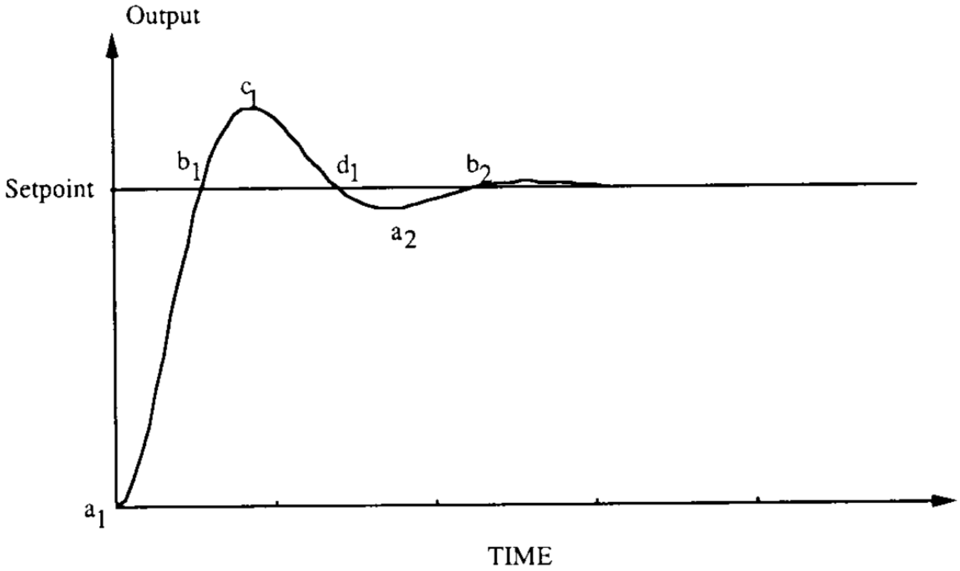
Rule i:

$$\text{If } e \text{ is } A_i, \dot{e} \text{ is } B_i, \text{ Then } \underbrace{k'_p \text{ is } \{\cdots\}}_{k'_{pi}}, \underbrace{k'_d \text{ is } \{\cdots\}}_{k'_{di}}, \alpha = \alpha_i$$

$$\mu_i \propto \mu_{A_i}(e)\mu_{B_i}(\dot{e}) \qquad \sum \mu_i = 1$$

$$k'_p = \sum_i \mu_i k'_{pi}, \quad k'_d = \sum_i \mu_i k'_{di}, \quad \alpha = \sum_i \mu_i \alpha_i$$

به‌صورت کلی توابع و قوانین فازی را بر اساس منطق کنترلی ؟؟ تعیین می‌کنیم. مثلا در شروع فاصله تا مرجع زیاد است و باید بهره‌های تناسبی و انتگرالی بالا باشد. چون مشتق خطا عدد بزرگی خواهد بود ضریب مشتقی را کم در نظر می‌گیریم. حال وقتی به نقطهٔ هدف رسیدیم باید کاری کنیم سیستم تا جای ممکن در محل مطلوب خود بماند. بنابراین در این‌جا بهره‌های تناسبی و انتگرالی را کم می‌کنیم. و برای تغییرات کم خطا و استفاده از خواص خوب مشتق‌گیر، بهرهٔ مشقی را زیاد در نظر می‌گیریم. در ادامه و جایی که اورشوت داریم هم همانند نقطهٔ اول و الی آخر. در ضرایبی که ما تعیین کردیم رفتار k_i برخلاف رفتار α است. حال برای تعیین قوانین در حالت کلی مطابق صفحه ۱۴۵، رابطهٔ ۸۱ و رابطهٔ ۸۲ پیش می‌رویم. در این جدول ستون مربوط به خطا و سطور مربوط به مشتق خطا هستند. این جداول مطابق منطقی که در



شکل ۵۳: نمودار کلی برای تعیین توابع و قوانین فازی.

توضیحات آورده‌شده برای شکل ۵۳ گفته شده نوشته شده‌اند.

	NB	NM	NS	ZO	PS	PM	PB
NB	B	B	B	B	B	B	B
NM	S	B	B	B	B	B	S
NS	S	S	B	B	B	S	S
ZO	S	S	S	B	S	S	S
PS	S	S	B	B	B	S	S
PM	S	B	B	B	B	B	S
PB	B	B	B	B	B	B	B

(۸۰)

	NB	NM	NS	ZO	PS	PM	PB
NB	S	S	S	S	S	S	S
NM	B	B	S	S	S	B	B
NS	B	B	B	S	B	B	B
ZO	B	B	B	B	B	B	B
PS	B	B	B	S	B	B	B
PM	B	B	S	S	S	B	B
PB	S	S	S	S	S	S	S

(۸۱)

	NB	NM	NS	ZO	PS	PM	PB
NB	۲	۲	۲	۲	۲	۲	۲
NM	۳	۳	۲	۲	۲	۳	۳
NS	۴	۳	۳	۲	۳	۳	۴
ZO	۵	۴	۳	۳	۳	۴	۵
PS	۴	۳	۳	۲	۳	۳	۴
PM	۳	۳	۲	۲	۲	۳	۳
PB	۲	۲	۲	۲	۲	۲	۲

(۸۲)

حال با استفاده از هستهٔ اصلی مطرح‌شده در شکل ۴۹ و رابطهٔ ۸۳ ضرایب کنترلر تنظیم‌شده به دست می‌آیند. طبق یک قاعدهٔ سرانگشتی مطرح‌شده در شکل ۵۰ مقادیر بیشینه و کمینهٔ مطرح‌شده در رابطهٔ ۸۳ به صورتی که در رابطهٔ ۸۴ آورده شده تعیین می‌شوند. به نوعی ما داریم برنامه‌ریزی می‌کنیم که از کدام حالت از کنترلرها استفاده کنیم.

$$
\begin{aligned}
k_p &= k_p^{\min} + \left(k_p^{\max} - k_p^{\min}\right) k_p' \\
k_d &= k_d^{\min} + \left(k_d^{\max} - k_d^{\min}\right) k_d' \\
k_i &= \frac{k_p^{2}}{\alpha k_d}
\end{aligned}
\qquad (۸۳)
$$

$$\begin{cases} k_p^{\min} = ۰.۳۲k_u \\ k_p^{\max} = ۰.۶k_u \end{cases} \qquad \begin{cases} k_d^{\min} \equiv ۰/۰۸k_uT_u \\ k_d^{\max} = ۰/۱۵k_uT_u \end{cases} \tag{۸۴}$$

با این مقدمه از بحث‌های تئوری به‌صورت دقیق‌تر وارد فاز شبیه‌سازی می‌شویم. در قدم اول با استفاده از دستور fuzzy قواعد، توابع تعلق و هرچیز دیگر مربوط به سیستم فازی را می‌سازیم. نمای کلی به صورتی است که در شکل ۵۵ نشان داده شده است. در ورودی خطا و مشتق آن و در خروجی سه ضریب تعیین‌شده در رابطهٔ ۷۸ را معین می‌کنیم. مطابق توضیحات در بخش تئوری توابع تعلق مربوط به خطا به صورتی که

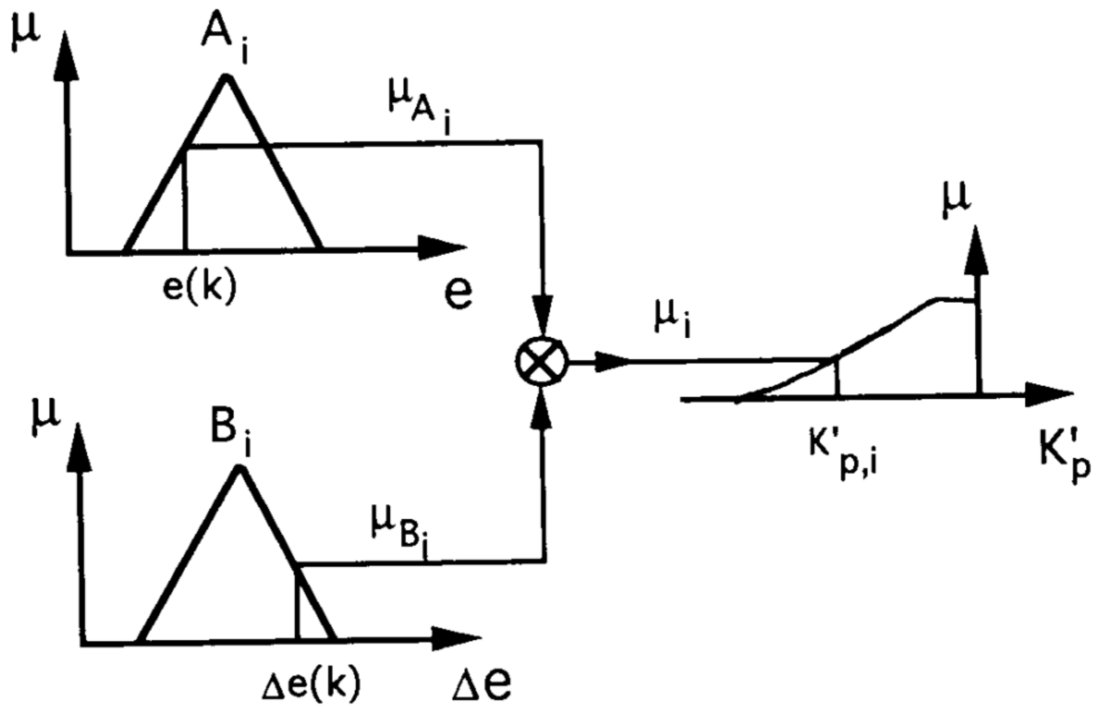
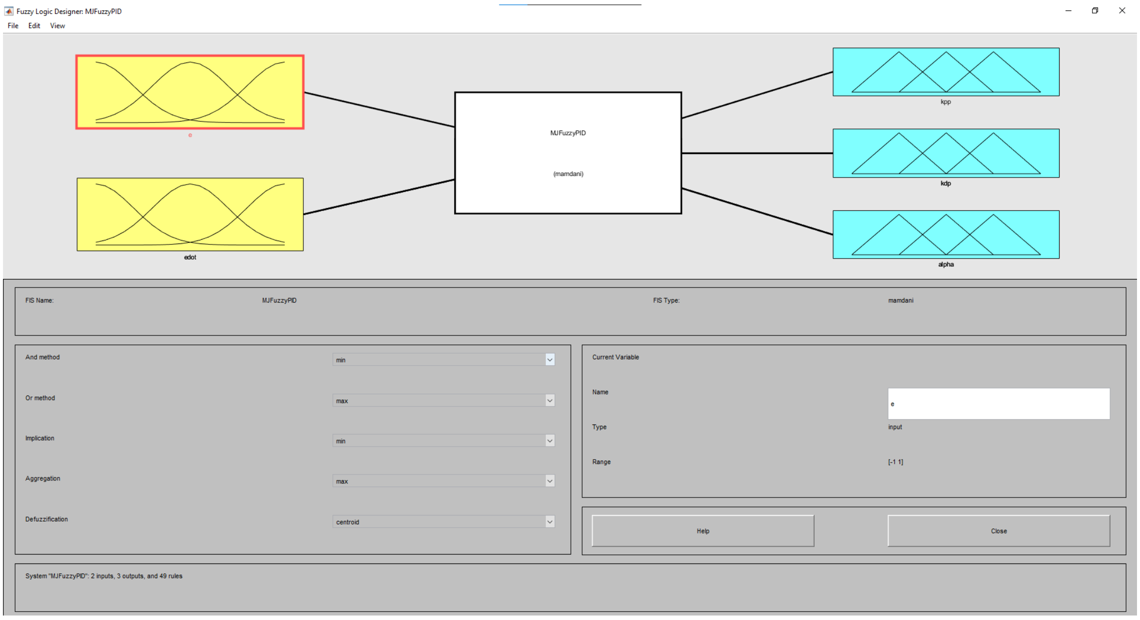


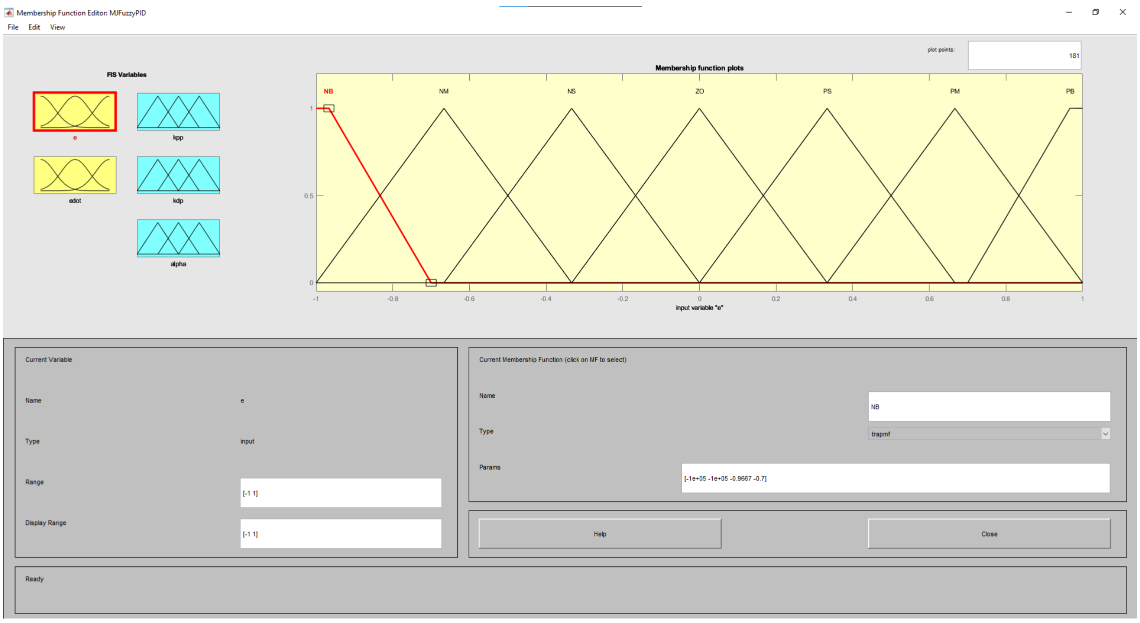
Fig. 6. Implication process of a fuzzy rule.

شکل ۵۴: نمای کلی استنتاج فازی.

در شکل ۵۶ آورده شده و توابع تعلق مربوط به مشتق خطا به صورتی که در شکل ۵۷ آورده شده در نظر گرفته شده است. باید توجه داشت موارد مطرح‌شده در شبیه‌سازی حالت بهتری است که با توجه به سیستم انتخابی و بر شانهٔ تئوری مطرح‌شده در قبل ساخته شده است. هم‌چنین مطابق توضیحات در بخش تئوری توابع تعلق مربوط به خروجی‌ها و ضرایب درنظرگرفته‌شده به صورتی که در شکل ۵۸، شکل ۵۹ و شکل ۶۰ نشان داده شده است تعیین می‌شود. مجدداً باید توجه داشت موارد مطرح‌شده در شبیه‌سازی حالت بهتری

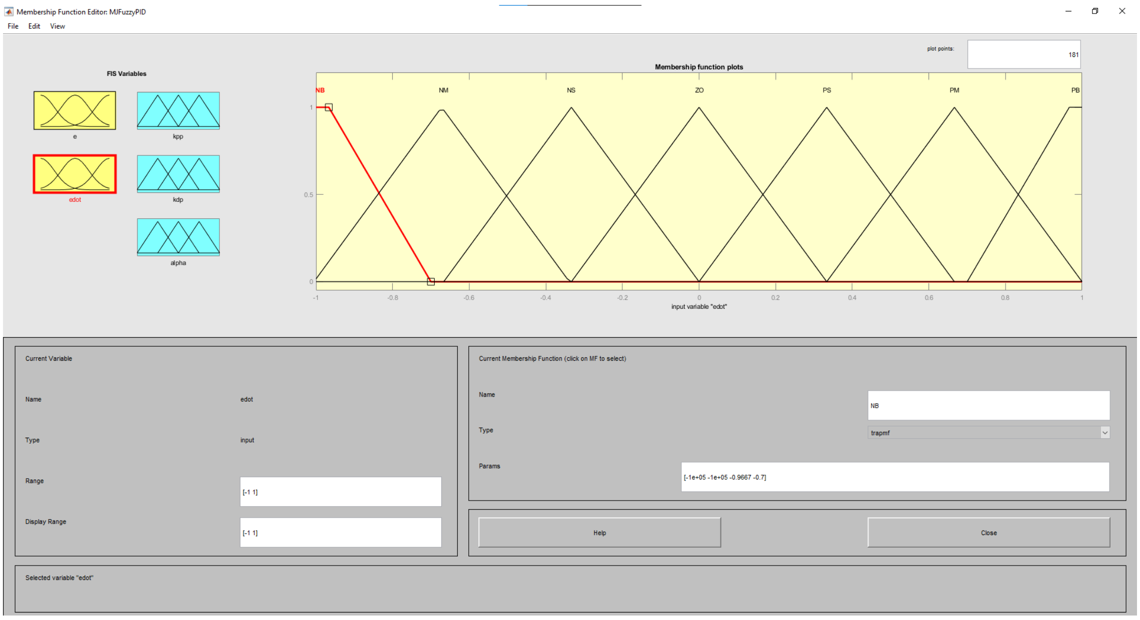


شکل ۵۵: نمای کلی سیستم فازی.

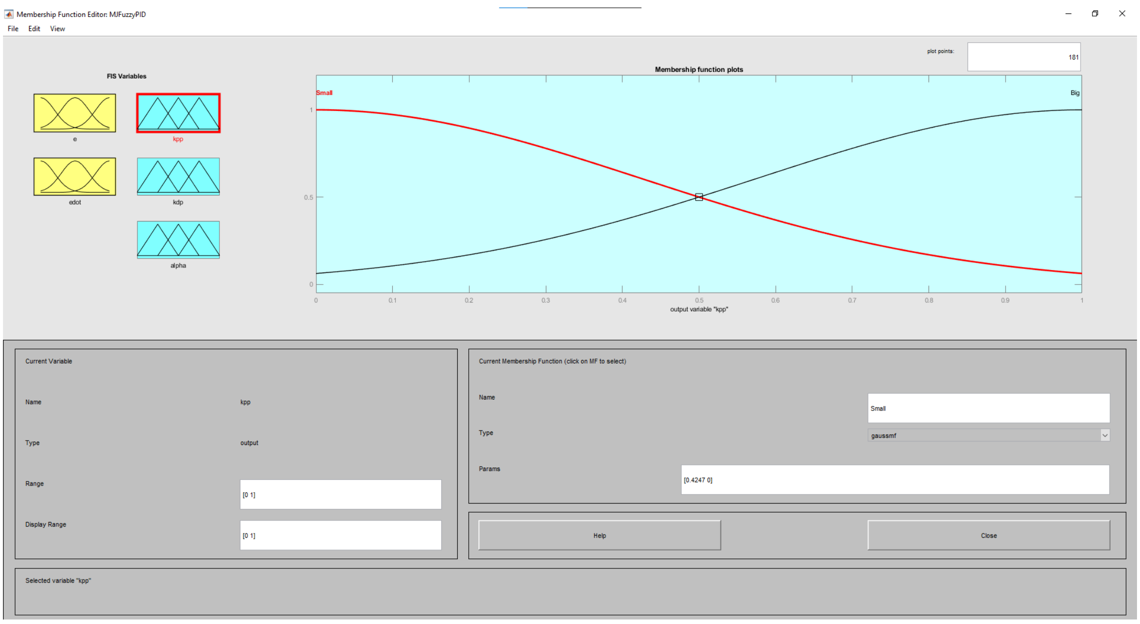


شکل ۵۶: توابع تعلق مربوط به خطا.

است که با توجه به سیستم انتخابی و بر شانهٔ تئوری مطرح‌شده در قبل ساخته شده است. مجدداً مطابق توضیحات در بخش تئوری و هم‌چنین به فراخور سیستم و شرایط موجود ۴۹ قانون به صورتی که بخشی از آن‌ها در شکل ۶۱ و شکل ۶۲ نشان داده شده‌اند معین گردیده‌اند. سطح سیستم فازی برای ضریب تناسبی به صورتی خواهد بود که در شکل ۶۳ و شکل ۶۴ نشان داده شده است. این سطح برای ضریب مشتقی و α نیز به صورت‌هایی که در شکل ۶۵ و شکل ۶۶ نشان داده شده خواهد بود. نمای کلی بلوک کنترلی به صورتی است که در شکل ۶۷ آورده شده است. مغز کنترلی FuzzyPID به صورتی است که در

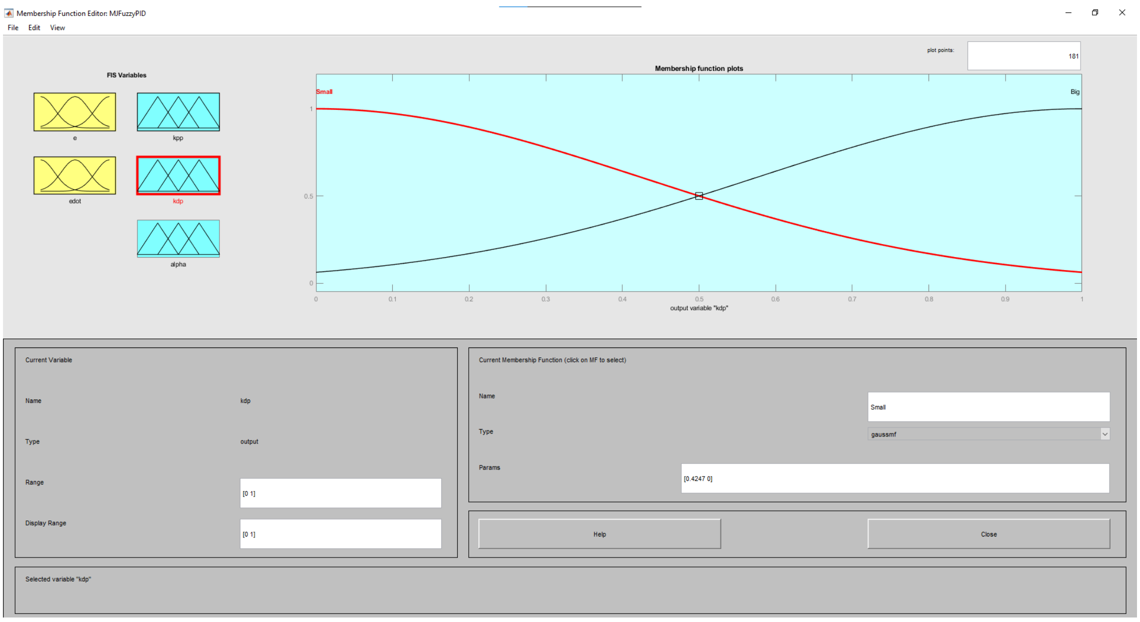


شکل ۵۷: توابع تعلق مربوط به مشتق خطا.

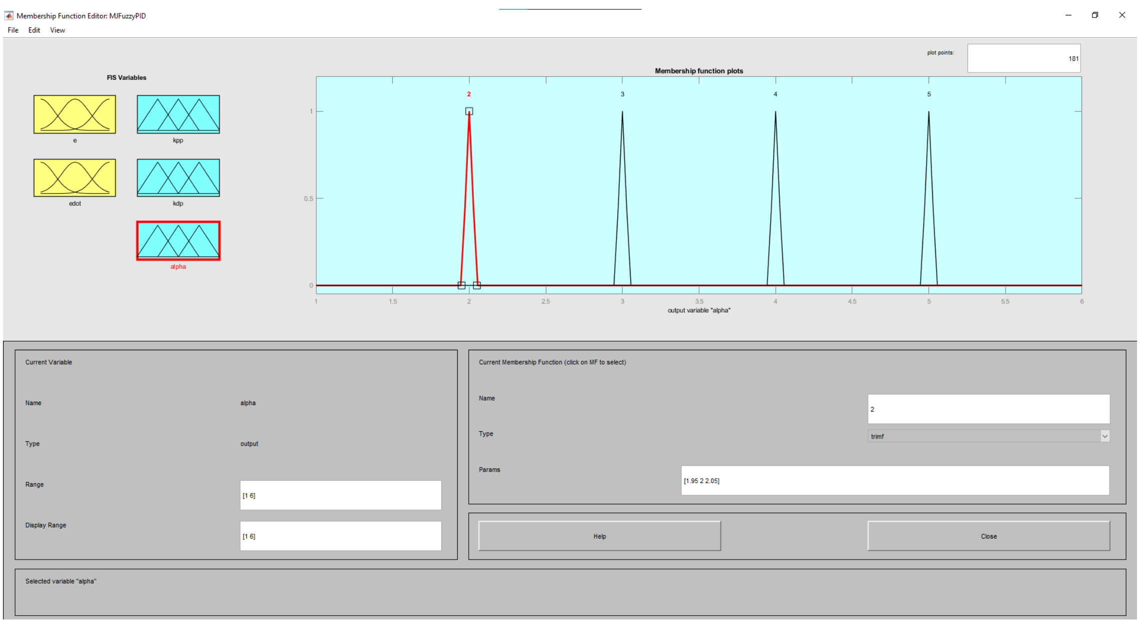


شکل ۵۸: توابع تعلق مربوط به k_pp.

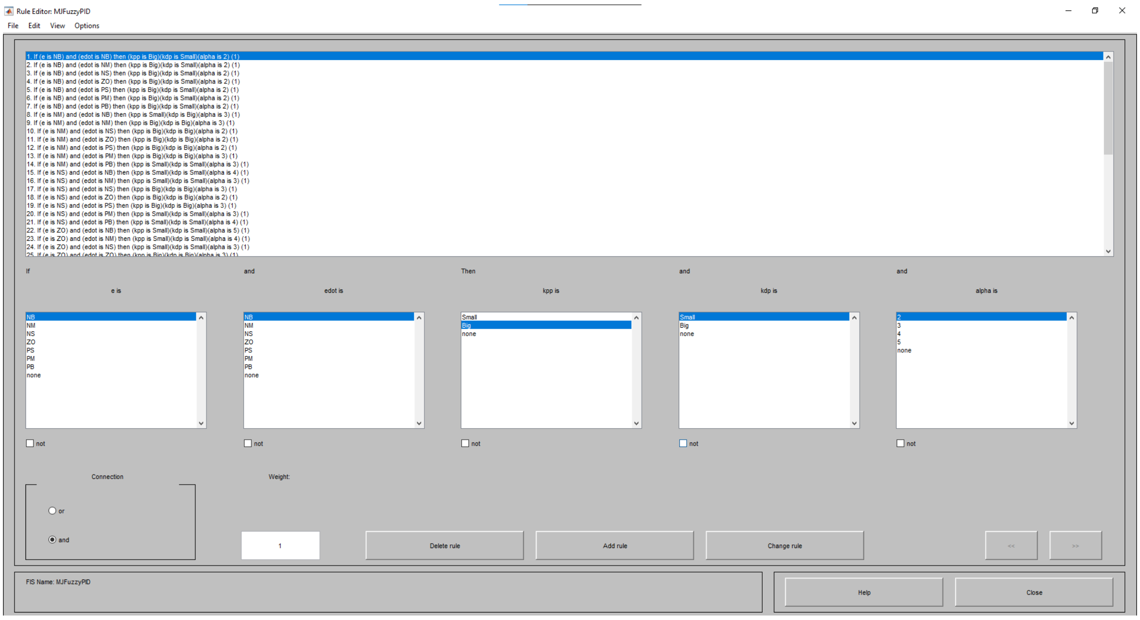
شکل ۶۸ آورده شده است. سیستم فازی به صورتی که در شکل ۶۹ نمایش داده شده تعیین می‌شود. محیط سیمولینک کلی و نتیجه هم به‌ترتیب در شکل ۷۰ و شکل ۷۱ نشان داده شده.



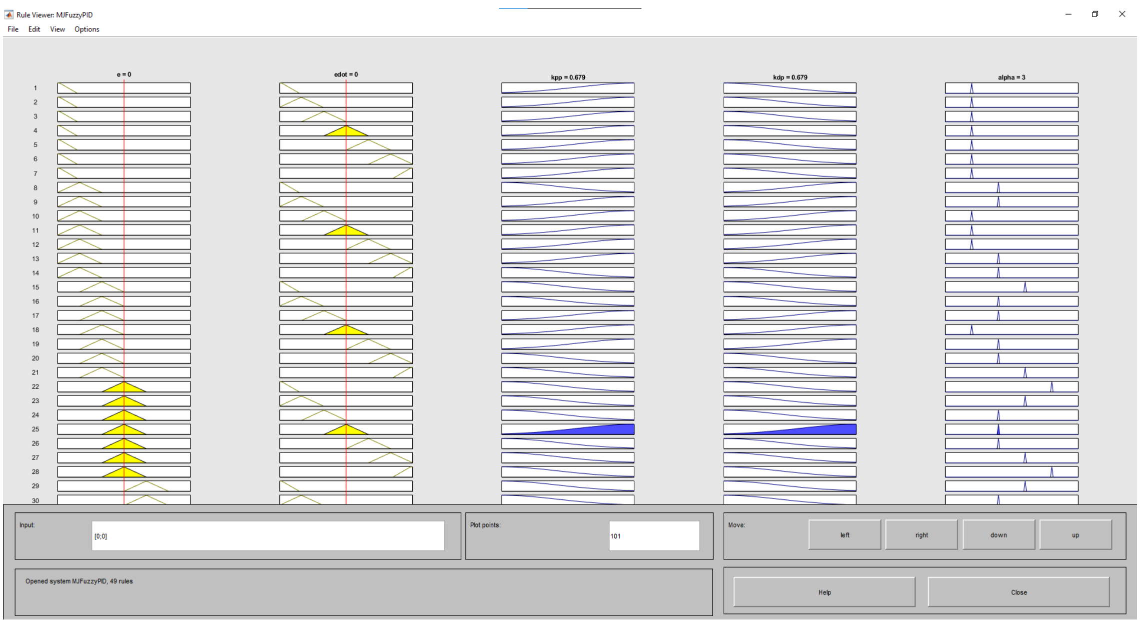
شکل ۵۹: توابع تعلق مربوط به مشتق k_dp.



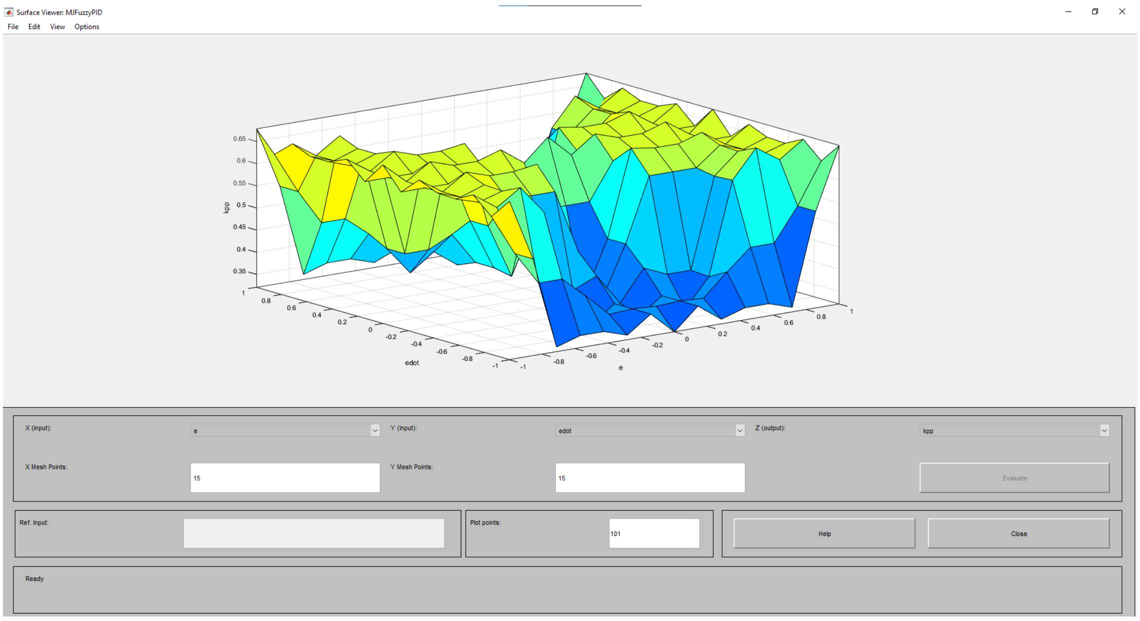
شکل ۶۰: توابع تعلق مربوط به مشتق α.



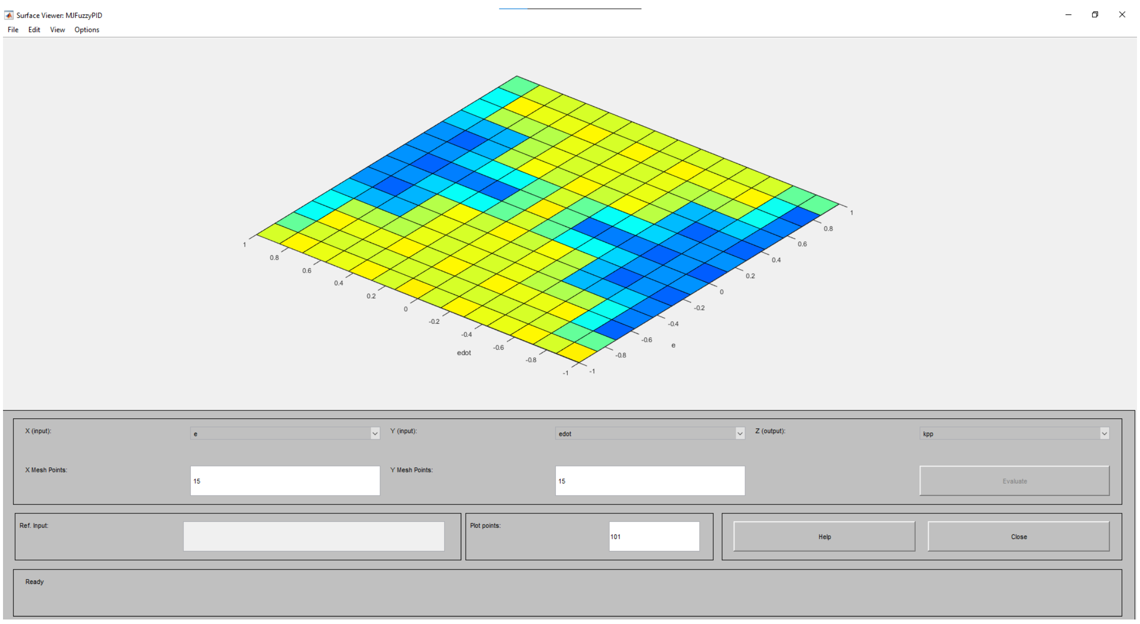
شکل ۶۱: تعیین قوانین فازی.



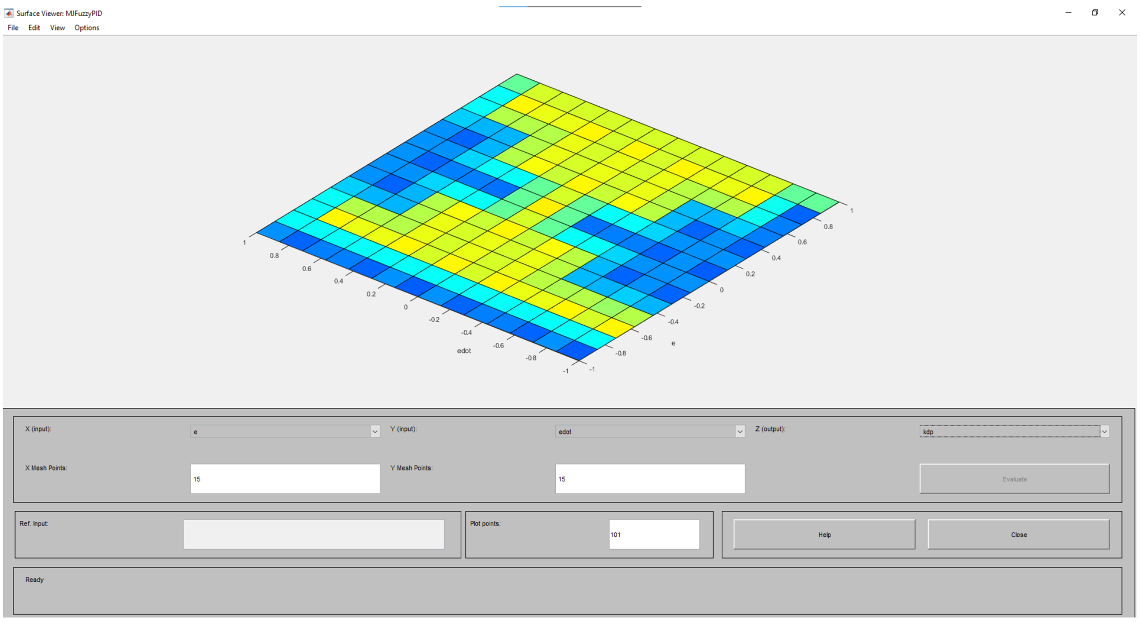
شکل ۶۲: نمایش قوانین فازی.



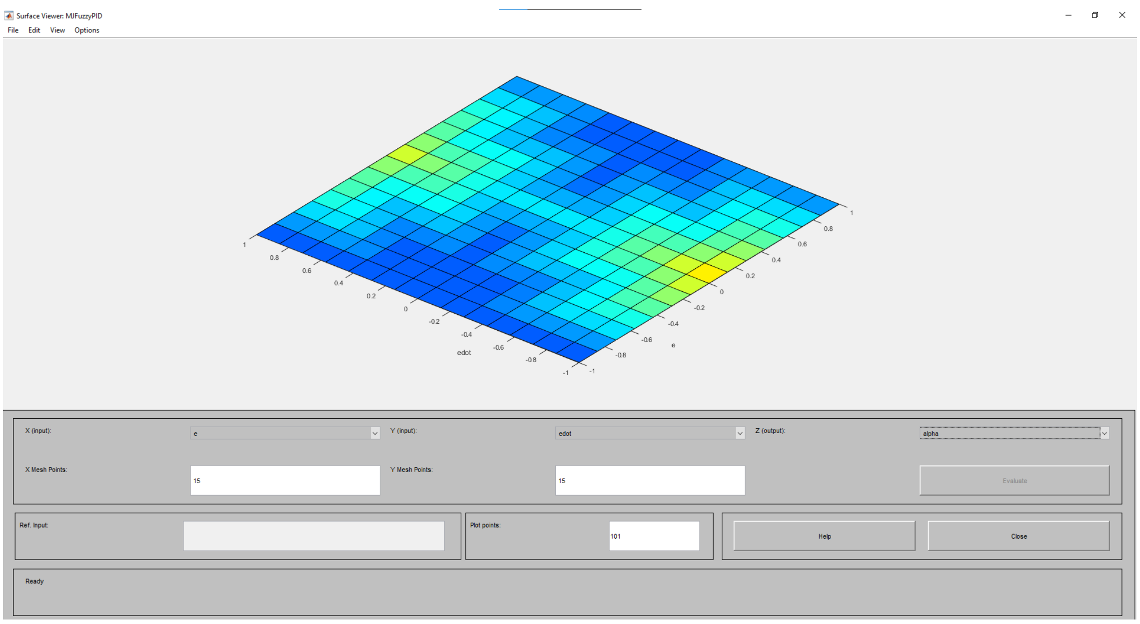
شکل ۶۳: سطح سیستم فازی.



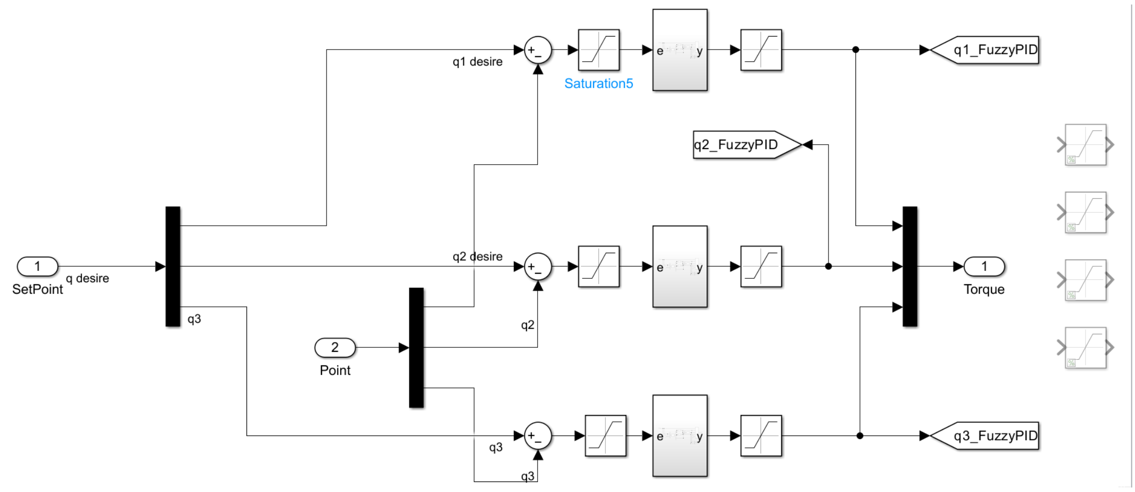
شکل ۶۴: سطح سیستم فازی.



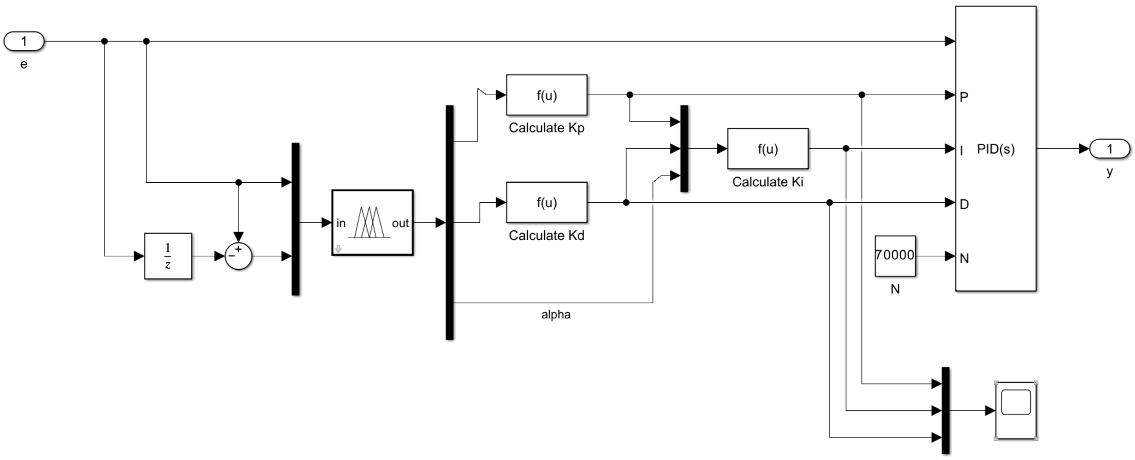
شکل ۶۵: سطح سیستم فازی.



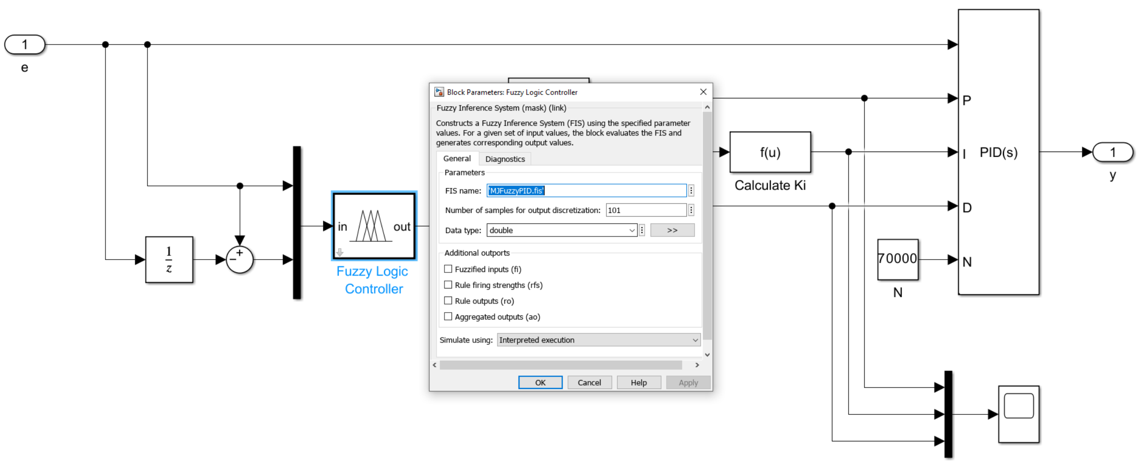
شکل ۶۶: سطح سیستم فازی.



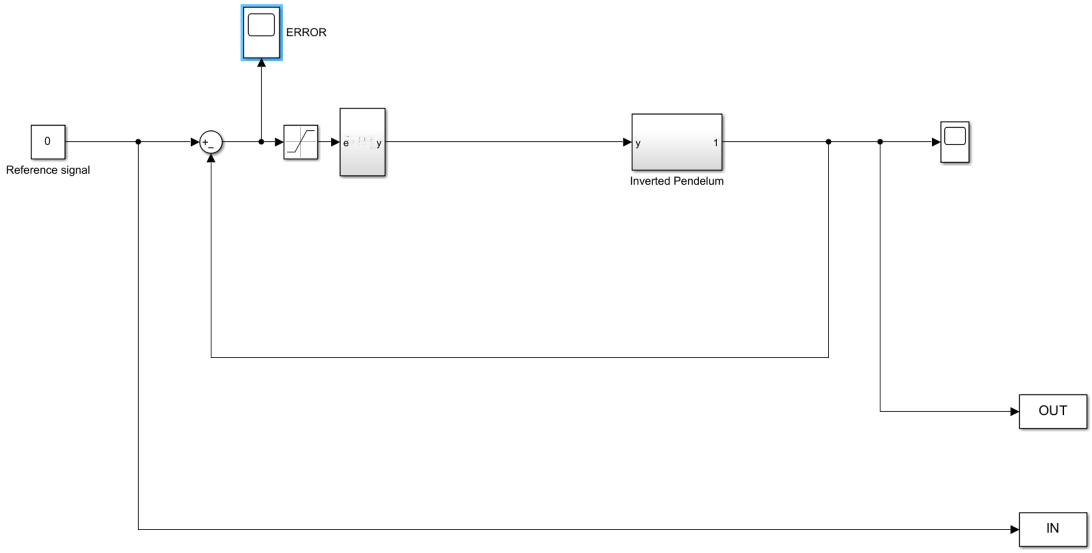
شکل ۶۷: نمای کلی بلوک کنترلی.



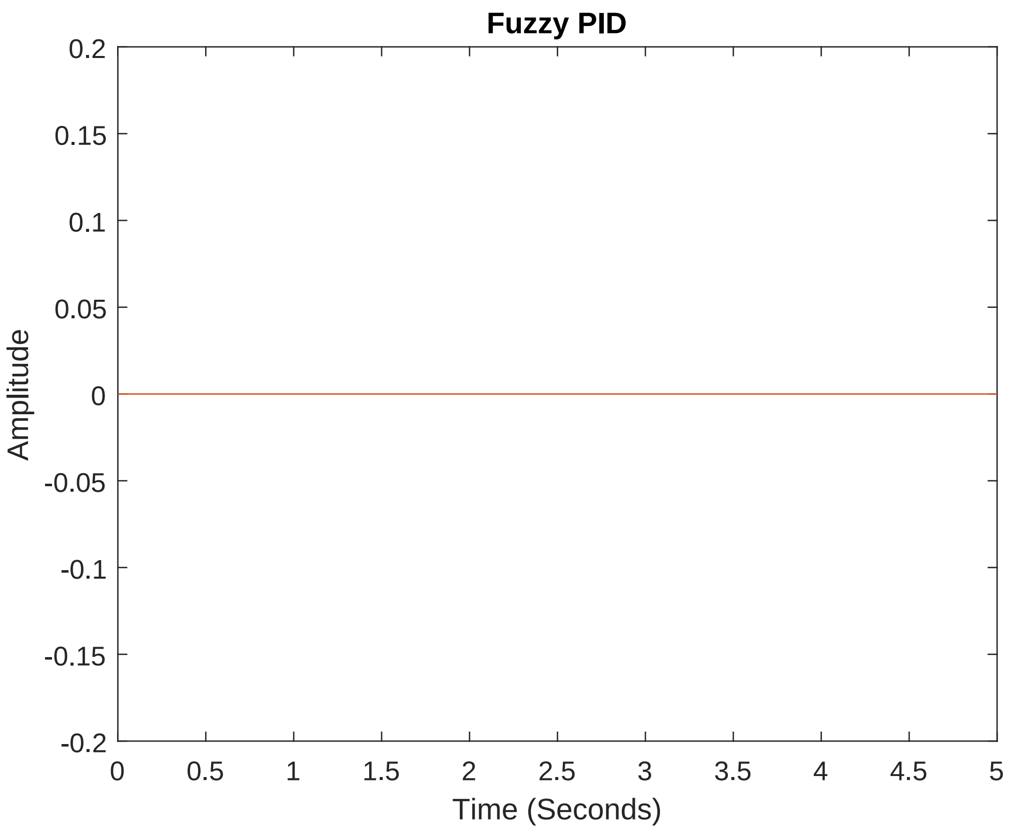
شکل ۶۸: مغز کنترلی FuzzyPID.



شکل ۶۹: مغز کنترلی FuzzyPID.



شکل ۷۰: سیمولینک.



شکل ۷۱: نتیجه.