

```

1 from plague import Plague
2 import numpy as np
3 import random
4 import matplotlib.pyplot as plt
5
6
7 def fuzzy_controller(infected_percentage, effective_infection):
8     # Calculating the membership for the current effective infection rate
9     effective_memberships = [effective_infection_low(effective_infection),
10                             effective_infection_good(effective_infection),
11                             effective_infection_high(effective_infection)]
12
13     # Calculating the membership for the infected bots rate
14     memberships = [calculate_membership_low_infected(infected_percentage),
15                   calculate_membership_good_infected(infected_percentage),
16                   calculate_membership_high_infected(infected_percentage)]
17
18     best_member = memberships.index(max(memberships))
19     best_effective = effective_memberships.index(max(effective_memberships))
20
21     # According to the highest memberships, evaluate the output control variable by using The Max
Criterion method
22     # The following conditional statements are the implementation of the Table 1 in the report.
23     if best_member == 0:
24         if best_effective == 0 or best_effective == 1:
25             return output_controller_high_positive(memberships[0])
26         else:
27             return output_controller_low_positive(memberships[0])
28     elif best_member == 1:
29         if best_effective == 2:
30             return output_controller_high_negative(memberships[1])
31         else:
32             return output_controller_good(memberships[1])
33     else:
34         if best_effective == 0 or best_effective == 1:
35             return output_controller_high_negative(memberships[2])
36         else:
37             return output_controller_low_negative(memberships[2])
38
39
40 # Calculate the membership for the current effective infection rate (Low)
41 def effective_infection_low(x):
42     if -0.2 >= x >= -1:
43         return 1
44     elif 0 >= x >= -0.2:
45         value = -5 * x
46         return value
47     else:
48         return 0
49
50
51 # Calculate the membership for the current effective infection rate (High)
52 def effective_infection_high(x):
53     if 1 >= x >= 0.2:
54         return 1
55     elif 0 <= x <= 0.2:
56         value = 5 * x
57         return value
58     else:
59         return 0
60
61
62 # Calculate the membership for the current effective infection rate (Good)
63 def effective_infection_good(x):
64     if 0 >= x >= -0.2:
65         return_value = 5 * x + 1
66     elif 0.2 >= x >= 0:
67         return_value = -5 * x + 1
68     else:
69         return_value = 0
70     return return_value
71
72
73 # Calculate the membership for the infected bots (Low)
74 def calculate_membership_low_infected(x):
75     if x <= 0.4:
76         return 1

```

```

77     elif x >= 0.6:
78         return 0
79     else:
80         slope = -(1 / 0.2)
81         value = 1 - min(1, slope * (0.4 - x))
82         return value
83
84
85 # Calculate the membership for the infected bots (Good)
86 def calculate_membership_good_infected(x):
87     if 0.65 >= x >= 0.6:
88         return_value = -10 * x + 7
89     elif 0.6 >= x >= 0.55:
90         return_value = (10 * x) - 5
91     else:
92         return_value = 0
93     return return_value
94
95
96 # Calculate the membership for the infected bots (High)
97 def calculate_membership_high_infected(x):
98     if x >= 0.8:
99         return 1
100    elif x <= 0.6:
101        return 0
102    else:
103        slope = (1 / 0.2)
104        value = 1 - min(1, slope * (0.8 - x))
105        return value
106
107
108 # Calculating the output control variable from membership (Good)
109 def output_controller_good(x):
110     first_point = (x - 1) * 3 / 100
111     second_point = -((x - 1) * 3 / 100)
112     return_value = random.uniform(first_point, second_point)
113     return return_value
114
115
116 # Calculating the output control variable from membership (Low (+))
117 def output_controller_low_positive(x):
118     first_point = 0.03 * x
119     second_point = -(x-2) * 0.03
120     return_value = random.uniform(first_point, second_point)
121     return return_value
122
123
124 # Calculating the output control variable from membership (Low (-))
125 def output_controller_low_negative(x):
126     first_point = ((x - 2) * 3 / 100)
127     second_point = first_point + 2 * (-0.03 - first_point)
128     return_value = random.uniform(first_point, second_point)
129     return return_value
130
131
132 # Calculating the output control variable from membership (High (+))
133 def output_controller_high_positive(x):
134     if x == 1:
135         return_value = random.uniform(0.09, 0.15)
136     else:
137         second_point = (x + 0.5) * 0.06
138         return_value = random.uniform(second_point, 0.09)
139     return return_value
140
141
142 # Calculating the output control variable from membership (High (-))
143 def output_controller_high_negative(x):
144     if x == 1:
145         return_value = random.uniform(-0.15, -0.09)
146     else:
147         second_point = (x + 0.5) * -0.06
148         return_value = random.uniform(-0.09, second_point)
149     return return_value
150
151
152 plague = Plague()
153

```

```
154 number_iterations = 200
155 counter = 0
156 infected_bots_plot = np.empty(shape=(200,))
157 effective_infection_rates = np.empty(shape=(200,))
158 # Loop for spreading virus for 20 days
159 while counter != number_iterations:
160     # Get the rates for the infected bot and effective infection
161     infected_bots, effective_infection_rate = plague.checkInfectionStatus()
162     # Add them to array for debugging and plotting
163     infected_bots_plot[counter] = infected_bots
164     effective_infection_rates[counter] = effective_infection_rate
165     # Use infected bots rate in order to generate a control variable
166     control_variable = fuzzy_controller(infected_bots, effective_infection_rate)
167     # Spread virus
168     plague.spreadPlague(control_variable)
169     # Increase the loop iteration
170     counter += 1
171
172
173 # Plotting
174 plt.ylabel("Infection rate")
175 plt.xlabel("Steps")
176 plt.plot(plague.infected_percentage_curve_)
177 plt.show()
178
179 plt.ylabel("Infection cost")
180 plt.xlabel("Steps")
181 plt.plot(plague.infection_rate_curve_)
182 plt.show()
183
184 plt.ylabel("effective infection rate")
185 plt.xlabel("Steps")
186 plt.plot(effective_infection_rates)
187 plt.show()
188
189 # Computation of the total cost until equilibrium
190 cost_sum = sum(plague.infected_percentage_curve_[1:77])
191
192 plague.viewPlague(77, cost_sum)
```