EE496 Homework 3 - Fuzzy Control

Berken Utku Demirel - 2166221

1 Plague v1

1.1 Set Partitioning

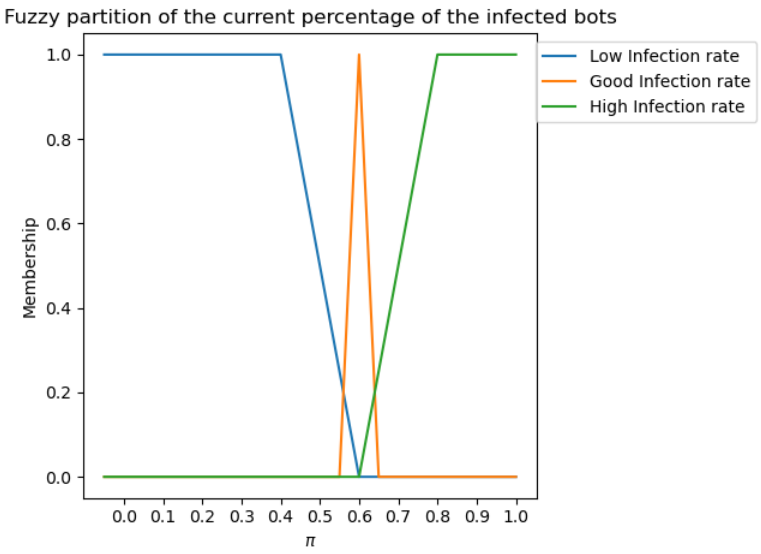

Figure 1: The partition for the infected bots π

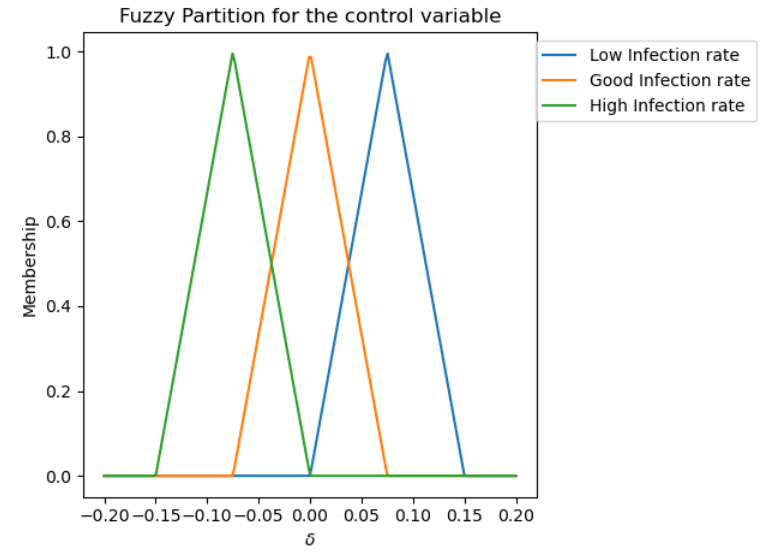

Figure 2: The partition for the control variable δ

Figure 1 shows the partition of the measurement into three fuzzy sets. This partition is done according to the percentage of the infected bots. Since we aim to reach the %60 of infection and maintain that percentage, the center of the orange triangle, which represents the good infection rate, is 0.6, and its limit is very narrow.
Figure 2 shows the partition of the output δ. It is composed of three equal triangles which have an equal base width. The shape difference between these two partitions is coming from our purpose. Since the output controller does not need to converge a special value, the limit of the base of triangles is equal.

1.2 Fuzzy Control Rules

The list for the control rules is given below.

If the infection rate is lower than the desired rate (%60), then the output control variable will be positive, which is the blue triangle.
If the infection rate is bigger than the desired rate (%60), then the output control variable will be negative, which is the green triangle.
If the infection rate is approximately (%60), then the output control variable will be approximately zero, which is the orange triangle.

The triangles, which are mentioned above, are in figure 2.

1.3 Fuzzification and Defuzzification Interface

The fuzzification and defuzzification are implemented as follows.
First, the membership values for each fuzzy set is calculated according to the infection rate measurement; then, the output controller is derived from the memberships by using the **The Max Criterion Method**. In that method, first, the highest membership is found, then the interval, which that membership corresponds to, is found, and an arbitrary value is chosen in this interval.
The Max Criterion method as defuzzification, a non-deterministic controller due to its randomness. However, our control output intervals are not too wide, so the deviation of the output control is not so much.

1.4 Simulation

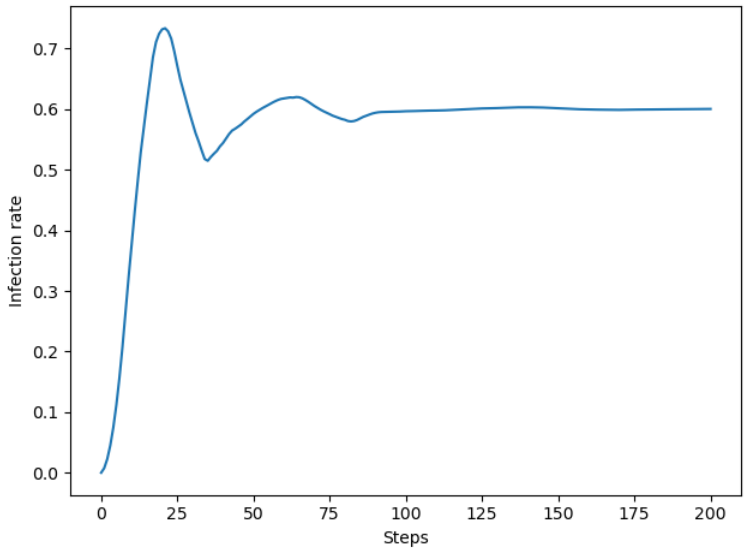

Figure 3: The result of $infected\_percentage\_curve\_$ attribute for plague v1

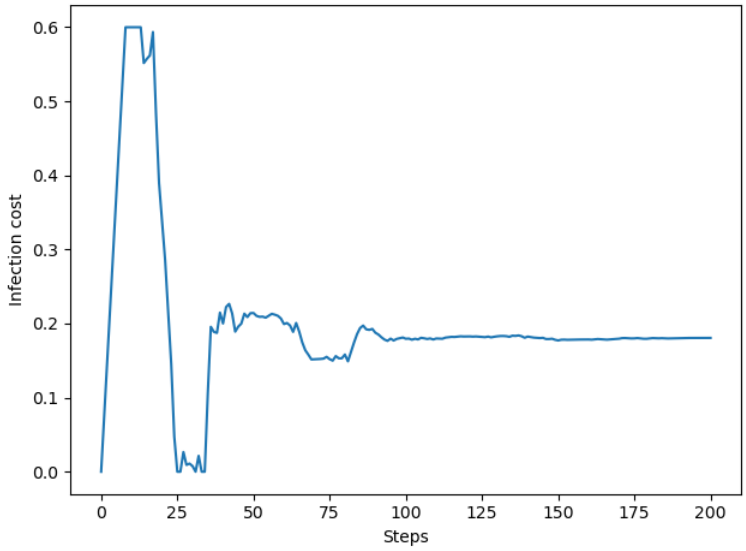

Figure 4: The result of $infection\_rate\_curve\_$ attribute for plague v1

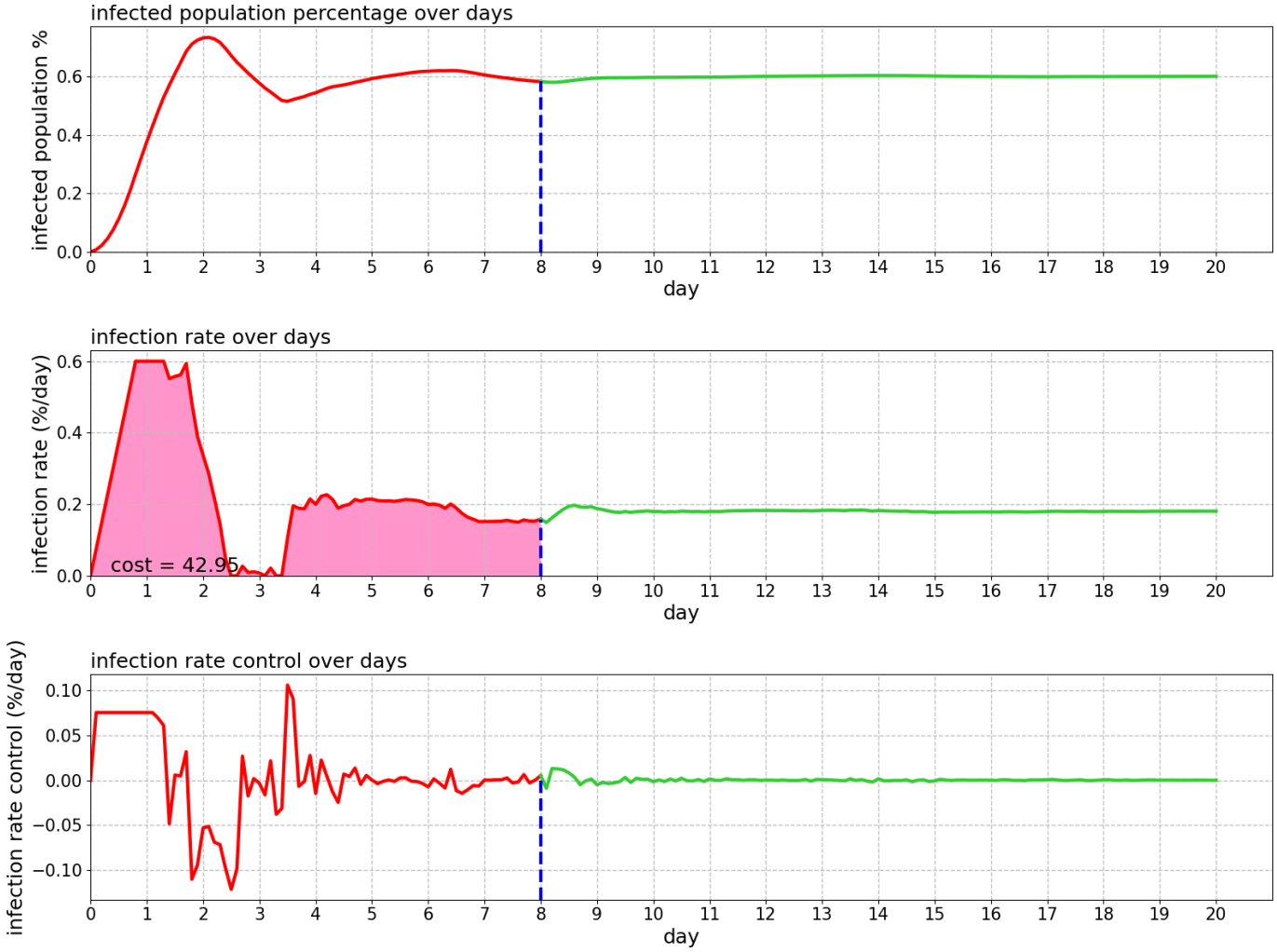

Figure 5: The result of the *viewPlague* function

2 Plague v2

2.1 Set Partitioning

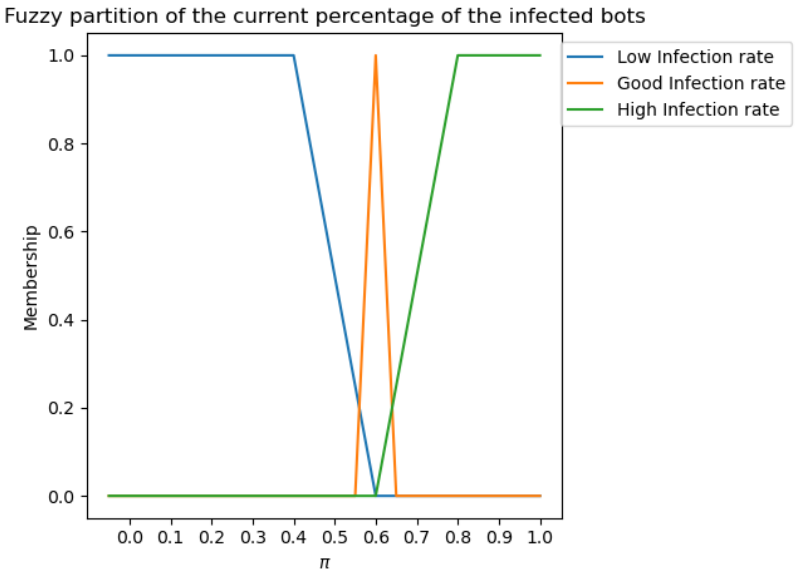

Figure 6: The partition for the infected bots π

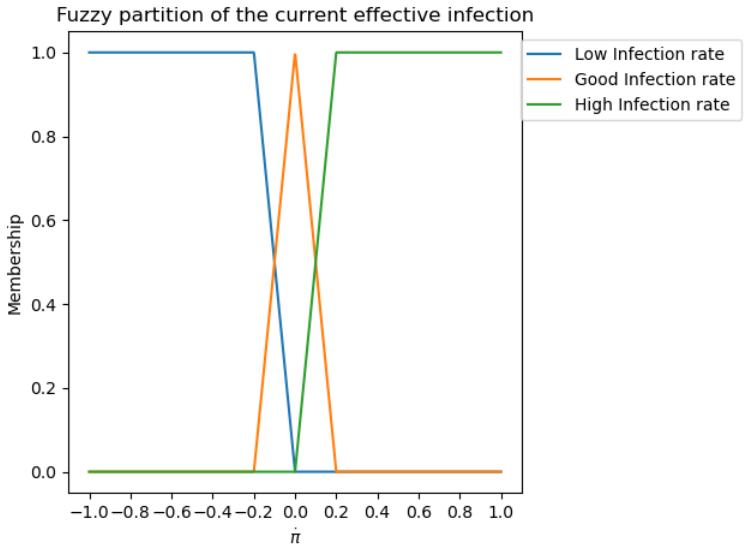

Figure 7: The partition for the current effective infected bots $\dot{\pi}$

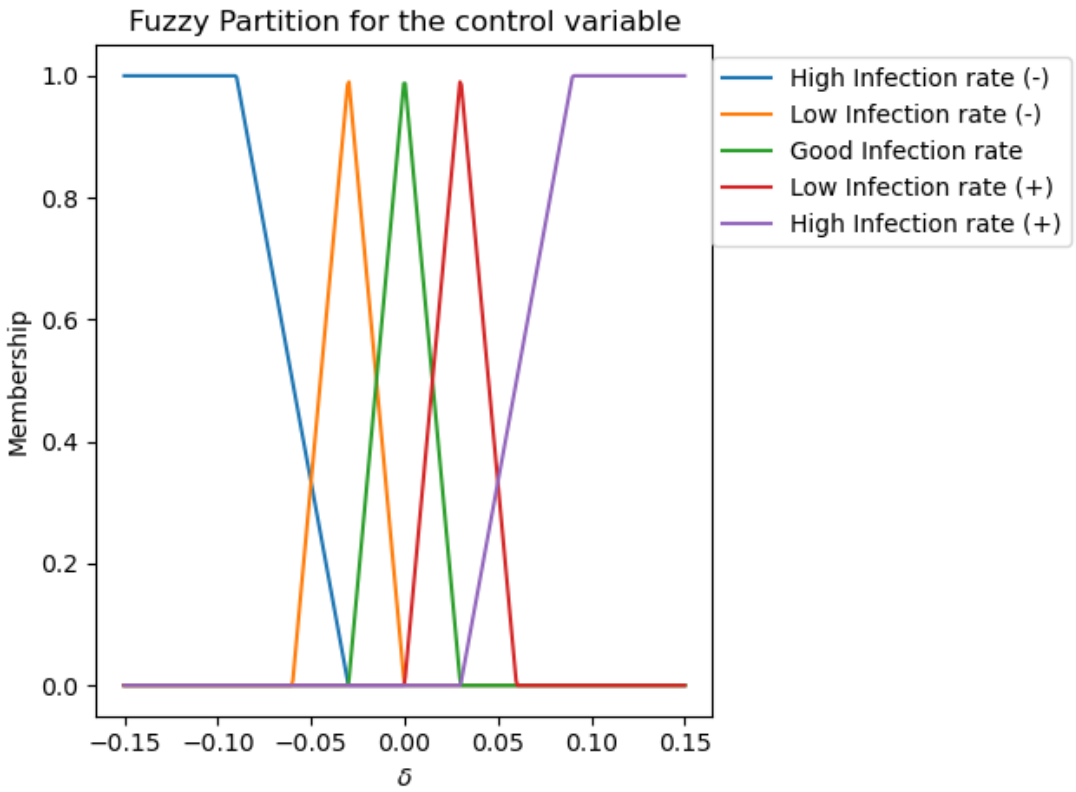

Figure 8: The output partition set

For the Plague v2, we have two measurements (π and $\dot{\pi}$) and one output (δ) to be partitioned. The partition for the infected bots percentage(π) is not different from plague v1 since our purpose, which is to infect 60% of bots population, is the same.
While determining the partition for the current effective infection rate($\dot{\pi}$), I have observed that when the system comes to the equilibrium, the effective infection rate is approximately 0.15-0.2. For this reason, The good infection rate is defined as a triangle, which has a center at the origin with 0.4 wide which is shown in figure 7. The other two fuzzy sets for the effective infection rate are composed of the trapezoid.

In the Plague v2, the output(δ) partition is set to 5 fuzzy sets. Unlike the output partition in Plague v1, the two new triangles are added, which can be seen from the figure 8. These two triangles, which have a 0.06 broad base, represent positive and negative low infection rates. The center of these triangles base is chosen as close to zero, in order to make more precise changes in the infection rate by changing the control variable.

2.2 Fuzzy Control Rules

The fuzzy control rules for the plague v2 are shown in the table 1.

Infected bots/ Effective Infection	Low	Good	High
Too Low	High (+)	High (+)	Small (+)
Good	Good	Good	High (-)
Too Many	High (-)	High (-)	Small (-)

Table 1: Table to test captions and labels

The table shows how the output control is chosen according to the current percentage of the infected bots and the effective infection rate. If the current percentage of infected bots is smaller than 0.6 (Too Low in the table), the control variable should be increased; however, this increase is also affected by the effective infection rate. When the effective infection rate is also high, the control variable is increased by a smaller value (Small (+) in the table). In this way, the control variable is changed smoothly compared to plague v1. The same logic is also valid when the infected bots are so many (Too Many in the Table) than the desired value. If the number of infected bots is high, and the effective infection rate is small or good, the control variable should be decreased highly in order to balance the infection rate. The last case where the percentage of the infected bots are approximately 0.6 (Good in the Table), the output control variable is also chosen as Good since, in this interval, changing the control variable too much is not a good idea. However, if the effective infection rate is high, it is observed that keeping the control variable High (-) has a good effect on decreasing overshoot.
For the defuzzification part, first, the membership of the infected bots are calculated, then the region (Low, Good, High in the Table) for the current effective infection rate is found. While determining the region of these two inputs, the interval of the maximum membership value is chosen. According to these two regions, the interval for the output control is chosen, as shown in table 1.
Lastly, in order to map the membership value to output control variable **The Max Criterion** method is used again.

2.3 Implementation and Simulation

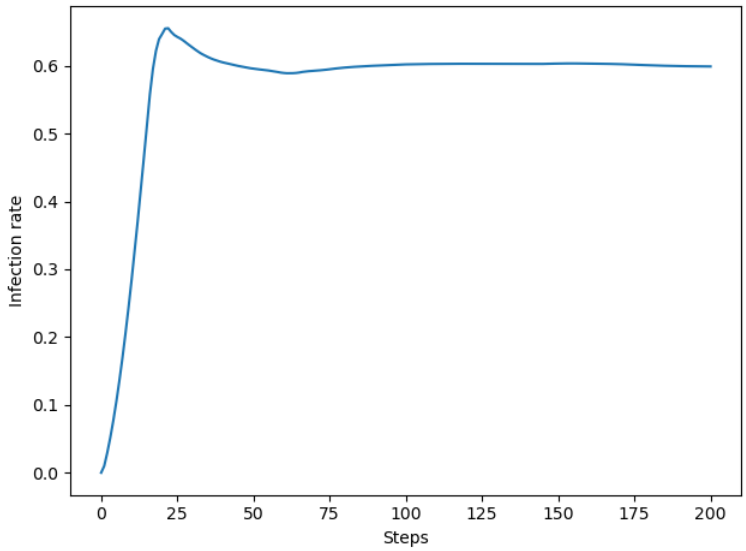

Figure 9: The result of $infected\_percentage\_curve\_$ attribute for plague v2

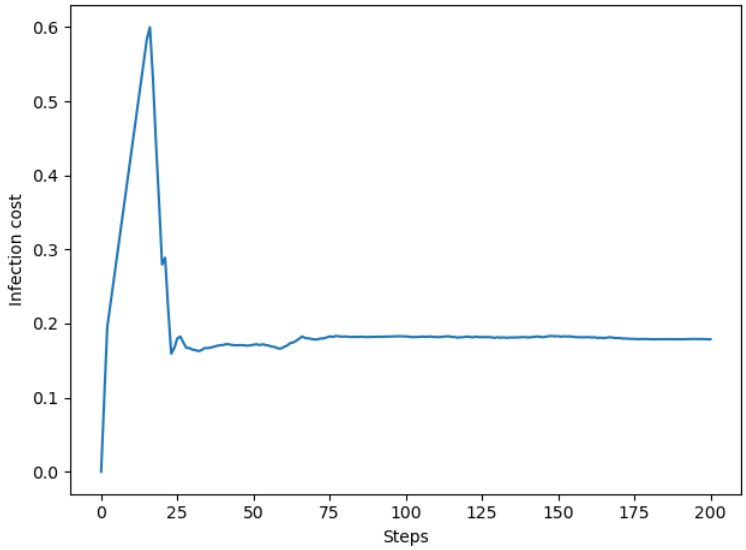

Figure 10: The result of $infection\_rate\_curve\_$ attribute for plague v2

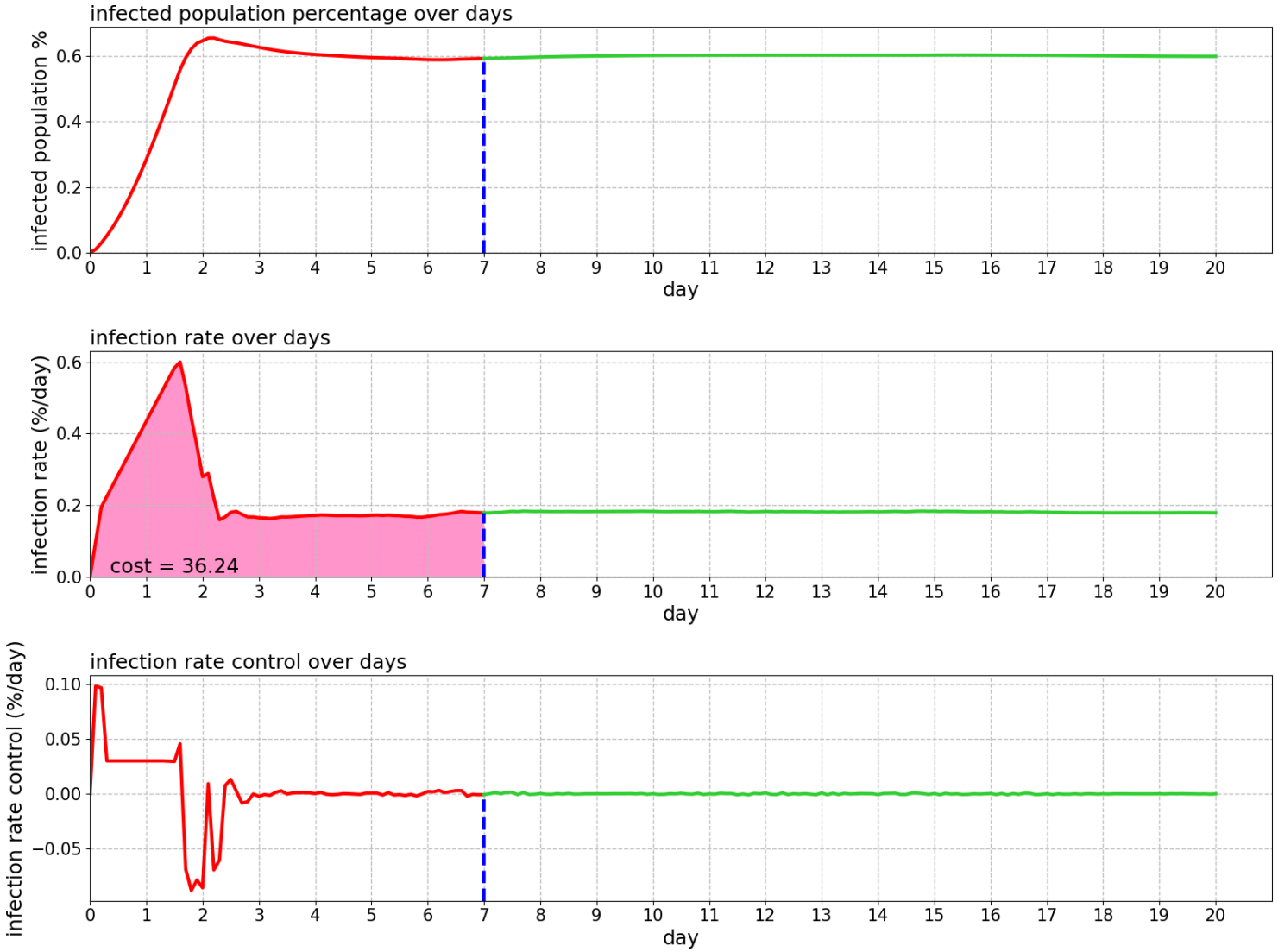

Figure 11: The result of the *viewPlague* function

2.4 Comparison

When we compare the infected population percentage of the two plagues, we can easily see that when the second input parameter, which is the current effective infection rate, is added to our fuzzy control, the system performance is increased. The reason why we introduce the effective infection rate was to decrease the overshoot and increase the convergence rate to the equilibrium. This performance increase can be seen in the figures 5 and 11.
When we look at figure 5, it can be seen that the infected bots percentage exceeds 0.6 by approximately 0.1, so we can say the overshoot is approximately 0.1 since the desired value is 0.6. However, figure 11 is examined; the overshoot is decreased to less than 0.5. This comparison clearly shows the increased performance when the effective infection rate is introduced to the system.
In addition, When we look at the same figures, it is clear that the convergence rate to equilibrium is also increased. In the plague v1, the system converges to 0.6 on day 8. However, the system converges approximately on day 7 in plague v2. This comparison of the control system also shows the increasing performance of the system by using the current effective infection rate. Also, by this improvement, the cost of the plague v2, which is 36.24, is lower than the cost of plague v1, which is 42.95.

All in all, the introduction of a current effective infection rate ($\dot{\pi}$) to the system as a new measurement improves the system performance by decreasing overshoot and increasing convergence rate to the equilibrium, which results in a decrease in the cost of the system.

3 Appendix

File - C:\Users\berkenutku\Desktop\496_hw3\hw3.py

```python
from plague import Plague
import numpy as np
import random
import matplotlib.pyplot as plt


def fuzzy_controller(infected_percentage):
    # Calculate the memberships of the infected bots rate for 3 different regions
    memberships = [calculate_membership_low_infected(infected_percentage),
                   calculate_membership_good_infected(infected_percentage),
                   calculate_membership_high_infected(infected_percentage)]

    best_member = memberships.index(max(memberships))
    # According to the highest memberships, evaluate the output control variable by using The Max Criterion method
    if best_member == 0:
        return output_controller_low(memberships[0])
    elif best_member == 1:
        return output_controller_good(memberships[1])
    else:
        return output_controller_high(memberships[2])


# Calculating the membership for low infected bots
def calculate_membership_low_infected(x):
    if 0.4 >= x > 0:
        return_value = 1
    elif 0.55 >= x >= 0.4:
        return_value = -(x / 0.15) + 3.6666
    else:
        return_value = 0
    return return_value


# Calculating the output control variable from membership (Low)
def output_controller_low(x):
    first_point = (0.15 * x / 2)
    second_point = -((x - 2) * 0.15 / 2)
    return_value = random.uniform(first_point, second_point)
    return return_value


# Calculating the membership for good rate infected bots
def calculate_membership_good_infected(x):
    if 0.7 >= x >= 0.6:
        return_value = -10 * x + 7
    elif 0.6 >= x >= 0.5:
        return_value = (10 * x) - 5
    else:
        return_value = 0
    return return_value


# Calculating the output control variable from membership (Good)
def output_controller_good(x):
    first_point = (x - 1) * 0.15 / 2
    second_point = -((x - 1) * 0.15 / 2)
    return_value = random.uniform(first_point, second_point)
    return return_value


# Calculating the membership for High rate infected bots
def calculate_membership_high_infected(x):
    if 1 >= x >= 0.8:
        return_value = 1
    elif 0.8 >= x >= 0.65:
        return_value = (x / 0.15) - 4.33
    else:
        return_value = 0
    return return_value


# Calculating the output control variable from membership (High)
def output_controller_high(x):
    first_point = ((x - 2) * 0.15 / 2)
    second_point = -(0.15 * x / 2)
    return_value = random.uniform(first_point, second_point)
```

File - C:\Users\berkenutku\Desktop\496_hw3\hw3.py

```
77      return return_value
78
79
80  plague = Plague()
81
82  number_iterations = 200
83  counter = 0
84  infected_bots_plot = np.empty(shape=(200,))
85  effective_infection_rates = np.empty(shape=(200,))
86  # Loop for spreading virus for 20 days
87  while counter != number_iterations:
88      # Get the rates for the infected bot and effective infection
89      infected_bots, effective_infection_rate = plague.checkInfectionStatus()
90      # Add them to array for debugging and plotting
91      infected_bots_plot[counter] = infected_bots
92      effective_infection_rates[counter] = effective_infection_rate
93      # Use infected bots rate in order to generate a control variable
94      control_variable = fuzzy_controller(infected_bots)
95      # Spread virus
96      plague.spreadPlague(control_variable)
97      # Increase the loop iteration
98      counter += 1
99
100 # Plotting
101 plt.ylabel("Infection rate")
102 plt.xlabel("Steps")
103 plt.plot(plague.infected_percentage_curve_)
104 plt.show()
105
106 plt.ylabel("Infection cost")
107 plt.xlabel("Steps")
108 plt.plot(plague.infection_rate_curve_)
109 plt.show()
110
111 plt.ylabel("effective infection rate")
112 plt.xlabel("Steps")
113 plt.plot(effective_infection_rates)
114 plt.show()
115
116 # Computation of the total cost until equilibrium
117 cost_sum = sum(plague.infected_percentage_curve_[1:100])
118
119 plague.viewPlague(100, cost_sum)
120
```

File - C:\Users\berkenutku\Desktop\496_hw3\plot_of_partioning.py

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


# Plotting the output control variable (Good)
def output_controller_good(x):
    if 0 >= x >= -0.15 / 2:
        return_value = (x * 2 / 0.15) + 1
    elif x <= -0.15 / 2 or x >= 0.15 / 2:
        return_value = 0
    else:
        return_value = -(x * 2 / 0.15) + 1
    return return_value


# Plotting the output control variable (High)
def output_controller_high(x):
    if x >= 0 or x < -0.15:
        return_value = 0
    elif -0.15 / 2 >= x >= -0.15:
        return_value = 2 * x / 0.15 + 2
    else:
        return_value = - (2 * x / 0.15)
    return return_value


# Plotting the output control variable (Low)
def output_controller_small(x):
    if x <= 0 or x > 0.15:
        return_value = 0
    elif 0.15 / 2 >= x >= 0:
        return_value = 2 * x / 0.15
    else:
        return_value = - (2 * x / 0.15) + 2
    return return_value

# X-axis definition for the output partition plot
t = np.linspace(-0.2, 0.2, 200)
counter = 0
# Define intervals for the output control variable partition
good_triangle = np.zeros(shape=t.shape)
high_triangle = np.zeros(shape=t.shape)
small_triangle = np.zeros(shape=t.shape)

for i in t:
    good_triangle[counter] = output_controller_good(i)
    high_triangle[counter] = output_controller_high(i)
    small_triangle[counter] = output_controller_small(i)
    counter += 1

plt.ylabel("Membership")
plt.xlabel("$\delta$")
plt.title('Fuzzy Partition for the control variable')
plt.plot(t, small_triangle, label='Low Infection rate')
plt.plot(t, good_triangle, label='Good Infection rate')
plt.plot(t, high_triangle, label='High Infection rate')
plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")

plt.show()


# Plotting the input partition for the rate of infected bots (High)
def controller_high(x):
    if 1 >= x >= 0.8:
        return_value = 1
    elif 0.8 >= x >= 0.65:
        return_value = (x / 0.15) - 4.33
    else:
        return_value = 0
    return return_value


# Plotting the input partition for the rate of infected bots (Good)
def controller_good(x):
    if 0.7 >= x >= 0.6:
        return_value = -10 * x + 7
    elif 0.6 >= x >= 0.5:
```

File - C:\Users\berkenutku\Desktop\496_hw3\plot_of_partioning.py

```
78          return_value = (10 * x) - 5
79      else:
80          return_value = 0
81      return return_value
82
83
84  # Plotting the input partition for the rate of infected bots (Low)
85  def controller_small(x):
86      if 0.4 >= x > 0:
87          return_value = 1
88      elif 0.55 >= x >= 0.4:
89          return_value = -(x / 0.15) + 3.666
90      else:
91          return_value = 0
92      return return_value
93
94  # X-axis definition for the input measurement partition
95  k = np.linspace(-0.05, 1, 400)
96
97  counter = 0
98  good_triangle = np.zeros(shape=k.shape)
99  high_triangle = np.zeros(shape=k.shape)
100 small_triangle = np.zeros(shape=k.shape)
101 for x in k:
102     good_triangle[counter] = controller_good(x)
103     high_triangle[counter] = controller_high(x)
104     small_triangle[counter] = controller_small(x)
105     counter += 1
106
107 plt.ylabel("Membership")
108 plt.xlabel("$\pi$")
109 plt.title('Fuzzy partition of the current percentage of the infected bots')
110 plt.plot(k, small_triangle, label='Low Infection rate')
111 plt.plot(k, good_triangle, label='Good Infection rate')
112 plt.plot(k, high_triangle, label='High Infection rate')
113 plt.xticks(np.arange(0, 1.1, 0.1))
114 plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")
115
116 plt.show()
117
```

File - C:\Users\berkenutku\Desktop\496_hw3\plague2.py

```python
from plague import Plague
import numpy as np
import random
import matplotlib.pyplot as plt


def fuzzy_controller(infected_percentage, effective_infection):
    # Calculating the membership for the current effective infection rate
    effective_memberships = [effective_infection_low(effective_infection),
                             effective_infection_good(effective_infection),
                             effective_infection_high(effective_infection)]

    # Calculating the membership for the infected bots rate
    memberships = [calculate_membership_low_infected(infected_percentage),
                   calculate_membership_good_infected(infected_percentage),
                   calculate_membership_high_infected(infected_percentage)]

    best_member = memberships.index(max(memberships))
    best_effective = effective_memberships.index(max(effective_memberships))

    # According to the highest memberships, evaluate the output control variable by using The Max Criterion method
    # The following conditional statements are the implementation of the Table 1 in the report.
    if best_member == 0:
        if best_effective == 0 or best_effective == 1:
            return output_controller_high_positive(memberships[0])
        else:
            return output_controller_low_positive(memberships[0])
    elif best_member == 1:
        if best_effective == 2:
            return output_controller_high_negative(memberships[1])
        else:
            return output_controller_good(memberships[1])
    else:
        if best_effective == 0 or best_effective == 1:
            return output_controller_high_negative(memberships[2])
        else:
            return output_controller_low_negative(memberships[2])


# Calculate the membership for the current effective infection rate (Low)
def effective_infection_low(x):
    if -0.2 >= x >= -1:
        return 1
    elif 0 >= x >= -0.2:
        value = -5 * x
        return value
    else:
        return 0


# Calculate the membership for the current effective infection rate (High)
def effective_infection_high(x):
    if 1 >= x >= 0.2:
        return 1
    elif 0 <= x <= 0.2:
        value = 5 * x
        return value
    else:
        return 0


# Calculate the membership for the current effective infection rate (Good)
def effective_infection_good(x):
    if 0 >= x >= -0.2:
        return_value = 5 * x + 1
    elif 0.2 >= x >= 0:
        return_value = -5 * x + 1
    else:
        return_value = 0
    return return_value


# Calculate the membership for the infected bots (Low)
def calculate_membership_low_infected(x):
    if x <= 0.4:
        return 1
```

File - C:\Users\berkenutku\Desktop\496_hw3\plague2.py

```
    elif x >= 0.6:
        return 0
    else:
        slope = -(1 / 0.2)
        value = 1 - min(1, slope * (0.4 - x))
        return value


# Calculate the membership for the infected bots (Good)
def calculate_membership_good_infected(x):
    if 0.65 >= x >= 0.6:
        return_value = -10 * x + 7
    elif 0.6 >= x >= 0.55:
        return_value = (10 * x) - 5
    else:
        return_value = 0
    return return_value


# Calculate the membership for the infected bots (High)
def calculate_membership_high_infected(x):
    if x >= 0.8:
        return 1
    elif x <= 0.6:
        return 0
    else:
        slope = (1 / 0.2)
        value = 1 - min(1, slope * (0.8 - x))
        return value


# Calculating the output control variable from membership (Good)
def output_controller_good(x):
    first_point = (x - 1) * 3 / 100
    second_point = -((x - 1) * 3 / 100)
    return_value = random.uniform(first_point, second_point)
    return return_value


# Calculating the output control variable from membership (Low (+))
def output_controller_low_positive(x):
    first_point = 0.03 * x
    second_point = -(x-2) * 0.03
    return_value = random.uniform(first_point, second_point)
    return return_value


# Calculating the output control variable from membership (Low (-))
def output_controller_low_negative(x):
    first_point = ((x - 2) * 3 / 100)
    second_point = first_point + 2 * (-0.03 - first_point)
    return_value = random.uniform(first_point, second_point)
    return return_value


# Calculating the output control variable from membership (High (+))
def output_controller_high_positive(x):
    if x == 1:
        return_value = random.uniform(0.09, 0.15)
    else:
        second_point = (x + 0.5) * 0.06
        return_value = random.uniform(second_point, 0.09)
    return return_value


# Calculating the output control variable from membership (High (-))
def output_controller_high_negative(x):
    if x == 1:
        return_value = random.uniform(-0.15, -0.09)
    else:
        second_point = (x + 0.5) * -0.06
        return_value = random.uniform(-0.09, second_point)
    return return_value


plague = Plague()

```

File - C:\Users\berkenutku\Desktop\496_hw3\plague2.py

```
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)
```

File - C:\Users\berkenutku\Desktop\496_hw3\plot_v2.py

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


# Plotting the output control variable (High (-))
def output_controller_high_negative(x):
    if -0.09 >= x >= -0.15:
        return_value = 1
    elif -0.03 >= x >= -0.09:
        return_value = -(100 / 6) * x - 0.5
    else:
        return_value = 0
    return return_value


# Plotting the output control variable (Small (-))
def output_controller_small_negative(x):
    if 0 >= x >= -0.03:
        return_value = -(100 / 3) * x
    elif -0.03 >= x >= -0.06:
        return_value = (100 / 3) * x + 2
    else:
        return_value = 0
    return return_value


# Plotting the output control variable (Good)
def output_controller_good(x):
    if 0 >= x >= -0.03:
        return_value = (100 / 3) * x + 1
    elif 0.03 >= x >= 0:
        return_value = - (100 / 3) * x + 1
    else:
        return_value = 0
    return return_value


# Plotting the output control variable (Small (+))
def output_controller_small_positive(x):
    if 0.06 >= x >= 0.03:
        return_value = - (100 / 3) * x + 2
    elif 0.03 >= x >= 0:
        return_value = (100 / 3) * x
    else:
        return_value = 0
    return return_value


# Plotting the output control variable (High (+))
def output_controller_high_positive(x):
    if 0.15 >= x >= 0.09:
        return_value = 1
    elif 0.09 >= x >= 0.03:
        return_value = (100 / 6) * x - 0.5
    else:
        return_value = 0
    return return_value


# Define x-axis for the output control value
t = np.linspace(-0.15, 0.15, 400)
counter = 0
high_negative = np.zeros(shape=t.shape)
small_negative = np.zeros(shape=t.shape)
good = np.zeros(shape=t.shape)
small_positive = np.zeros(shape=t.shape)
high_positive = np.zeros(shape=t.shape)

for i in t:
    high_negative[counter] = output_controller_high_negative(i)
    small_negative[counter] = output_controller_small_negative(i)
    good[counter] = output_controller_good(i)
    small_positive[counter] = output_controller_small_positive(i)
    high_positive[counter] = output_controller_high_positive(i)

    counter += 1

```

File - C:\Users\berkenutku\Desktop\496_hw3\plot_v2.py

```
plt.ylabel("Membership")
plt.xlabel("$\delta$")
plt.title('Fuzzy Partition for the control variable')
plt.plot(t, high_negative, label='High Infection rate (-)')
plt.plot(t, small_negative, label='Low Infection rate (-)')
plt.plot(t, good, label='Good Infection rate')
plt.plot(t, small_positive, label='Low Infection rate (+)')
plt.plot(t, high_positive, label='High Infection rate (+)')

plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")

plt.show()


# Calculating the membership for the infected bots rate (High)
def controller_high(x):
    if x >= 0.8:
        return 1
    elif x <= 0.6:
        return 0
    else:
        value = 5 * x - 3
        return value


# Calculating the membership for the infected bots rate (Good)
def controller_good(x):
    if 0.65 >= x >= 0.6:
        return_value = -20 * x + 13
    elif 0.6 >= x >= 0.55:
        return_value = (20 * x) - 11
    else:
        return_value = 0
    return return_value


# Calculating the membership for the infected bots rate (Low)
def controller_small(x):
    if x <= 0.4:
        return 1
    elif x >= 0.6:
        return 0
    else:
        value = -5 * x + 3
        return value


# Define x-axis for the infected bots rate
k = np.linspace(-0.05, 1, 400)

counter = 0
good_triangle = np.zeros(shape=k.shape)
high_triangle = np.zeros(shape=k.shape)
small_triangle = np.zeros(shape=k.shape)
for x in k:
    good_triangle[counter] = controller_good(x)
    high_triangle[counter] = controller_high(x)
    small_triangle[counter] = controller_small(x)
    counter += 1

plt.ylabel("Membership")
plt.xlabel("$\pi$")
plt.title('Fuzzy partition of the current percentage of the infected bots')
plt.plot(k, small_triangle, label='Low Infection rate')
plt.plot(k, good_triangle, label='Good Infection rate')
plt.plot(k, high_triangle, label='High Infection rate')
plt.xticks(np.arange(0, 1.1, 0.1))
plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")

plt.show()


# Calculating the membership for the effective infection rate(High)
def effective_controller_high(x):
    if 1 >= x >= 0.2:
        return 1
    elif 0 <= x <= 0.2:
```

File - C:\Users\berkenutku\Desktop\496_hw3\plot_v2.py

```
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 
```