



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

تمرین درس محاسبات نرم  
در رشته مهندسی برق گرایش مهندسی کنترل

عنوان

تمرین چهارم: شبکه‌های عصبی

نگارش

محمدحسین احمدی

استاد درس

دکتر مهدی علیاری شوره‌دلی

بهمن‌ماه ۱۴۰۱

# فهرست مطالب

## فهرست شکل ها

ج

### پاسخ سوالات

۱	پاسخ سوال ۲.۶
۱	راه حل اول
۴	پاسخ سوال ۳.۶
۱۴	پاسخ سوال ۳.۷
۱۸	پاسخ سوال ۴.۱۱
۱۸	پاسخ سوال ۴.۱۲
۲۰	i
۲۱	ii
۲۲	iii
۲۲	iv
۲۳	پاسخ سوال ۷.۱۰
۲۳	i
۲۵	ii
۲۶	iii
۲۸	iv
۲۹	پاسخ سوال ۷.۱۱
۳۴	پاسخ سوال ۹.۱۰
۳۴	راه حل اول
۳۹	راه حل دوم
۴۹	پاسخ سوال ۱۱.۱
۴۹	i
۵۷	ii
۶۵	iii
۷۵	iv
۷۹	پاسخ سوال ۱۱.۲
۷۹	i

۸۲	.....	ii
۸۷	.....	iii
۹۱	.....	iv
۹۶	.....	پاسخ سوال ۱۱.۱۴
۹۶	.....	i
۹۷	.....	ii
۹۷	.....	پاسخ سوال ۱۱.۲۵
۹۷	.....	راه حل اول
۱۰۵	.....	راه حل دوم
۱۰۸	.....	پاسخ سوال ۱۲.۱۴
۱۱۱	.....	تخمین توابع مقاله ژنگ با MLP
۱۱۱	.....	i
۱۱۵	.....	ii
۱۲۰	.....	iii
۱۲۴	.....	iv
۱۲۹	.....	طراحی کنترل کننده Fuzzy PID
۱۲۹	.....	راه حل اول
۱۴۰	.....	راه حل دوم

# فهرست شکل‌ها

۱	نتایج سوال ۲۰۶ . . . . .	۳
۲	نمودار سوال ۳۰۶ . . . . .	۱۴
۳	نمودار سوال ۳۰۷ . . . . .	۱۷
۴	نمودار قسمت اول سوال ۴۰۱۲ . . . . .	۲۱
۵	نمودار قسمت چهارم سوال ۴۰۱۲ . . . . .	۲۳
۶	مرز سوال پنجم . . . . .	۲۶
۷	بازسازی اعداد از روی قسمتی از آن‌ها (Pseudoinverse) . . . . .	۳۳
۸	بازسازی اعداد از روی قسمتی از آن‌ها (Hebb) . . . . .	۳۳
۹	بازسازی اعداد از روی نویزی شده آن‌ها (Pseudoinverse) . . . . .	۳۴
۱۰	بازسازی اعداد از روی نویزی شده آن‌ها (Hebb) . . . . .	۳۴
۱۱	پاسخ سوال ۹۰۱۰ . . . . .	۳۶
۱۲	پاسخ سوال ۹۰۱۰ . . . . .	۳۶
۱۳	پاسخ سوال ۹۰۱۰ . . . . .	۳۹
۱۴	مرزهای تصمیم‌گیری قسمت اول سوال ۱۱-۱ ( $t_1$ ) . . . . .	۵۰
۱۵	مرزهای تصمیم‌گیری قسمت اول سوال ۱۱-۱ ( $t_2$ ) . . . . .	۵۲
۱۶	مرزهای تصمیم‌گیری قسمت اول سوال ۱۱-۱ . . . . .	۵۴
۱۷	مرزهای تصمیم‌گیری قسمت دوم سوال ۱۱-۱ ( $t_1$ ) . . . . .	۵۸
۱۸	مرزهای تصمیم‌گیری قسمت دوم سوال ۱۱-۱ ( $t_2$ ) . . . . .	۵۹
۱۹	مرزهای تصمیم‌گیری قسمت دوم سوال ۱۱-۱ . . . . .	۶۱
۲۰	مرزهای تصمیم‌گیری قسمت سوم سوال ۱۱-۱ ( $t_1$ ) . . . . .	۶۶
۲۱	مرزهای تصمیم‌گیری قسمت سوم سوال ۱۱-۱ ( $t_2$ ) . . . . .	۶۷
۲۲	مرزهای تصمیم‌گیری قسمت سوم سوال ۱۱-۱ ( $t_3$ ) . . . . .	۶۹
۲۳	مرزهای تصمیم‌گیری قسمت سوم سوال ۱۱-۱ . . . . .	۷۱
۲۴	نتیجه سوال ۱۱-۲-۱ . . . . .	۸۳
۲۵	نتیجه سوال ۱۱-۲-۱ . . . . .	۸۳
۲۶	نتیجه سوال ۱۱-۲-۲ . . . . .	۸۷
۲۷	نتیجه سوال ۱۱-۲-۲ . . . . .	۸۷
۲۸	نتیجه سوال ۱۱-۲-۳ . . . . .	۹۱
۲۹	نتیجه سوال ۱۱-۲-۳ . . . . .	۹۱

۳۰	نتیجه سوال ۱۱-۲-۴	۹۵
۳۱	نتیجه سوال ۱۱-۲-۴	۹۵
۳۲	نتیجه سوال ۱۱-۲۵	۱۰۰
۳۳	نتیجه سوال ۱۱-۲۵	۱۰۲
۳۴	نتیجه سوال ۱۱-۲۵	۱۰۳
۳۵	نتیجه سوال ۱۱-۲۵	۱۰۴
۳۶	نتیجه سوال ۱۱-۲۵	۱۰۷
۳۷	نتیجه قسمت اول سوال اول	۱۱۵
۳۸	نتیجه قسمت اول سوال اول	۱۱۶
۳۹	نتیجه قسمت دوم سوال اول	۱۲۰
۴۰	نتیجه قسمت دوم سوال اول	۱۲۱
۴۱	سیمولینک	۱۲۵
۴۲	نتیجه قسمت سوم سوال اول	۱۲۵
۴۳	نتیجه قسمت سوم سوال اول	۱۲۶
۴۴	سیمولینک	۱۲۷
۴۵	نتیجه قسمت چهارم سوال اول	۱۲۸
۴۶	نتیجه قسمت چهارم سوال اول	۱۲۸
۴۷	نتیجه راه حل اول سوال طراحی	۱۴۰
۴۸	طرح کلی کنترل مدنظر ما	۱۴۱
۴۹	هسته کلی سیستم مدنظر ما	۱۴۲
۵۰	مقاله مورد توجه برای ایده تنظیم قواعد	۱۴۲
۵۱	تقسیمات کلی فازی خطا و مشتق آن	۱۴۳
۵۲	توابع عضویت بزرگ و کوچک	۱۴۴
۵۳	نمودار کلی برای تعیین توابع و قوانین فازی	۱۴۵
۵۴	نمای کلی استنتاج فازی	۱۴۷
۵۵	نمای کلی سیستم فازی	۱۴۸
۵۶	توابع تعلق مربوط به خطا	۱۴۸
۵۷	توابع تعلق مربوط به مشتق خطا	۱۴۹
۵۸	توابع تعلق مربوط به $k_{pp}$	۱۴۹
۵۹	توابع تعلق مربوط به مشتق $k_{dp}$	۱۵۰
۶۰	توابع تعلق مربوط به مشتق $\alpha$	۱۵۰
۶۱	تعیین قوانین فازی	۱۵۱
۶۲	نمایش قوانین فازی	۱۵۱
۶۳	سطح سیستم فازی	۱۵۲
۶۴	سطح سیستم فازی	۱۵۲
۶۵	سطح سیستم فازی	۱۵۳
۶۶	سطح سیستم فازی	۱۵۳
۶۷	نمای کلی بلوک کنترلی	۱۵۴

۱۵۴	.....	FuzzyPID مغز کنترلی	۶۸
۱۵۴	.....	FuzzyPID مغز کنترلی	۶۹
۱۵۵	.....	سیمولینک	۷۰
۱۵۵	.....	نتیجه	۷۱

# پاسخ سوالات

## پاسخ سوال ۲.۶

### راه حل اول

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

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def hardlim(x):
5     return np.where(x >= 0, 1, 0)
6
7 def hardlims(x):
8     return np.where(x >= 0, 1, -1)
9
10 def purelin(x):
11     return x
12
13 def satlin(x):
14     return np.maximum(0, np.minimum(x, 1))
15
16 def logsig(x):
17     return 1/(1+np.exp(-x))
18
19 def htansig(x):
20     return (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
21
22 def poslin(x):
23     return np.maximum(0, x)
24
25 def compet(x):
26     return (x==np.max(x)).astype(int)
```

```

21 def plotfunc(k) :
22     plt.plot(p, k, '-b')
23     plt.xlabel('p', color = '#1C2833' )
24     plt.ylabel('n', color = '#1C2833' )
25     plt.grid()
26
27 weights = [2, 1, 1, -1]
28 bias = [2, -1, 0]
29 p = np.linspace(-3, 3, 100) #Plot indicated variable versus p for -3<p<3
30
31 #i.
32 n_1 = p*weights[0] + bias[0]
33 #ii.
34 a_1 = satlin(n_1)
35 #iii.
36 n_2 = p*weights[1] + bias[1]
37 #iv.
38 a_2 = satlin(n_2)
39 # v.
40 n_n = a_1*weights[2] + a_2*weights[3] + bias[2]
41 #vi.
42 a_a = purelin(n_n)
43
44 # plot i .
45 plt.title('Graph of $n_1^1$')
46 plotfunc(n_1)
47 plt.savefig('020601.pdf')
48 plt.show()
49 # plot ii.
50 plt.title('Graph of $a_1^1$')
51 plotfunc(a_1)
52 plt.savefig('020602.pdf')
53 plt.show()
54 # plot iii.
55 plt.title('Graph of $n_2^1$')

```

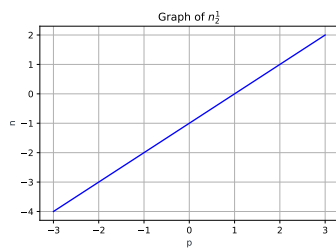


```

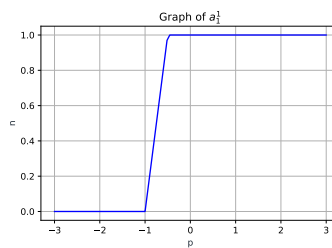
56 plotfunc(n_2)
57 plt.savefig('020603.pdf')
58 plt.show()
59 # plot iv.
60 plt.title('Graph of $a_2^1$')
61 plotfunc(a_2)
62 plt.savefig('020604.pdf')
63 plt.show()
64 # plot v.
65 plt.title('Graph of $n^2$')
66 plotfunc(n_n)
67 plt.savefig('020605.pdf')
68 plt.show()
69 # plot vi.
70 plt.title('Graph of $a^2$')
71 plotfunc(a_a)
72 plt.savefig('020606.pdf')
73 plt.show()

```

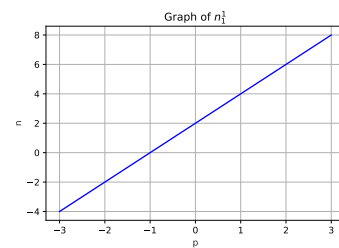
نتایج به صورتی است که در شکل ۱ نشان داده شده است.



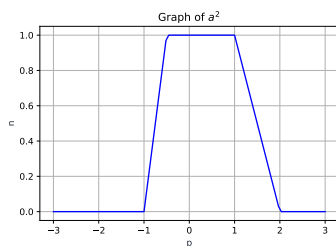
iii (ج)



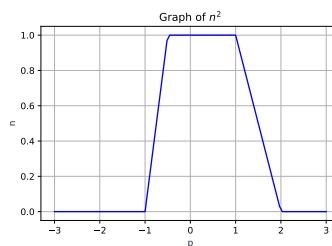
ii (ب)



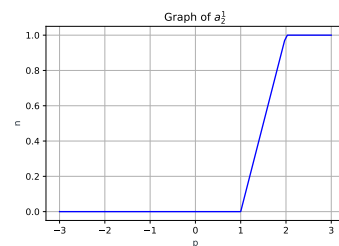
i (آ)



vi (و)



v (ه)



iv (د)

شکل ۱: نتایج سوال ۲.۶.

## توضیح پوشه کدها

در پوشه اصلی این سوال (Q1\_02\_06)، فایل‌های پایتون مربوط به این سوال قرار گرفته‌اند. راه حل دومی هم برای سوال در نظر گرفته شده است که در همان پوشه موجود است. هم‌چنین فایل اجرای برخط کدها از طریق **پیوند گوگل کولب** در دسترس است.

## پاسخ سوال ۳.۶

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

```

1  ## transfer_functions.py ##
2
3  import numpy as np
4
5  # Transfer functions as defined by Neural Network Toolbox for MATLAB
6
7  def hardlim(n):
8      """
9      Hard Limit
10     """
11     if n < 0:
12         return 0
13     else:
14         return 1
15
16  def hardlims(n):
17      """
18      Symmetrical Hard Limit
19     """
20     if n < 0:
21         return -1
22     else:
23         return 1

```

```
24
25 def purelin(n):
26     """
27     Linear
28     """
29     return n
30
31 def satlin(n):
32     """
33     Saturating Linear
34     """
35     if n < 0:
36         return 0
37     elif n > 1:
38         return 1
39     else:
40         return n
41
42 def satlins(n):
43     """
44     Symmetric Saturating Linear
45     """
46     if n < -1:
47         return -1
48     elif n > 1:
49         return 1
50     else:
51         return n
52
53 def logsig(n):
54     """
55     Log-Sigmoid
56     """
57     return 1/(1+np.exp(-1*n))
58
```

```

59 def tansig(n):
60     """
61     Hyperbolic Tangent Sigmoid
62     """
63     return (np.exp(n) - np.exp(-1*n))/(np.exp(n) + np.exp(-1*n))
64
65 def poslin(n):
66     """
67     Positive Linear
68     """
69     if n < 0:
70         return 0
71     else:
72         return n
73
74 def compet(n):
75     """
76     Competitive
77     """
78     return (n > 0).astype(np.int)
79
80 ## Perceptron.py ##
81
82 import numpy as np
83
84 from transfer_functions import *
85
86 class Perceptron(object):
87
88     def __init__(self, W, b, transfer_function=hardlims):
89         self.Weights = W
90         self.bias = b
91         self.transfer_function = np.vectorize(transfer_function)
92
93     def classify(self, prototype):
94         net_input = self.Weights.dot(prototype) + self.bias

```

```

94         return self.transfer_function(net_input)
95
96
97 if __name__ == "__main__":
98     # prototype = [shape, texture, weight] as a column vector
99     orange_prototype = np.array([1, -1, -1]).reshape((3, 1))
100     apple_prototype = np.array([1, 1, -1]).reshape((3, 1))
101
102     # Weight matrix and bias determined by decision boundary.
103     decision_boundary = (orange_prototype != apple_prototype).astype(np.int).
104     reshape((1, len(orange_prototype)))
105     print(decision_boundary)
106
107     bias = 0
108
109     fruit_perceptron = Perceptron(W=decision_boundary, b=bias)
110
111     test_prototype = np.array([-1, -1, -1]).reshape((3, 1))
112     print(fruit_perceptron.classify(test_prototype))
113
114 ## HammingNetwork.py ##
115
116 import numpy as np
117
118 from transfer_functions import *
119
120 class HammingNetwork(object):
121
122     def __init__(self, prototypes):
123         self.feedForwardLayer = self.FeedForwardLayer(W=prototypes)
124         self.recurrentLayer = self.RecurrentLayer()
125
126     def classify(self, obj):
127         a1 = self.feedForwardLayer.propagate(obj=obj)
128         recurrent_result = self.recurrentLayer.propagate(initial_a=a1)
129         return compet(recurrent_result)

```

```

128
129 class FeedFowardLayer(object):
130     def __init__(self, W, transfer_function = purelin):
131         self.Weights = W
132         self.bias = np.array(self.Weights.shape[0]).repeat(self.Weights.
133 shape[0], axis=0).reshape((self.Weights.shape[0], 1))
134         self.transfer_function = np.vectorize(transfer_function, otypes=[np
135 .float])
136
137     def propagate(self, obj):
138         return self.transfer_function(self.Weights.dot(obj) + self.bias)
139
140
141 class RecurrentLayer(object):
142     def __init__(self, W = None, transfer_function = poslin):
143         if W is None:
144             self.Weights = W
145         else:
146             self.Weights = None
147         self.transfer_function = np.vectorize(transfer_function, otypes=[np
148 .float])
149
150     def propagate(self, initial_a):
151         if self.Weights is None:
152             s = initial_a.shape[0]
153             epsilon = 1 / (s - 1)
154             epsilon -= 0.01
155             epsilon *= -1
156             self.Weights = np.ones((s, s))
157             for i in range(s):
158                 for j in range(s):
159                     if i != j:
160                         self.Weights[i][j] = epsilon
161
162             a2 = self.transfer_function(self.Weights.dot(initial_a))
163
164             while True:

```

```

160         a3 = self.transfer_function(self.Weights.dot(a2))
161         if a2.all() != a3.all():
162             a2 = a3
163         else:
164             return a3
165
166
167 if __name__ == "__main__":
168     # prototype = [shape, texture, weight] as a column vector
169     orange_prototype = np.array([1, -1, -1]).reshape((3, 1))
170     apple_prototype = np.array([1, 1, -1]).reshape((3, 1))
171     prototypes = np.array([orange_prototype.T[0], apple_prototype.T[0]])
172
173     test_fruit = np.array([-1, -1, -1]).reshape((3, 1))
174
175     hammingFruitClassifier = HammingNetwork(prototypes=prototypes)
176     print(hammingFruitClassifier.classify(obj=test_fruit))
177
178 ## HopfieldNetwork.py ##
179
180 import numpy as np
181
182 from transfer_functions import *
183
184 class HopfieldNetwork(object):
185
186     def __init__(self, weights = np.array([[0.2, 0, 0], [0, 1.2, 0], [0, 0,
187         0.2]]), transfer_function = satlins, bias=np.array([0.9, 0, -0.9])):
188         self.Weights = weights
189         self.bias = bias.reshape((weights.shape[0], 1))
190         self.transfer_function = np.vectorize(transfer_function, otypes=[np.
191             float])
192         self.activations = list([])
193
194     def classify(self, a0):

```

```

193     self.activations.append(a0)
194     a1 = self.transfer_function(self.Weights.dot(a0) + self.bias)
195     self.activations.append(a1)
196     if np.array_equal(a0, a1):
197         return a1
198     else:
199         while True:
200             an = self.transfer_function(self.Weights.dot(a1) + self.bias)
201             self.activations.append(an)
202             if not np.array_equal(an, a1):
203                 a1 = an
204             else:
205                 return an
206
207 if __name__ == "__main__":
208     test_obj = np.array([-1, -1, -1]).reshape((3, 1))
209
210     test_hopfield = HopfieldNetwork()
211     print(test_hopfield.classify(test_obj))

```

در ادامه دستوراتی مطابق زیر را اجرا می‌کنیم:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 from Perceptron import Perceptron
5 from HammingNetwork import HammingNetwork
6 from HopfieldNetwork import HopfieldNetwork
7
8 if __name__ == "__main__":
9     print("We have the following two prototype vectors:",
10           "\n\tp1 = [-1, 1].T    p2 = [1,1].T")
11     p1 = np.array([-1, 1]).reshape((2, 1))
12     p2 = np.array([1, 1]).reshape((2, 1))
13     prototypes = np.array([p1.T[0], p2.T[0]])
14

```



```

15     x = np.linspace(0, 0, 100)
16     y = np.linspace(0, 2, 100)
17
18     print("\ni.    Find and sketch a decision boundary for a perceptron network
19           that will recognize these two vectors.")
20     plt.scatter(prototypes[0], prototypes[1])
21     plt.plot(x, y, color='black')
22     plt.grid(True)
23     plt.show()
24
25     print("\nii.   Find weights and bias that will produce the decision
26           boundary you found in part i.")
27
28     Weights = np.array([1, 0])
29     bias = 0
30
31     e3_6_perceptron = Perceptron(W = Weights, b = bias)
32
33     print("p1", p1, "classification:", e3_6_perceptron.classify(p1))
34     print("p2", p2, "classification:", e3_6_perceptron.classify(p2))
35
36     print("\niv.   For the vector given below, calculate the net input, n, and
37           the network output, a, for the network you have designed. Does the network
38           produce a good output? Explain.",
39           "\n\tp_test = [0.5, -0.5].T")
40
41     p_test = np.array([0.5, -0.5]).reshape((2,1))
42     print("p_test Classification = ", e3_6_perceptron.classify(p_test))
43     print("This is a good classification because the test array lands on the
44           right side of the purposed decision boundary and gets classified as such.")
45
46     print("\nv.    Design a Hamming network to recognize the two vectors used
47           in part i.")
48
49     e3_6_hamming = HammingNetwork(prototypes=prototypes)
50     print("p1", p1, "classification:", e3_6_hamming.classify(p1))
51     print("p2", p2, "classification:", e3_6_hamming.classify(p2))

```

```

44
45     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.")
46     print("p_test Classification = ", e3_6_hamming.classify(p_test))
47     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.")
48
49     print("\nvii. Design a Hopfield network to recognize the two vectors used
        in part i.")
50     Weights = np.array([[1.2, 0], [0, 0.2]])
51     bias = np.array([0, 0.9])
52
53     e3_6_hopfield = HopfieldNetwork(weights=Weights, bias = bias)
54     print("p1", p1, "classification:", e3_6_hopfield.classify(p1))
55     print("p2", p2, "classification:", e3_6_hopfield.classify(p2))
56
57     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.")
58     print("p_test Classification = ", e3_6_hopfield.classify(p_test))
59     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:

```

1 We have the following two prototype vectors:
2     p1 = [-1, 1].T     p2 = [1,1].T
3
4 i. Find and sketch a decision boundary for a perceptron network that will
    recognize these two vectors.
5 In-Text...
6
7 ii. Find weights and bias that will produce the decision boundary you found
    in part i.

```

```

8 p1 [[-1]
9 [ 1]] classification: [-1]
10 p2 [[1]
11 [1]] classification: [1]
12
13 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.
14 p_test = [0.5, -0.5].T
15 p_test Classification = [1]
16 This is a good classification because the test array lands on the right side of
      the purposed decision boundary and gets classified as such.
17
18 v. Design a Hamming network to recognize the two vectors used in part i.
19 p1 [[-1]
20 [ 1]] classification: [[1]
21 [0]]
22 p2 [[1]
23 [1]] classification: [[0]
24 [1]]
25
26 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.
27 p_test Classification = [[0]
28 [1]]
29 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.
30
31 vii. Design a Hopfield network to recognize the two vectors used in part i.
32 p1 [[-1]
33 [ 1]] classification: [[-1.]
34 [ 1.]]
35 p2 [[1]
36 [1]] classification: [[1.]
37 [1.]]

```

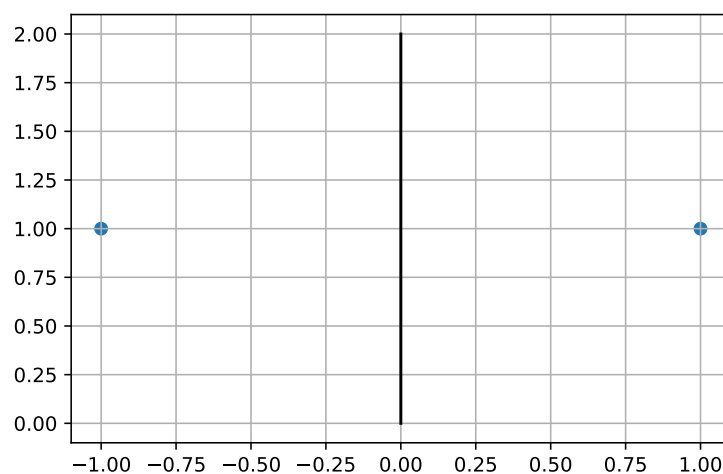
38

39 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.

40 p\_test Classification = [[1.]

41 [1.]]

42 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 پوشه کدها

در پوشه اصلی این سوال (Q2\_03\_06\_07)، فایل‌های پایتون مربوط به این سوال قرار گرفته‌اند. هم‌چنین فایل اجرای برخط کدها از طریق **پیوند گوگل کولب** 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:

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

```

3 from HammingNetwork import HammingNetwork
4
5 if __name__ == "__main__":
6     print("We want to design a Hamming network to recognize the following
7     prototype vectors:",
8         "\n\tp1 = [1, 1].T", "p2 = [-1, -1].T", "p3 = [-1, 1].T")
9     p1 = np.array([1, 1]).reshape((2, 1))
10    p2 = np.array([-1, -1]).reshape((2, 1))
11    p3 = np.array([-1, 1]).reshape((2, 1))
12
13    prototypes = np.array([p1.T, p2.T, p3.T]).reshape((3, 2))
14    print("prototypes", prototypes)
15
16    e3_7_hamming = HammingNetwork(prototypes=prototypes)
17
18    print("p1", p1, "classification:", e3_7_hamming.classify(p1))
19    print("p2", p2, "classification:", e3_7_hamming.classify(p2))
20    print("p3", p3, "classification:", e3_7_hamming.classify(p3))
21
22    print("\ni. Find the weight matrices and bias vectors for the Hamming
23    network.",
24        "\n\tFeedForwardLayer", "Weights", e3_7_hamming.feedForwardLayer.
25        Weights, "bias", e3_7_hamming.feedForwardLayer.bias,
26        "\n\tRecurrentLayer", "Weights", e3_7_hamming.recurrentLayer.Weights,
27        "bias", "There are no biases in the recurrent layer.")
28
29    print("\niii. Apply the following input vector and calculate the total
30    network response (iterating the second layer to convergence). Explain the
31    meaning of the final network output.",
32        "\n\tp_test = [1, 0].T")
33    p_test = np.array([1, 0]).reshape((2, 1))
34    print("\tClassification:", e3_7_hamming.classify(p_test))
35
36    print("\niv. Sketch the decision boundaries for this network. Explain how
37    you determined the boundaries.")

```

```

31 x = np.linspace(-2, 2, 100)
32 y = np.linspace(0, 0, 100)
33
34 x2 = np.linspace(0, 0, 100)
35 y2 = np.linspace(-2, 2, 100)
36
37 plt.scatter(prototypes[:, 0], prototypes[:, 1])
38 plt.plot(x, y, color='black')
39 plt.plot(x2, y2, color='black')
40 plt.grid(True)
41 plt.savefig('030701.pdf')
42 plt.show()

```

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

```

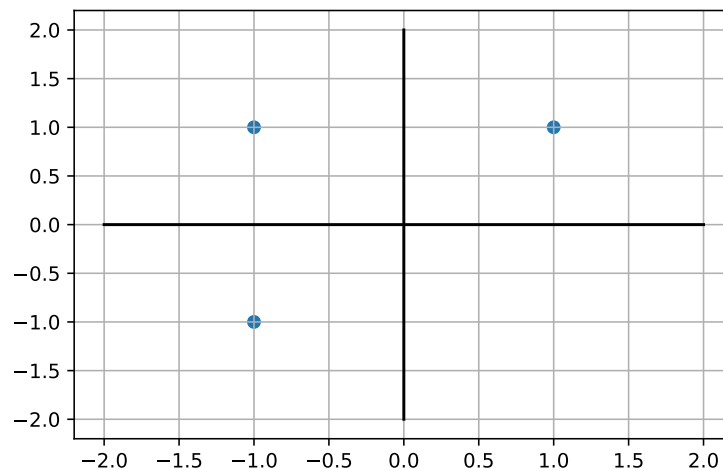
1 We want to design a Hamming network to recognize the following prototype
  vectors:
2 p1 = [1, 1].T p2 = [-1, -1].T p3 = [-1, 1].T
3 prototypes [[ 1  1]
4 [-1 -1]
5 [-1  1]]
6 p1 [[1]
7 [1]] classification: [[1]
8 [0]
9 [0]]
10 p2 [[-1]
11 [-1]] classification: [[0]
12 [1]
13 [0]]
14 p3 [[-1]
15 [ 1]] classification: [[0]
16 [0]
17 [1]]
18
19 i. Find the weight matrices and bias vectors for the Hamming network.
20 FeedForwardLayer Weights [[ 1  1]

```

```

21 [-1 -1]
22 [-1  1]] bias [[3]
23 [3]
24 [3]]
25 RecurrentLayer Weights [[ 1.   -0.49 -0.49]
26 [-0.49  1.   -0.49]
27 [-0.49 -0.49  1.  ]] bias There are no biases in the recurrent layer.
28
29 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.
30 p_test = [1, 0].T
31 Classification: [[1]
32 [0]
33 [0]]
34
35 iv. Sketch the decision boundaries for this network. Explain how you
      determined the boundaries.
36 In-Text...

```



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

## توضیح پوشه کدها

در پوشه اصلی این سوال (Q2\_03\_06\_07)، فایل‌های پایتون مربوط به این سوال قرار گرفته‌اند. هم‌چنین فایل اجرای برخط کدها از طریق [پیوند گوگل کولب](#) در دسترس است.

## پاسخ سوال ۴.۱۱

## پاسخ سوال ۴.۱۲

برای حل این سوال تابع زیر را تعریف می‌کنیم:

```

1 function E412(p, t)
2 error = 1;
3 W = [rand rand; rand rand];
4 b = [rand rand]';
5 count = 0;
6 while (error ~= 0 & count<=50)
7     count = count + 1;
8     error = 0;
9     for i = 1 : 8
10        a = hardlim(W * p(:, i) + b);
11        e = t(:, i) - a;
12        newW = W + e * p(:, i)';
13        W = newW;
14        b = b + e;
15        if (~isequal(e, [0, 0]'))
16            error = error + 1;
17        end
18    end
19 end
20 if error ~=0
21     fprintf('Program terminated with no satisfying weight or bias after 50
    iterations.\n');

```



```

22     return;
23 end
24
25 W
26 b
27
28 figure
29 hold on
30
31 plot(p(1,1), p(2,1), 'r*');
32 plot(p(1,2), p(2,2), 'r*');
33 plot(p(1,3), p(2,3), 'g*');
34 plot(p(1,4), p(2,4), 'g*');
35 plot(p(1,5), p(2,5), 'b*');
36 plot(p(1,6), p(2,6), 'b*');
37 plot(p(1,7), p(2,7), 'c*');
38 plot(p(1,8), p(2,8), 'c*');
39
40 x = -3:1:3;
41 y = -(W(1,1) * x + b(1)) / W(1,2);
42 plot(x, y);
43 x = -3:1:3;
44 y = -(W(2,1) * x + b(2)) / W(2,2);
45 plot(x, y);

```

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

```

1 clc;
2 clear all;
3 close all;
4 disp("4 - 12 - i:")
5 disp("_____")
6 p = [1 1 2 2 -1 -2 -1 -2; 1 2 2 0 2 1 -1 -2];
7 t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
8 hold on
9 plot(p(1,1), p(2,1), 'R*');

```

```

10 plot(p(1,2), p(2,2), 'R*');
11 plot(p(1,3), p(2,3), 'G*');
12 plot(p(1,4), p(2,4), 'G*');
13 plot(p(1,5), p(2,5), 'B*');
14 plot(p(1,6), p(2,6), 'B*');
15 plot(p(1,7), p(2,7), 'Y*');
16 plot(p(1,8), p(2,8), 'Y*');
17 x = -2.5:1:2.5;
18 y = x;
19 plot(x, y)
20 disp("4 - 12 - ii:")
21 disp("_____")
22 p = [1 1 2 2 -1 -2 -1 -2; 1 2 2 0 2 1 -1 -2];
23 t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
24 E412(p, t)
25 disp("4 - 12 - iii & iv:")
26 disp("_____")
27 p = [1 1 2 2 -1 -2 -1 -2; 1 2 1.5 0 2 1 -1 -2];
28 t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
29 E412(p, t)

```

i

اگر بردار ورودی  $p_3 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$  تغییر دهیم مسأله با دو شبکهٔ دینورونه به صورت خطی تفکیک پذیر 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 می‌گیرند که هرچقدر هم مرز و خط را بچرخانیم امکان طبقه‌بندی صحیح نقاط را نخواهیم داشت.

```

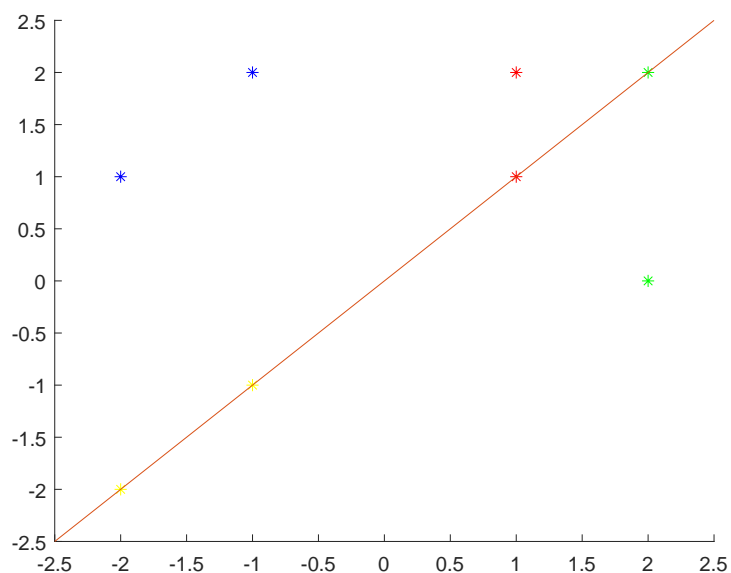
1 p = [1 1 2 2 -1 -2 -1 -2; 1 2 2 0 2 1 -1 -2];
2 t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
3 hold on
4 plot(p(1,1), p(2,1), 'R*');

```

```

5 plot(p(1,2), p(2,2), 'R*');
6 plot(p(1,3), p(2,3), 'G*');
7 plot(p(1,4), p(2,4), 'G*');
8 plot(p(1,5), p(2,5), 'B*');
9 plot(p(1,6), p(2,6), 'B*');
10 plot(p(1,7), p(2,7), 'Y*');
11 plot(p(1,8), p(2,8), 'Y*');
12 x = -2.5:1:2.5;
13 y = x;
14 plot(x, y)

```



شکل ۴: نمودار قسمت اول سوال ۴.۱۲.

ii

با فراخوانی تابع تعریف شده به صورت زیر، نتیجه نشان داده شده به دست آورده می شود که این نشان می دهد که شبکه پرسپترون نمی تواند مرزهای تصمیم گیری را برای تفکیک بردارهای ورودی به شکل صحیح پیدا کند.

```

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

```

```

4 -----
5
6 Program terminated with no satisfying weight or bias after 50 iterations.

```

iii

با تغییر نقطه  $p_3$  به  $p_3 = \begin{bmatrix} 2 \\ 1/5 \end{bmatrix}$ ، مسأله به صورت خطی تفکیک پذیر است.

iv

با فراخوانی تابع تعریف شده به صورت زیر، نتیجه نشان داده شده در زیر و در شکل ۵ به دست آورده می شود

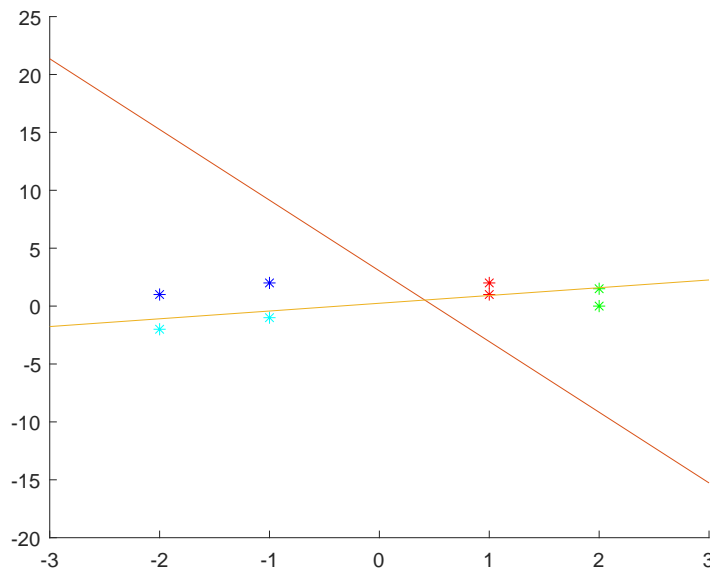
```

1 p = [1 1 2 2 -1 -2 -1 -2; 1 2 1.5 0 2 1 -1 -2];
2 t = [0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1];
3 E412(p, t)
4 -----
5
6 W =
7     -3.6404     1.0268
8     1.5004    -1.6730
9
10 b =
11     0.2590
12     0.0459

```

توضیح پوشه کدها

در پوشه اصلی این سوال (Q4\_04\_12)، فایل های کد مربوط به این سوال قرار گرفته اند.



شکل ۵: نمودار قسمت چهارم سوال ۴.۱۲.

## پاسخ سوال ۷.۱۰

i

ورودی و اهداف را به صورت زیر در نظر می گیریم:

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_1 = [1] \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, t_2 = [-1] \right\} \quad (1)$$

برای محاسبات خواسته شده از دستورات زیر استفاده می کنیم. نتایج در ادامه کد آورده شده است.

```

1 clc
2 clear all
3 close all
4
5 P = [1 1; 1 -1];
6 p1 = [1; 1];
7 p2 = [1; -1];
8 T = [1 -1];
9

```

```

10 Wh = T * P'
11 PPseudoinverse = inv(P'*P)*P'
12 W = T * pinv(P)
13
14 a1h = Wh*p1
15 a1p = W*p1
16 -----
17
18 Wh =
19      0      2
20
21 PPseudoinverse =
22      0.5000      0.5000
23      0.5000     -0.5000
24
25 W =
26     -0.0000      1.0000
27
28 a1h =
29      2
30
31 a1p =
32      1.0000

```

در واقع:

$$p = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad T = \begin{bmatrix} 1 & -1 \end{bmatrix} \quad (۲)$$

$$w^h = TP^T = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 2 \end{bmatrix} \quad (۳)$$

هم چنین همین دستورات را در محیط پایتون به صورت زیر نوشته ایم و نتایج را در ادامه کد گزارش می کنیم:

```

1 import numpy as np
2
3 P = np.array([[1, 1], [1, -1]])
4 p1 = np.array([[1], [1]])
5 p2 = np.array([[1], [-1]])
6 T = np.array([1, -1])
7
8 Wh = np.dot(T, P.T)
9 PPseudoinverse = np.dot(np.linalg.inv(np.dot(P.T, P)), P.T)
10 W = np.dot(T, np.linalg.pinv(P))
11
12 a1h = np.dot(Wh, p1)
13 a1p = np.dot(W, p1)
14
15 print("Wh = ", Wh)
16 print("PPseudoinverse = ", PPseudoinverse)
17 print("W = ", W)
18 print("a1h = ", a1h)
19 print("a1p = ", a1p)
20 -----
21
22 Wh = [0 2]
23 PPseudoinverse = [[ 0.5  0.5]
24 [ 0.5 -0.5]]
25 W = [-2.22044605e-16  1.00000000e+00]
26 a1h = [2]
27 a1p = [1.]

```

ii

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

```

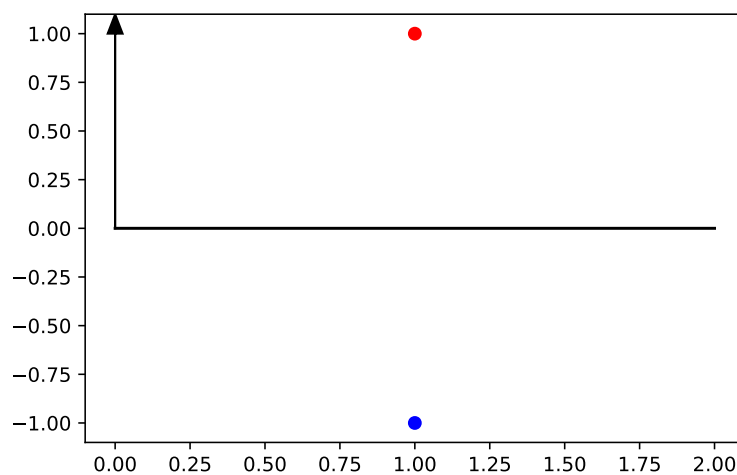
1 import numpy as np
2 import matplotlib.pyplot as plt

```

```

3
4 p1 = np.array([1,1])
5 p2 = np.array([1,-1])
6 plt.plot(p1[0], p1[1], 'ro')
7 plt.plot(p2[0], p2[1], 'bo')
8
9 plt.plot([0,2],[0,0], 'k-')
10 plt.arrow(0,0,0,1,head_width=0.05, head_length=0.1, fc='k', ec='k')
11
12 plt.savefig('E710.pdf')
13 plt.show()

```



شکل ۶: مرز سوال پنجم.

iii

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

```

1 clc
2 clear all
3 close all
4
5 P = [1 1; 1 -1];

```



```

6 p1 = [1; 1];
7 p2 = [1; -1];
8 T = [1 -1];
9
10 Wh = T * P'
11 PPseudoinverse = inv(P'*P)*P'
12 W = T * pinv(P)
13
14 a1h = Wh*p1
15 a1p = W*p1
16 -----
17
18 Wh =
19      0      2
20
21 PPseudoinverse =
22      0.5000      0.5000
23      0.5000     -0.5000
24
25 W =
26     -0.0000      1.0000
27
28 a1h =
29      2
30
31 a1p =
32      1.0000

```

درواقع:

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

iv

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

```
1  clc
2  clear all
3  close all
4
5  P = [1 1; 1 -1];
6  p1 = [1; 1];
7  p2 = [1; -1];
8  T = [1 -1];
9
10 Wh = T * P'
11 PPseudoinverse = inv(P'*P)*P'
12 W = T * pinv(P)
13
14 a1h = Wh*p1
15 a1p = W*p1
16 -----
17
18 Wh =
19      0      2
20
21 PPseudoinverse =
22      0.5000      0.5000
23      0.5000     -0.5000
24
25 W =
26     -0.0000      1.0000
27
28 a1h =
29      2
30
31 a1p =
32      1.0000
```

همان طور که مشاهده می شود:

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

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

که این نشان می دهد  $w^h p_1$  به دلیل نرمال نبودن ( $(p_1^T p_1) \neq 1$ ) برابر با هدف نشده؛ اما در روش شبه معکوس علاوه بر تعامد با هدف هم هم سنگ شده ایم.

#### توضیح پوشه کدها

در پوشه اصلی این سوال (Q5\_07\_10)، فایل های کد مربوط به این سوال قرار گرفته اند. هم چنین فایل اجرای برخط کدها از طریق [پیوند گوگل کولب](#) در دسترس است.

## پاسخ سوال ۷.۱۱

برای حل این سوال از دستورات زیر که در پوشه مربوط به کدها نیز آمده است استفاده می کنیم:

```
1 % Autoassociative network for digit recognition using Hebbian learning
2 clc;
3 clear all;
4 close all;
5
6 %% load training images(5x7 px)
7 M=7; N=5;
8 P=zeros(M*N,10);
9 for n=0:9
10     s=num2str(n);
11     RGB=imread(s,'png');
12     m=n+1;
```

```

13     P(:,m)=reshape(rgb2gray(RGB),[M*N,1]); %column prototype vector
14 end
15 P = P/255*2-1; %normalizes data to be either -1 or 1
16
17 %% compute weight matrix using Hebb rule
18 W=zeros(M*N,M*N);
19 for n=1:10
20     W=W+P(:,n)*P(:,n)';
21 end
22
23 % simple Hebb rule did not make it recognize images well
24 % try again using pseudoinverse
25 T=P;
26 W1=T*pinv(P);
27
28 %% test out network with training images
29 for n = 10:-1:1
30     a=hardlims(W*P(:,n));
31     outputImg=reshape(a,[M,N]);
32     figure;
33     imshow(outputImg,'InitialMagnification','Fit')
34 end
35
36 %% test out network with noisy inputs
37 P_noisy = P + randi([-1,1],M*N,10);
38
39 for n = 10:-1:1
40     outputImg1=reshape(P_noisy(:,n),[M,N]);
41     figure;
42     subplot(1,2,1)
43     imshow(outputImg1,'InitialMagnification','Fit')
44     title('noisy image (Hebb)')
45
46     subplot(1,2,2)
47     a=hardlims(W*P_noisy(:,n));

```

```

48     outputImg2=reshape(a,[M,N]);
49     imshow(outputImg2,'InitialMagnification','Fit')
50     title('reconstructed image (Hebb)')
51 end
52
53 for n = 10:-1:1
54     outputImg1=reshape(P_noisy(:,n),[M,N]);
55     figure;
56     subplot(1,2,1)
57     imshow(outputImg1,'InitialMagnification','Fit')
58     title('noisy image (Pseudoinverse)')
59
60     subplot(1,2,2)
61     a=hardlims(W1*P_noisy(:,n));
62     outputImg2=reshape(a,[M,N]);
63     imshow(outputImg2,'InitialMagnification','Fit')
64     title('reconstructed image (Pseudoinverse)')
65 end
66
67 %% test out network with parts of image missing
68 P_partial=P;
69 P_partial(8:17,:)=1;
70
71 for n = 10:-1:1
72     outputImg1=reshape(P_partial(:,n),[M,N]);
73     figure;
74     subplot(1,2,1)
75     imshow(outputImg1,'InitialMagnification','Fit')
76     title('partial image (Hebb)')
77
78     subplot(1,2,2)
79     a=hardlims(W*P_partial(:,n));
80     outputImg2=reshape(a,[M,N]);
81     imshow(outputImg2,'InitialMagnification','Fit')
82     title('reconstructed image (Hebb)')

```

```

83 end
84
85 P_partial=P;
86 P_partial(8:17,:)=1;
87
88 for n = 10:-1:1
89     outputImg1=reshape(P_partial(:,n),[M,N]);
90     figure;
91     subplot(1,2,1)
92     imshow(outputImg1,'InitialMagnification','Fit')
93     title('partial image (Pseudoinverse)')
94
95     subplot(1,2,2)
96     a=hardlims(W1*P_partial(:,n));
97     outputImg2=reshape(a,[M,N]);
98     imshow(outputImg2,'InitialMagnification','Fit')
99     title('reconstructed image (Pseudoinverse)')
100 end

```

تابع hardlims هم که در کد اصلی بالا تعریف شده است به صورت زیر تعریف می گردد:

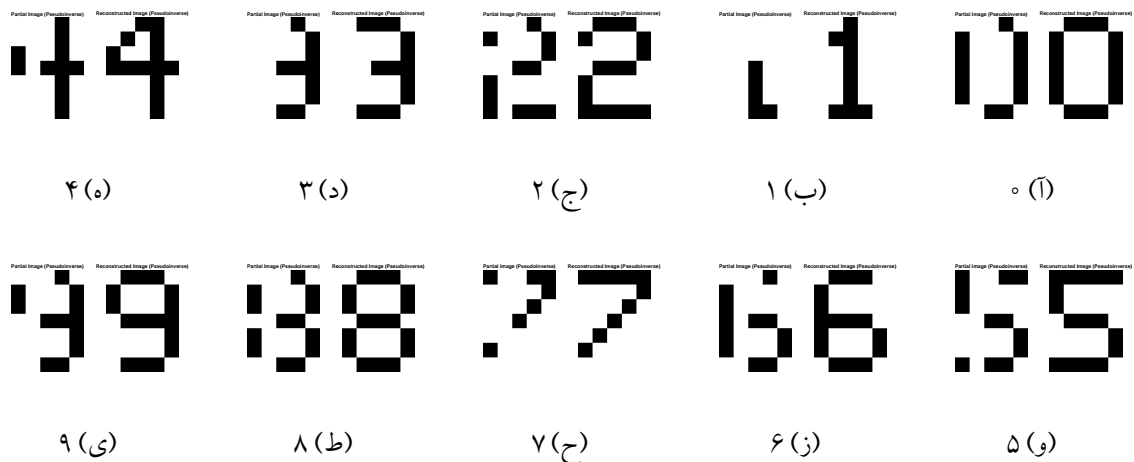
```

1 function [ a ] = hardlims( n )
2 %   hardlims Symmetric Hard Limit transfer function
3 %   Accepts column vector. Returns column vector.
4 %   behaves like unit step of amplitude 2 and vertical offset of -1
5 %   a = -1 for n < 0
6 %   a =  1 for n >= 0
7   dims = size(n);
8   a = zeros(dims);
9   for i = 1:dims(1)
10       for j = 1:dims(2)
11           if n(i,j) < 0
12               a(i,j) = -1;
13           else
14               a(i,j) = 1;
15           end

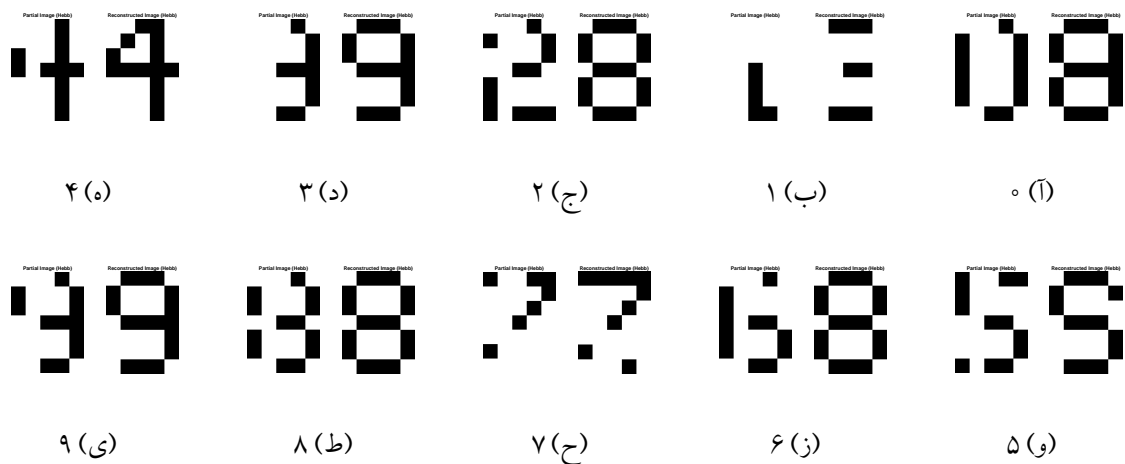
```

16 end  
17 end  
18 end

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



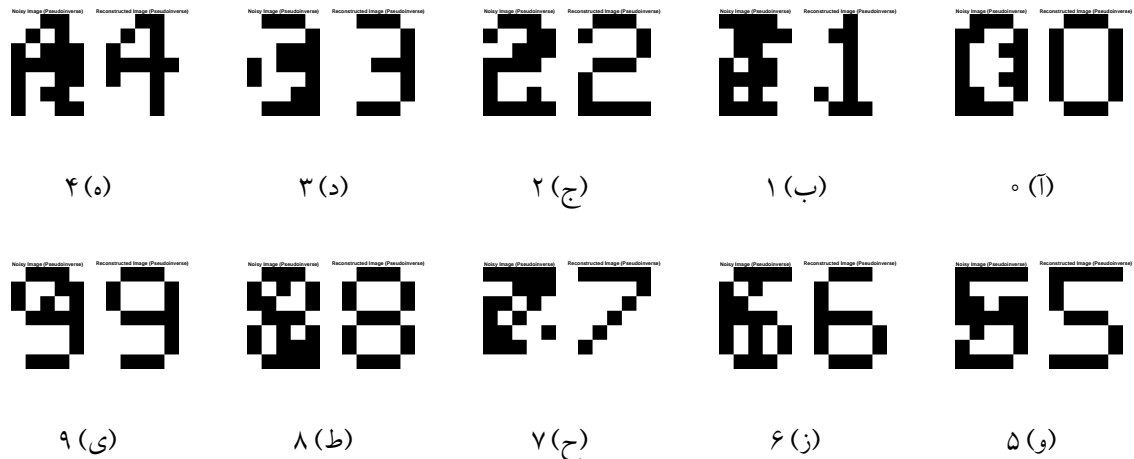
شکل ۷: بازسازی اعداد از روی قسمتی از آن‌ها (Pseudoinverse).



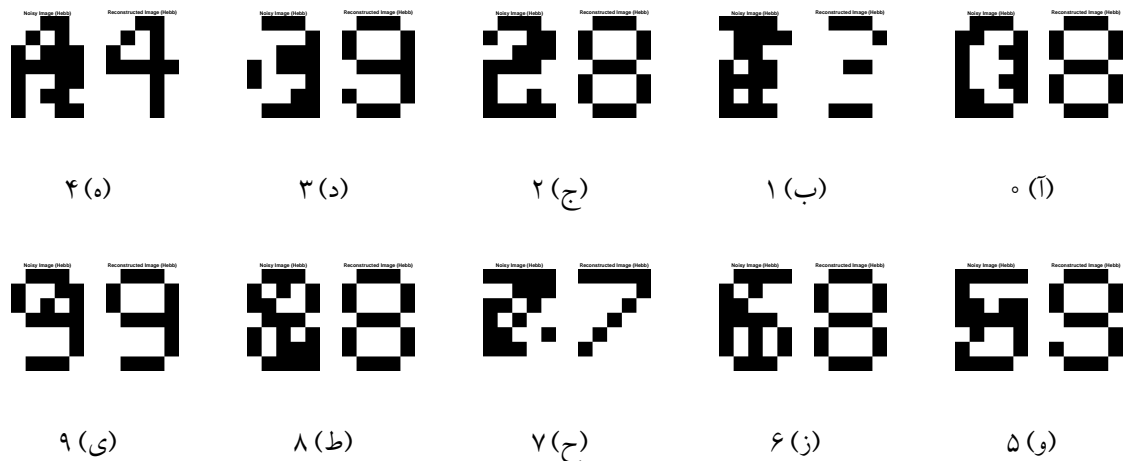
شکل ۸: بازسازی اعداد از روی قسمتی از آن‌ها (Hebb).

### توضیح پوشه کدها

در پوشه اصلی این سوال (Q6\_07\_11)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند.



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



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

## پاسخ سوال ۹.۱۰

### راه حل اول

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

```
1 clear
2 %plot the function contour lines
3 [X,Y] = meshgrid(-1 : .1 : 1);
4 Z = (X+Y).^4 - 12*X.*Y + X + Y + 1;
5 N = 10;
```



```

6 figure;
7 contour(X, Y, Z, N), title('Steepest Descent Trajectory for X_0 = [0 0]');
8 hold on;
9
10 x = [0 0]'; %Initialize x
11 G = zeros(2,1);
12 G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1;
13 G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
14 alfa = 0.01;
15 dx = [1e4 1e4]';
16 small = [1.0e-5,1.0e-5]';
17
18 %Find the stationary point
19 while (abs(dx(1)) >= small(1) | abs(dx(2))>= small(2))
20     plot(x(1), x(2), 'k.')
21     old = x;
22     x = x - alfa * G;
23     dx = x - old;
24     G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1; G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
25 end
26 x
27 plot(x(1), x(2), 'ro')
28 -----
29
30 x =
31     -0.6504
32     -0.6504

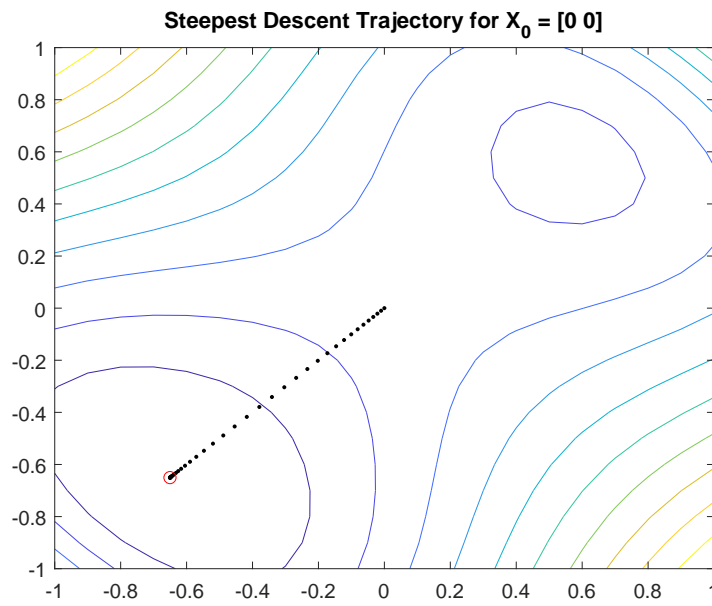
```

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

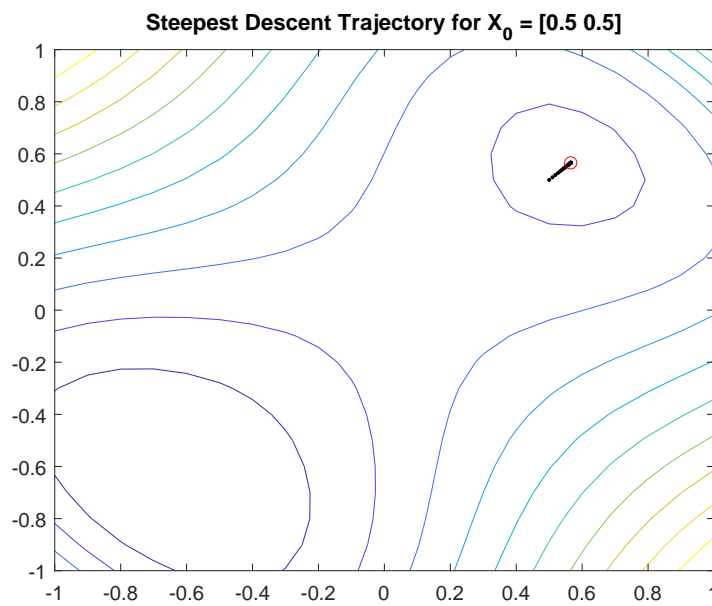
```

1 clc
2 clear all
3 close all
4

```



شکل ۱۱: پاسخ سوال ۹.۱۰.



شکل ۱۲: پاسخ سوال ۹.۱۰.

```

5 %plot the function contour lines
6 [X,Y] = meshgrid(-1 : .1 : 1);
7 Z = (X+Y).^4 - 12*X.*Y + X + Y + 1;
8 N = 10;
9 figure;
10 contour(X, Y, Z, N), title('Newton''s Method Trajectory for X_0 = [0 0]');

```

```

11 hold on;
12
13 G = zeros(2,1);
14 A = zeros(2);
15 x = [0.5 0.5]'; %Initialize x
16 % G = grad(x);
17 G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1;
18 G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
19 % G = jacobian(Z,x)
20 % G = [-12*Y + 4*(X + Y)^3 + 1; -12*X + 4*(X + Y)^3 + 1]
21 % A = hessian(x);
22 A(1,1) = 12*(x(1) + x(2))^2; A(1,2) = 12*((x(1) + x(2))^2 - 12);
23 A(2,1) = 12*((x(1) + x(2))^2 - 12); A(2,2) = 12*(x(1) + x(2))^2;
24 % A = jacobian(G,x);
25 % A = [12*(X + Y)^2 12*((X + Y)^2 - 1); 12*((X + Y)^2 - 1), 12*(X + Y)^2]
26 dx = [1e2 1e2]';
27 small = [1.0e-3,1.0e-3]';
28
29 %Find the stationary point
30 while (abs(dx(1)) >= small(1) | abs(dx(2)) >= small(2))
31     plot(x(1), x(2), 'k.')
32     old = x;
33     x = x - inv(A).*G;
34     dx = x - old;
35 %     G = grad(x);
36     G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1; G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
37     %G = [-12*Y + 4*(X + Y)^3 + 1; -12*X + 4*(X + Y)^3 + 1]
38 %     A = hessian(x);
39     A(1,1) = 12*(x(1) + x(2))^2; A(1,2) = 12*((x(1) + x(2))^2 - 1);
40     A(2,1) = 12*((x(1) + x(2))^2 - 1); A(2,2) = 12*(x(1) + x(2))^2;
41     %A = [12*(X + Y)^2 12*((X + Y)^2 - 1); 12*((X + Y)^2 - 1), 12*(X + Y)^2]
42 end
43 x
44 plot(x(1), x(2), 'ro')
45 -----

```

```

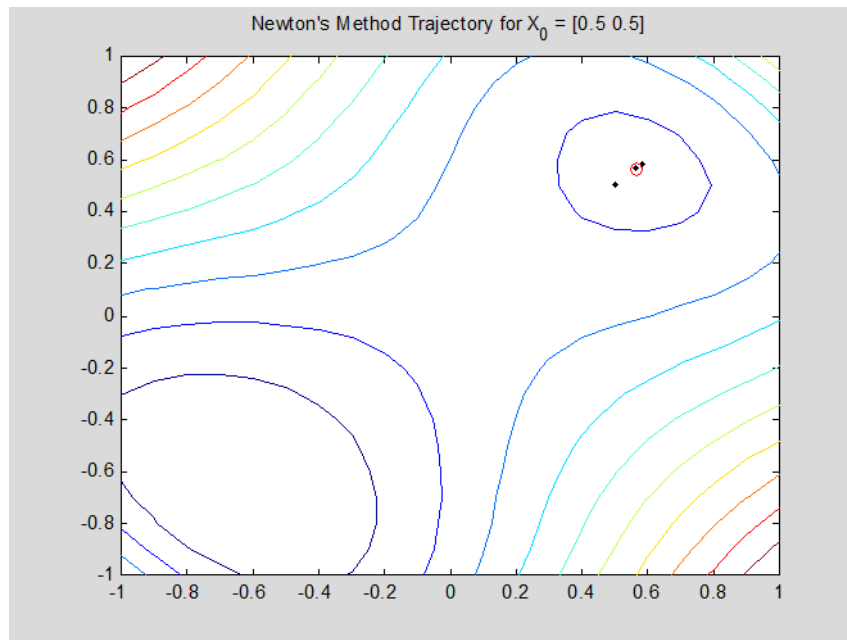
46
47 clc
48 clear all
49 close all
50
51 %plot the function contour lines
52 [X,Y] = meshgrid(-1 : .1 : 1);
53 Z = (X+Y).^4 - 12*X.*Y + X + Y + 1;
54 N = 10;
55 figure;
56 contour(X, Y, Z, N), title('Newton''s Method Trajectory for X_0 = [0 0]');
57 hold on;
58
59 G = zeros(2,1);
60 A = zeros(2);
61 x = [0.5 0.5]'; %Initialize x
62 % G = grad(x);
63 G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1;
64 G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
65 % G = jacobian(Z,x)
66 % G = [-12*Y + 4*(X + Y)^3 + 1; -12*X + 4*(X + Y)^3 + 1]
67 % A = hessian(x);
68 A(1,1) = 12*(x(1) + x(2))^2; A(1,2) = 12*((x(1) + x(2))^2 - 12);
69 A(2,1) = 12*((x(1) + x(2))^2 - 12); A(2,2) = 12*(x(1) + x(2))^2;
70 % A = jacobian(G,x);
71 % A = [12*(X + Y)^2 12*((X + Y)^2 - 1); 12*((X + Y)^2 - 1), 12*(X + Y)^2]
72 dx = [1e2 1e2]';
73 small = [1.0e-3,1.0e-3]';
74
75 %Find the stationary point
76 while (abs(dx(1)) >= small(1) | abs(dx(2))>= small(2))
77     plot(x(1), x(2), 'k.')
78     old = x;
79     x = x - inv(A).*G;
80     dx = x - old;

```

```

81 % G = grad(x);
82 G(1) = 4*(x(1)+x(2))^3 - 12*x(2) + 1; G(2) = 4*(x(1)+x(2))^3 - 12*x(1) + 1;
83 %G = [-12*Y + 4*(X + Y)^3 + 1; -12*X + 4*(X + Y)^3 + 1]
84 % A = hessian(x);
85 A(1,1) = 12*(x(1) + x(2))^2; A(1,2) = 12*((x(1) + x(2))^2 - 1);
86 A(2,1) = 12*((x(1) + x(2))^2 - 1); A(2,2) = 12*(x(1) + x(2))^2;
87 %A = [12*(X + Y)^2 12*((X + Y)^2 - 1); 12*((X + Y)^2 - 1), 12*(X + Y)^2]
88 end
89 x
90 plot(x(1), x(2), 'ro')

```



شکل ۱۳: پاسخ سوال ۹.۱۰.

راه حل دوم

برای تابع

$$F(X) = (x_1 + x_2)^4 - 12x_1x_2 + x_1 + x_2 - 1 \quad (۷)$$

گرایان را تشکیل می‌دهیم:

$$\nabla F(X) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(x) \\ \frac{\partial}{\partial x_2} F(x) \end{bmatrix} = \begin{bmatrix} 4(x_1 + x_2)^3 - 12x_2 + 1 \\ 4(x_1 + x_2)^3 - 12x_1 + 1 \end{bmatrix} \quad (8)$$

بنابراین نقاط ایستا به صورت زیر به دست می‌آید:

$$x^1 = \begin{bmatrix} -0.6504 \\ -0.6504 \end{bmatrix}, x^2 = \begin{bmatrix} 0.085 \\ 0.085 \end{bmatrix}, x^3 = \begin{bmatrix} 0.5655 \\ 0.5655 \end{bmatrix} \quad (9)$$

حال ماتریس هسیان را تشکیل می‌دهیم:

$$\nabla^2 F(X) = \begin{bmatrix} 12(x_1 + x_2)^2 & 12(x_1 + x_2)^2 - 12 \\ 12(x_1 + x_2)^2 - 12 & 12(x_1 + x_2)^2 \end{bmatrix} \quad (10)$$

ادامه حل این راه حل 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 نوشته شده و نتایج دیگر صرف نظر می‌شود. همگی از طریق پوشه کدها در دسترس هستند.

```

1 # Steepest Descent
2 #
3 # Requires an initial point x - matrix nx1
4 # Requires a function parameters to the gradient and hessian matrices of the
   function
5 # Requires a maximum number of iterations, used to stop if algorithm diverges
6 # Requires a tolerance for exiting condition
7 # Requires a mask boolean for whether or not to print out values inbetween
   iterations
8
9 steepest_descent = function(init_point, gradient, hessian, maxIter, tol, mask)
10 {
11     x = init_point
12

```

```

13  for(i in 1:maxIter) {
14
15      if(!mask) { # if the user does not want print out statements
16          print(sprintf("      Iteration: %d", i))
17          print("x value:")
18          print(x)
19      }
20
21      g = gradient(x) # find gradient at x
22
23      if(!mask) { # if the user does not want print out statements
24          print("gradient value:")
25          print(g)
26      }
27
28      # calculate the frobenius norm of the gradient vector evaluated at x
29      # remember, stationary points occur when gradient is zero, so we take
30      # the square root of the sum of squares of the gradient values and
31      # compare to the tolerance
32      if(norm(g, "F") < tol) { # convert gradient into single value
33          # through frobenius norm
34          print("Stationary Point Found at point:")
35          print(x)
36          print(sprintf("Iterations taken: %d", i))
37          return(x)
38      }
39
40      # creating the search direction vectors
41      p = -g
42
43      # Here we have a dynamic learning rate that is minimized against
44      # the line of the next iteration
45      alpha = -(t(g)%*%p) / (t(p)%*%hessian(x)%*%p)
46
47      x = x - alpha[1]*g # find next x

```

```
48
49
50 }
51
52 print("MAXIMUM ITERATIONS REACHED")
53 print("Current point is: ")
54 print(x)
55 print("Error: ")
56 print(abs(norm(g, "F")-tol)) # computing the error
57 return(x)
58 }
59
60
61 # example function:
62 # (x+y)^4-12xy+x+y-1
63 # Has three stationary points
64 # 1) [-0.65, -0.65]
65 # 2) [0.5654, 0.564]
66 # 3) [0.0849, 0.0849]
67
68 gradient = function(X) {
69   x = X[1,1]
70   y = X[2,1]
71   g1 = 4*(x+y)^3-12*y+1
72   g2 = 4*(x+y)^3-12*x+1
73   matrix(c(g1, g2), ncol=1)
74 }
75
76 hessian = function(X) {
77   x = X[1,1]
78   y = X[2,1]
79   fxx = 12*(x + y)^2
80   fxy = 12*(x + y)^2 - 12
81   matrix(c(fxx, fxy, fxy, fxx), ncol=2)
82 }
```



```

83
84 # initial points
85 p1 = matrix(c(-1, -1), ncol=1)
86
87 # Function call
88 min = steepest_descent(p1,gradient, hessian, 1000, 1e-10, 1)
89 -----
90
91 [1] "Stationary Point Found at point:"
92      [,1]
93 [1,] -0.6504198
94 [2,] -0.6504198
95 [1] "Iterations taken: 7"

```

```

1
2 # Newton's Method
3 #
4 # Requires an initial point x - matrix nx1
5 # Requires a function parameters to the gradient and hessian matrices of the
   function
6 # Requires a maximum number of iterations, used to stop if algorithm diverges
7 # Requires a tolerance for exiting condition
8 # Requires a mask boolean for whether or not to print out values inbetween
   iterations
9
10 newtons_method = function(init_point, gradient, hessian, maxIter, tol, mask) {
11
12
13   x = init_point
14
15
16   for(i in 1:maxIter) {
17
18     if(!mask) { # if the user does not want print out statements
19       print(sprintf("      Iteration: %d", i))

```

```

20     print("x value:")
21     print(x)
22 }
23
24 A = hessian(x)
25 g = gradient(x)
26 x = x - solve(A)%*%g # solve means inverse, A^-1
27
28 if(!mask) { # if the user does not want print out statements
29     print("gradient value:")
30     print(g)
31 }
32
33 # calculate the frobenius norm of the gradient vector evaluated at x
34 # remember, stationary points occur when gradient is zero, so we take
35 # the square root of the sum of squares of the gradient values and
36 # compare to the tolerance
37 if(norm(g, "F") < tol) { # convert gradient into single value
38     # through frobenius norm
39     print("Stationary Point Found at point:")
40     print(x)
41     print(sprintf("Iterations taken: %d", i))
42     return(x)
43 }
44 }
45
46 print("MAXIMUM ITERATIONS REACHED")
47 print("Current point is: ")
48 print(x)
49 print("Error: ")
50 print(abs(norm(g, "F")-tol)) # computing the error
51 return(x)
52 }
53
54

```

```

55 # example function:
56 # (x+y)^4-12xy+x+y-1
57 # Has three stationary points
58 # 1) [-0.65, -0.65]
59 # 2) [0.5654, 0.564]
60 # 3) [0.0849, 0.0849]
61
62 gradient = function(X) {
63   x = X[1,1]
64   y = X[2,1]
65   g1 = 4*(x+y)^3-12*y+1
66   g2 = 4*(x+y)^3-12*x+1
67   matrix(c(g1, g2), ncol=1)
68 }
69
70 hessian = function(X) {
71   x = X[1,1]
72   y = X[2,1]
73   fxx = 12*(x + y)^2
74   fxy = 12*(x + y)^2 - 12
75   matrix(c(fxx, fxy, fxy, fxx), ncol=2)
76 }
77
78 # initial points
79 p1 = matrix(c(-1, -1), ncol=1)
80
81 # Function call
82 min = newtons_method(p1,gradient, hessian, 1000, 1e-10, 1)
83 -----
84
85 [1] "Stationary Point Found at point:"
86      [,1]
87 [1,] -0.6504198
88 [2,] -0.6504198
89 [1] "Iterations taken: 7"

```

```

1
2 # Conjugate Gradient method
3 #
4 # Requires an initial point x - matrix nx1
5 # Requires a function parameters to the gradient and hessian matrices of the
   function
6 # Requires a maximum number of iterations, used to stop if algorithm diverges
7 # Requires a tolerance for exiting condition
8 # Requires a mask boolean for whether or not to print out values inbetween
   iterations
9 # Requires a value ranging from 1,2,3 for the type of beta method
10
11 conjugate_gradient = function(init_point, gradient, hessian, maxIter, tol, mask,
   betaMethod) {
12
13     x = init_point
14
15     for(i in 1:maxIter) {
16
17         if(!mask) { # if the user does not want print out statements
18             print(sprintf("      Iteration: %d", i))
19             print("x value:")
20             print(x)
21         }
22
23         g = gradient(x) # find gradient at x
24
25         if(!mask) { # if the user does not want print out statements
26             print("gradient value:")
27             print(g)
28         }
29
30         # calculate the frobenius norm of the gradient vector evaluated at x
31         # remember, stationary points occur when gradient is zero, so we take
32         # the square root of the sum of squares of the gradient values and

```

```

33     # compare to the tolerance
34     if(norm(g, "F") < tol) { # convert gradient into single value
35         # through frobenius norm
36         print("Stationary Point Found at point:")
37         print(x)
38         print(sprintf("Iterations taken: %d", i))
39         return(x)
40     }
41
42     # creating the search direction vectors
43     if(i==1) { # first search direction
44         p = -g
45     }
46     else { # use beta
47         if(betaMethod == 1) { # Common choice
48             beta = (t(g-g0)%*%g)/(t(g0)%*%p)
49         }
50         else if (betaMethod == 2) { # hestens and stiefel
51             beta = (t(g)%*%g)/(t(g0)%*%g)
52         }
53         else { # Fletcher and Reeves
54             beta = (t(g-g0)%*%g)/(t(g0)%*%g0)
55         }
56
57         if(!mask) { # if the user does not want print out statements
58             print("beta value:")
59             print(beta)
60         }
61
62         p = -g+beta[1]*p
63     }
64
65
66     alpha = -(t(g)%*%p) / (t(p)%*%hessian(x)%*%p) # learning rate
67     x = x - alpha[1]*g # find next x

```

```

68
69     g0 = g # update previous gradient to current
70 }
71
72 print("MAXIMUM ITERATIONS REACHED")
73 print("Current point is: ")
74 print(x)
75 print("Error: ")
76 print(abs(norm(g, "F")-tol)) # computing the error
77 return(x)
78 }
79
80 # example function:
81 # (x+y)^4-12xy+x+y-1
82 # Has three stationary points
83 # 1) [-0.65, -0.65]
84 # 2) [0.5654, 0.564]
85 # 3) [0.0849, 0.0849]
86
87 gradient = function(X) {
88     x = X[1,1]
89     y = X[2,1]
90     g1 = 4*(x+y)^3-12*y+1
91     g2 = 4*(x+y)^3-12*x+1
92     matrix(c(g1, g2), ncol=1)
93 }
94
95 hessian = function(X) {
96     x = X[1,1]
97     y = X[2,1]
98     fxx = 12*(x + y)^2
99     fxy = 12*(x + y)^2 - 12
100     matrix(c(fxx, fxy, fxy, fxx), ncol=2)
101 }
102

```

```

103 # initial points
104 p1 = matrix(c(-1, -1), ncol=1)
105
106 # Function call
107 min = conjugate_gradient(p1,gradient, hessian, 1000, 1e-10, 1,1)
108 -----
109
110 [1] "Stationary Point Found at point:"
111      [,1]
112 [1,] -0.6504198
113 [2,] -0.6504198
114 [1] "Iterations taken: 29"

```

### توضیح پوشه کدها

در پوشه اصلی این سوال (Q7\_09\_10)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند. هم‌چنین فایل اجرای برخی 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 لایه تصمیم‌گیری اولیه برای ایجاد مرز برای دسته‌ها، و یک لایه AND برای تثبیت آن لایه‌ها، و یک لایه OR برای ترکیب دسته‌ها خواهد بود.

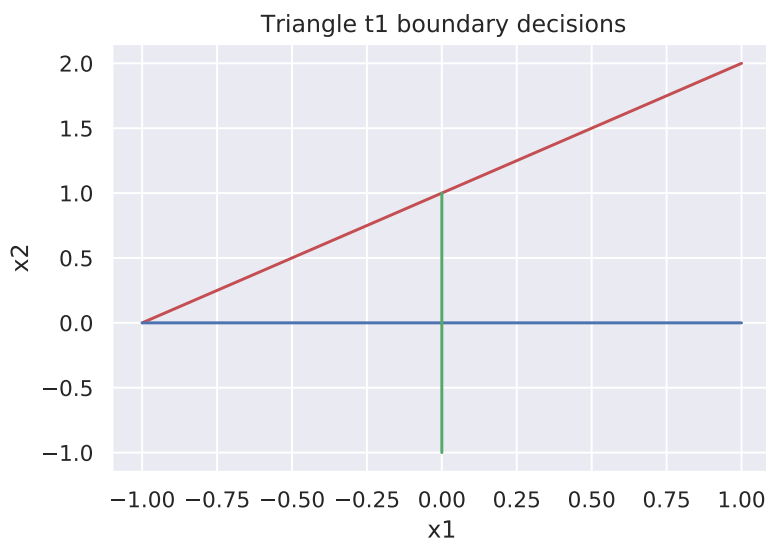
i

در این قسمت دو مثلث با نقاط انتهایی  $(0, 1), (0, 0), (-1, 0)$  و  $(0, -1), (-1, -1), (-1, 0)$  مشاهده می‌شود. مثلث نخست که نام  $t_1$  را برای آن برمی‌گزینیم، می‌تواند با خطوط  $y_1 = x_1 + 1, y_2 = 0, x_3 = 0$  تعیین شود. این ناحیه مثلثی شکل با بهره‌گیری از دستورات زیر در شکل ۱۴ نشان داده شده است.

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Plot 3 lines: 1-  $y_1 = x_1 + 1$ , 2-  $y_2 = 0$ , 3-  $x_3 = 0$ 
5 x1 = np.linspace(-1, 1, 100)
6 y1 = x1 + 1
7 y2 = np.zeros(100)
8 x3 = np.zeros(100)
9
10 plt.plot(x1, y1, 'r', x1, y2, 'b', x3, x1, 'g')
11 plt.xlabel('x1')
12 plt.ylabel('x2')
13 plt.title('Triangle t1 boundary decisions')
14 plt.savefig('E111_1.pdf')
15 plt.show()

```



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

این سه معادله 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:

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

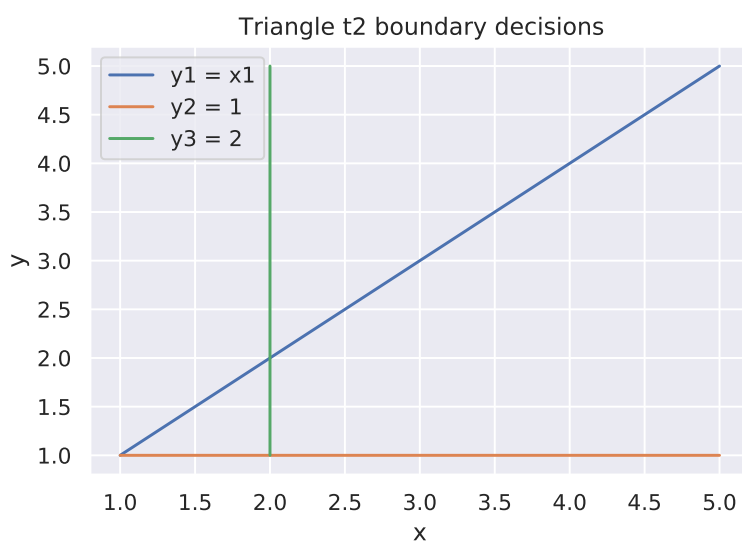
از طرف دیگر مثلث دوم با نام  $t_2$  با خطوط  $x_3 = 2$ ,  $y_2 = 1$ ,  $y_1 = x_1$  ساخته می‌شود. این ناحیه با بهره‌گیری از دستورات زیر در شکل ۱۵ نشان داده شده است.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
5 x1 = np.linspace(-1, 1, 100)
6 y1 = x1 + 1
7 y2 = np.zeros(100)
8 x3 = np.zeros(100)
9
10 plt.plot(x1, y1, 'r', x1, y2, 'b', x3, x1, 'g')
11 plt.xlabel('x1')
12 plt.ylabel('x2')
13 plt.title('Triangle t1 boundary decisions')
14 plt.savefig('E111_1.pdf')
15 plt.show()
```

این سه معادله 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  $2 + p_1 -$ .



شکل ۱۵: مرزهای تصمیم‌گیری قسمت اول سوال ۱۱-۱ (t<sub>۲</sub>).

- 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:

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

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

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

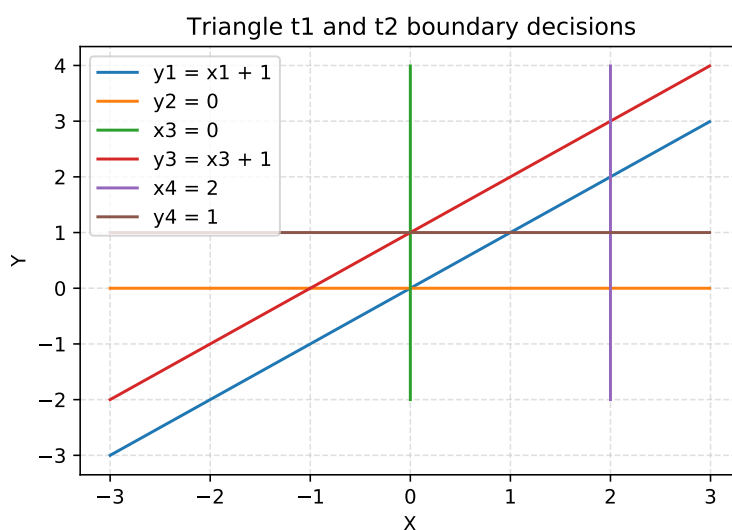
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Create the vectors X and Y
5 x = np.arange(-3, 3, 0.01)
6 y1 = x
7 y2 = np.zeros(600)
8 x3 = np.zeros(600)
```

```

9 y3 = x + 1
10 x4 = 2*np.ones(600)
11 y4 = 1*np.ones(600)
12
13 # Create the plot
14 plt.plot(x, y1, label='y1 = x1 + 1')
15 plt.plot(x, y2, label='y2 = 0')
16 plt.plot(x3, y3, label='x3 = 0')
17 plt.plot(x, y3, label='y3 = x3 + 1')
18 plt.plot(x4, y3, label='x4 = 2')
19 plt.plot(x, y4, label='y4 = 1')
20
21 # Add a title
22 plt.title('Triangle t1 and t2 boundary decisions')
23
24 # Add X and y Label
25 plt.xlabel('X')
26 plt.ylabel('Y')
27
28 # Add a grid
29 plt.grid(alpha=.4,linestyle='--')
30
31 # Add a Legend
32 plt.legend()
33
34 # Save the plot
35 plt.savefig('E1113.pdf')
36
37 # Show the plot
38 plt.show()

```

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



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

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

$$w_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (14)$$

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

$$b_2 = \begin{bmatrix} -2 & -2 \end{bmatrix} \quad (15)$$

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 دسته‌ی 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 انجام شود؛ در نتیجه داریم:

$$w_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}, b_3 = [1] \quad (16)$$

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

$$w^T = \begin{bmatrix} 1 & -1 & 0 & 1 & -1 & 0 \\ -1 & 0 & 1 & -1 & 0 & 1 \end{bmatrix}, b^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 & -1 \end{bmatrix} \quad (17)$$

در ادامه، در لایه پنهان برای عملیات AND داریم:

$$w_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}, b_2 = \begin{bmatrix} -2 & -2 \end{bmatrix} \quad (18)$$

در نهایت، در لایه انتهایی برای عملیات OR داریم:

$$w_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}, b_3 = [1] \quad (19)$$

از نمودار می‌فهمیم که نقطه  $(-0.5, 0.25)$  باید در مثلث اول، و در نتیجه دسته یک، و نقطه  $(0.5, 1)$  باید خارج از هر دو مثلث یعنی دسته دو طبقه‌بندی شود. حال نقاط را با استفاده از دستورات زیر تست می‌کنیم. نتایج ذیل هربخش آورده شده است.

```
1 import numpy as np
2
3 def hardlims(x):
4     result = np.zeros((x.shape[0], x.shape[1]))
5     for i in range(x.shape[0]):
6         for j in range(x.shape[1]):
7             if x[i,j] < 0:
8                 result[i,j] = -1
9             else:
10                 result[i,j] = 1
11     return result
12
13 w1 = np.array([[1, -1], [-1, 0], [0, 1], [1, -1], [-1, 0], [0, 1]])
14 b1 = np.array([[1], [0], [0], [0], [2], [-1]])
15 w2 = np.array([[1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1]])
16 b2 = np.array([[ -2], [ -2]])
```

```
17 w3 = np.array([[1, 1]])
18 b3 = 1
19
20 p1 = np.array([[-0.5], [0.25]])
21 p2 = np.array([[0.5], [1]])
22
23 # Test
24 a11 = hardlims(np.dot(w1, p1) + b1)
25 print(a11)
26
27 [[ 1.]
28  [ 1.]
29  [ 1.]
30 [-1.]
31  [ 1.]
32 [-1.]]
33
34 a21 = hardlims(np.dot(w1, p2) + b1)
35 print(a21)
36
37 [[ 1.]
38 [-1.]
39  [ 1.]
40 [-1.]
41  [ 1.]
42  [ 1.]]
43
44 w2
45
46 array([[1, 1, 1, 0, 0, 0],
47        [0, 0, 0, 1, 1, 1]])
48
49 a12 = hardlims(np.dot(w2, a11) + b2)
50 print(a12)
51
```

```

52 [[ 1.]
53    [-1.]]
54
55 a22 = hardlims(np.dot(w2, a21) + b2)
56 print(a22)
57
58 [[-1.]
59    [-1.]]
60
61 a13 = hardlims(np.dot(w3, a12) + b3)
62 print(a13)
63
64 [[1.]]
65
66 a23 = hardlims(np.dot(w3, a22) + b3)
67 print(a23)
68
69 [[-1.]]

```

ii

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

```

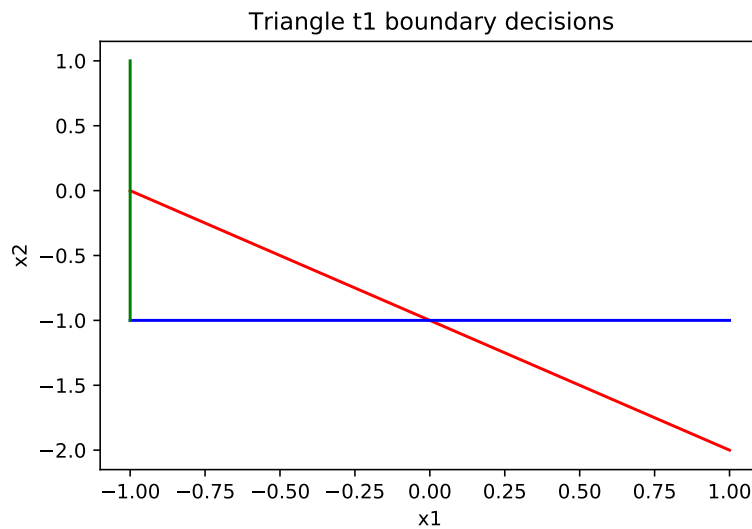
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
5 x1 = np.linspace(-1, 1, 100)
6 y1 = -x1 - 1
7 y2 = -1*np.ones(100)
8 x3 = -1*np.ones(100)
9

```

```

10 plt.plot(x1, y1, 'r', x1, y2, 'b', x3, x1, 'g')
11 plt.xlabel('x1')
12 plt.ylabel('x2')
13 plt.title('Triangle t1 boundary decisions')
14 plt.savefig('E1121.pdf')
15 plt.show()

```



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

این سه معادله خط به اوزان زیر می‌انجامد:

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

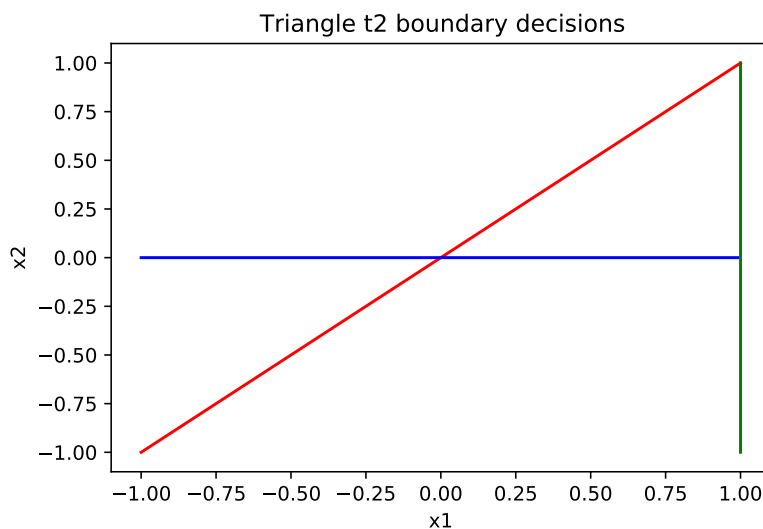
بنابراین، ماتریس اوزان و بایاس برای طبقه‌بندی مثلث به صورت زیر خواهد بود:

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



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.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Plot 3 lines: 1-  $y_1 = x_1 + 1$ , 2-  $y_2 = 0$ , 3-  $x_3 = 0$ 
5 x1 = np.linspace(-1, 1, 100)
6 y1 = x1
7 y2 = np.zeros(100)
8 x2 = 1*np.ones(100)
9
10 plt.plot(x1, y1, 'r', x1, y2, 'b', x2, x1, 'g')
11 plt.xlabel('x1')
12 plt.ylabel('x2')
13 plt.title('Triangle t2 boundary decisions')
14 plt.savefig('E1122.pdf')
15 plt.show()
```



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

این سه معادله 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  $x_2 = 1$  می‌خواهیم هر مقداری در سمت چپ خط عمودی یک شود؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم  $-p_1 + 1$ .

• 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:

$$w^T = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, b^T = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \quad (21)$$

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

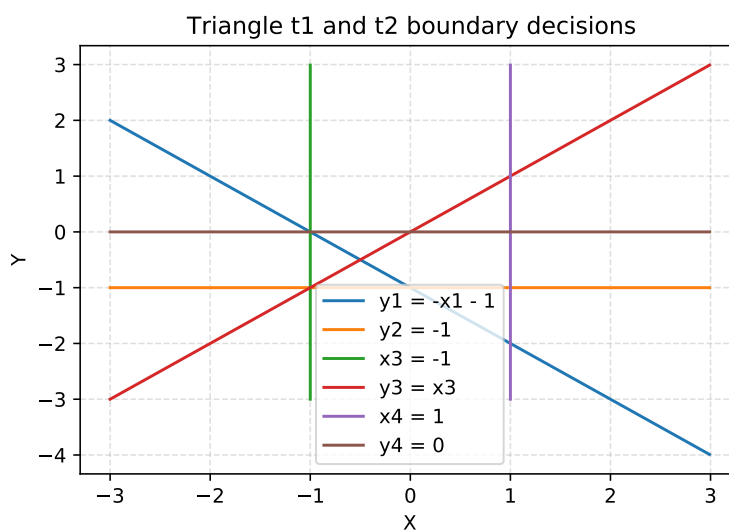
$$w^T = \begin{bmatrix} -1 & 1 & 0 & 1 & -1 & 0 \\ -1 & 0 & 1 & -1 & 0 & 1 \end{bmatrix}, b^T = \begin{bmatrix} -1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \quad (22)$$

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Create the vectors X and Y
5 x = np.arange(-3, 3, 0.01)
6 y1 = -x-1
7 y2 = -1*np.ones(600)
8 x3 = -1*np.ones(600)
9 y3 = x
10 x4 = 1*np.ones(600)
11 y4 = np.zeros(600)
12
13 # Create the plot
14 plt.plot(x, y1, label='y1 = -x1 - 1')
15 plt.plot(x, y2, label='y2 = -1')
16 plt.plot(x3, y3, label='x3 = -1')
17 plt.plot(x, y3, label='y3 = x3')
18 plt.plot(x4, y3, label='x4 = 1')
```

```

19 plt.plot(x, y4, label='y4 = 0')
20
21 # Add a title
22 plt.title('Triangle t1 and t2 boundary decisions')
23
24 # Add X and y Label
25 plt.xlabel('X')
26 plt.ylabel('Y')
27
28 # Add a grid
29 plt.grid(alpha=.4,linestyle='--')
30
31 # Add a Legend
32 plt.legend()
33
34 # Save the plot
35 plt.savefig('E1123.pdf')
36
37 # Show the plot
38 plt.show()

```



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

در ادامه به ماتریس وزنی نیاز داریم که مرزهای تصمیم‌گیری را به گونه‌ای که به فرم مثلثی دربیانند ترکیب

کند (مرزهای از حالت بی سر و ته بودن درآیند). برای مثلث اول، سه ستون اول ماتریس وزن اول مرزهای را می‌سازند و در نتیجه باید این سه قسمت در نظر گرفته شوند و بقیه نادیده گرفته شوند. این کار می‌تواند با یک سطر به صورت  $[1, 1, 1, 0, 0, 0]$  انجام گیرد. همین کار را برای مثلث دوم تکرار می‌کنیم. بنابراین، این کار را با استفاده از ماتریس وزن لایه‌ی دوم انجام می‌دهیم:

$$w_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (23)$$

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

$$b_2 = \begin{bmatrix} -2 & -2 \end{bmatrix} \quad (24)$$

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 دسته‌ی 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 انجام شود؛ در نتیجه داریم:

$$w_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}, b_3 = [1] \quad (25)$$

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

$$w^T = \begin{bmatrix} -1 & 1 & 0 & 1 & -1 & 0 \\ -1 & 0 & 1 & -1 & 0 & 1 \end{bmatrix}, b^T = \begin{bmatrix} -1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \quad (26)$$

در ادامه، در لایه‌ی پنهان برای عملیات AND داریم:

$$w_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}, b_2 = \begin{bmatrix} -2 & -2 \end{bmatrix} \quad (27)$$

در نهایت، در لایه انتهایی برای عملیات OR داریم:

$$w_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}, b_3 = [1] \quad (28)$$

از نمودار می‌فهمیم که نقطه  $(-0.5, -0.5)$  باید در مثلث اول، و در نتیجه دسته یک، و نقطه  $(0.5, 1)$  باید خارج از هر دو مثلث یعنی دسته 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.

```

1 import numpy as np
2
3 def hardlims(x):
4     result = np.zeros((x.shape[0], x.shape[1]))
5     for i in range(x.shape[0]):
6         for j in range(x.shape[1]):
7             if x[i,j] < 0:
8                 result[i,j] = -1
9             else:
10                 result[i,j] = 1
11     return result
12
13 w1 = np.array([[ -1, -1], [1, 0], [0, 1], [1, -1], [-1, 0], [0, 1]])
14 b1 = np.array([[ -1], [1], [1], [0], [1], [0]])
15 w2 = np.array([[1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1]])
16 b2 = np.array([[ -2], [-2]])
17 w3 = np.array([[1, 1]])
18 b3 = 1
19
20 p1 = np.array([[ -0.5], [-0.5]])
21 p2 = np.array([[0.5], [1]])
22
23 # Test
24 a11 = hardlims(np.dot(w1, p1) + b1)
25 print(a11)
26

```

```
27 [[ 1.]
28  [ 1.]
29  [ 1.]
30  [ 1.]
31  [ 1.]
32 [-1.]]
33
34 a21 = hardlims(np.dot(w1, p2) + b1)
35 print(a21)
36
37 [[-1.]
38  [ 1.]
39  [ 1.]
40 [-1.]
41  [ 1.]
42  [ 1.]]
43
44 w2
45
46 array([[1, 1, 1, 0, 0, 0],
47        [0, 0, 0, 1, 1, 1]])
48
49 a12 = hardlims(np.dot(w2, a11) + b2)
50 print(a12)
51
52 [[ 1.]
53 [-1.]]
54
55 a22 = hardlims(np.dot(w2, a21) + b2)
56 print(a22)
57
58 [[-1.]
59 [-1.]]
60
61 a13 = hardlims(np.dot(w3, a12) + b3)
```

```

62 print(a13)
63
64 [[1.]]
65
66 a23 = hardlims(np.dot(w3, a22) + b3)
67 print(a23)
68
69 [[-1.]]

```

iii

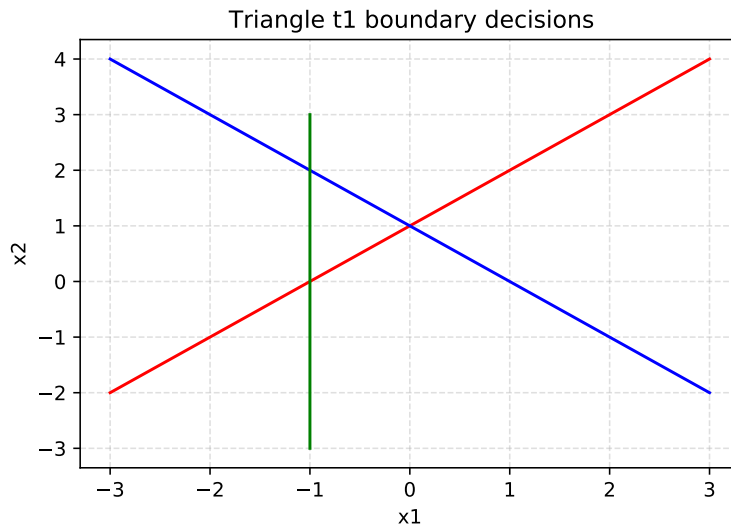
در این قسمت سه شکل (شامل دو شکل مثلثی و یک شکل مربعی) با نقاط انتهایی  $(-1, 0), (-1, 2), (0, 1)$ ،  $(-1, -2), (1, -2), (1, -1), (0, 2), (0, -1), (-1, -1), (0, 0), (1, 1), (0, 0)$  مشاهده می‌شود. شکل نخست که نام  $t_1$  را برای آن برمی‌گزینیم، می‌تواند با خطوط  $x_3 = -1, y_2 = -x_2 + 1, y_1 = x_1 + 1$  تعیین شود. این ناحیه مثلثی شکل با بهره‌گیری از دستورات زیر در شکل ۲۰ نشان داده شده است.

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
5 x1 = np.linspace(-3, 3, 600)
6 y1 = x1 + 1
7 y2 = -x1 + 1
8 x3 = -1*np.ones(600)
9
10 plt.plot(x1, y1, 'r', x1, y2, 'b', x3, x1, 'g')
11 plt.xlabel('x1')
12 plt.ylabel('x2')
13 # Add a grid
14 plt.grid(alpha=.4,linestyle='--')
15 plt.title('Triangle t1 boundary decisions')
16 plt.savefig('E1131.pdf')
17 plt.show()

```

این سه معادله خط به اوزان زیر می‌انجامد:



شکل ۲۰: مرزهای تصمیم‌گیری قسمت سوم سوال ۱۱-۱ ( $t_1$ ).

- برای خط  $y_1 = x_1 + 1$ ، می‌خواهیم تمام نقاط بالای خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم:  $-p_1 + p_2 - 1$ .
- برای خط  $y_2 = -x_2 + 1$ ، می‌خواهیم تمام نقاط زیر خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلث در طبقه‌بندی یک گردد، بنابراین داریم:  $-p_1 - p_2 + 1$ .
- برای خط  $x_3 = -1$  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:

$$w^T = \begin{bmatrix} -1 & -1 & 1 \\ 1 & -1 & 0 \end{bmatrix}, b^T = \begin{bmatrix} -1 & 1 & 1 \end{bmatrix} \quad (29)$$

از طرف دیگر مثلث دوم با نام  $t_2$  با خطوط  $y_1 = x_1 - 1$ ،  $y_2 = -x_2 - 1$ ،  $y_3 = -2$  ساخته می‌شود. این ناحیه با بهره‌گیری از دستورات زیر در شکل ۲۲ نشان داده شده است.

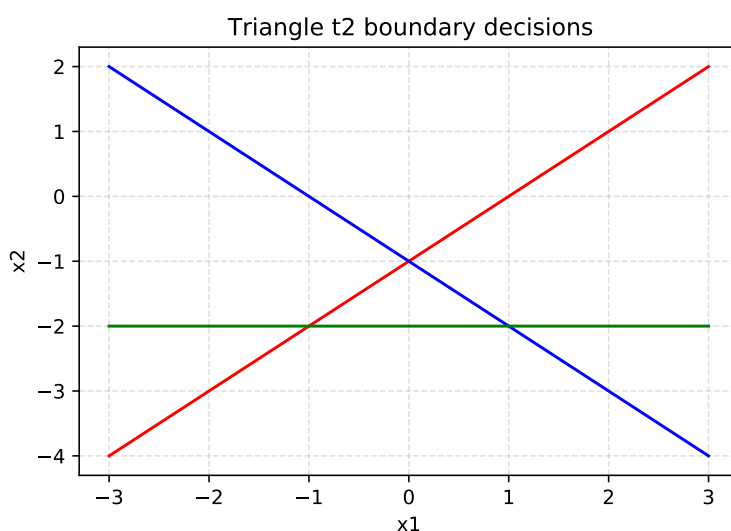
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Plot 3 lines: 1-  $y_1 = x_1 - 1$ , 2-  $y_2 = -x_2 - 1$ , 3-  $x_3 = 0$ 
5 x1 = np.linspace(-3, 3, 600)
```



```

6 y1 = x1 - 1
7 y2 = -x1 - 1
8 y3 = -2*np.ones(600)
9
10 plt.plot(x1, y1, 'r', x1, y2, 'b', x1, y3, 'g')
11 plt.xlabel('x1')
12 plt.ylabel('x2')
13 # Add a grid
14 plt.grid(alpha=.4,linestyle='--')
15 plt.title('Triangle t2 boundary decisions')
16 plt.savefig('E1132.pdf')
17 plt.show()

```



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

این سه معادله 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  $y_2 = -x_2 - 1$ ، 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:

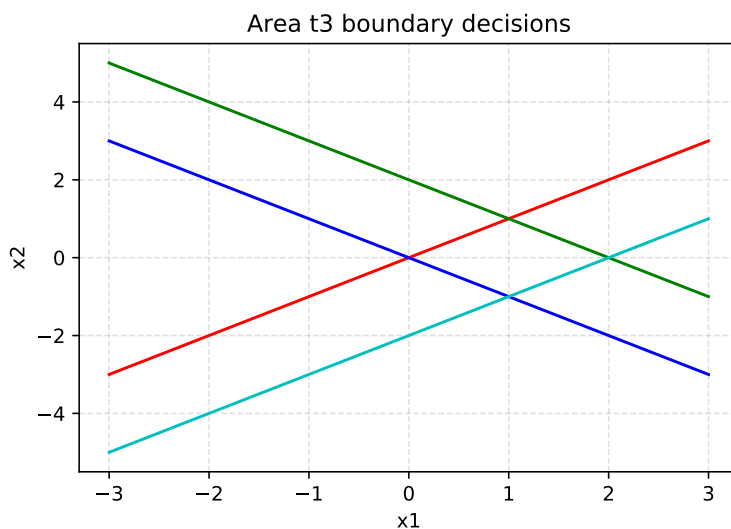
$$w^T = \begin{bmatrix} 1 & -1 & 0 \\ -1 & -1 & 1 \end{bmatrix}, b^T = \begin{bmatrix} -1 & -1 & 2 \end{bmatrix} \quad (30)$$

ناحیه 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.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Plot 3 lines: 1- y1 = x1 + 1, 2- y2 = 0, 3- x3 = 0
5 x1 = np.linspace(-3, 3, 600)
6 y1 = x1
7 y2 = -x1
8 y3 = -x1 + 2
9 y4 = x1 - 2
10
11 plt.plot(x1, y1, 'r', x1, y2, 'b', x1, y3, 'g', x1, y4, 'c')
12 plt.xlabel('x1')
13 plt.ylabel('x2')
14 # Add a grid
15 plt.grid(alpha=.4, linestyle='--')
16 plt.title('Area t3 boundary decisions')
17 plt.savefig('E1133.pdf')
18 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 ۱۱-۱ (۳۳).  
 شکل ۲۲: مرزهای تصمیم‌گیری قسمت سوم سوال ۱۱-۱ (۳۳).

مثلاً در طبقه‌بندی یک گردد، بنابراین داریم:  $-p_1 - p_2 + 2$ .

• برای خط  $y = x - 2$ ، می‌خواهیم تمام نقاط بالای خط یک شوند؛ چراکه، می‌خواهیم مساحت مثلاً

در طبقه‌بندی یک گردد، بنابراین داریم:  $-p_1 = +p_2 + 2$ .

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

$$w^T = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix}, b^T = \begin{bmatrix} 0 & 0 & 2 & 2 \end{bmatrix} \quad (31)$$

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

$$w^T = \begin{bmatrix} -1 & -1 & 1 & 1 & -1 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & -1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}, b^T = \begin{bmatrix} -1 & 1 & 1 & -1 & -1 & 2 & 0 & 0 & 2 & 2 \end{bmatrix} \quad (32)$$

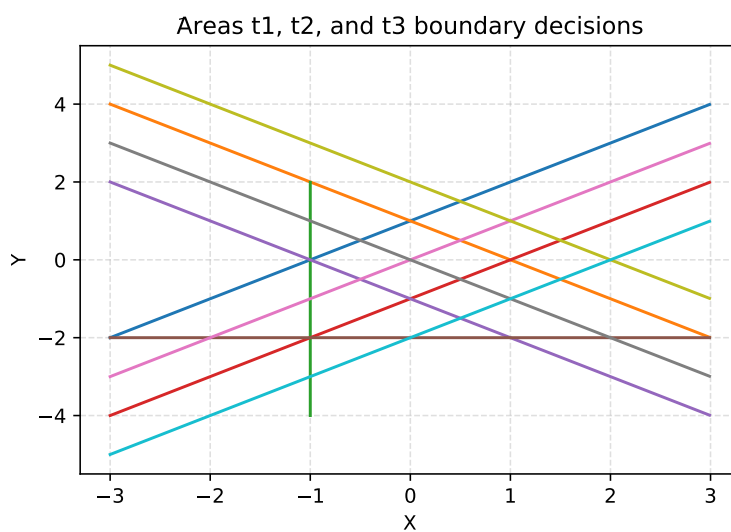
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Create the vectors X and Y
5 x = np.arange(-3, 3, 0.01)
```

```
6 y1 = x + 1
7 y2 = -x + 1
8 x3 = -1*np.ones(600)
9 y3 = x - 1
10 y4 = -x -1
11 y5 = -2*np.ones(600)
12 y6 = x
13 y7 = -x
14 y8 = -x + 2
15 y9 = x -2
16
17 # Create the plot
18 plt.plot(x, y1, label='y1 = -x1 - 1')
19 plt.plot(x, y2, label='y2 = -1')
20 plt.plot(x3, y3, label='x3 = -1')
21 plt.plot(x, y3, label='y3 = x3')
22 plt.plot(x4, y3, label='x4 = 1')
23 plt.plot(x, y4, label='y4 = 0')
24 plt.plot(x, y5, label='y1 = -x1 - 1')
25 plt.plot(x, y6, label='y2 = -1')
26 plt.plot(x, y7, label='x3 = -1')
27 plt.plot(x, y8, label='y3 = x3')
28 plt.plot(x, y9, label='x4 = 1')
29 # plt.plot(x, y4, label='y4 = 0')
30
31 # Add a title
32 plt.title('Triangle t1 and t2 boundary decisions')
33
34 # Add X and y Label
35 plt.xlabel('X')
36 plt.ylabel('Y')
37
38 # Add a grid
39 plt.grid(alpha=.4,linestyle='--')
40
```

```

41 # Add a Legend
42 # plt.legend()
43
44 # Save the plot
45 plt.savefig('E1134.pdf')
46
47 # Show the plot
48 plt.show()

```



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

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

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

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

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

$$b_2 = \begin{bmatrix} -2 & -2 & -3 \end{bmatrix} \quad (34)$$

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 دسته 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 انجام شود؛ در نتیجه داریم:

$$w_3 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, b_3 = [2] \quad (35)$$

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

$$w^T = \begin{bmatrix} -1 & -1 & 1 & 1 & -1 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & -1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}, b^T = \begin{bmatrix} -1 & 1 & 1 & -1 & -1 & 2 & 0 & 0 & 2 & 2 \end{bmatrix} \quad (36)$$

در ادامه، در لایه پنهان برای عملیات AND داریم:

$$w_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}, b_2 = \begin{bmatrix} -2 & -2 & -3 \end{bmatrix} \quad (37)$$

در نهایت، در لایه انتهایی برای عملیات OR داریم:

$$w_3 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, b_3 = [2] \quad (38)$$

از نمودار می‌فهمیم که نقطه  $(0.5, 0.5)$  باید در مثلث اول، و در نتیجه دسته یک، و نقطه  $(-2, -1)$  باید خارج از هر دو مثلث یعنی دسته دو طبقه‌بندی شود. حال نقاط را با استفاده از دستورات زیر تست می‌کنیم. نتایج ذیل هربخش آورده شده است. همان‌طور که مشاهده می‌شود نقطه اول یک و درون نواحی و نقطه دوم ۱- و خارج از نواحی تشخیص داده شده است.

```

1 import numpy as np
2
3 def hardlims(x):
4     result = np.zeros((x.shape[0], x.shape[1]))
5     for i in range(x.shape[0]):
6         for j in range(x.shape[1]):
7             if x[i,j] < 0:
8                 result[i,j] = -1
9             else:
10                 result[i,j] = 1
11     return result
12
13 w1 = np.array([[ -1, 1], [ -1, -1], [ 1, 0], [ 1, -1], [ -1, -1], [ 0, 1], [ 1, -1],
14               [ 1, 1], [ -1, -1], [ -1, 1]])
15 b1 = np.array([[ -1], [ 1], [ 1], [ -1], [ -1], [ 2], [ 0], [ 0], [ 2], [ 2]])
16 w2 = np.array([[ 1, 1, 1, 0, 0, 0, 0, 0, 0, 0], [ 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
17               [ 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]])
18 b2 = np.array([[ -2], [ -2], [ -3]])
19 w3 = np.array([[ 1, 1, 1]])
20 b3 = 2
21
22 p1 = np.array([[ -0.5], [ 0.5]])
23 p2 = np.array([[ -2], [ -1]])
24
25 # Test
26
27 a11 = hardlims(np.dot(w1, p1) + b1)
28 print(a11)
29
30 [[ 1.]
31  [ 1.]
32  [ 1.]
33  [-1.]
34  [-1.]
35  [ 1.]
36  [-1.]

```

```
34 [ 1.]
35 [ 1.]
36 [ 1.]]
37
38 a21 = hardlims(np.dot(w1, p2) + b1)
39 print(a21)
40
41 [[ 1.]
42 [ 1.]
43 [-1.]
44 [-1.]
45 [ 1.]
46 [ 1.]
47 [-1.]
48 [-1.]
49 [ 1.]
50 [ 1.]]
51
52 w2
53
54 array([[1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
55        [0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
56        [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]])
57
58 a12 = hardlims(np.dot(w2, a11) + b2)
59 print(a12)
60
61 [[ 1.]
62 [-1.]
63 [-1.]]
64
65 a22 = hardlims(np.dot(w2, a21) + b2)
66 print(a22)
67
68 [[-1.]
```



```

69 [-1.]
70 [-1.]]
71
72 a13 = hardlims(np.dot(w3, a12) + b3)
73 print(a13)
74
75 [[1.]]
76
77 a23 = hardlims(np.dot(w3, a22) + b3)
78 print(a23)
79
80 [[-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  $y = -x$ ,  $y = 1$ ,  $y = -2x$  تعیین شود. این سه معادله 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:

$$w^T = \begin{bmatrix} -1 & 0.5 & 0 \\ -1 & 1 & -1 \end{bmatrix}, b^T = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \quad (39)$$

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 معادله 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:

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

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

$$w^T = \begin{bmatrix} -1 & 0.5 & 0 & 1 & -\frac{2}{3} & 1 \\ 0 & 0 & 1 & -1 & 1 & 0 \end{bmatrix}, b^T = \begin{bmatrix} 0 & 0 & 1 & -1 & \frac{1}{3} & -1 \end{bmatrix} \quad (41)$$

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

$$w_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (42)$$

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

$$b_2 = \begin{bmatrix} -2 & -2 \end{bmatrix} \quad (43)$$

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 دسته 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 انجام شود؛ در نتیجه داریم:

$$w_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}, b_3 = [1] \quad (44)$$

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

$$w^T = \begin{bmatrix} -1 & 2 & 0 & 1 & -\frac{2}{3} & 1 \\ 0 & 0 & 1 & -1 & 1 & 0 \end{bmatrix}, b^T = \begin{bmatrix} 0 & 0 & 1 & -1 & \frac{1}{3} & -1 \end{bmatrix} \quad (45)$$

در ادامه، در لایه پنهان برای عملیات AND داریم:

$$w_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}, b_2 = \begin{bmatrix} -2 & -2 \end{bmatrix} \quad (46)$$

در نهایت، در لایه انتهایی برای عملیات OR داریم:

$$w_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}, b_3 = [1] \quad (47)$$

از نمودار می‌فهمیم که نقطه  $(-1, 0.5)$  باید در مثلث اول، و در نتیجه دسته یک، و نقطه  $(0, -1)$  باید خارج از هر دو مثلث یعنی دسته 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.

```
1 import numpy as np
```

```
2
```

```
3 def hardlims(x):
4     result = np.zeros((x.shape[0], x.shape[1]))
5     for i in range(x.shape[0]):
6         for j in range(x.shape[1]):
7             if x[i,j] < 0:
8                 result[i,j] = -1
9             else:
10                result[i,j] = 1
11        return result
12
13 w1 = np.array([[ -1, 0], [0.5, 0], [0, 1], [1, -1], [-2/3, 1], [1, 0]])
14 b1 = np.array([[0], [0], [1], [-1], [1/3], [-1]])
15 w2 = np.array([[1, 1, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1]])
16 b2 = np.array([[ -2], [-2]])
17 w3 = np.array([[1, 1]])
18 b3 = 1
19
20 p1 = np.array([[ -1], [0.5]])
21 p2 = np.array([[0], [-1]])
22
23 # Test
24 a11 = hardlims(np.dot(w1, p1) + b1)
25 print(a11)
26
27 a21 = hardlims(np.dot(w1, p2) + b1)
28 print(a21)
29
30 w2
31
32 a12 = hardlims(np.dot(w2, a11) + b2)
33 print(a12)
34
35 a22 = hardlims(np.dot(w2, a21) + b2)
36 print(a22)
37
```

```

38 a13 = hardlims(np.dot(w3, a12) + b3)
39 print(a13)
40
41 a23 = hardlims(np.dot(w3, a22) + b3)
42 print(a23)

```

### توضیح پوشه کدها

در پوشه اصلی این سوال (Q8\_\_11\_01)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند. هم‌چنین فایل اجرای برخط کدها از طریق [پیوند گوگل کولب](#) 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.

```

1 % Solve an Input-Output Fitting problem with a Neural Network
2 % Script generated by Neural Fitting app
3 % Created 18-Jan-2023 11:57:05
4 %
5 % This script assumes these variables are defined:
6 %
7 % TestInputs - input data.
8 % TestTargets - target data.
9 clc
10 close all
11 clear all
12
13 %% Load Data
14
15 x = [-2 -1.5 -1 0.5 0 0.5 1 1.5 2];
16 t = [-2 -2 -1 0 0 0 1 2 2];

```

```
17
18 % Choose a Training Function
19 % For a list of all training functions type: help nntrain
20 % 'trainlm' is usually fastest.
21 % 'trainbr' takes longer but may be better for challenging problems.
22 % 'trainscg' uses less memory. Suitable in low memory situations.
23 trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation.
24
25 % Create a Fitting Network
26 hiddenLayerSize = 4;
27 net = fitnet(hiddenLayerSize,trainFcn);
28
29 % Choose Input and Output Pre/Post-Processing Functions
30 % For a list of all processing functions type: help nnprocess
31 net.input.processFcns = {'removeconstantrows','mapminmax'};
32 net.output.processFcns = {'removeconstantrows','mapminmax'};
33
34 % Setup Division of Data for Training, Validation, Testing
35 % For a list of all data division functions type: help nndivision
36 net.divideFcn = 'dividerand'; % Divide data randomly
37 net.divideMode = 'sample'; % Divide up every sample
38 net.divideParam.trainRatio = 80/100;
39 net.divideParam.valRatio = 10/100;
40 net.divideParam.testRatio = 10/100;
41
42 % Choose a Performance Function
43 % For a list of all performance functions type: help nnperformance
44 net.performFcn = 'mse'; % Mean Squared Error
45
46 % Choose Plot Functions
47 % For a list of all plot functions type: help nnplot
48 net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
49     'plotregression', 'plotfit'};
50
51 % Train the Network
```

```
52 [net,tr] = train(net,x,t);
53
54 % Test the Network
55 y = net(x);
56 e = gsubtract(t,y);
57 performance = perform(net,t,y)
58
59 % Recalculate Training, Validation and Test Performance
60 trainTargets = t .* tr.trainMask{1};
61 valTargets = t .* tr.valMask{1};
62 testTargets = t .* tr.testMask{1};
63 trainPerformance = perform(net,trainTargets,y)
64 valPerformance = perform(net,valTargets,y)
65 testPerformance = perform(net,testTargets,y)
66
67 % View the Network
68 view(net)
69
70 % Plots
71 % Uncomment these lines to enable various plots.
72 %figure, plotperform(tr)
73 %figure, plottrainstate(tr)
74 %figure, ploterrhist(e)
75 %figure, plotregression(t,y)
76 %figure, plotfit(net,x,t)
77
78 % Deployment
79 % Change the (false) values to (true) to enable the following code blocks.
80 % See the help for each generation function for more information.
81 if (false)
82     % Generate MATLAB function for neural network for application
83     % deployment in MATLAB scripts or with MATLAB Compiler and Builder
84     % tools, or simply to examine the calculations your trained neural
85     % network performs.
86     genFunction(net,'myNeuralNetworkFunction');
```

```

87     y = myNeuralNetworkFunction(x);
88 end
89 if (false)
90     % Generate a matrix-only MATLAB function for neural network code
91     % generation with MATLAB Coder tools.
92     genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
93     y = myNeuralNetworkFunction(x);
94 end
95 if (false)
96     % Generate a Simulink diagram for simulation or deployment with.
97     % Simulink Coder tools.
98     gensim(net);
99 end
100
101 -----
102
103 performance =
104     0.0016
105
106 trainPerformance =
107     2.3843e-05
108
109 valPerformance =
110     0.0124
111
112 testPerformance =
113     0.0014

```

ii

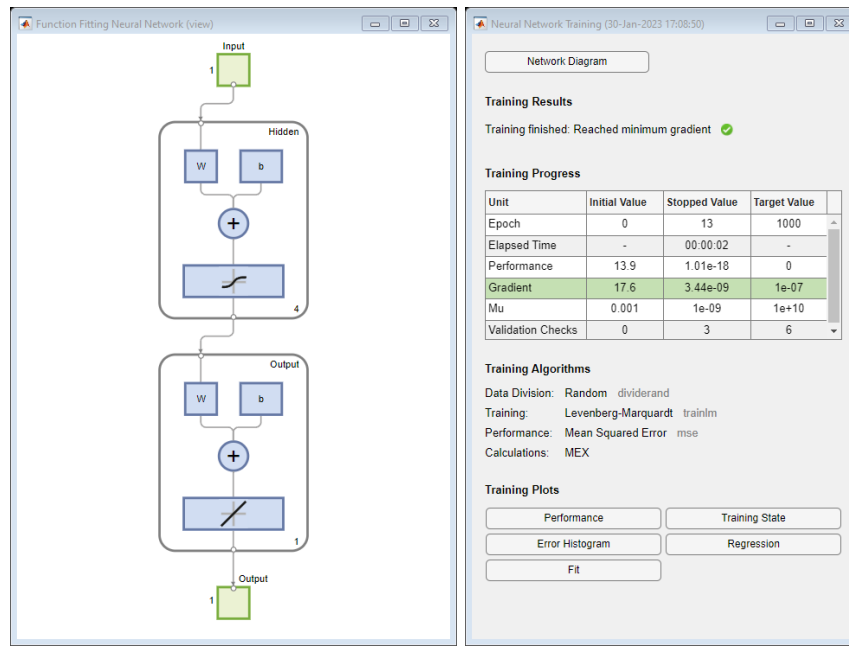
برای این قسمت از دستورات زیر استفاده می‌کنیم و نتایج مطابق شکل ۲۶ و شکل ۲۷ خواهد بود که نشان می‌دهد هدف سوال به خوبی به سرانجام رسیده است.

```

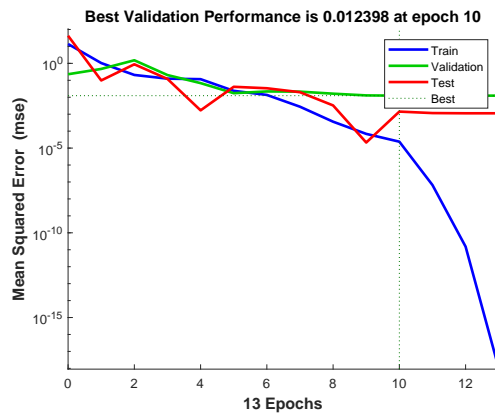
1 % Solve an Input-Output Fitting problem with a Neural Network
2 % Script generated by Neural Fitting app

```

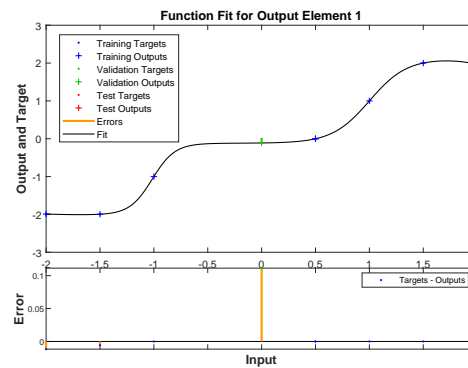




شکل ۲۴: نتیجه سوال ۱۱-۲-۱.



(ب)



(آ)

شکل ۲۵: نتیجه سوال ۱۱-۲-۱.

```

3 % Created 18-Jan-2023 11:57:05
4 %
5 % This script assumes these variables are defined:
6 %
7 % TestInputs - input data.
8 % TestTargets - target data.
9 clc
10 close all

```

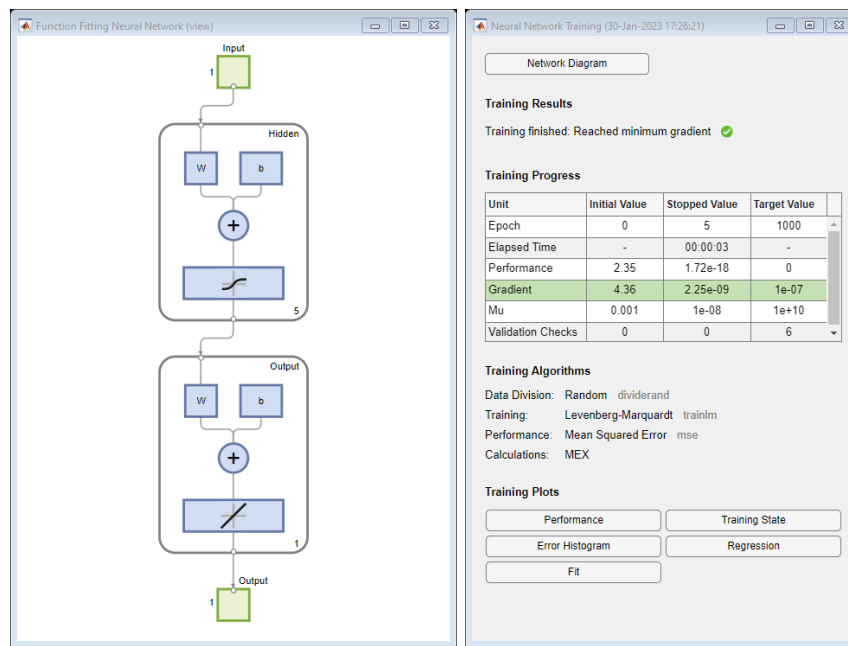
```
11 clear all
12
13 %% Load Data
14
15 x = [-2 -1.5 -1 0.5 0 0.5 1 1.5 2];
16 t = [0 0 -1 -2 -2 -2 -1 0 0];
17
18 % Choose a Training Function
19 % For a list of all training functions type: help nntrain
20 % 'trainlm' is usually fastest.
21 % 'trainbr' takes longer but may be better for challenging problems.
22 % 'trainscg' uses less memory. Suitable in low memory situations.
23 trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation.
24
25 % Create a Fitting Network
26 hiddenLayerSize = 5;
27 net = fitnet(hiddenLayerSize, trainFcn);
28
29 % Choose Input and Output Pre/Post-Processing Functions
30 % For a list of all processing functions type: help nnprocess
31 net.input.processFcns = {'removeconstantrows', 'mapminmax'};
32 net.output.processFcns = {'removeconstantrows', 'mapminmax'};
33
34 % Setup Division of Data for Training, Validation, Testing
35 % For a list of all data division functions type: help nndivision
36 net.divideFcn = 'dividerand'; % Divide data randomly
37 net.divideMode = 'sample'; % Divide up every sample
38 net.divideParam.trainRatio = 80/100;
39 net.divideParam.valRatio = 10/100;
40 net.divideParam.testRatio = 10/100;
41
42 % Choose a Performance Function
43 % For a list of all performance functions type: help nnperformance
44 net.performFcn = 'mse'; % Mean Squared Error
45
```

```

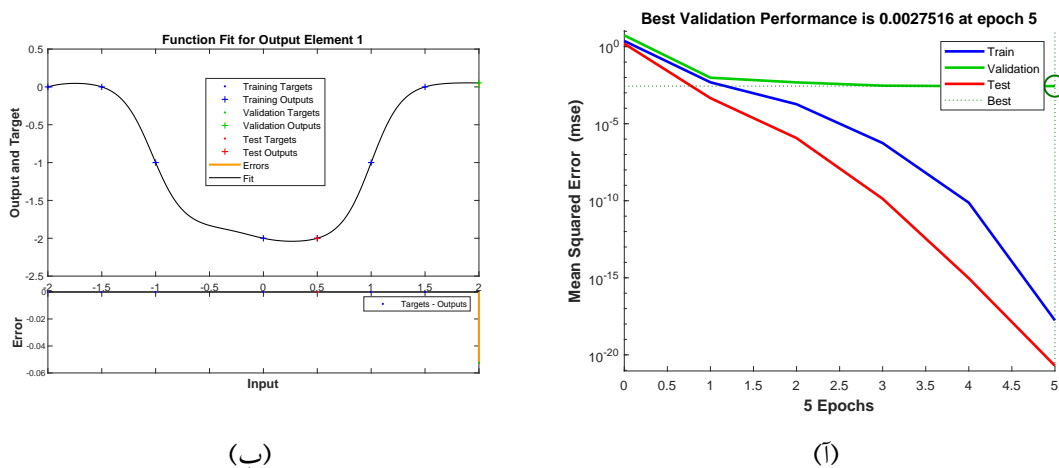
46 % Choose Plot Functions
47 % For a list of all plot functions type: help nnplot
48 net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
49     'plotregression', 'plotfit'};
50
51 % Train the Network
52 [net,tr] = train(net,x,t);
53
54 % Test the Network
55 y = net(x);
56 e = gsubtract(t,y);
57 performance = perform(net,t,y)
58
59 % Recalculate Training, Validation and Test Performance
60 trainTargets = t .* tr.trainMask{1};
61 valTargets = t .* tr.valMask{1};
62 testTargets = t .* tr.testMask{1};
63 trainPerformance = perform(net,trainTargets,y)
64 valPerformance = perform(net,valTargets,y)
65 testPerformance = perform(net,testTargets,y)
66
67 % View the Network
68 view(net)
69
70 % Plots
71 % Uncomment these lines to enable various plots.
72 %figure, plotperform(tr)
73 %figure, plottrainstate(tr)
74 %figure, ploterrhist(e)
75 %figure, plotregression(t,y)
76 %figure, plotfit(net,x,t)
77
78 % Deployment
79 % Change the (false) values to (true) to enable the following code blocks.
80 % See the help for each generation function for more information.

```

```
81 if (false)
82     % Generate MATLAB function for neural network for application
83     % deployment in MATLAB scripts or with MATLAB Compiler and Builder
84     % tools, or simply to examine the calculations your trained neural
85     % network performs.
86     genFunction(net, 'myNeuralNetworkFunction');
87     y = myNeuralNetworkFunction(x);
88 end
89 if (false)
90     % Generate a matrix-only MATLAB function for neural network code
91     % generation with MATLAB Coder tools.
92     genFunction(net, 'myNeuralNetworkFunction', 'MatrixOnly', 'yes');
93     y = myNeuralNetworkFunction(x);
94 end
95 if (false)
96     % Generate a Simulink diagram for simulation or deployment with.
97     % Simulink Coder tools.
98     gensim(net);
99 end
100 -----
101
102 performance =
103     3.0573e-04
104
105 trainPerformance =
106     1.7216e-18
107
108 valPerformance =
109     0.0028
110
111 testPerformance =
112     1.9680e-21
```



شکل ۲۶: نتیجه سوال ۱۱-۲-۲.



(ب)

(آ)

شکل ۲۷: نتیجه سوال ۱۱-۲-۲.

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.

```

1 % Solve an Input-Output Fitting problem with a Neural Network
2 % Script generated by Neural Fitting app
3 % Created 18-Jan-2023 11:57:05
4 %
5 % This script assumes these variables are defined:

```

```
6 %
7 % TestInputs - input data.
8 % TestTargets - target data.
9 clc
10 close all
11 clear all
12
13 %% Load Data
14
15 x = [-2 -1.5 -1 0.5 0 0.5 1 1.5 2];
16 t = [0 0 1 2 2 2 1 0 0];
17
18 % Choose a Training Function
19 % For a list of all training functions type: help nntrain
20 % 'trainlm' is usually fastest.
21 % 'trainbr' takes longer but may be better for challenging problems.
22 % 'trainscg' uses less memory. Suitable in low memory situations.
23 trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation.
24
25 % Create a Fitting Network
26 hiddenLayerSize = 5;
27 net = fitnet(hiddenLayerSize,trainFcn);
28
29 % Choose Input and Output Pre/Post-Processing Functions
30 % For a list of all processing functions type: help nnprocess
31 net.input.processFcns = {'removeconstantrows','mapminmax'};
32 net.output.processFcns = {'removeconstantrows','mapminmax'};
33
34 % Setup Division of Data for Training, Validation, Testing
35 % For a list of all data division functions type: help nndivision
36 net.divideFcn = 'dividerand'; % Divide data randomly
37 net.divideMode = 'sample'; % Divide up every sample
38 net.divideParam.trainRatio = 80/100;
39 net.divideParam.valRatio = 10/100;
40 net.divideParam.testRatio = 10/100;
```

```
41
42 % Choose a Performance Function
43 % For a list of all performance functions type: help nnperformance
44 net.performFcn = 'mse'; % Mean Squared Error
45
46 % Choose Plot Functions
47 % For a list of all plot functions type: help nnplot
48 net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
49     'plotregression', 'plotfit'};
50
51 % Train the Network
52 [net,tr] = train(net,x,t);
53
54 % Test the Network
55 y = net(x);
56 e = gsubtract(t,y);
57 performance = perform(net,t,y)
58
59 % Recalculate Training, Validation and Test Performance
60 trainTargets = t .* tr.trainMask{1};
61 valTargets = t .* tr.valMask{1};
62 testTargets = t .* tr.testMask{1};
63 trainPerformance = perform(net,trainTargets,y)
64 valPerformance = perform(net,valTargets,y)
65 testPerformance = perform(net,testTargets,y)
66
67 % View the Network
68 view(net)
69
70 % Plots
71 % Uncomment these lines to enable various plots.
72 %figure, plotperform(tr)
73 %figure, plottrainstate(tr)
74 %figure, ploterrhist(e)
75 %figure, plotregression(t,y)
```

```

76 %figure, plotfit(net,x,t)
77
78 % Deployment
79 % Change the (false) values to (true) to enable the following code blocks.
80 % See the help for each generation function for more information.
81 if (false)
82     % Generate MATLAB function for neural network for application
83     % deployment in MATLAB scripts or with MATLAB Compiler and Builder
84     % tools, or simply to examine the calculations your trained neural
85     % network performs.
86     genFunction(net,'myNeuralNetworkFunction');
87     y = myNeuralNetworkFunction(x);
88 end
89 if (false)
90     % Generate a matrix-only MATLAB function for neural network code
91     % generation with MATLAB Coder tools.
92     genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
93     y = myNeuralNetworkFunction(x);
94 end
95 if (false)
96     % Generate a Simulink diagram for simulation or deployment with.
97     % Simulink Coder tools.
98     gensim(net);
99 end
100 -----
101
102 performance =
103     6.6775e-05
104
105 trainPerformance =
106     3.6007e-27
107
108 valPerformance =
109     2.2159e-27
110

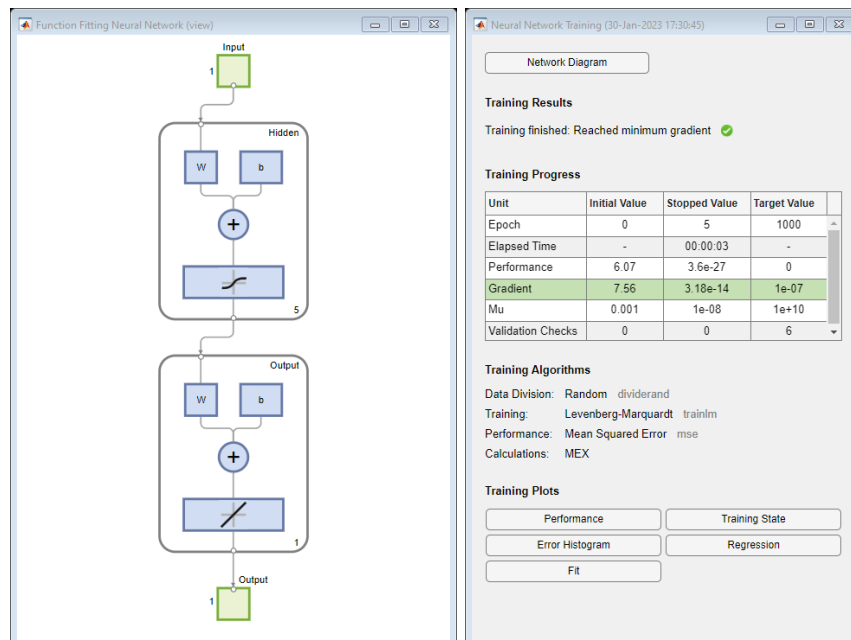
```



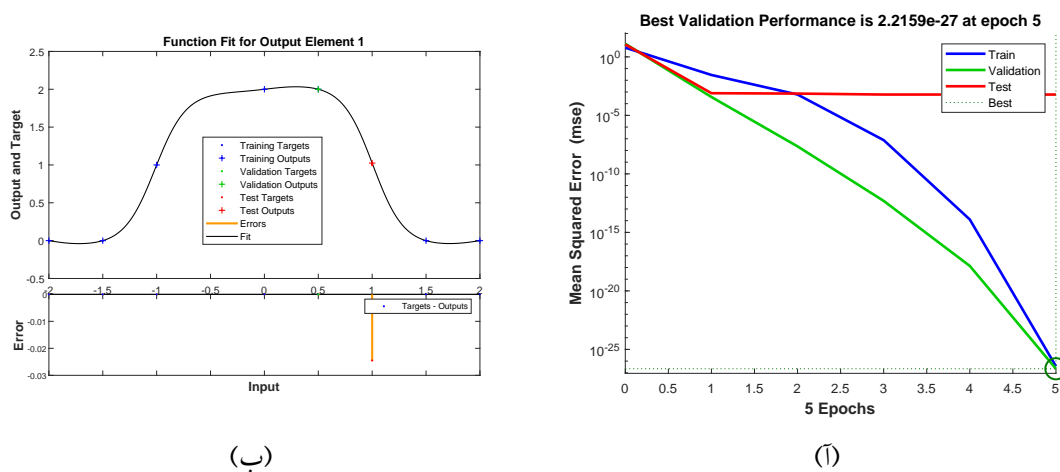
```

111 testPerformance =
112     6.0097e-04

```



شکل ۲۸: نتیجه سوال ۱۱-۲-۳.



شکل ۲۹: نتیجه سوال ۱۱-۲-۳.

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.

```

1 % Solve an Input-Output Fitting problem with a Neural Network
2 % Script generated by Neural Fitting app
3 % Created 18-Jan-2023 11:57:05
4 %
5 % This script assumes these variables are defined:
6 %
7 % TestInputs - input data.
8 % TestTargets - target data.
9
10 clc
11 close all
12 clear all
13
14 %% Load Data
15
16 x = [-2 -1.5 -1 0.5 0 0.5 1 1.5 2];
17 t = [0 0 1 2 2 2 1 0 0];
18
19 % Choose a Training Function
20 % For a list of all training functions type: help nntrain
21 % 'trainlm' is usually fastest.
22 % 'trainbr' takes longer but may be better for challenging problems.
23 % 'trainscg' uses less memory. Suitable in low memory situations.
24 trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation.
25
26 % Create a Fitting Network
27 hiddenLayerSize = 5;
28 net = fitnet(hiddenLayerSize,trainFcn);
29
30 % Choose Input and Output Pre/Post-Processing Functions
31 % For a list of all processing functions type: help nnprocess
32 net.input.processFcns = {'removeconstantrows','mapminmax'};
33 net.output.processFcns = {'removeconstantrows','mapminmax'};
34
35 % Setup Division of Data for Training, Validation, Testing
36 % For a list of all data division functions type: help nndivision

```

```
36 net.divideFcn = 'dividerand'; % Divide data randomly
37 net.divideMode = 'sample'; % Divide up every sample
38 net.divideParam.trainRatio = 80/100;
39 net.divideParam.valRatio = 10/100;
40 net.divideParam.testRatio = 10/100;
41
42 % Choose a Performance Function
43 % For a list of all performance functions type: help nnperformance
44 net.performFcn = 'mse'; % Mean Squared Error
45
46 % Choose Plot Functions
47 % For a list of all plot functions type: help nnplot
48 net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
49               'plotregression', 'plotfit'};
50
51 % Train the Network
52 [net,tr] = train(net,x,t);
53
54 % Test the Network
55 y = net(x);
56 e = gsubtract(t,y);
57 performance = perform(net,t,y)
58
59 % Recalculate Training, Validation and Test Performance
60 trainTargets = t .* tr.trainMask{1};
61 valTargets = t .* tr.valMask{1};
62 testTargets = t .* tr.testMask{1};
63 trainPerformance = perform(net,trainTargets,y)
64 valPerformance = perform(net,valTargets,y)
65 testPerformance = perform(net,testTargets,y)
66
67 % View the Network
68 view(net)
69
70 % Plots
```

```

71 % Uncomment these lines to enable various plots.
72 %figure, plotperform(tr)
73 %figure, plottrainstate(tr)
74 %figure, ploterrhist(e)
75 %figure, plotregression(t,y)
76 %figure, plotfit(net,x,t)
77
78 % Deployment
79 % Change the (false) values to (true) to enable the following code blocks.
80 % See the help for each generation function for more information.
81 if (false)
82     % Generate MATLAB function for neural network for application
83     % deployment in MATLAB scripts or with MATLAB Compiler and Builder
84     % tools, or simply to examine the calculations your trained neural
85     % network performs.
86     genFunction(net,'myNeuralNetworkFunction');
87     y = myNeuralNetworkFunction(x);
88 end
89 if (false)
90     % Generate a matrix-only MATLAB function for neural network code
91     % generation with MATLAB Coder tools.
92     genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
93     y = myNeuralNetworkFunction(x);
94 end
95 if (false)
96     % Generate a Simulink diagram for simulation or deployment with.
97     % Simulink Coder tools.
98     gensim(net);
99 end
100 -----
101
102 performance =
103     0.0713
104
105 trainPerformance =

```

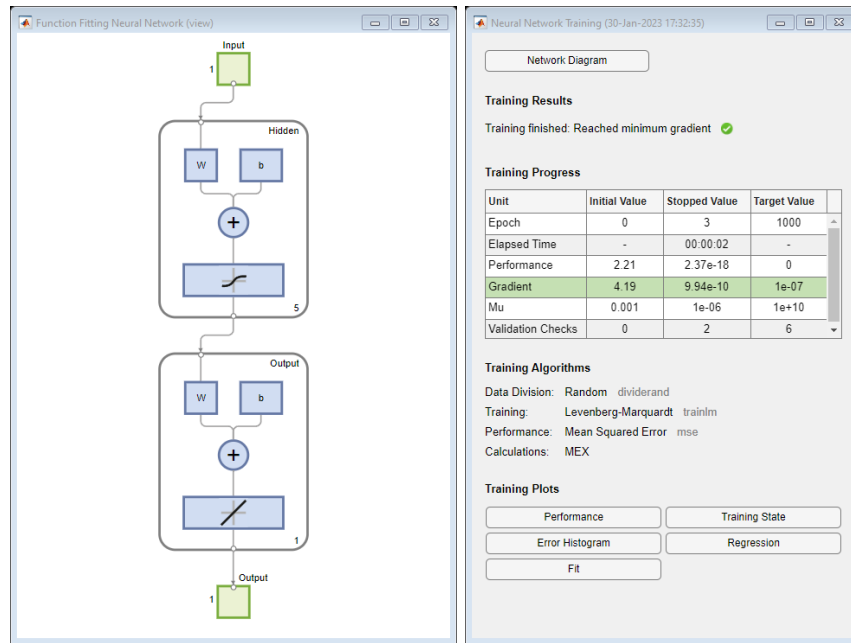
2.8563e-05

valPerformance =

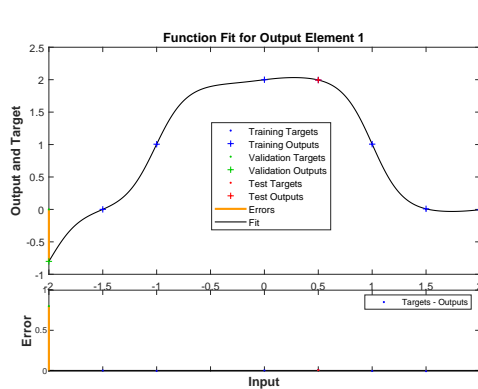
0.6419

testPerformance =

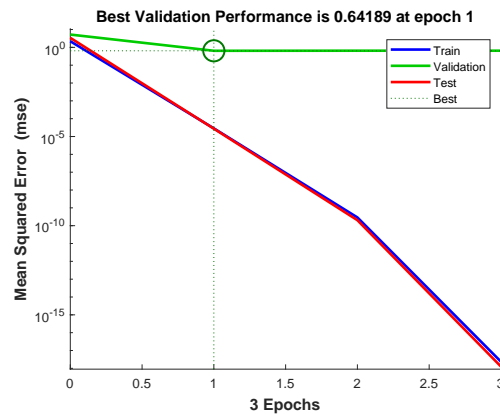
2.7548e-05



شکل ٣٠: نتیجه سوال ١١-٢-٤.



(ب)



(ا)

شکل ٣١: نتیجه سوال ١١-٢-٤.

## توضیح پوشه کدها

در پوشه اصلی این سوال (Q9\_11\_02)، فایل‌های کد مربوط به این سوال قرار گرفته‌اند.

## پاسخ سوال ۱۱.۱۴

با در نظر گرفتن توابع تبدیل هر لایه مشتقات آن‌ها را محاسبه می‌کنیم:

$$\begin{aligned} f^1(n) = \frac{1}{1 + e^{-n}} &\implies \dot{f}^1(n) = \frac{1}{1 + e^n} \\ f^2(n) = n &\implies \dot{f}^2(n) = 1 \end{aligned} \quad (48)$$

ورودی به صورت  $p = 1$  داده شده و بایاس‌ها و اوزان به صورت رابطه ۴۹ تعریف می‌شوند.

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

i

برای مسیر پیش‌رو می‌نویسیم:

$$\begin{aligned} a^\circ &= 1 \\ a^1 &= \log \operatorname{sig} \left( \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right) = \begin{bmatrix} 0.7311 \\ 0.8808 \end{bmatrix} \\ a^2 &= \text{pureline}(n^2) = n^2 \implies \frac{\partial a^2}{\partial n^2} = 1 \end{aligned} \quad (50)$$

$$\begin{aligned} \frac{\partial a^2}{\partial n_1^1} &= \frac{\partial a^2}{\partial n^2} \times \frac{\partial n^2}{\partial n_1^1} = 1 \times (w_1^2 \dot{f}^1(n^1)) = 2 \times \begin{bmatrix} 0 & 0.1192 \end{bmatrix} = \begin{bmatrix} 0 & 0.2384 \end{bmatrix} \\ \frac{\partial a^2}{\partial n_2^1} &= \frac{\partial a^2}{\partial n^2} \times \frac{\partial n^2}{\partial n_2^1} = 1 \times (w_2^2 \dot{f}^1(n^1)) = 3 \times \begin{bmatrix} 0 & 0.1192 \end{bmatrix} = \begin{bmatrix} 0 & 0.3576 \end{bmatrix} \end{aligned} \quad (51)$$

ii

می نویسیم:

$$\frac{\partial a^2}{\partial p} = \frac{\partial a^2}{\partial n_1^1} \times \frac{\partial n_1^1}{\partial p} + \frac{\partial a^2}{\partial n_2^1} \times \frac{\partial n_2^1}{\partial p} = \begin{bmatrix} 0 & 0.2384 \end{bmatrix} \times 2 + \begin{bmatrix} 0 & 0.7152 \end{bmatrix} \times 1 = \begin{bmatrix} 0 & 1.192 \end{bmatrix} \quad (52)$$

## پاسخ سوال ۱۱.۲۵

### راه حل اول

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

```

1 clc
2 close all
3 clear all
4
5 tic
6
7 %Initialize data
8 W1 = rand(2,1) - 0.5;
9 W2 = rand(1,2) - 0.5;
10 b1 = rand(2,1) - 0.5;
11 b2 = rand - 0.5;
12 a1 = zeros(2,1);
13
14 %Output the initial set
15 W1_0 = W1
16 b1_0 = b1
17 W2_0 = W2
18 b2_0 = b2
19
20 alfa = 0.2;      %learning rate
21 tol = 0.001;    %tol: tolerance
22 mse = 1;        %mse: mean square error

```

```

23 iter = 0;
24
25 figure;
26 while (mse > tol && iter<=1500)
27     mse = 0;
28     i = 0;
29     iter = iter + 1;
30     for P = -2 : .1 : 2
31         i = i + 1;
32         T = 1 + sin(pi*P/2);
33         a1 = logsig(W1*P + b1);
34         a2 = purelin(W2*a1 + b2);
35         mse = mse + (T - a2)^2;
36         A(i) = a2;
37
38         dlogsig = [(1 - a1(1))* a1(1) 0;0 (1 - a1(2))* a1(2)];
39         s2 = -2 * (T - a2);
40         s1 = dlogsig * W2' * s2;
41
42         W2 = W2 - alfa * s2 * a1';
43         W1 = W1 - alfa * s1 * P;
44         b2 = b2 - alfa * s2;
45         b1 = b1 - alfa * s1;
46     end
47     P = -2 : .1 : 2;
48     if (mod(iter,10) == 0)
49         plot(P,A,'g:')
50     end
51     hold on;
52 end
53
54 %Display in graph
55 P = -2 : .1 : 2;
56 T = 1 + sin(pi*P/2);
57 %figure;

```



```

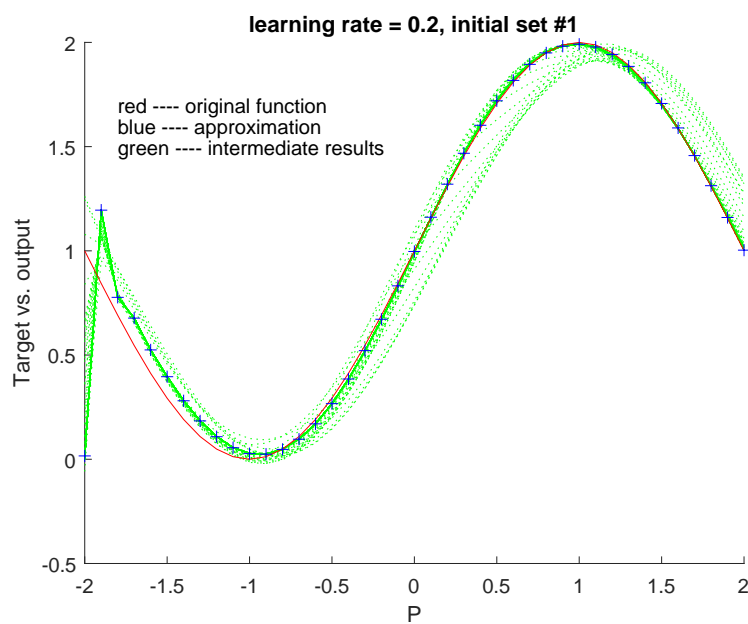
58 plot(P,T,'r-',P,A,'b+')
59 title('Fig6.1 learning rate = 0.2, initial set #1');
60 text(-1.8,1.7,'red ---- original function');
61 text(-1.8,1.6,'blue ---- approximation');
62 text(-1.8,1.5,'green ---- intermediate results');
63 xlabel('P'), ylabel('Target vs. output');
64 W1
65 b1
66 W2
67 b2
68 iter
69
70 toc
71 -----
72
73 W1_0 =
74     0.1892
75     0.2482
76
77 b1_0 =
78    -0.2710
79     0.4133
80
81 W2_0 =
82    -0.0495    -0.4162
83
84 b2_0 =
85    -0.3476
86
87 W1 =
88    -1.7262
89    -2.6171
90
91 b1 =
92     3.3501

```

```

93     0.5607
94
95 W2 =
96     3.6560    -2.9336
97
98 b2 =
99     -0.7111
100
101 iter =
102     1501
103
104 Elapsed time is 1.047967 seconds.

```



شکل ۳۲: نتیجه سوال ۱۱-۲۵.

با تغییر مقادیر اولیه داریم:

```

1 W1_0 =
2     0.1477
3     -0.0491
4
5 b1_0 =
6     0.2447

```

```

7      -0.3110
8
9  W2_0 =
10      0.0470    -0.2037
11
12  b2_0 =
13      0.1868
14
15  W1 =
16      -1.5950
17      -2.5929
18
19  b1 =
20      3.0224
21      0.5975
22
23  W2 =
24      3.8997    -3.1101
25
26  b2 =
27      -0.7619
28
29  iter =
30      1501
31
32  Elapsed time is 1.023033 seconds.

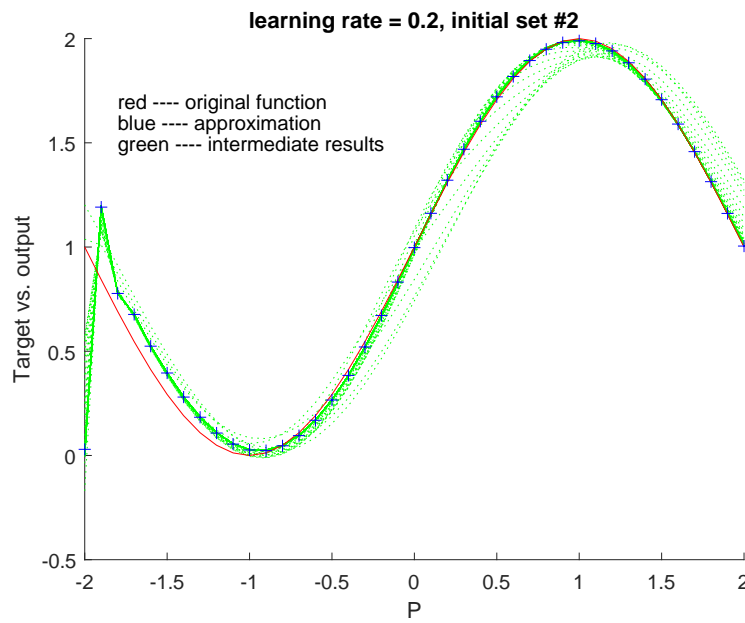
```

حال با افزایش مقدار نرخ یادگیری به ۴٪ داریم:

```

1
2  W1_0 =
3      0.2948
4      0.1443
5
6  b1_0 =
7      0.0328

```



شکل ۳۳: نتیجه سوال ۱۱-۲۵.

```

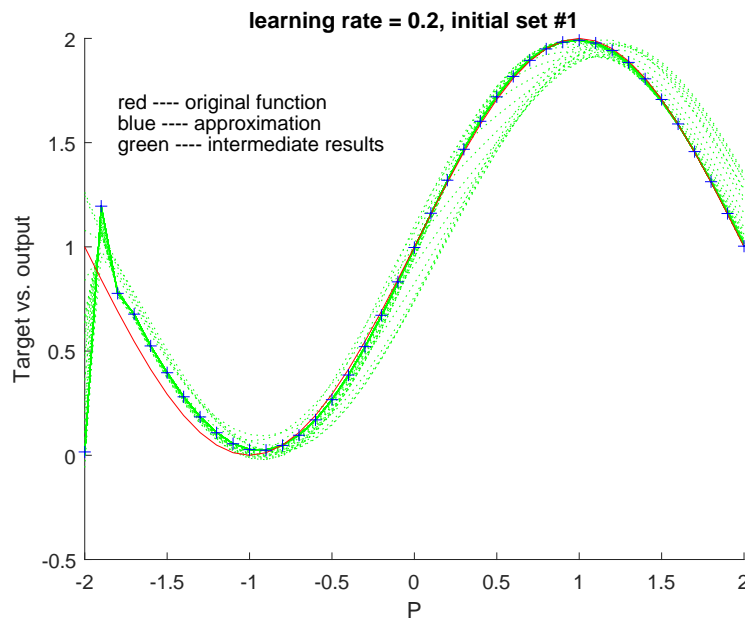
8      -0.1493
9
10 w2_0 =
11      -0.1214      0.3116
12
13 b2_0 =
14      0.4390
15
16 w1 =
17      -2.7608
18      2.1593
19
20 b1 =
21      0.3814
22      -4.3332
23
24 w2 =
25      -2.5486      -3.0327
26
27 b2 =

```

```

28      2.5222
29
30 iter =
31      1501
32
33 Elapsed time is 1.087420 seconds.

```



شکل ۳۴: نتیجه سوال ۱۱-۲۵.

با افزایش مقدار نرخ یادگیری به ۰/۸ داریم:

```

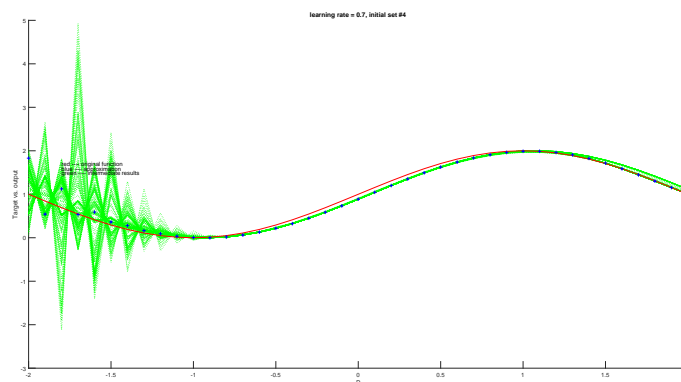
1 W1_0 =
2     -0.2695
3     0.3443
4
5 b1_0 =
6     -0.3293
7     -0.2723
8
9 W2_0 =
10    -0.3052    -0.2741
11
12 b2_0 =

```

```

13     -0.0643
14
15 W1 =
16     4.6886
17     -5.8325
18
19 b1 =
20     -8.5763
21     -11.8243
22
23 W2 =
24     -1.4307    -0.8085
25
26 b2 =
27     1.9758
28
29 iter =
30     1501
31
32 Elapsed time is 1.120159 seconds.

```



شکل ۳۵: نتیجه سوال ۱۱-۲۵.

همان طور که مشاهده می شود با زیاد شدن نرخ یادگیری، سرعت گام ها بیشتر می شود و اگر این نرخ از عددی مانند ۵۵٪ بیشتر شود پاسخ ما به سمت واگرایی می رود. هم چنین مقادیر اولیه اوزان و بایاس ها در پاسخ نهایی تأثیر به سزایی دارد.

## راه حل دوم

در راه حل دوم از دستورات زیر استفاده می کنیم و نتایج به ازای نرخ یادگیری ۳٪ به صورتی است که در شکل ۳۶ آورده شده است.

```

1 clf; clc; clear all; close all;
2
3 % Init min/max
4 xMax = 0.5;
5 xMin = -xMax;
6 difference = xMax-xMin;
7
8 % Init w1, b1, w2, b2
9
10 W1(:, 1) = xMin + rand(1, 10)*difference; % For the first layer (logsig)
11 B1(:, 1) = xMin + rand(1, 10)*difference; % The new values of the weights,
    biases will be added to each column
12
13 W2(1, :) =(xMin+rand(1,10)*difference)'; % For the second layer (purelin)
14 B2(1) = xMin + rand(1, 1)*(difference); % The new values of the weights, biases
    will be added to each row
15
16 i = 1;
17 learningRate = 0.01;
18
19 while (1)
20 for p = -2:0.01:2
21
22 % FEED-FORWARD PHASE
23
24 product = (W1(:, i).* p + B1(:, i)); % Compute activation for 1st layer
25 A1(:, i) = 1./(1+exp(-product));
26 A2(i) = W2(i, :)*A1(:, i) + B2(1); % Compute activation for 2nd layer
27
28 % COMPUTE ERROR THROUGH THE COST FUNCTION
29

```

```

30 error = 1 + sin(p * (pi/2)) - A2(i);
31
32 % BACKPROPAGATION PREPARATION
33
34 % Compute derivative for 1st activation, compute sensitivities
35 derivative = [];
36 for row = 1:10
37     element = (1-A1(row,i))*A1(row,i);
38     derivative = [derivative element];
39 end
40 R = diag(derivative);
41
42 s2 = -2 * 1 * error;
43
44 s1 = R * W2(i, :) * s2;
45
46 % UPDATE WEIGHTS AND BIASES
47
48 W1(:, i+1) = W1(:, i) - learningRate * s1 * p;
49 B1(:, i+1) = B1(:, i) - learningRate * s1;
50
51 W2(i+1, :) = W2(i, :) - learningRate * s2 * A1(:,i)';
52 B2(i+1) = B2(i) - learningRate * s2;
53
54 figure(1);
55 plot(p, error, '*b');
56 hold on;
57 xlabel('p');
58 ylabel('error');
59 legend('error');
60
61 figure(2);
62 plot(p, A2(i), '*g');
63 hold on;
64 xlabel('p');

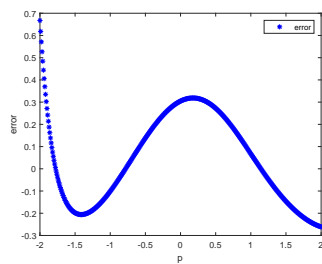
```



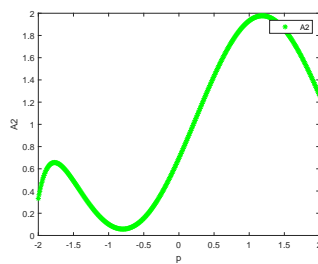
```

65 ylabel('A2');
66 legend('A2');
67
68 figure(3)
69 plot(p,1 + sin(p*(pi/2)),'*r');
70 hold on;
71 xlabel('p');
72 ylabel('1 + sin(p*pi/2)');
73 legend('1 + sin(p*pi/2)');
74
75 i = i + 1;
76
77 end
78
79 if (abs(error) < 0.01 || i<1500)
80 return % End if absolute error is smaller than the predefined threshold
81 end
82
83 end

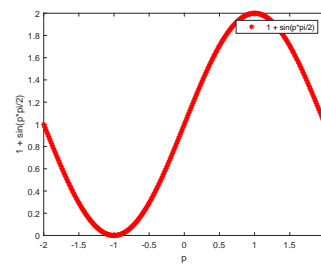
```



(ج) خطا



(ب) خروجی



(آ) اصلی

شکل ۳۶: نتیجه سوال ۱۱-۲۵.

### توضیح پوشه کدها

در پوشه 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 محاسبه خطا است:

$$a_1^o = p_1 = [1] \quad (53)$$

از صفحه پانزدهم فصل یازدهم کتاب مرجع داریم:

$$\begin{aligned} n_1^1 &= [-0.75; -0.54] \\ a_1^1 &= [0.321; 0.368] \\ n_1^2 &= [0.09 - 0.17] * [0.321; 0.368] + [0.48] = [0.4463] \\ a_1^2 &= [0.4463]; \end{aligned} \quad (54)$$

خطا برابر است با:

$$e_1 = t - a = (1 + \sin(1 * \pi/4)) - a_1^2 = 1.261 \quad (55)$$

می‌نویسیم:

$$\begin{aligned} a_2^o &= p_2 = [0] \\ n_2^1 &= [-0.27; -0.41] * [0] + [-0.48; -0.13] = [-0.48; -0.13] \\ a_2^1 &= \log \text{sig}([-0.48; -0.13]) = [0.3823; 0.5325]; \\ n_2^2 &= [0.09 - 0.17] * [0.3823; 0.5325] + [0.48] = [0.4239] \\ a_2^2 &= [0.4239]; \end{aligned} \quad (56)$$

خطا برابر است با:

$$e_1 = t - a = (1 + \sin(0 * \pi/4)) - a_2^2 = 0.5761 \quad (57)$$

گام بعدی آغازگری پس انتشار یا حساسیت مارکوات و استفاده از رابطه ۵۸ و رابطه ۵۹ است.

$$\tilde{S}_q^M = -\dot{F}^M(n_q^M) \quad (58)$$

$$\tilde{S}_q^m = F^m(n_q^m)(W^{m+1})^T \tilde{S}_q^{m+1} \quad (59)$$

در نتیجه می نویسیم:

$$\begin{aligned} \tilde{S}_1^2 &= -\dot{F}^2(n_1^2) = -[1] \\ \tilde{S}_1^1 &= \dot{F}^1(n_1^1)(w^2)^T \tilde{S}_1^2 \\ &= \left[ (1 - a_{1,1})^1 \right]^* a_{1,1}^1, 0; 0, (1 - a_{1,1})^2 \left[ a_{1,1}^2 \right]^* [0.09, -0.17]' * [-1] \\ &= [(1 - 0.321) * 0.321, 0; 0, (1 - 0.368) * 0.368] * [0.09, -0.17]' * [-1] \\ &= [-0.0196; 0.0395]; \end{aligned} \quad (60)$$

$$\begin{aligned} \tilde{S}_2^2 &= -\dot{F}^2(n_2^2) = -[1] \\ \tilde{S}_2^1 &= \dot{F}^1(n_2^1)(w^2)^T \tilde{S}_2^2 \\ &= \left[ (1 - a_{1,2})^1 \right]^* a_{1,2}^1, 0; 0, (1 - a_{1,2})^2 \left[ a_{1,2}^2 \right]^* [0.09, -0.17]' * [-1] \\ &= [(1 - 0.3823) * 0.3823, 0; 0, (1 - 0.5325) * 0.5325] * [0.09, -0.17]' * [-1] \\ &= [-0.0213; 0.0423]; \end{aligned} \quad (61)$$

$$\tilde{S}^1 = [\tilde{S}_1^1 \mid \tilde{S}_2^1] = [-0.0196 - 0.0213; 0.0395 \mid 0.0423] \quad (62)$$

حال ماتریس ژاکوبی را محاسبه می‌کنیم:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial v_1}{\partial x_1} & \frac{\partial v_1}{\partial x_2} & \frac{\partial v_1}{\partial x_3} & \frac{\partial v_1}{\partial x_4} \\ \frac{\partial v_2}{\partial x_1} & \frac{\partial v_2}{\partial x_2} & \frac{\partial v_2}{\partial x_3} & \frac{\partial v_2}{\partial x_4} \end{bmatrix} = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_{111}} & \frac{\partial e_{1,1}}{\partial w_{211}} & \frac{\partial e_{1,1}}{\partial b_{111}} & \frac{\partial e_{1,1}}{\partial b_{211}} & \frac{\partial e_{1,1}}{\partial w_{112}} & \frac{\partial e_{1,1}}{\partial w_{122}} & \frac{\partial e_{1,1}}{\partial b_{12}} \\ \frac{\partial e_{1,2}}{\partial w_{111}} & \frac{\partial e_{1,2}}{\partial w_{211}} & \frac{\partial e_{1,2}}{\partial b_{111}} & \frac{\partial e_{1,2}}{\partial b_{211}} & \frac{\partial e_{1,2}}{\partial w_{112}} & \frac{\partial e_{1,2}}{\partial w_{122}} & \frac{\partial e_{1,2}}{\partial b_{12}} \end{bmatrix} \quad (63)$$

می‌نویسیم:

$$\begin{aligned} [\mathbf{J}]_{1,1} &= S^1_{1,1} * a_{1,1} = [-0.196] * 1 = [-0.196] ; \\ [\mathbf{J}]_{1,2} &= S^1_{2,1} * a_{1,1} = [0.395] * 1 = [0.395]; \\ [\mathbf{J}]_{1,3} &= S^1_{1,1} = [-0.196] = [-0.196]; \\ [\mathbf{J}]_{1,4} &= S^1_{2,1} = [0.395] = [0.395]; \\ [\mathbf{J}]_{1,5} &= S^2_{1,1} * a_{1,1} = [-1] * [0.321] = [-0.321]; \\ [\mathbf{J}]_{1,6} &= S^2_{1,1} * a_{2,1} = [-1] * [0.368] = [-0.368] ; \\ [\mathbf{J}]_{1,7} &= S^2_{1,1} = [-1]; \end{aligned} \quad (64)$$

$$\begin{aligned} [\mathbf{J}]_{2,1} &= S^1_{1,2} * a_{1,2} = [-0.213] * 0 = [0] \\ [\mathbf{J}]_{2,2} &= S^1_{2,2} * a_{1,2} = [0.423] * 0 = [0]; \\ [\mathbf{J}]_{2,3} &= S^1_{1,2} = [-0.213]; \\ [\mathbf{J}]_{2,4} &= S^1_{2,2} = [0.423] ; \\ [\mathbf{J}]_{2,5} &= S^2_{1,1} * a_{1,2} = [-1] * [-0.3823] = [0.3823]; \\ [\mathbf{J}]_{2,6} &= S^2_{1,1} * a_{2,2} = [-1] * [0.5325] = [-0.5325]; \\ [\mathbf{J}]_{2,7} &= S^2_{1,1} = [-1]; \end{aligned} \quad (65)$$

بنابراین ماتریس ژاکوبی به صورت زیر خواهد بود:

$$J(x) = \begin{bmatrix} -0.196 & 0.395 & -0.196 & 0.395 & -0.321 & -0.368 & -1 \\ 0 & 0 & -0.213 & 0.423 & 0.3823 & -0.5325 & -1 \end{bmatrix} \quad (66)$$

## تخمین توابع مقاله 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.

```

1 % Solve an Input-Output Fitting problem with a Neural Network
2 % Script generated by Neural Fitting app
3 % Created 18-Jan-2023 11:57:05
4 %
5 % This script assumes these variables are defined:
6 %
7 % TestInputs - input data.
8 % TestTargets - target data.
9 clc
10 close all
11 clear all
12
13 %% Load Data
14 f=@(x,y) sin(x)./x .* sin(y)./y;
15
16 xmin=1;
17 xmax=6;
18 ymin=1;
19 ymax=6;
20
21 x = xmin + (xmax-xmin)*rand(26,1);
22 y = ymin + (ymax-ymin)*rand(26,1);

```

```

23
24 F=f(x,y);
25
26 TrainInputs=[x,y];
27 TrainTargets=F;
28 TrainData=[TrainInputs TrainTargets];
29
30 xx = xmin + (xmax-xmin)*rand(1000,1);
31 yy = ymin + (ymax-ymin)*rand(1000,1);
32
33 FF=f(xx,yy);
34
35 TestInputs=[xx,yy];
36 TestTargets=FF;
37 TestData=[TestInputs TestTargets];
38
39
40 x = TestInputs';
41 t = TestTargets';
42
43 % Choose a Training Function
44 % For a list of all training functions type: help nntrain
45 % 'trainlm' is usually fastest.
46 % 'trainbr' takes longer but may be better for challenging problems.
47 % 'trainscg' uses less memory. Suitable in low memory situations.
48 trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation.
49
50 % Create a Fitting Network
51 hiddenLayerSize = 10;
52 net = fitnet(hiddenLayerSize,trainFcn);
53
54 % Choose Input and Output Pre/Post-Processing Functions
55 % For a list of all processing functions type: help nnprocess
56 net.input.processFcns = {'removeconstantrows','mapminmax'};
57 net.output.processFcns = {'removeconstantrows','mapminmax'};

```

```

58
59 % Setup Division of Data for Training, Validation, Testing
60 % For a list of all data division functions type: help nndivision
61 net.divideFcn = 'dividerand'; % Divide data randomly
62 net.divideMode = 'sample'; % Divide up every sample
63 net.divideParam.trainRatio = 80/100;
64 net.divideParam.valRatio = 10/100;
65 net.divideParam.testRatio = 10/100;
66
67 % Choose a Performance Function
68 % For a list of all performance functions type: help nnperformance
69 net.performFcn = 'mse'; % Mean Squared Error
70
71 % Choose Plot Functions
72 % For a list of all plot functions type: help nnplot
73 net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
74               'plotregression', 'plotfit'};
75
76 % Train the Network
77 [net,tr] = train(net,x,t);
78
79 % Test the Network
80 y = net(x);
81 e = gsubtract(t,y);
82 performance = perform(net,t,y)
83
84 % Recalculate Training, Validation and Test Performance
85 trainTargets = t .* tr.trainMask{1};
86 valTargets = t .* tr.valMask{1};
87 testTargets = t .* tr.testMask{1};
88 trainPerformance = perform(net,trainTargets,y)
89 valPerformance = perform(net,valTargets,y)
90 testPerformance = perform(net,testTargets,y)
91
92 % View the Network

```

```

93 view(net)
94
95 % Plots
96 % Uncomment these lines to enable various plots.
97 %figure, plotperform(tr)
98 %figure, plottrainstate(tr)
99 %figure, ploterrhist(e)
100 %figure, plotregression(t,y)
101 %figure, plotfit(net,x,t)
102
103 % Deployment
104 % Change the (false) values to (true) to enable the following code blocks.
105 % See the help for each generation function for more information.
106 if (false)
107     % Generate MATLAB function for neural network for application
108     % deployment in MATLAB scripts or with MATLAB Compiler and Builder
109     % tools, or simply to examine the calculations your trained neural
110     % network performs.
111     genFunction(net,'myNeuralNetworkFunction');
112     y = myNeuralNetworkFunction(x);
113 end
114 if (false)
115     % Generate a matrix-only MATLAB function for neural network code
116     % generation with MATLAB Coder tools.
117     genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
118     y = myNeuralNetworkFunction(x);
119 end
120 if (false)
121     % Generate a Simulink diagram for simulation or deployment with.
122     % Simulink Coder tools.
123     gensim(net);
124 end
125 -----
126
127 performance =

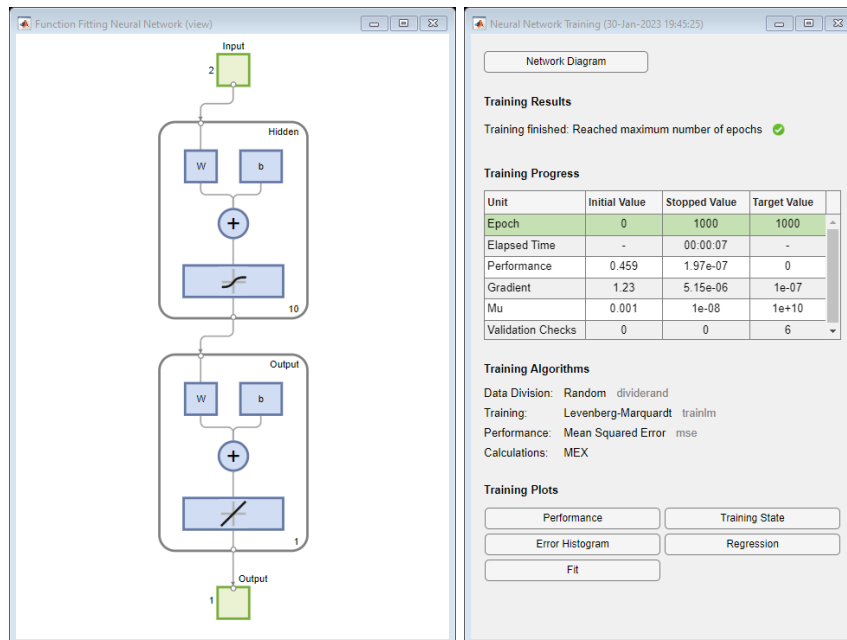
```



```

128     1.9310e-07
129
130 trainPerformance =
131     1.9719e-07
132
133 valPerformance =
134     1.8551e-07
135
136 testPerformance =
137     1.6801e-07

```



شکل ۳۷: نتیجه قسمت اول سوال اول.

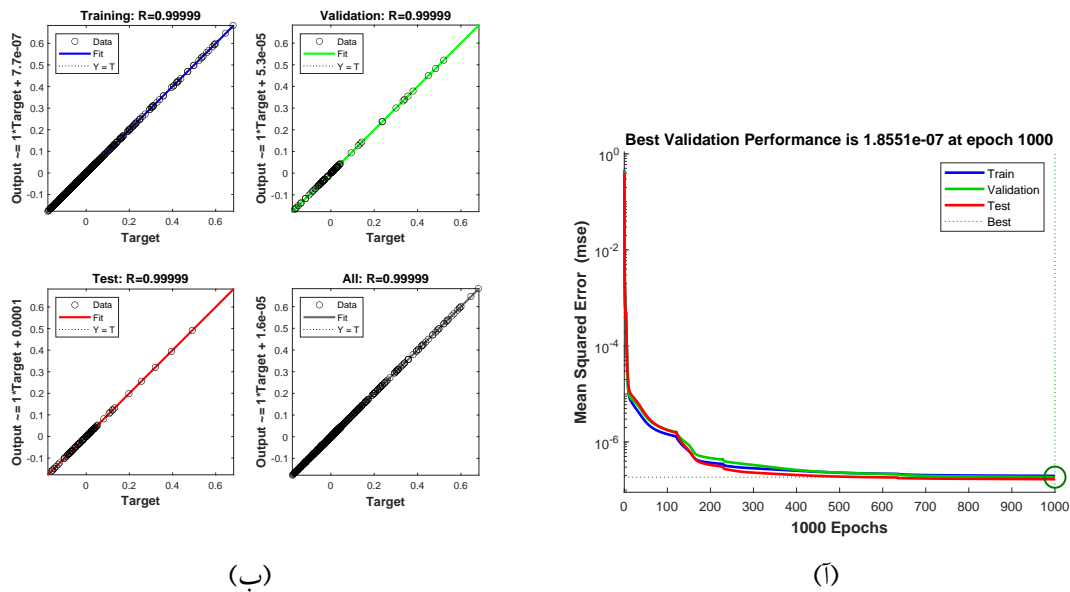
ii

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.

```

1 % Solve an Input-Output Fitting problem with a Neural Network
2 % Script generated by Neural Fitting app
3 % Created 18-Jan-2023 11:57:05
4 %

```



شکل ۳۸: نتیجه قسمت اول سوال اول.

```

5 % This script assumes these variables are defined:
6 %
7 % TestInputs - input data.
8 % TestTargets - target data.
9 clc
10 close all
11 clear all
12
13 %% Load Data
14 f=@(x,y,z) (1+x.^0.5+y.^-1+z.^(-1.5)).^2;
15
16 xmin=1;
17 xmax=6;
18 ymin=1;
19 ymax=6;
20 zmin=1;
21 zmax=6;
22 x = xmin + (xmax-xmin)*rand(26,1);
23 y = ymin + (ymax-ymin)*rand(26,1);
24 z = zmin + (zmax-zmin)*rand(26,1);

```

```
25 F=f(x,y,z);
26
27 TrainInputs=[x,y,z];
28 TrainTargets=F;
29 TrainData=[TrainInputs TrainTargets];
30
31 xx = xmin + (xmax-xmin)*rand(101,1);
32 yy = ymin + (ymax-ymin)*rand(101,1);
33 zz = zmin + (zmax-zmin)*rand(101,1);
34 FF=f(xx,yy,zz);
35
36 TestInputs=[xx,yy,zz];
37 TestTargets=FF;
38 TestData=[TestInputs TestTargets];
39
40
41 x = TestInputs';
42 t = TestTargets';
43
44 % Choose a Training Function
45 % For a list of all training functions type: help nntrain
46 % 'trainlm' is usually fastest.
47 % 'trainbr' takes longer but may be better for challenging problems.
48 % 'trainscg' uses less memory. Suitable in low memory situations.
49 trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation.
50
51 % Create a Fitting Network
52 hiddenLayerSize = 20;
53 net = fitnet(hiddenLayerSize,trainFcn);
54
55 % Choose Input and Output Pre/Post-Processing Functions
56 % For a list of all processing functions type: help nnprocess
57 net.input.processFcns = {'removeconstantrows','mapminmax'};
58 net.output.processFcns = {'removeconstantrows','mapminmax'};
59
```

```
60 % Setup Division of Data for Training, Validation, Testing
61 % For a list of all data division functions type: help nndivision
62 net.divideFcn = 'dividerand'; % Divide data randomly
63 net.divideMode = 'sample'; % Divide up every sample
64 net.divideParam.trainRatio = 80/100;
65 net.divideParam.valRatio = 10/100;
66 net.divideParam.testRatio = 10/100;
67
68 % Choose a Performance Function
69 % For a list of all performance functions type: help nnperformance
70 net.performFcn = 'mse'; % Mean Squared Error
71
72 % Choose Plot Functions
73 % For a list of all plot functions type: help nnplot
74 net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
75     'plotregression', 'plotfit'};
76
77 % Train the Network
78 [net,tr] = train(net,x,t);
79
80 % Test the Network
81 y = net(x);
82 e = gsubtract(t,y);
83 performance = perform(net,t,y)
84
85 % Recalculate Training, Validation and Test Performance
86 trainTargets = t .* tr.trainMask{1};
87 valTargets = t .* tr.valMask{1};
88 testTargets = t .* tr.testMask{1};
89 trainPerformance = perform(net,trainTargets,y)
90 valPerformance = perform(net,valTargets,y)
91 testPerformance = perform(net,testTargets,y)
92
93 % View the Network
94 view(net)
```

```

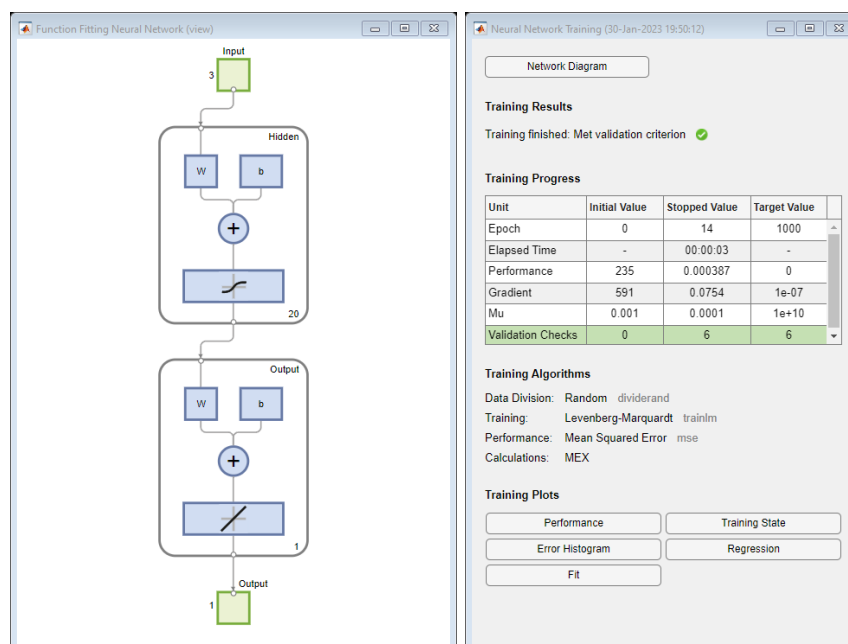
95
96 % Plots
97 % Uncomment these lines to enable various plots.
98 %figure, plotperform(tr)
99 %figure, plottrainstate(tr)
100 %figure, ploterrhist(e)
101 %figure, plotregression(t,y)
102 %figure, plotfit(net,x,t)
103
104 % Deployment
105 % Change the (false) values to (true) to enable the following code blocks.
106 % See the help for each generation function for more information.
107 if (false)
108     % Generate MATLAB function for neural network for application
109     % deployment in MATLAB scripts or with MATLAB Compiler and Builder
110     % tools, or simply to examine the calculations your trained neural
111     % network performs.
112     genFunction(net,'myNeuralNetworkFunction');
113     y = myNeuralNetworkFunction(x);
114 end
115 if (false)
116     % Generate a matrix-only MATLAB function for neural network code
117     % generation with MATLAB Coder tools.
118     genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
119     y = myNeuralNetworkFunction(x);
120 end
121 if (false)
122     % Generate a Simulink diagram for simulation or deployment with.
123     % Simulink Coder tools.
124     gensim(net);
125 end
126 -----
127
128 performance =
129     0.0529

```

```

130
131 trainPerformance =
132     0.0630
133
134 valPerformance =
135     0.0079
136
137 testPerformance =
138     0.0168

```



شکل ۳۹: نتیجه قسمت دوم سوال اول.

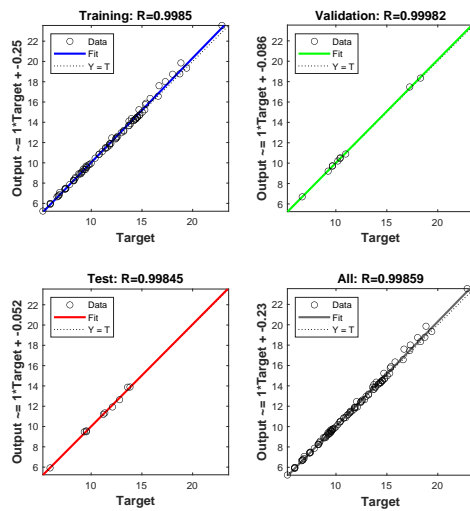
iii

برای این مثال از محیط سیمولینک آورده شده در شکل ۴۱ و دستورات زیر استفاده می کنیم.

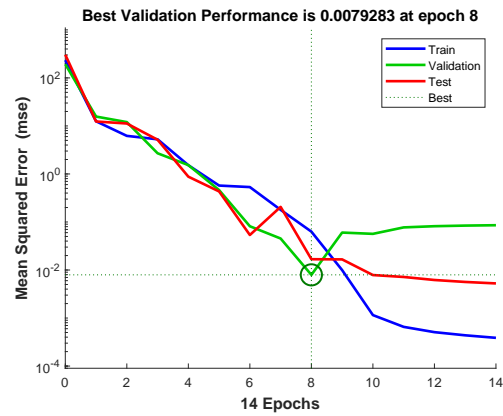
```

1 clc;
2 clear all;
3 close all;
4
5 for k=1:700
6     if k<=500

```



(ب)



(آ)

شکل ۴۰: نتیجه قسمت دوم سوال اول.

```

7      o=@(x) sin((2*pi*x)/250);
8
9      else
10         o=@(x) 0.5*sin((2*pi*x)/250) + 0.5*sin((2*pi*x)/25);
11     end
12     u(k)=o(k);
13
14 k=1:700;
15
16 figure
17 plot(k,u,LineWidth=2)
18 xlabel("time")
19
20 o=@(y) 0.6*sin(pi*y)+0.3*sin(3*pi*y)+0.1*sin(5*pi*y);
21 f=o(u);
22
23 figure
24 plot(k,f,LineWidth=2)
25 xlabel("time")

```

```

27 train_data=k';
28 train_target=f';
29 net = feedforwardnet(30);
30
31 [trainedNet,tr] = train(net,train_data',train_target')
32
33 kk=k(1:500);
34
35 test_data=kk';
36
37 for i=1:500
38 NNsOutput(i)=sim(trainedNet,test_data(i,:));
39 end
40
41 ff=f(1:500);
42 plot(1:500,ff,'r',1:500,NNsOutput,'b',LineWidth=2)
43 legend('Data','MLP Output')
44
45
46 time=kk;
47 F=NNsOutput;
48
49 out=sim('anf33.slx');
50
51 yk=out.Output;
52 yk=yk.Data;
53
54 yhk=out.Output1;
55 yhk=yhk.Data;
56
57 figure
58 plot(yhk,'b',LineWidth=2)
59 hold on
60 plot(yk,'r--',LineWidth=2)
61 legend("yhk","yk")

```



```

62 axis([0 500 -6 6])clc;
63 clear all;
64 close all;
65
66 for k=1:700
67     if k<=500
68         o=@(x) sin((2*pi*x)/250);
69     else
70         o=@(x) 0.5*sin((2*pi*x)/250) + 0.5*sin((2*pi*x)/25);
71     end
72     u(k)=o(k);
73 end
74
75 k=1:700;
76
77 figure
78 plot(k,u,LineWidth=2)
79 xlabel("time")
80
81 o=@(y) 0.6*sin(pi*y)+0.3*sin(3*pi*y)+0.1*sin(5*pi*y);
82 f=o(u);
83
84 figure
85 plot(k,f,LineWidth=2)
86 xlabel("time")
87
88 train_data=k';
89 train_target=f';
90 net = feedforwardnet(30);
91
92 [trainedNet,tr] = train(net,train_data',train_target')
93
94 kk=k(1:500);
95
96 test_data=kk';

```

```

97
98 for i=1:500
99     NNsOutput(i)=sim(trainedNet,test_data(i,:));
100 end
101
102 ff=f(1:500);
103 plot(1:500,ff,'r',1:500,NNsOutput,'b',LineWidth=2)
104 legend('Data','MLP Output')
105
106
107 time=kk;
108 F=NNsOutput;
109
110 out=sim('anf33.slx');
111
112 yk=out.Output;
113 yk=yk.Data;
114
115 yhk=out.Output1;
116 yhk=yhk.Data;
117
118 figure
119 plot(yhk,'b',LineWidth=2)
120 hold on
121 plot(yk,'r--',LineWidth=2)
122 legend("yhk","yk")
123 axis([0 500 -6 6])

```

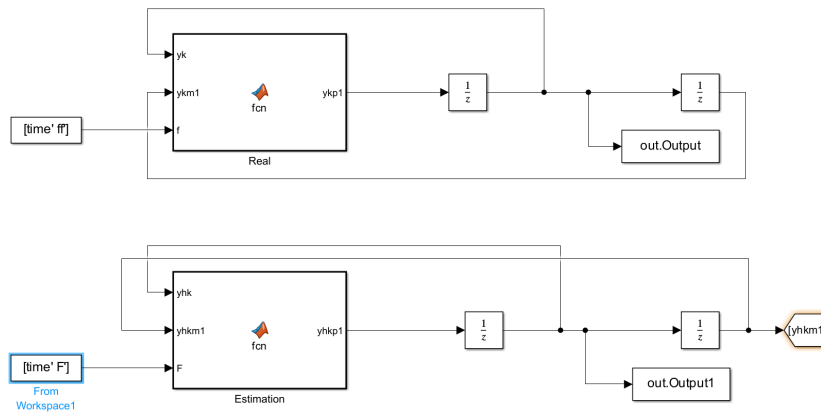
iv

برای این مثال از محیط سیمولینک آورده شده در شکل ۴۴ و دستورات زیر استفاده می کنیم.

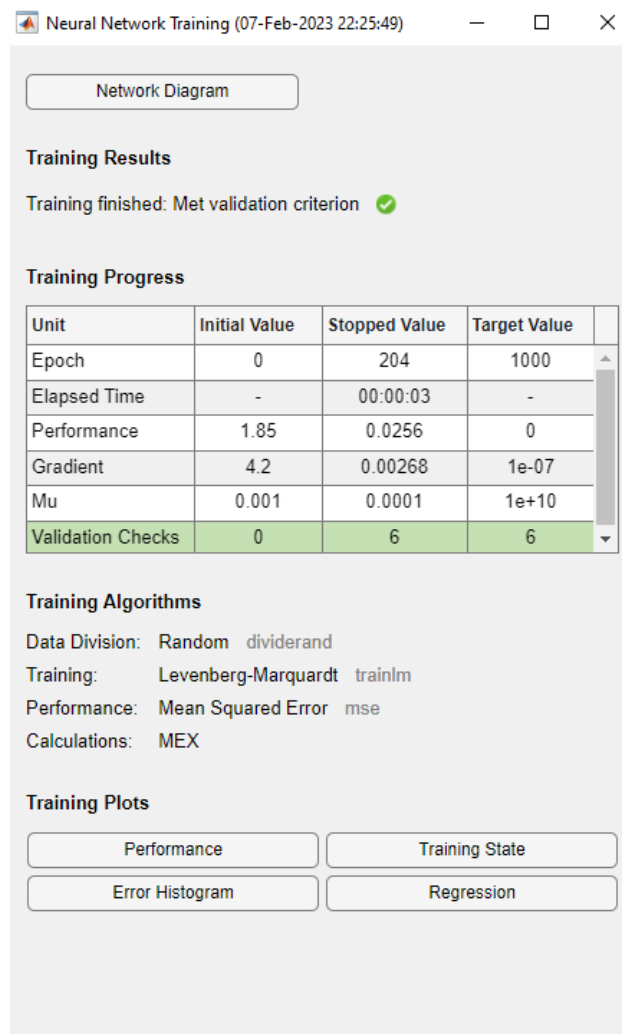
```

1 clc;
2 clear all;
3 close all;

```



شکل ۴۱: سیمولینک.

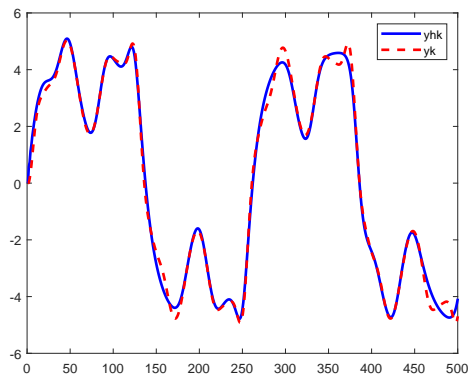


شکل ۴۲: نتیجه قسمت سوم سوال اول.

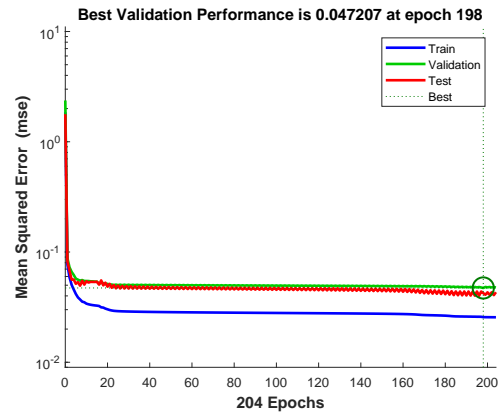
```

4
5 out=sim('MLPQ4.slx');
6

```



(ب)



(I)

شکل ۴۳: نتیجه قسمت سوم سوال اول.

```

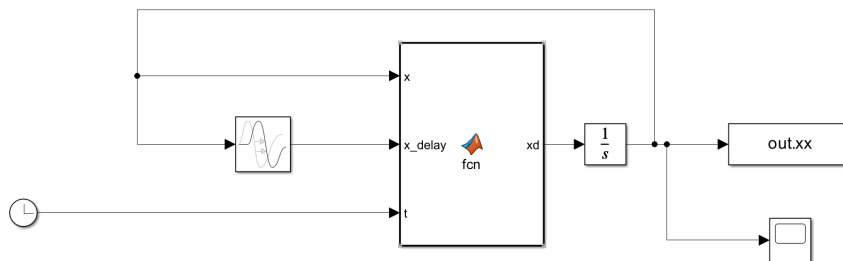
7 x=out.xx;
8 x=x.Data;
9
10 x_start = 117 ;
11 x_end = 1118 ;
12
13 x=[x(x_start-18:x_end-18,1),x(x_start-12:x_end-12,1),x(x_start-6:x_end-6,1),x(
    x_start:x_end,1),x(x_start+6:x_end+6,1)];
14
15 x_train=x(1:500,1:4);
16 y_train=x(1:500,5);
17
18 net = feedforwardnet(30);
19
20 [trainedNet,tr] = train(net,x_train',y_train')
21
22
23 x_test=x(501:1000,1:4);
24 y_test=x(501:1000,5);
25
26
27 for i=1:500
28 NNSOutput(i)=sim(trainedNet,x_test(i,:)');

```

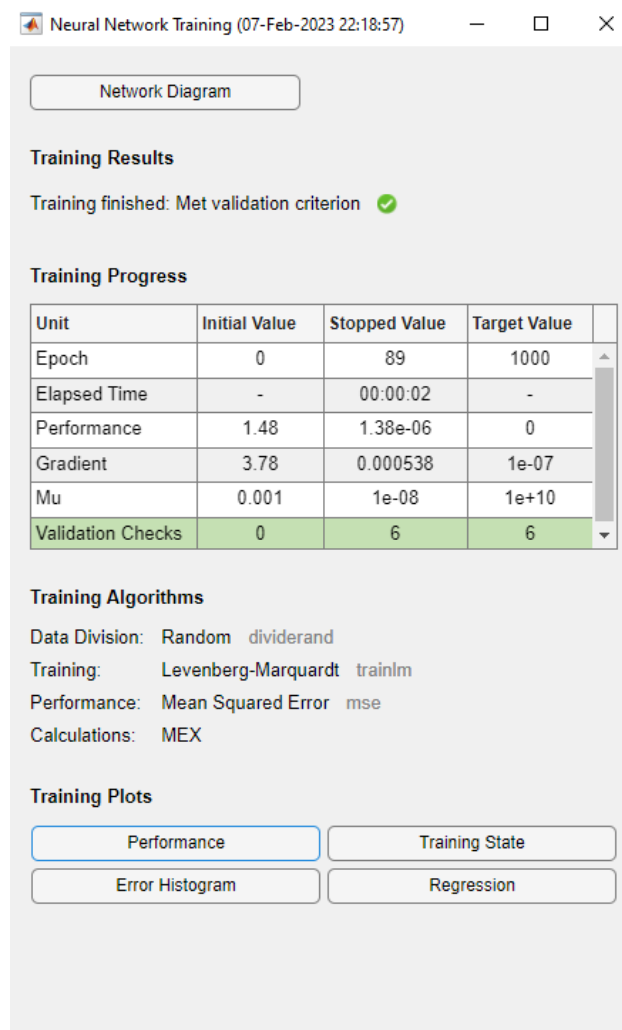
```

29 end
30
31 figure
32 plot(1:500,NNsOutput,'k',LineWidth=1.2)
33 hold on
34 plot(1:500,y_test,'r--',LineWidth=2)
35 legend('Data','MLP Output')

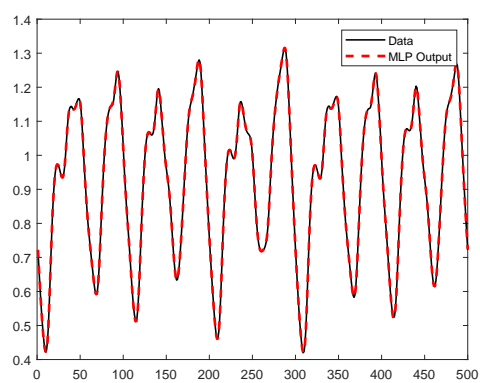
```



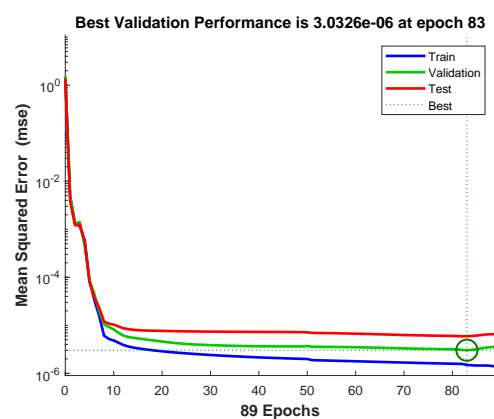
شکل ۴۴: سیمولینک.



شکل ۴۵: نتیجه قسمت چهارم سوال اول.



(ب)



(ا)

شکل ۴۶: نتیجه قسمت چهارم سوال اول.

جدول ۱: پارامترهای سیستم

۲ kg	M
۰/۸ kg	m
۰/۰۰۵ kg/s	$b_x$
۰/۰۰۰۵ kgm <sup>۲</sup> /s	$\theta_x$
۰/۲۵ m	l

## طراحی کنترل کننده Fuzzy PID

### راه حل اول

از پارامترهای آورده شده در جدول ۱ برای سیستم خود استفاده می کنیم. معادله لاگرانژ به صورت زیر خواهد بود:

$$L = T - V \quad (۶۷)$$

$$= \frac{1}{2} M v^2 + \frac{1}{2} m v^2 - mgl \cos \theta$$

که در آن:

$$v = \dot{x}$$

$$v^2 = \left( \frac{d}{dt} (x - l \sin \theta) \right)^2 + \left( \frac{d}{dt} (l \cos \theta) \right)^2 \quad (۶۸)$$

$$= \dot{x}^2 - 2l\dot{x}\dot{\theta} \cos \theta + l^2 \dot{\theta}^2$$

حال می توانیم بنویسیم:

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

از تعریف معادلات لاگرانژ داریم:

$$\frac{d}{dt} \left( \frac{dL}{d\dot{x}} \right) - \frac{dL}{dx} = F - b_x \dot{\theta} \rightarrow (۱) \quad (۷۰)$$

$$\frac{d}{dt} \left( \frac{dL}{d\dot{\theta}} \right) - \frac{dL}{d\theta} = b_\theta \dot{\theta} \rightarrow (۲)$$

که در آن  $b_x$  و  $b_\theta$  ضرایب اصطکاک کارت و پاندول هستند. رابطه ۷۰ را می توان به صورت زیر بازنویسی کرد:

$$(M + m)\ddot{x} - ml\ddot{\theta} \cos \theta + ml\dot{\theta}^2 \sin \theta = F - b_x \dot{x} \rightarrow (۳) \quad (۷۱)$$

$$-\ddot{x} \cos \theta + l\ddot{\theta} - g \sin \theta = -b_\theta \dot{\theta} \rightarrow (۴)$$

معادلات آورده شده در رابطه ۷۱ می تواند با  $\ddot{x}$  و  $\ddot{\theta}$  بیان شود تا در حالت ODE45 مورد استفاده قرار گیرد. از معادله (۴) در رابطه ۷۱ می توانیم بنویسیم:  $\ddot{x} = \frac{1}{\cos \theta} (l\ddot{\theta} - g \sin \theta + b_\theta \dot{\theta})$  و اگر آن را جابگزین معادله (۳) در رابطه ۷۱ کنیم می توانیم بنویسیم:

$$\ddot{\theta} = \frac{1}{\frac{(M+m)l}{\cos \theta} - ml \cos \theta} \left\{ (M + m)g \tan \theta - \frac{(M + m)}{\cos \theta} b_\theta \dot{\theta} - ml\dot{\theta}^2 \sin \theta + F - b_x \dot{x} \right\} \quad (۷۲)$$

در ادامه، از معادله (۴) در رابطه ۷۱ می نویسیم:  $\ddot{\theta} = \frac{1}{l} (\ddot{x} \cos \theta + g \sin \theta - b_\theta \dot{\theta})$  و اگر آن را جابگزین معادله (۳) در رابطه ۷۱ کنیم می توانیم بنویسیم:

$$\ddot{x} = \frac{1}{M + m - m \cos^2 \theta} \left( mg \sin \theta \cos \theta - mb_\theta \dot{\theta} \cos \theta - ml\dot{\theta}^2 \sin \theta + F - b_x \dot{x} \right) \quad (۷۳)$$

بنابراین حالات در ODE45 به صورت زیر بیان می شوند:

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T$$

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{1}{M + m - m \cos^2 x_3} \left( mg \sin x_3 \cos x_3 - mb_\theta \cos x_3 \cdot x_4 - mlx_4^2 \sin x_3 + F - b_x x_2 \right) \quad (۷۴)$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = \frac{1}{\frac{M+m}{\cos x_3} l - ml \cos x_3} \left\{ (M + m)g \tan x_3 - \frac{M + m}{\cos x_3} b_\theta x_4 - mlx_4^2 \sin x_3 + F - b_x x_2 \right\}$$



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

```

1 function dX = diff_pendulum(~, X)
2 global M m bx bq l g J F
3 dX = zeros(4,1);
4
5 % Lagrangian equation
6 dX(1) = X(2);
7 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));
8 dX(3) = X(4);
9 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));
10
11 % Text book equation : Not good
12 % dX(1) = X(2);
13 % 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));
14 % dX(3) = X(4);
15 % 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)
      );
16
17 end
18 -----
19 function fuzzy = fuzzification(e, de)
20 % Get the normalized fuzzy output value from error, error_dot
21 % input(sigma) : e(error), de(delta error)
22 % output(fuzzy) : mu(e), mu(de) : normalized fuzzy value
23 % Fuzzy layer configuration : NB, NS, ZO, PS, PB
24 % 2017.12.07 Hyosung Hong
25
26 fuzzy = zeros(5,2); % Column: [NB, NS, ZO, PS, PB]' , Row: [mu(e), mu(de)]
27
28 for i=1:2

```

```

29     if i==1
30         sigma = e;
31     else
32         sigma = de;
33     end
34
35     if sigma < -2/3
36         fuzzy(1,i) = 1;
37     elseif sigma >= -2/3 && sigma < -1/3
38         fuzzy(1,i) = -3*sigma - 1;
39         fuzzy(2,i) = 3*sigma + 2;
40     elseif sigma >= -1/3 && sigma < 0
41         fuzzy(2,i) = -3*sigma;
42         fuzzy(3,i) = 3*sigma + 1;
43     elseif sigma >= 0 && sigma < 1/3
44         fuzzy(3,i) = -3*sigma + 1;
45         fuzzy(4,i) = 3*sigma;
46     elseif sigma >= 1/3 && sigma < 2/3
47         fuzzy(4,i) = -3*sigma + 2;
48         fuzzy(5,i) = 3*sigma - 1;
49     else
50         fuzzy(5,i) = 1;
51     end
52 end
53
54 end
55 -----
56
57 function u_tilde = defuzzification(u_indx)
58 % defuzzification to generate output u_tilde(normalized value) from fuzzy
59 % inference output u_indx
60 % input(u_indx) : fuzzy set output
61 % output(fuzzy) : normalized u_tilde
62 % Fuzzy layer configuration : NB(1), NS(2), ZO(3), PS(4), PB(5)

```

```

63 u_range = linspace(-1, 1, 100);
64 if isempty(u_indx) ~= 1      % if u_indx contains some data
65     u_candidate = zeros(size(u_indx,1), size(u_range,2));
66
67     for i=1:size(u_indx, 1)
68         switch u_indx(i,1)
69             case 1
70                 for j=1:size(u_range,2)
71                     if u_range(j) < -2/3
72                         u_candidate(i,j) = min(1, u_indx(i,2));
73                     elseif u_range(j) > -2/3 && u_range(j) < -1/3
74                         u_candidate(i,j) = min(-3*u_range(j) - 1, u_indx(i,2));
75                     end
76                 end
77
78             case 2
79                 for j=1:size(u_range,2)
80                     if u_range(j) >= -2/3 && u_range(j) < -1/3
81                         u_candidate(i,j) = min( 3*u_range(j) + 2, u_indx(i,2));
82                     elseif u_range(j) > -1/3 && u_range(j) < 0
83                         u_candidate(i,j) = min(-3*u_range(j)      , u_indx(i,2));
84                     end
85                 end
86
87             case 3
88                 for j=1:size(u_range,2)
89                     if u_range(j) >= -1/3 && u_range(j) < 0
90                         u_candidate(i,j) = min( 3*u_range(j) + 1, u_indx(i,2));
91                     elseif u_range(j) > 0 && u_range(j) < 1/3
92                         u_candidate(i,j) = min(-3*u_range(j) + 1, u_indx(i,2));
93                     end
94                 end
95
96             case 4
97                 for j=1:size(u_range,2)

```

```

98         if u_range(j) >= 0 && u_range(j) < 1/3
99             u_candidate(i,j) = min( 3*u_range(j)      , u_indx(i,2));
100         elseif u_range(j) > 1/3 && u_range(j) < 2/3
101             u_candidate(i,j) = min(-3*u_range(j) + 2, u_indx(i,2));
102         end
103     end
104
105     case 5
106         for j=1:size(u_range,2)
107             if u_range(j) >= 1/3 && u_range(j) < 2/3
108                 u_candidate(i,j) = min( 3*u_range(j) - 1, u_indx(i,2));
109             elseif u_range(j) > 2/3
110                 u_candidate(i,j) = min(1, u_indx(i,2));
111             end
112         end
113
114     otherwise
115         u_candidate(i,:) = 0;
116     end
117 end
118 else
119     u_candidate = zeros(1,size(u_range,2));
120 end
121
122 u_union = max(u_candidate, [], 1);
123
124 u_tilde = sum(u_range*u_union')/sum(u_union);
125
126 end
127 -----
128
129 function u_indx = fuzzy_inference(fuzzy)
130 % fuzzy inference to generate fuzzy set and corresponding output u set
131 % input(fuzzy) : normalized fuzzy value mu(e), mu(de)
132 % output(u_indx) : fuzzy set output

```

```

133 % Fuzzy layer configuration : NB(1), NS(2), ZO(3), PS(4), PB(5)
134
135 fuzzy_rule = [1 1 1 2 1;
136               1 1 2 5 4;
137               1 1 3 5 5;
138               4 1 4 5 5;
139               5 4 5 5 5];
140 fuzzy_indx = [];
141
142 for i=1:5
143     if fuzzy(i,1) > 0
144         for j=1:5
145             if fuzzy(j,2) > 0
146                 if fuzzy_rule(i,j) > 0      % only use the defined fuzzy rules
147                     fuzzy_indx = [fuzzy_indx; [i,j]];    % example: fuzzy_indx
148                     = [NS NS; ZO NB; ZO NS] to determine u_indx
149                 end
150             end
151         end
152     end
153
154 if isempty(fuzzy_indx) == 1
155     u_indx = [];
156 else
157     u_indx = zeros(size(fuzzy_indx));
158
159     for i=1:size(u_indx,1)
160         u_indx(i,:) = [fuzzy_rule(fuzzy_indx(i,1), fuzzy_indx(i,2)), min(fuzzy(
161             fuzzy_indx(i,1), 1), fuzzy(fuzzy_indx(i,2), 2))];
162         % example: u_indx = [NS, 0.2; ZO, 0.2; ZO, 0.7];
163     end
164 end
165 end

```

برای اجرا هم از دستورات زیر استفاده می‌کنیم:

```

1 % Cart and Pole Control
2 % Fuzzy control
3
4 close all; clear;
5 addpath ./sub_functions
6 global M m bx bq l g J F
7 %% pendulum parameters
8 M = 2;           % [kg]
9 m = 0.8;         % [kg]
10 bx = 0.005;      % [kg/s]
11 bq = 0.0005;     % [kg m^2/s]
12 l = 0.25;        % [m]
13 g = 9.81;        % [m/s^2]
14 J = 0.0326;      % [kg m]
15 F = 0;           % [N]
16 T_final = 25;    % [s]
17 Ts = 0.01;       % control step time [s]
18 video_flag = 1; % choose 1 if you want to save a video file of plot animation
19
20 %% initial condition
21 X = [-0.5; 0; 15*pi/180; 0];
22 xd = 0;
23 xd_dot = 0;
24 qd = 0;
25 qd_dot = 0;
26
27 %% Fuzzy Control
28 % Fuzzy data normalization
29 x_normal = 12;    % [m]
30 dx_normal = 1.5;  % [m/s]
31 q_normal = 360*pi/180; % [rad]
32 dq_normal = 180*pi/180; % [rad/s]
33 u_normal = 1000;  % [N]
34

```

```

35 % data save
36 X_Fuzzy = zeros(T_final/Ts+1,4);
37 time_Fuzzy = zeros(T_final/Ts+1,1);
38 F_save = zeros(T_final/Ts+1,1);
39 u_save = zeros(T_final/Ts+1,2);
40 X_Fuzzy(1,:) = X';
41 count = 2;
42 x_prev = X(1);
43 q_prev = X(3);
44 for time=Ts:Ts:T_final
45
46     ex = xd - X(1);           % error of cart position
47     dex = x_prev - X(1);      % error difference of cart position
48
49     eq = qd - X(3);           % error of pendulum angle
50     deq = q_prev - X(3);      % error difference of pendulum angle
51
52     % Fuzzy control (u_x: cart position, u_q: pendulum angle)
53     u_x = defuzzification(fuzzy_inference(fuzzification(ex/x_normal, dex/
dx_normal)))) * u_normal*1;
54     u_q = defuzzification(fuzzy_inference(fuzzification(eq/q_normal, deq/
dq_normal)))) * u_normal*2;
55     u_save(count,:) = [u_x, u_q];
56
57     F = -u_x + u_q;
58
59     x_prev = X(1);
60     q_prev = X(3);
61
62     [T, X_next] = ode45(@diff_pendulum, [0, Ts], X);
63     X = X_next(end,:);
64     X_Fuzzy(count,:) = X';
65     time_Fuzzy(count) = time;
66     F_save(count) = F;
67     count = count + 1;

```

```

68 end
69
70
71 %% Free fall (No control)
72 [T, Z_ode] = ode45(@diff_pendulum, [0, T_final], X);
73
74 %% Make video
75 if video_flag == 1
76     % Video file open
77     makeVideo = VideoWriter('Fuzzy');
78     % Frame Rate - Frames per second
79     makeVideo.FrameRate = 100;
80     % Quality - ۰.۰۰۰۰۰۰۰۰۰۰۰۰ (0 ~ 100)
81     makeVideo.Quality = 80;
82     open(makeVideo);
83 end
84
85 %% Plot
86 X_result = X_Fuzzy;
87 Time_result = time_Fuzzy;
88
89 figure(1)
90 axis_limit = 1;
91 for i=1:size(X_result,1)
92     cart_position_x = X_result(i,1);
93     pend_position_x = X_result(i,1) - l*sin(X_result(i,3));
94     pend_position_y = l*cos(X_result(i,3));
95
96     hold off
97     plot(pend_position_x, pend_position_y, 'ok', 'MarkerSize', 25, '
MarkerFaceColor',[0.2 0.9 0.2])    % pendulum
98     hold on
99     plot(cart_position_x, 0, 'sk', 'MarkerSize', 50, 'MarkerFaceColor',[0.8 0.8
0.8])    % cart
100     hold on

```



```

101     if cart_position_x > pend_position_x
102         plot(linspace(cart_position_x,pend_position_x,2), linspace(0,
pend_position_y,2), 'k', 'LineWidth', 3)    % rod
103     else
104         plot(linspace(cart_position_x,pend_position_x,2), linspace(0,
pend_position_y,2), 'r', 'LineWidth', 3)    % rod
105     end
106     grid on
107     xlabel('[m]')
108     ylabel('[m]')
109     axis([-axis_limit axis_limit -0.5 0.5])
110     sim_status = sprintf('Simulation time: %5.2f s',Time_result(i));
111     title(sim_status,'FontSize',13)
112
113     if video_flag == 1 && rem(Time_result(i),0.01) == 0
114         frame = getframe(gcf);
115         writeVideo(makeVideo,frame);
116     end
117
118     pause(Ts);
119 end
120
121 if video_flag == 1
122     close(makeVideo);
123 end
124
125 figure(2)
126 subplot(3,1,1)
127 plot(Time_result, X_result(:,1:2))
128 grid on
129 xlabel('Time [s]')
130 legend('cart position [m]', 'cart velocity [m/s]')
131
132 subplot(3,1,2)
133 plot(Time_result, X_result(:,3:4)*180/pi)

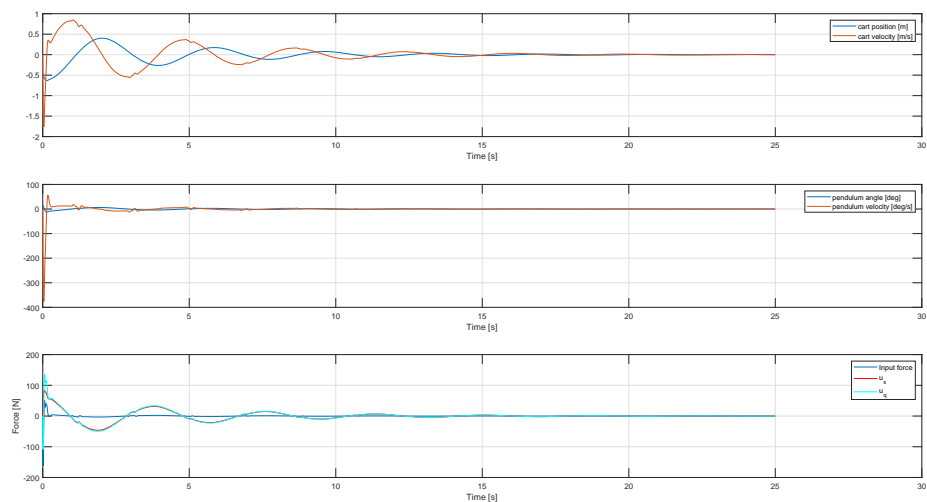
```

```

134 grid on
135 xlabel('Time [s]')
136 legend('pendulum angle [deg]', 'pendulum velocity [deg/s]')
137
138 subplot(3,1,3)
139 plot(Time_result, F_save)
140 hold on
141 plot(Time_result, u_save(:,1), 'r')
142 plot(Time_result, u_save(:,2), 'c')
143 grid on
144 xlabel('Time [s]')
145 ylabel('Force [N]')
146 legend('Input force', 'u_x', 'u_q')

```

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



شکل ۴۷: نتیجه راه حل اول سوال طراحی.

## راه حل دوم

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

تعیین مشخصه‌های کنترلی سیستم است.

$$G_c(s) = k_p + \frac{k_i}{s} + k_d \quad (75)$$

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

$$K_p \left( 1 + \frac{1}{T_i s} + T_d s \right) \quad (76)$$

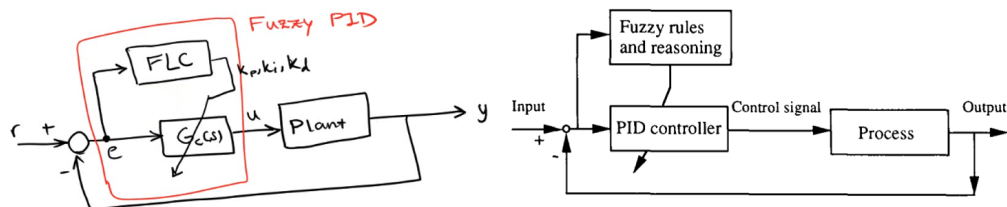
$$k_i = \frac{K_p}{T_i}, \quad k_d = K_p T_d$$

تنظیم ضرایب کنترلر PID راه حل قطعی ندارد و کار ساده‌ای نیست. هرچند برخی قواعد برای ساده‌تر شدن انتخاب ضرایب وجود دارند که از جمله آن‌ها می‌توان به قواعد زیگلر-نیکولز اشاره کرد (رابطه ۷۷) که در آن  $K_p, T_u$  با ایجاد حالت نوسانی در سیستم و به صورت بهره‌ی سیستم نوسانی و دوره‌ی زمانی نوسان تعیین می‌شوند (در مرز پایداری و ناپایداری).

Type Control	$K_p$	$K_i$	$K_d$
$P$	$0.5 K_u$	—	—
$PI$	$0.45 K_u$	$1/2 K_p / T_u$	—
$PD$	$0.7 K_u$	—	$K_p T_u / 8$
PID classic	$0.6 K_u$	$2 K_p / T_u$	$K_p T_u / 8$

(77)

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 در نظر گرفتن خطا و مشتق آن ضرایب کنترلر را تنظیم می‌کند. قبل از آغاز طراحی، خروجی‌ها و ضرایب را به صورتی که در رابطه ۷۸

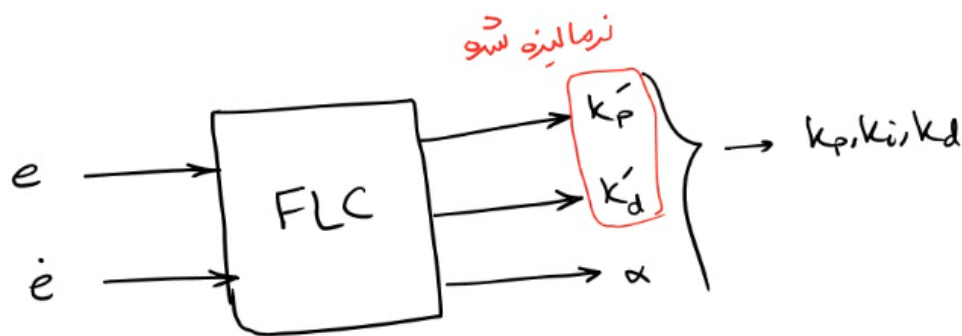


شکل ۴۸: طرح کلی کنترل مدنظر ما.

آورده شده است در نظر می گیریم.

$$k_p, k_i, k_d \rightsquigarrow k_p, k_d, \alpha \quad \alpha = \frac{T_i}{T_d}, k_i = \frac{k_p^2}{\alpha k_d} \quad (78)$$

حال ضرایب را به صورتی که در تصویر نشان داده شده نرمال سازی می کنیم (مستقل از سیستم و بهره آن). هم چنین می دانیم  $\alpha$  عددی بین ۲ تا ۵ است که در روش زیگلر-نیکولز همواره برابر با ۴ است. اما ما در روش خود می خواهیم آن را محدود به یک نقطه خاص نکنیم و آن را متغیر در نظر بگیریم (زیگلر-نیکولز تطبیقی). بنابراین هسته کلی سیستم مدنظر ما به صورتی خواهد بود که در شکل ۴۹ آورده شده است. حال



شکل ۴۹: هسته 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 آورده شده در شکل ۵۰ استفاده می کنیم. شکل کلی قواعد بر اساس خطا و تغییرات آن به صورت

## 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: مقاله مورد توجه برای ایده تنظیم قواعد.

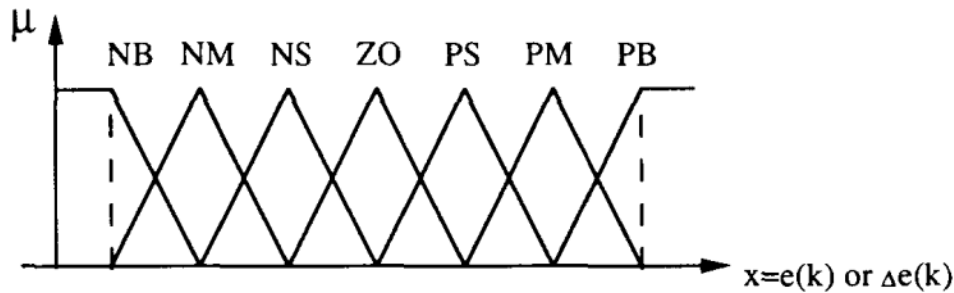
زیر تعیین می شود.

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

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

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

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

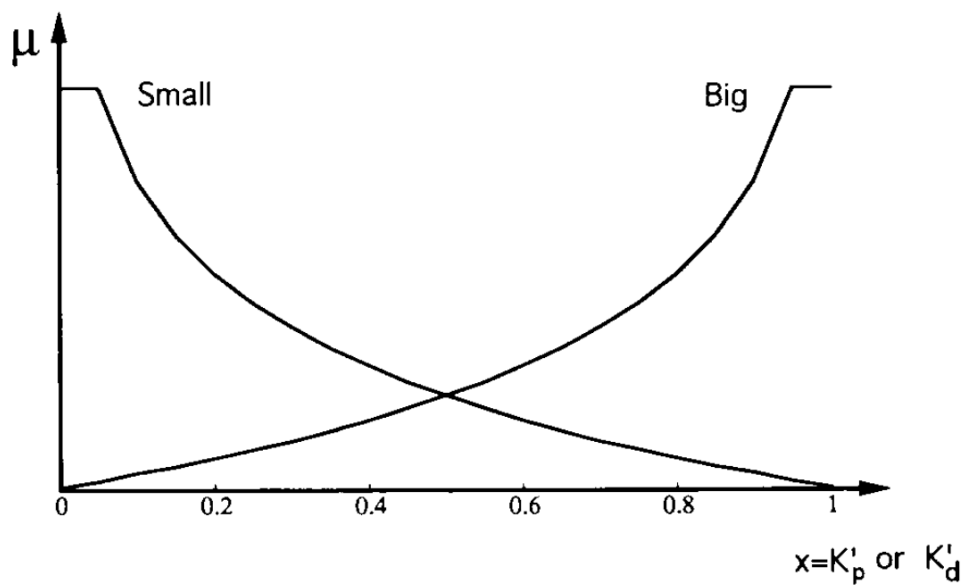


شکل ۵۱: تقسیمات کلی فازی خطا و مشتق آن.

رابطه ۷۹ و شکل ۵۲ آورده شده است. البته مقاله آورده شده در شکل ۵۰ به این نکته توجه نکرده که در جایی که  $x$  صفر شود مقدار تابع عضویت برابر با بی نهایت می شود و در واقع رابطه درست و صحیح به صورتی است که در صفحه ۱۴۳ نوشته شده. اعداد ۲ تا ۵ هم به صورت سینگلتون فازی اضافه می شوند.

$$\begin{aligned} \mu_{\text{Small}}(x) &= \min\left(-\frac{1}{4} \ln x, 1\right) \\ \mu_{\text{Big}}(x) &= \min\left(-\frac{1}{4} \ln(1-x), 1\right) \end{aligned} \quad (79)$$

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



شکل ۵۲: توابع عضویت بزرگ و کوچک.

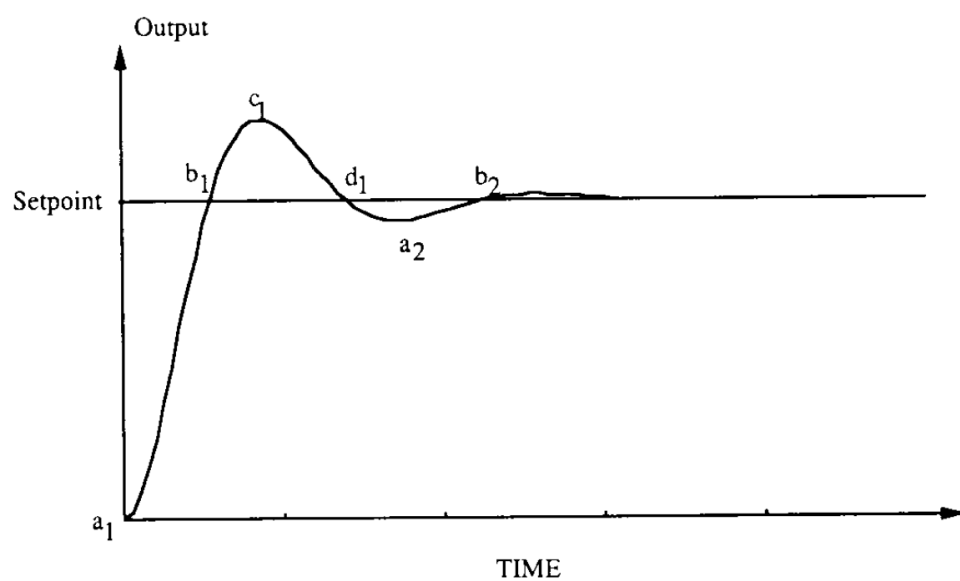
Rule i:

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

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

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

به صورت کلی توابع و قوانین فازی را بر اساس منطق کنترلی؟؟ تعیین می کنیم. مثلاً در شروع فاصله تا مرجع زیاد است و باید بهره های تناسبی و انتگرالی بالا باشد. چون مشتق خطا عدد بزرگی خواهد بود ضریب مشتقی را کم در نظر می گیریم. حال وقتی به نقطه هدف رسیدیم باید کاری کنیم سیستم تا جای ممکن در محل مطلوب خود بماند. بنابراین در این جا بهره های تناسبی و انتگرالی را کم می کنیم. و برای تغییرات کم خطا و استفاده از خواص خوب مشتق گیر، بهره مشتقی را زیاد در نظر می گیریم. در ادامه و جایی که اورشوت داریم هم همانند نقطه اول و الی آخر. در ضرایبی که ما تعیین کردیم رفتار  $k_i$  برخلاف رفتار  $\alpha$  است. حال برای تعیین قوانین در حالت کلی مطابق صفحه ۱۴۵، رابطه ۸۱ و رابطه ۸۲ پیش می رویم. 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 وقوانین فازی.

توضیحات آورده شده برای شکل ۵۳ گفته شده نوشته شده اند.

	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 هسته اصلی مطرح شده در شکل ۴۹ و رابطه ۸۳ ضرایب کنترلر تنظیم شده به دست می آیند. طبق یک قاعده سرانگشتی مطرح شده در شکل ۵۰ مقادیر بیشینه و کمینه مطرح شده در رابطه ۸۳ به صورتی که در رابطه ۸۴ آورده شده تعیین می شوند. به نوعی ما داریم برنامه ریزی می کنیم که از کدام حالت از کنترلرها استفاده کنیم.

$$k_p = k_p^{\min} + (k_p^{\max} - k_p^{\min}) k'_p$$

$$k_d = k_d^{\min} + (k_d^{\max} - k_d^{\min}) k'_d$$

$$k_i = \frac{k_p^2}{\alpha k_d}$$

(۸۳)



$$\begin{cases} k_p^{\min} = 0.32k_u \\ k_p^{\max} = 0.6k_u \end{cases} \quad \begin{cases} k_d^{\min} = 0.08k_uT_u \\ k_d^{\max} = 0.15k_uT_u \end{cases} \quad (84)$$

با این مقدمه از بحث‌های تئوری به صورت 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 تعیین شده در رابطه ۷۸ را معین می‌کنیم. مطابق توضیحات در بخش تئوری توابع تعلق مربوط به خطا به صورتی که

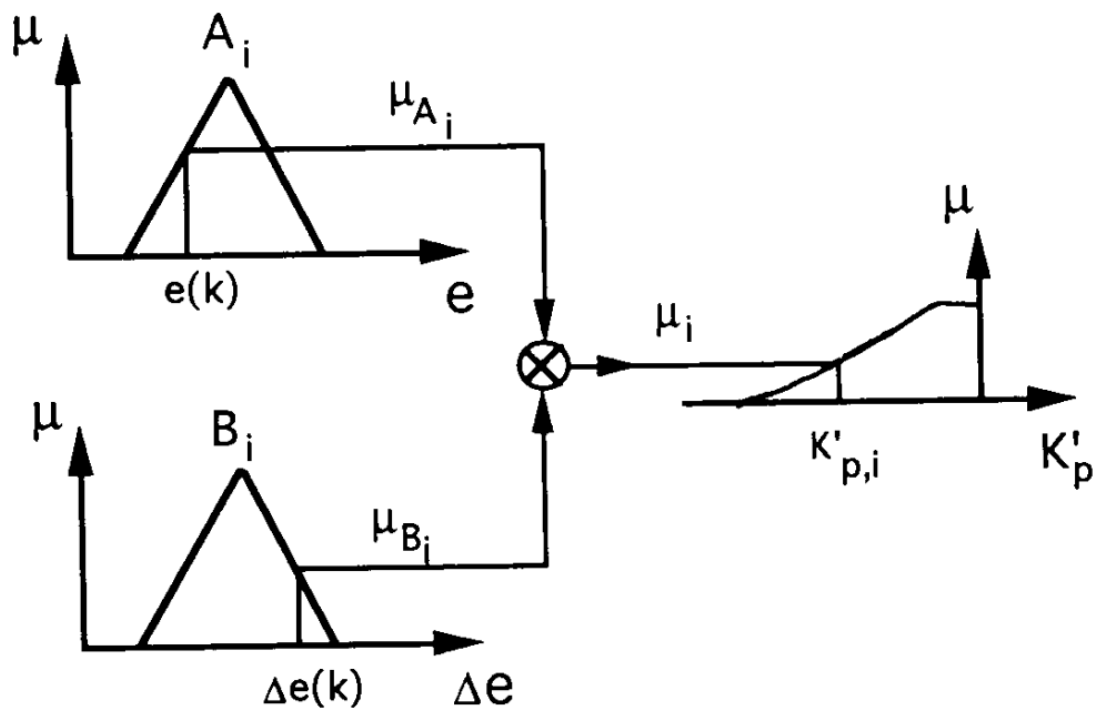
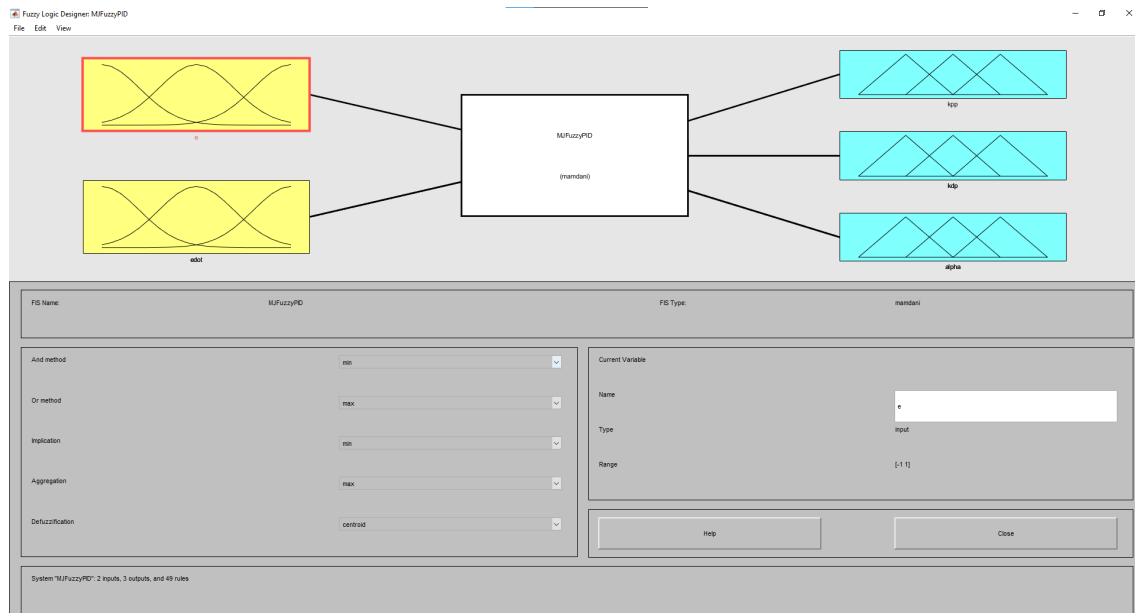


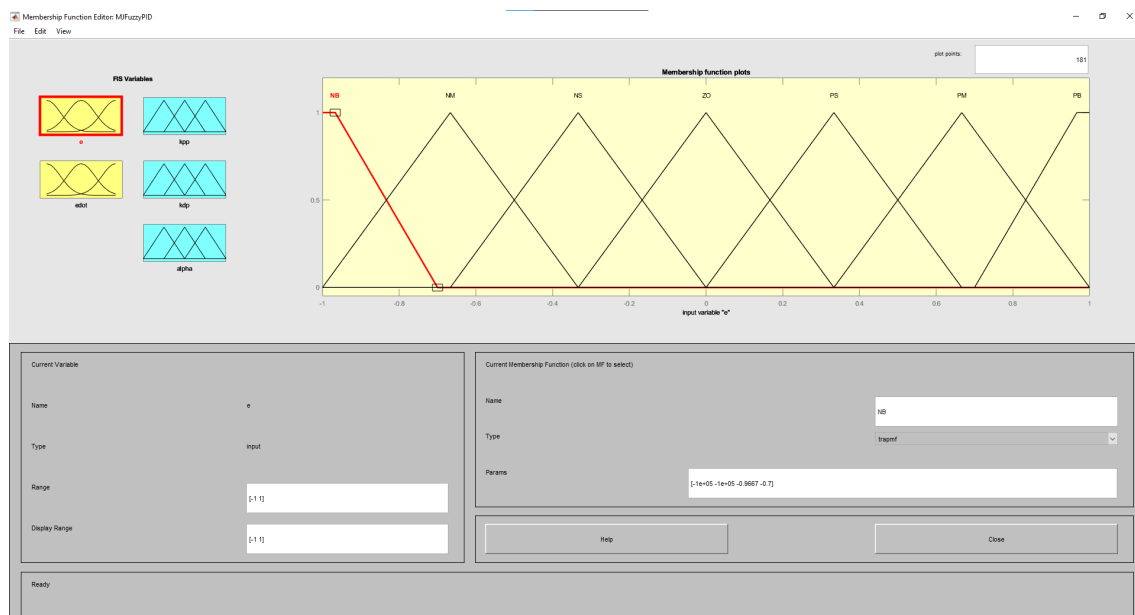
Fig. 6. Implication process of a fuzzy rule.

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 مطرح شده در شبیه‌سازی حالت بهتری است که با توجه به سیستم انتخابی و بر شانه تئوری مطرح شده در قبل ساخته شده است. هم چنین مطابق توضیحات در بخش تئوری توابع تعلق مربوط به خروجی‌ها و ضرایب در نظر گرفته شده به صورتی که در شکل ۵۸، شکل ۵۹ و شکل ۶۰ نشان داده شده است تعیین می‌شود. مجدداً باید توجه داشت موارد مطرح شده در شبیه‌سازی حالت بهتری

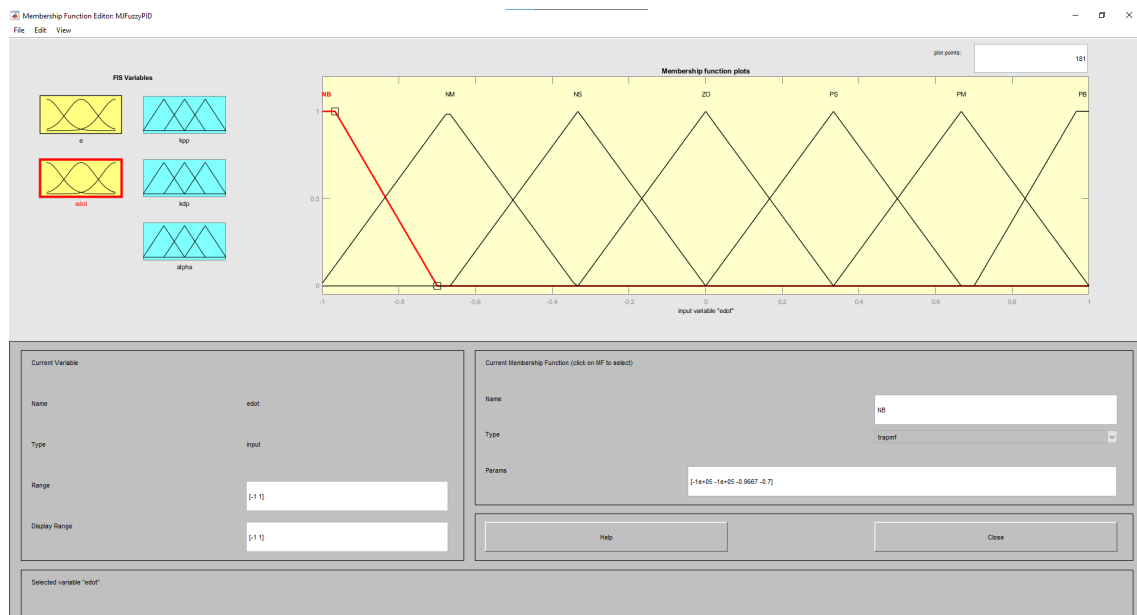


شکل ۵۵: نمای کلی سیستم فازی.

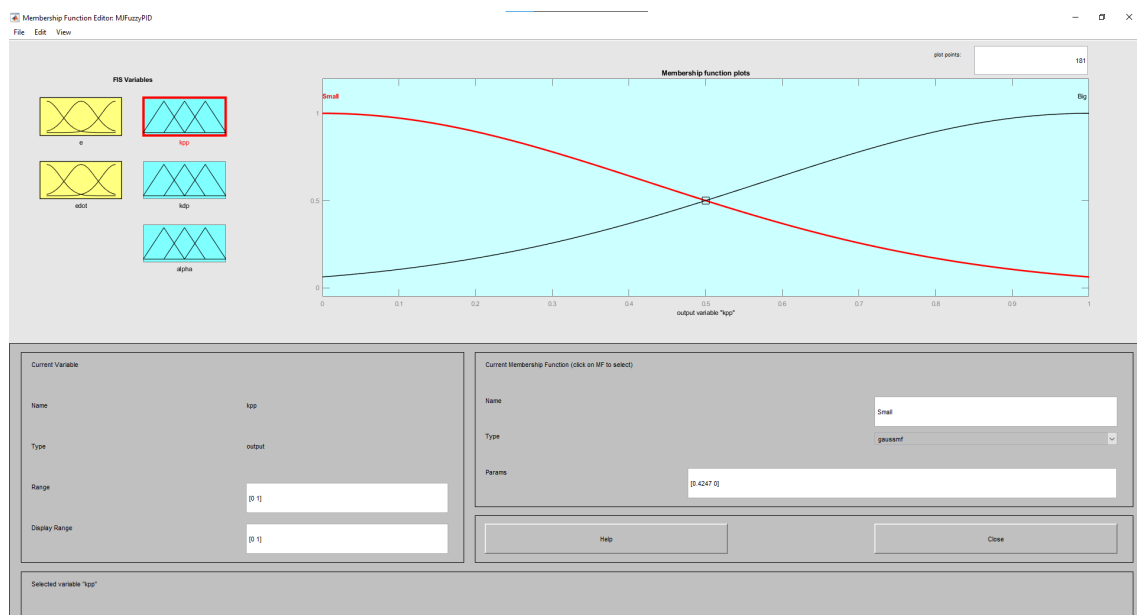


شکل ۵۶: توابع تعلق مربوط به خطا.

است که با توجه به سیستم انتخابی و بر شانه تئوری مطرح شده در قبل ساخته شده است. مجدداً مطابق توضیحات در بخش تئوری و هم چنین 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  $\alpha$  نیز به صورت‌هایی که در شکل ۶۵ و شکل ۶۶ نشان داده خواهد بود. نمای کلی بلوک کنترلی به صورتی است که در شکل ۶۷ آورده شده است. مغز کنترلی FuzzyPID به صورتی است که در

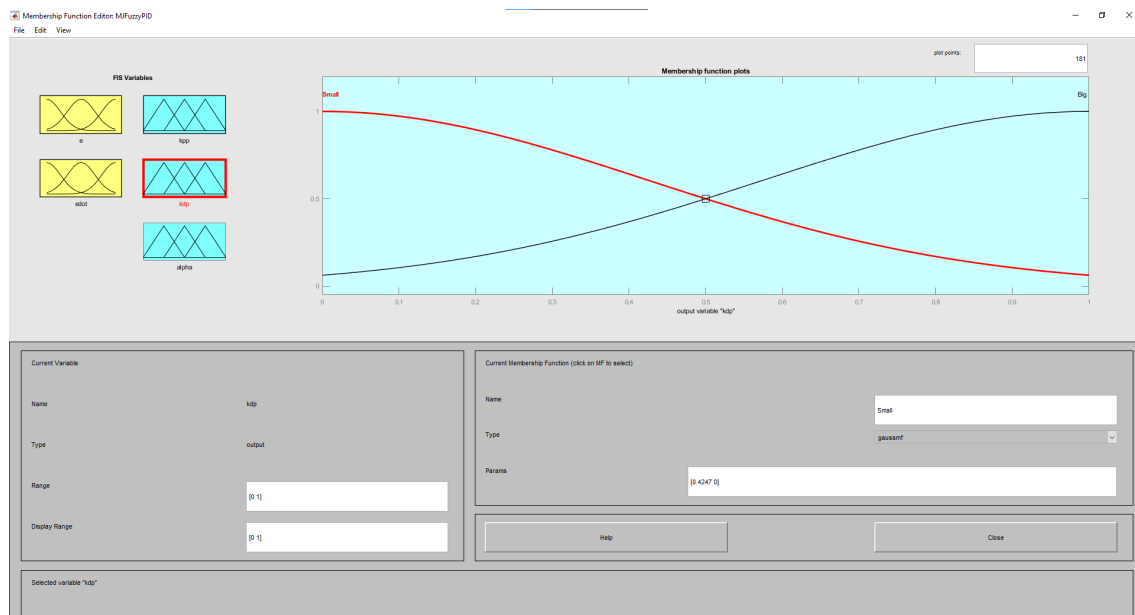


شکل ۵۷: توابع تعلق مربوط به مشتق خطا.

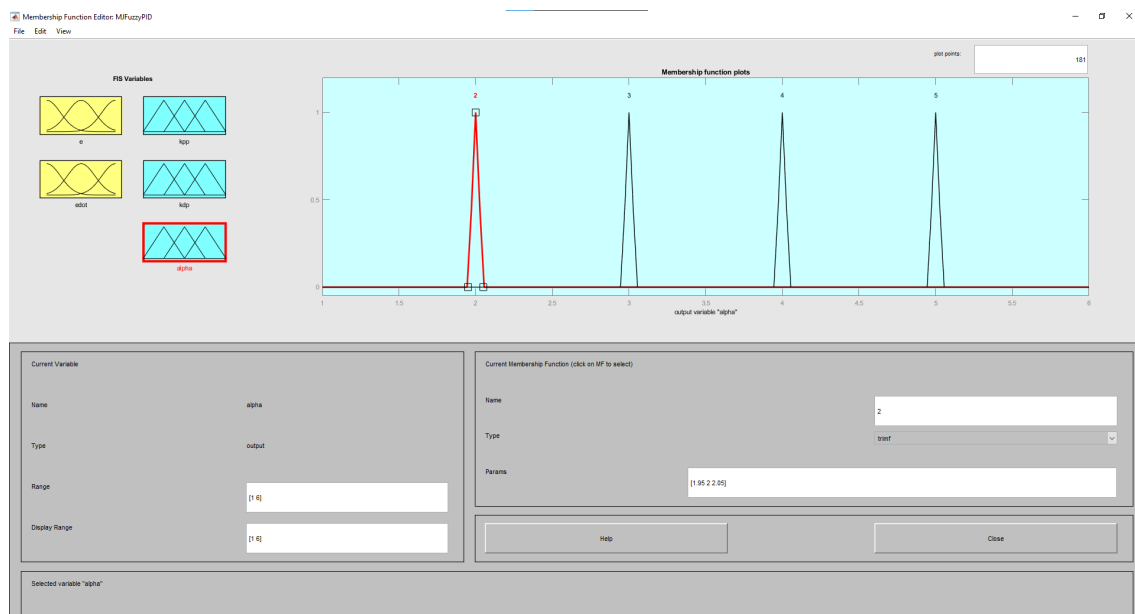


شکل ۵۸: توابع تعلق مربوط به  $k_{pp}$ .

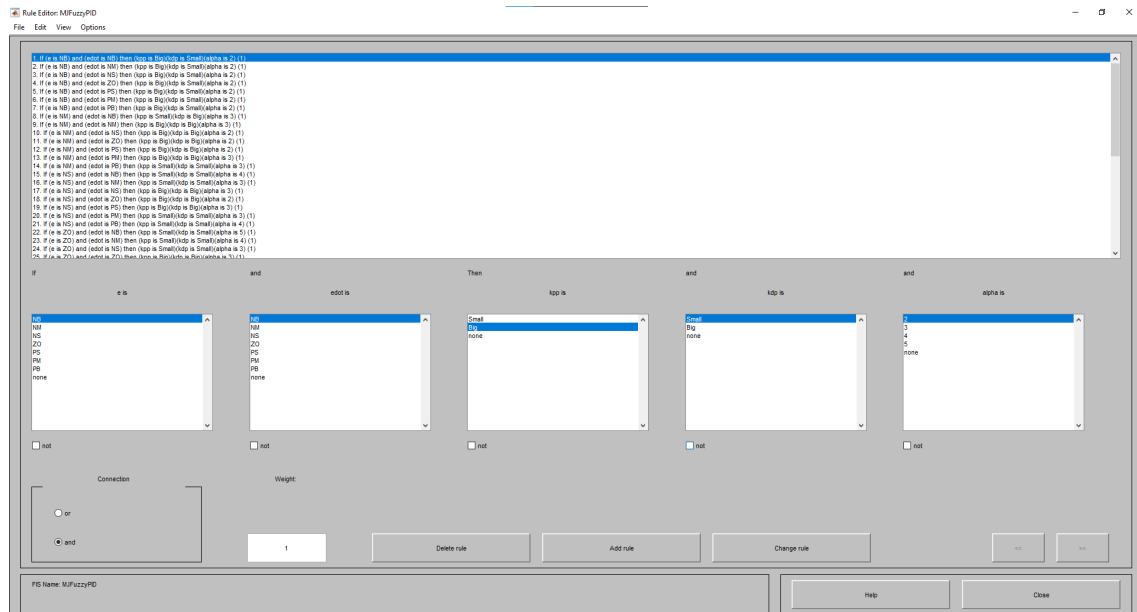
شکل ۶۸ آورده شده است. سیستم فازی به صورتی که در شکل ۶۹ نمایش داده شده تعیین می شود. محیط سیمولینک کلی و نتیجه هم به ترتیب در شکل ۷۰ و شکل ۷۱ نشان داده شده.



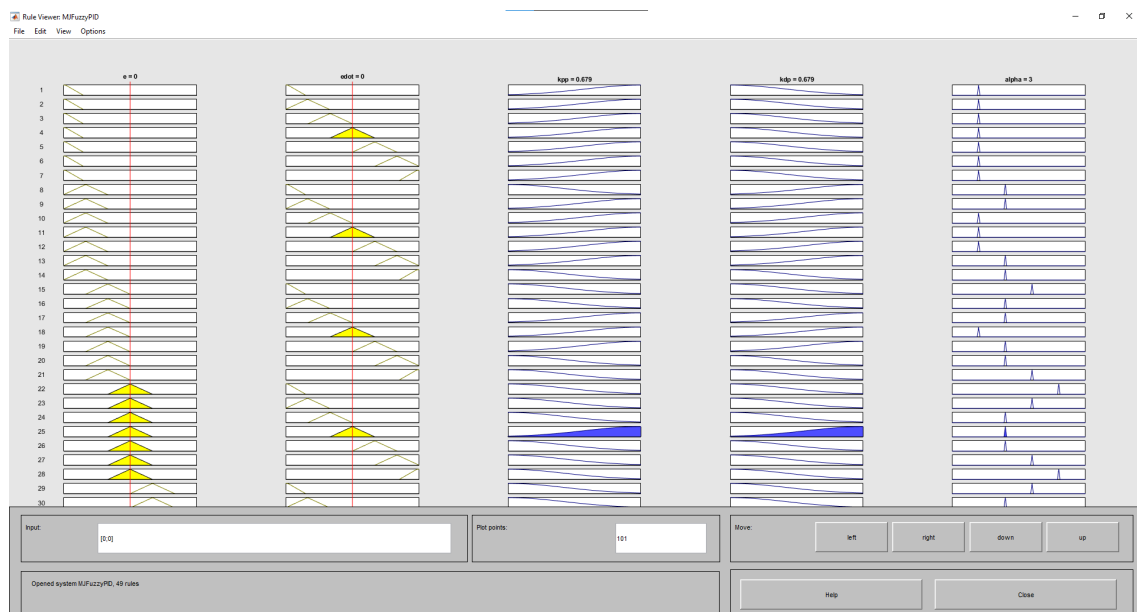
شکل ۵۹: توابع تعلق مربوط به مشتق  $k_{dp}$ .



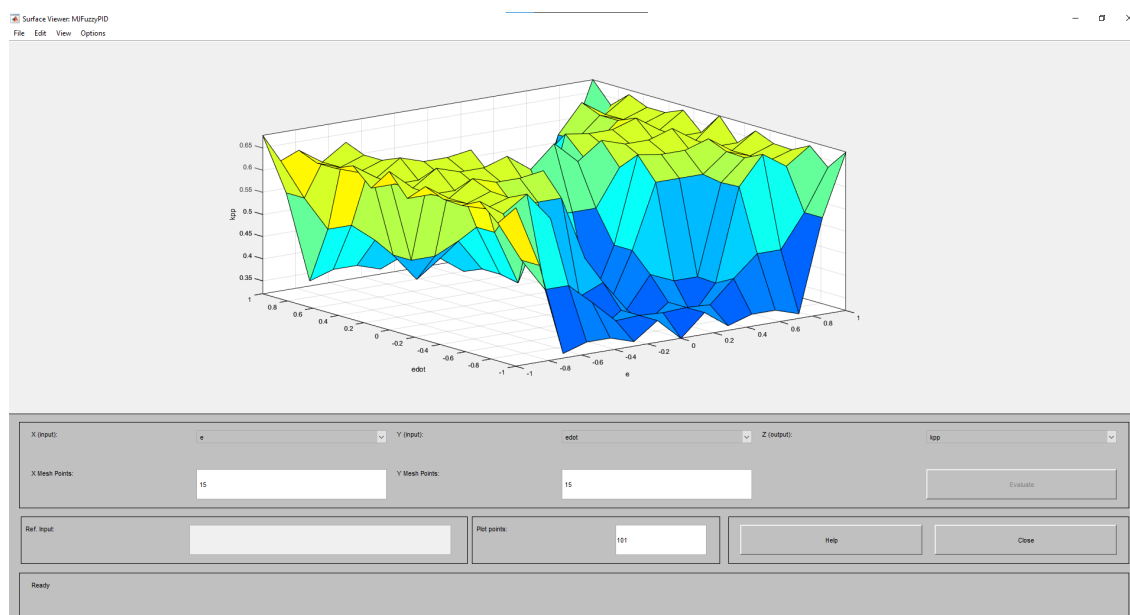
شکل ۶۰: توابع تعلق مربوط به مشتق  $\alpha$ .



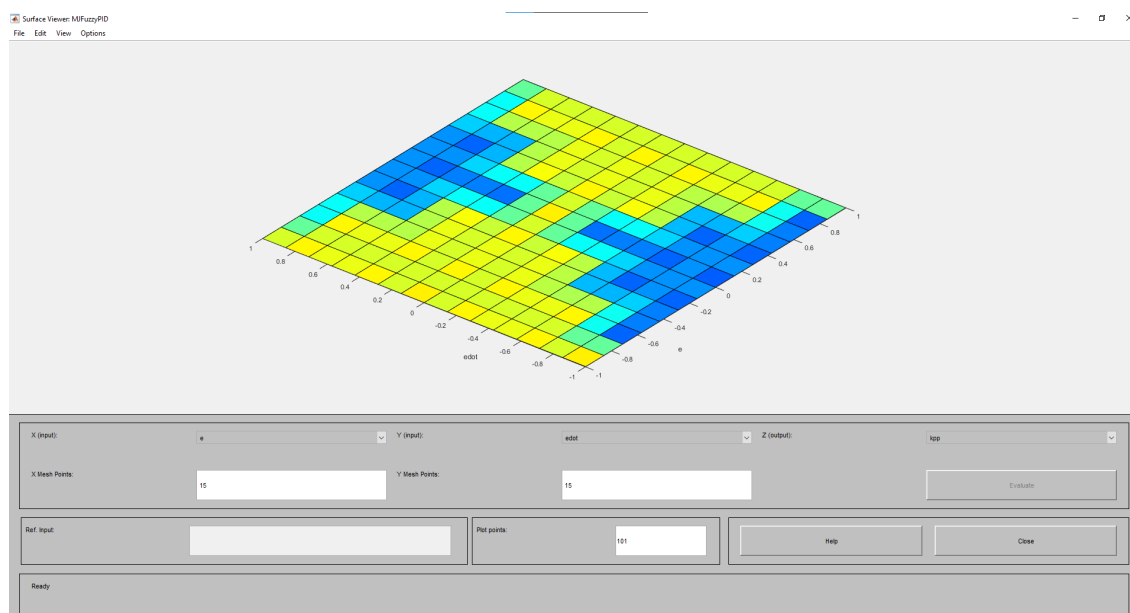
شکل ۶۱: تعیین قوانین فازی.



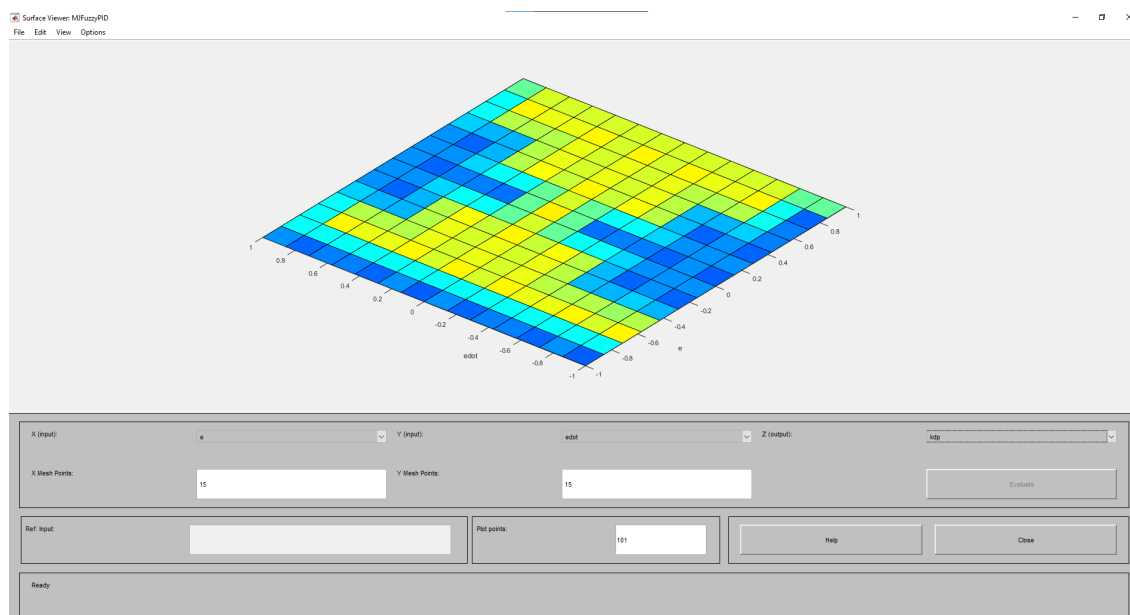
شکل ۶۲: نمایش قوانین فازی.



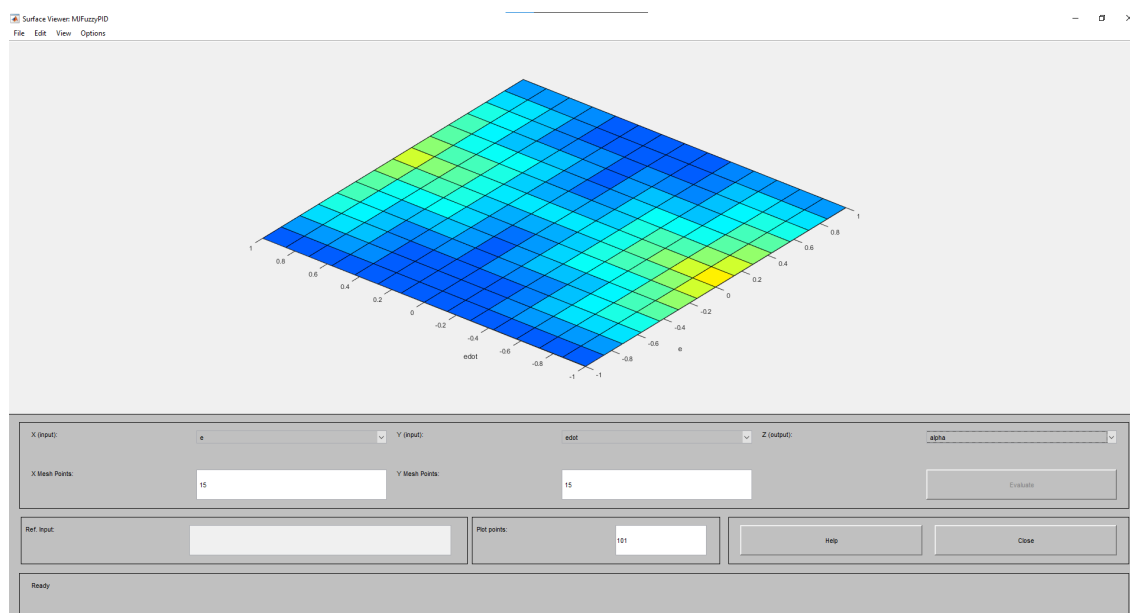
شکل ۶۳: سطح سیستم فازی.



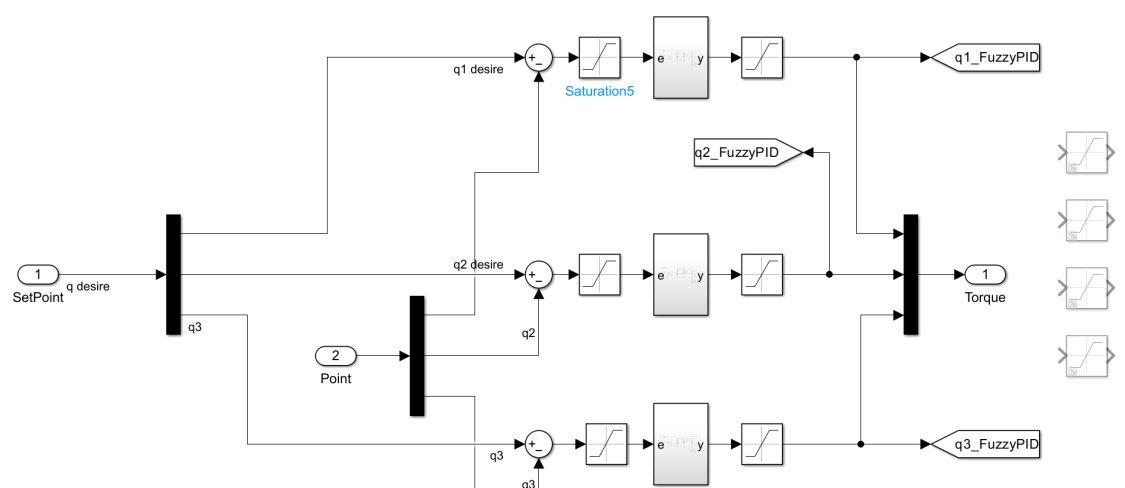
شکل ۶۴: سطح سیستم فازی.



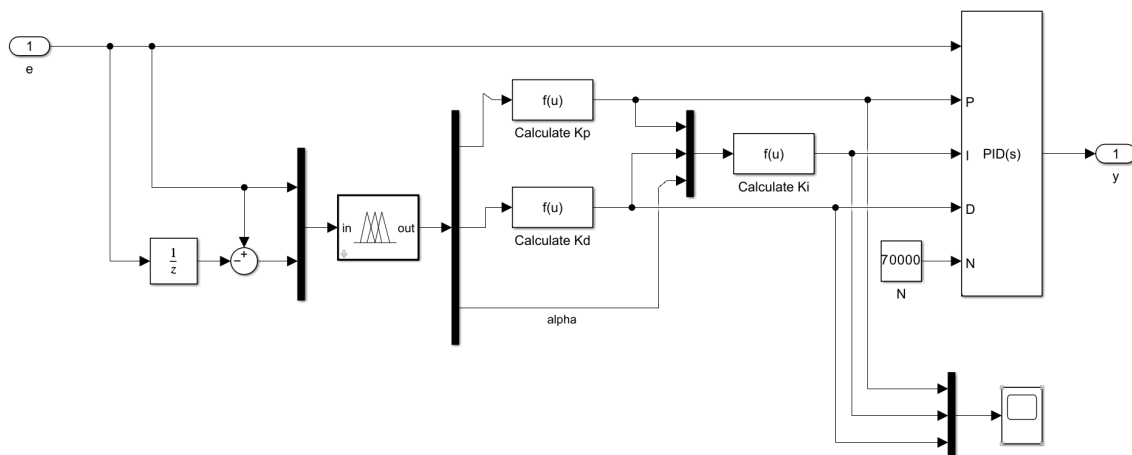
شکل ۶۵: سطح سیستم فازی.



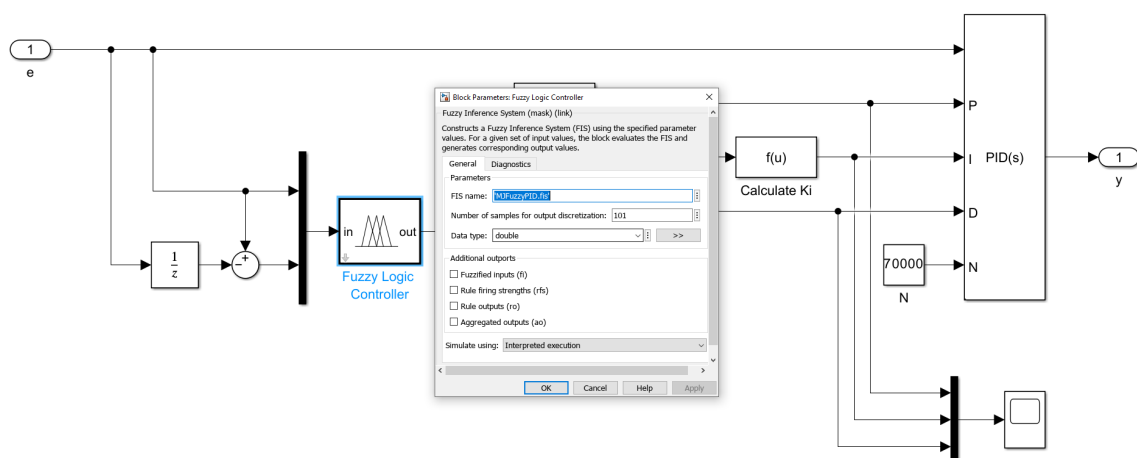
شکل ۶۶: سطح سیستم فازی.



شکل ۶۷: نمای کلی بلوک کنترلی.

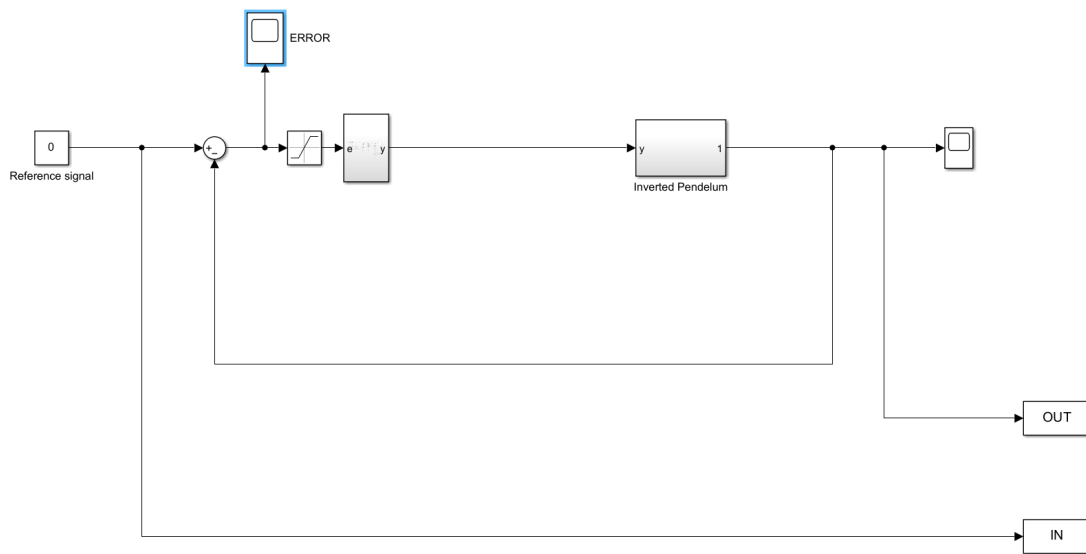


شکل ۶۸: مغز کنترلی FuzzyPID.

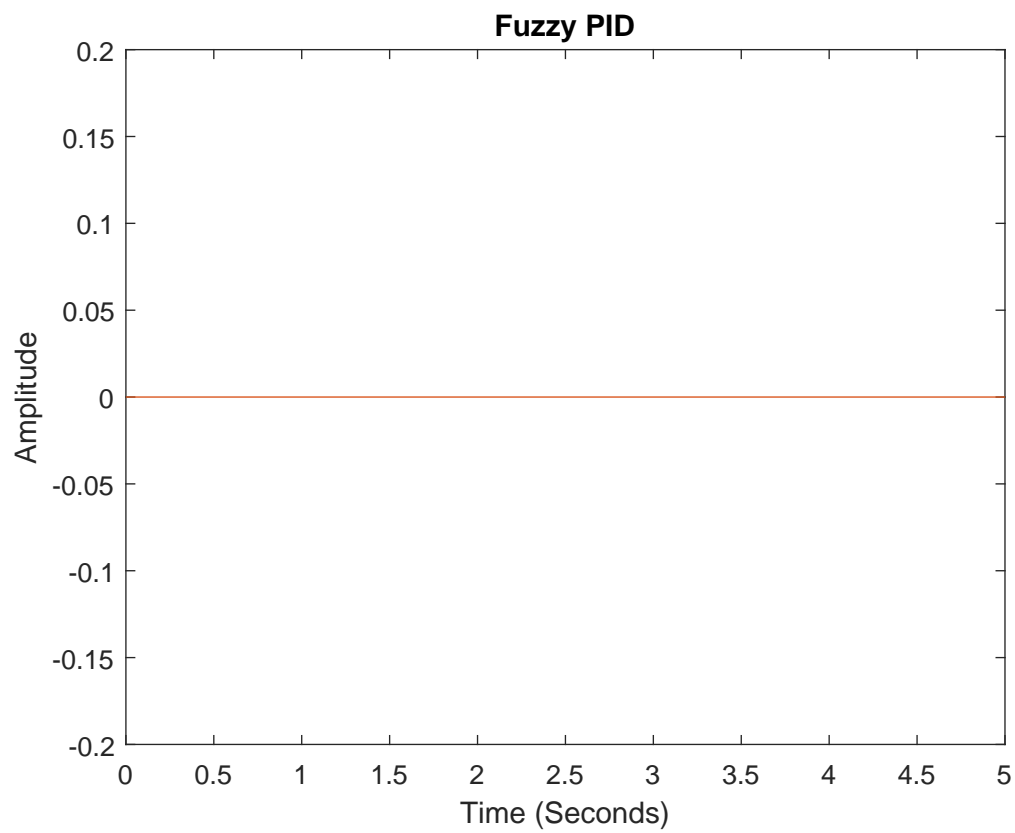


شکل ۶۹: مغز کنترلی FuzzyPID.





شکل ۷۰: سیمولینک.



شکل ۷۱: نتیجه.