

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4
5 # Plotting the output control variable (High (-))
6 def output_controller_high_negative(x):
7     if -0.09 >= x >= -0.15:
8         return_value = 1
9     elif -0.03 >= x >= -0.09:
10        return_value = -(100 / 6) * x - 0.5
11    else:
12        return_value = 0
13    return return_value
14
15
16 # Plotting the output control variable (Small (-))
17 def output_controller_small_negative(x):
18     if 0 >= x >= -0.03:
19         return_value = -(100 / 3) * x
20     elif -0.03 >= x >= -0.06:
21         return_value = (100 / 3) * x + 2
22     else:
23         return_value = 0
24    return return_value
25
26
27 # Plotting the output control variable (Good)
28 def output_controller_good(x):
29     if 0 >= x >= -0.03:
30         return_value = (100 / 3) * x + 1
31     elif 0.03 >= x >= 0:
32         return_value = - (100 / 3) * x + 1
33     else:
34         return_value = 0
35    return return_value
36
37
38 # Plotting the output control variable (Small (+))
39 def output_controller_small_positive(x):
40     if 0.06 >= x >= 0.03:
41         return_value = - (100 / 3) * x + 2
42     elif 0.03 >= x >= 0:
43         return_value = (100 / 3) * x
44     else:
45         return_value = 0
46    return return_value
47
48
49 # Plotting the output control variable (High (+))
50 def output_controller_high_positive(x):
51     if 0.15 >= x >= 0.09:
52         return_value = 1
53     elif 0.09 >= x >= 0.03:
54         return_value = (100 / 6) * x - 0.5
55     else:
56         return_value = 0
57    return return_value
58
59
60 # Define x-axis for the output control value
61 t = np.linspace(-0.15, 0.15, 400)
62 counter = 0
63 high_negative = np.zeros(shape=t.shape)
64 small_negative = np.zeros(shape=t.shape)
65 good = np.zeros(shape=t.shape)
66 small_positive = np.zeros(shape=t.shape)
67 high_positive = np.zeros(shape=t.shape)
68
69 for i in t:
70     high_negative[counter] = output_controller_high_negative(i)
71     small_negative[counter] = output_controller_small_negative(i)
72     good[counter] = output_controller_good(i)
73     small_positive[counter] = output_controller_small_positive(i)
74     high_positive[counter] = output_controller_high_positive(i)
75
76     counter += 1
77

```

```

78 plt.ylabel("Membership")
79 plt.xlabel("$\delta$")
80 plt.title('Fuzzy Partition for the control variable')
81 plt.plot(t, high_negative, label='High Infection rate (-)')
82 plt.plot(t, small_negative, label='Low Infection rate (-)')
83 plt.plot(t, good, label='Good Infection rate')
84 plt.plot(t, small_positive, label='Low Infection rate (+)')
85 plt.plot(t, high_positive, label='High Infection rate (+)')
86
87 plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")
88
89 plt.show()
90
91
92 # Calculating the membership for the infected bots rate (High)
93 def controller_high(x):
94     if x >= 0.8:
95         return 1
96     elif x <= 0.6:
97         return 0
98     else:
99         value = 5 * x - 3
100     return value
101
102
103 # Calculating the membership for the infected bots rate (Good)
104 def controller_good(x):
105     if 0.65 >= x >= 0.6:
106         return_value = -20 * x + 13
107     elif 0.6 >= x >= 0.55:
108         return_value = (20 * x) - 11
109     else:
110         return_value = 0
111     return return_value
112
113
114 # Calculating the membership for the infected bots rate (Low)
115 def controller_small(x):
116     if x <= 0.4:
117         return 1
118     elif x >= 0.6:
119         return 0
120     else:
121         value = -5 * x + 3
122     return value
123
124
125 # Define x-axis for the infected bots rate
126 k = np.linspace(-0.05, 1, 400)
127
128 counter = 0
129 good_triangle = np.zeros(shape=k.shape)
130 high_triangle = np.zeros(shape=k.shape)
131 small_triangle = np.zeros(shape=k.shape)
132 for x in k:
133     good_triangle[counter] = controller_good(x)
134     high_triangle[counter] = controller_high(x)
135     small_triangle[counter] = controller_small(x)
136     counter += 1
137
138 plt.ylabel("Membership")
139 plt.xlabel("$\pi$")
140 plt.title('Fuzzy partition of the current percentage of the infected bots')
141 plt.plot(k, small_triangle, label='Low Infection rate')
142 plt.plot(k, good_triangle, label='Good Infection rate')
143 plt.plot(k, high_triangle, label='High Infection rate')
144 plt.xticks(np.arange(0, 1.1, 0.1))
145 plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")
146
147 plt.show()
148
149
150 # Calculating the membership for the effective infection rate(High)
151 def effective_controller_high(x):
152     if 1 >= x >= 0.2:
153         return 1
154     elif 0 <= x <= 0.2:

```

```

155     value = 5 * x
156     return value
157 else:
158     return 0
159
160
161 # Calculating the membership for the effective infection rate(Good)
162 def effective_controller_good(x):
163     if 0 >= x >= -0.2:
164         return_value = 5 * x + 1
165     elif 0.2 >= x >= 0:
166         return_value = -5 * x + 1
167     else:
168         return_value = 0
169     return return_value
170
171
172 # Calculating the membership for the effective infection rate(Small)
173 def effective_controller_small(x):
174     if -0.2 >= x >= -1:
175         return 1
176     elif 0 >= x >= -0.2:
177         value = -5 * x
178         return value
179     else:
180         return 0
181
182 # Define x-axis for the current effective infection rate
183 m = np.linspace(-1, 1, 1000)
184
185 counter = 0
186 good_triangle = np.zeros(shape=m.shape)
187 high_triangle = np.zeros(shape=m.shape)
188 small_triangle = np.zeros(shape=m.shape)
189
190 for x in m:
191     good_triangle[counter] = effective_controller_good(x)
192     high_triangle[counter] = effective_controller_high(x)
193     small_triangle[counter] = effective_controller_small(x)
194     counter += 1
195
196 plt.ylabel("Membership")
197 plt.xlabel("$\dot{\pi}$")
198 plt.title('Fuzzy partition of the current effective infection')
199 plt.plot(m, small_triangle, label='Low Infection rate')
200 plt.plot(m, good_triangle, label='Good Infection rate')
201 plt.plot(m, high_triangle, label='High Infection rate')
202 plt.xticks(np.arange(-1, 1.1, 0.2))
203 plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")
204
205 plt.show()
206

```