EE449

Homework 3 – Fuzzy Control

Due: 23:55, 12/06/2022

Berkay İPEK

2304814

Sec1

Table of Contents

1. Vaccination v1

1.1 Set Partitioning

- *Partition the sets where the measurement and the output lie into 3 fuzzy sets. Namely, you are to represent [0, 1] and [−0.2, 0.2] with three fuzzy sets for each. You may experiment on different partitioning strategies by examining the performances. Decide your favorite partitioning and plot your fuzzy partitions for both sets in Python and include them in your report. Explain how you decide that partitioning.*

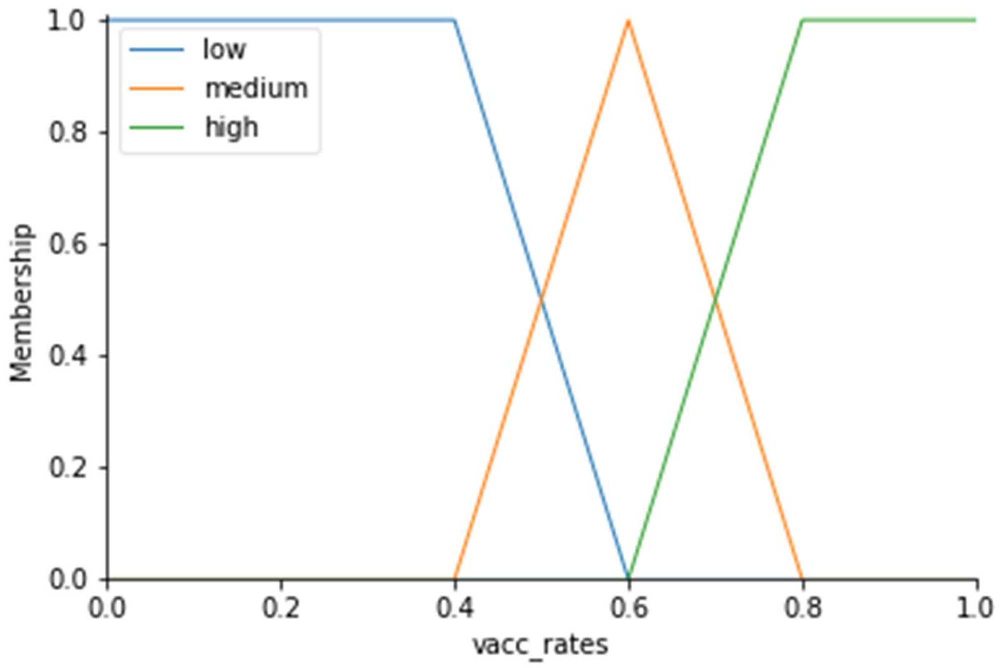


Figure 1: Vaccination Rate Partitioning

The center of medium should be 0.6, which is desired output. The control should give 0 when only vaccination rate reaches 60%. For medium and high rates, I changed initial parameters to see their effects. As high rate starting point to increase decreases (or low rate starting point to decrease increases), it is converging in a faster way. The reason is that the system labels a vaccination rate as medium in earlier stage. Therefore, it is output control will be smaller. If we want to get flawless result at the point 0.6, we can change these points (0.4 and 0.8) to wider points (0.3 and 0.9). However, I chose to converge them as soon as possible. Therefore, I gave their parameters as like that.

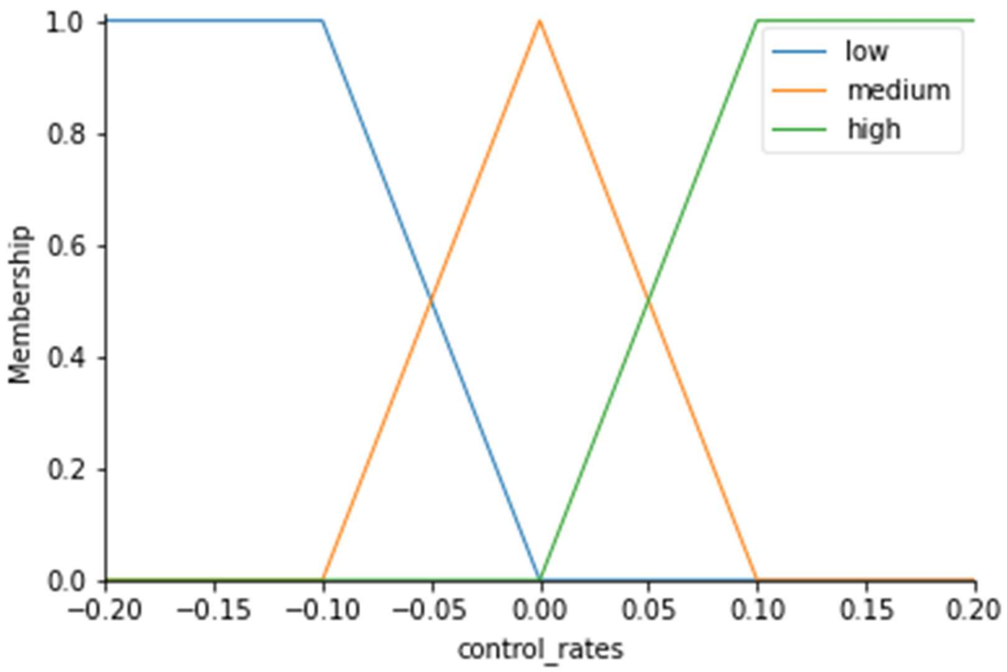


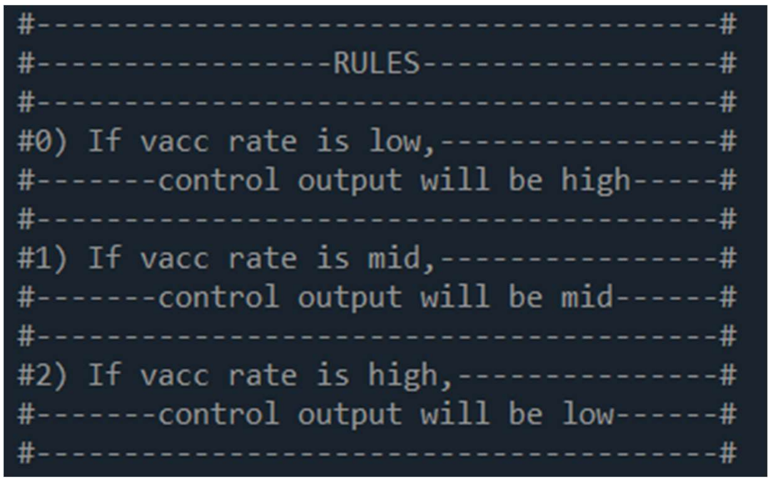
Figure 2: Control Rate Partitioning

The center of medium should be 0.0, which is desired output. The control should give 0 when only vaccination rate reaches 60%. For medium and high rates, I changed initial parameters to see their effects. The results were similar to previous one. As the medium one range increases, convergence time increases also since system response is much larger. Therefore, I gave these parameters like that.

For both input sets, I used trimf and trampf methods in the library of skfuzzy. (See the code in the appendix)

1.2 Fuzzy Control Rules

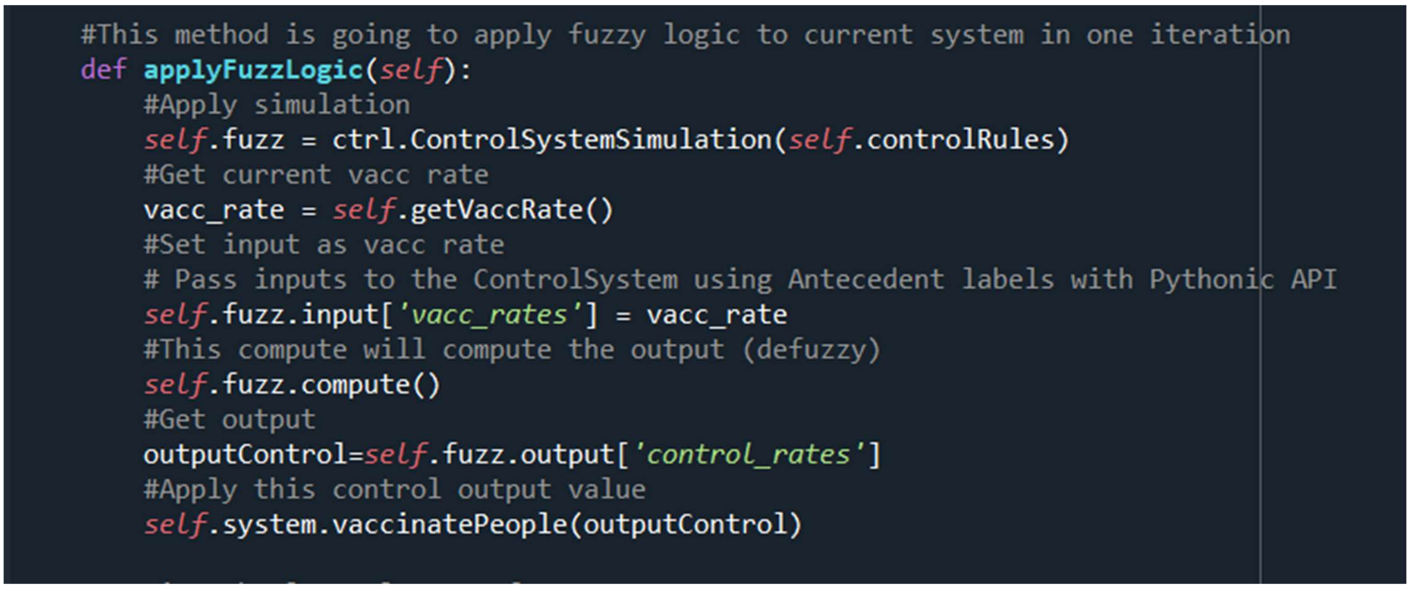
Considering your partitions, list your control rules.



```
#-----------------------------------------#
#----------------RULES-----------------#
#-----------------------------------------#
#0) If vacc rate is low,----------------#
#-------control output will be high-----#
#-----------------------------------------#
#1) If vacc rate is mid,----------------#
#-------control output will be mid------#
#-----------------------------------------#
#2) If vacc rate is high,---------------#
#-------control output will be low------#
#-----------------------------------------#
```

Figure 3: Control Rules

1.3 Fuzzification and Defuzzification Interface



```
#This method is going to apply fuzzy logic to current system in one iteration
def applyFuzzLogic(self):
    #Apply simulation
    self.fuzz = ctrl.ControlSystemSimulation(self.controlRules)
    #Get current vacc rate
    vacc_rate = self.getVaccRate()
    #Set input as vacc rate
    # Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
    self.fuzz.input['vacc_rates'] = vacc_rate
    #This compute will compute the output (defuzzy)
    self.fuzz.compute()
    #Get output
    outputControl=self.fuzz.output['control_rates']
    #Apply this control output value
    self.system.vaccinatePeople(outputControl)
```

Figure 4: Fuzzification and Defuzzification Interface

Firstly, I defined control rules as explained in previous part. Then, assigned it to a member of a class, called as controlRules(it is a list of rule objects in skfuzzy library). Then, I imported control class from skfuzzy library. It has a method called as ControlSystemSimulation with an argument of rules. That's how my fuzzification works.

For defuzzification, I used compute method for control system simulation variable in the skfuzzy library. Then, I got output by 'control_rates' key of the dictionary. As a result, I gave this control output to system.

1.4 Simulation

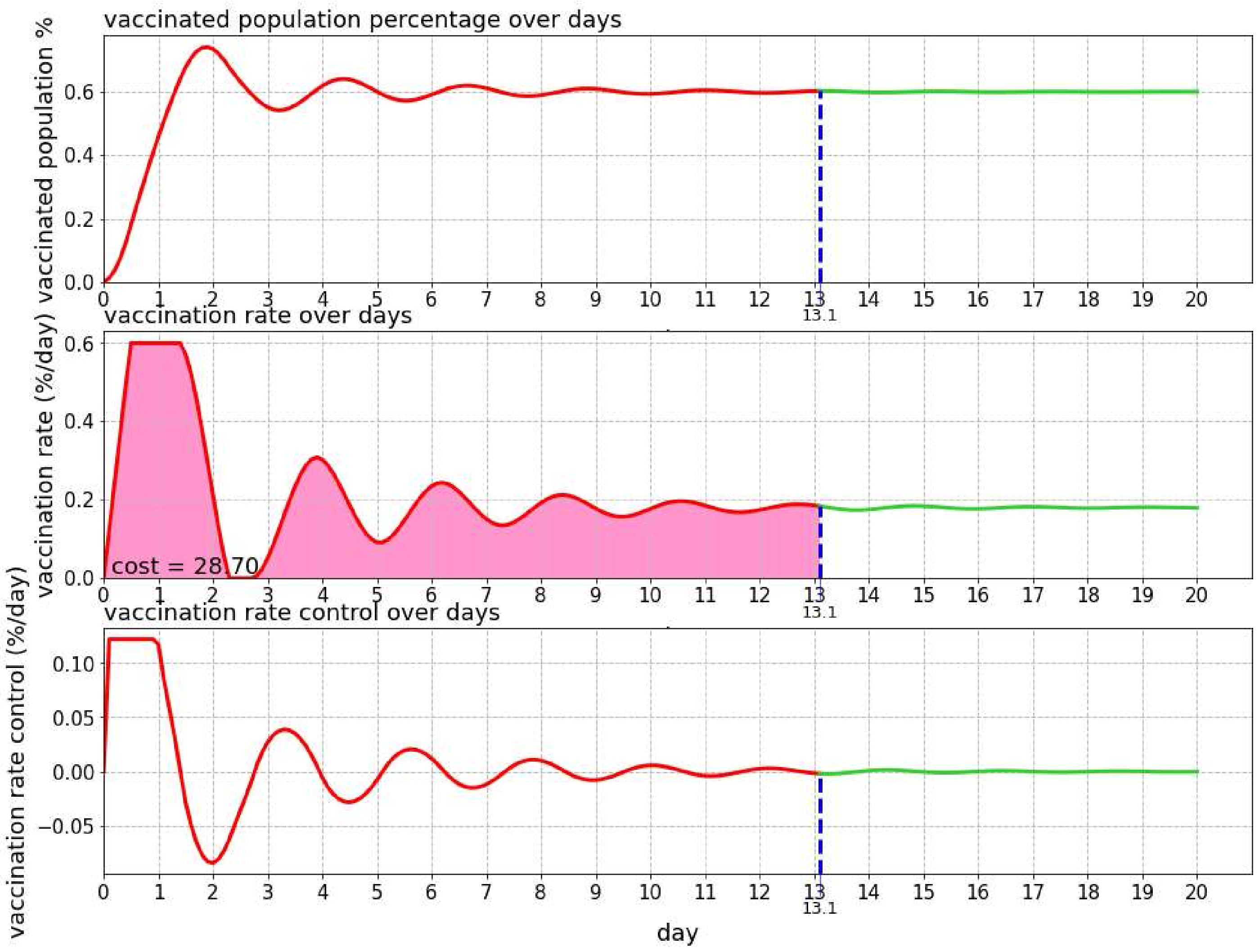


Figure 5: Simulation Result of Vaccination-1

2. Vaccination v2

2.1 Set Partitioning

- *Partition the sets where the measurements lie into 3 fuzzy sets. Namely, you are to represent [0, 1] and [−1, 1] with three fuzzy sets for each. Partition the output set into 5 fuzzy sets [−0.2, 0.2]. You may experiment on different partitioning strategies by examining the performances. Decide your favorite partitioning and plot your fuzzy partitions for both sets in Python and include them in your report. Explain how you decide that partitioning.*

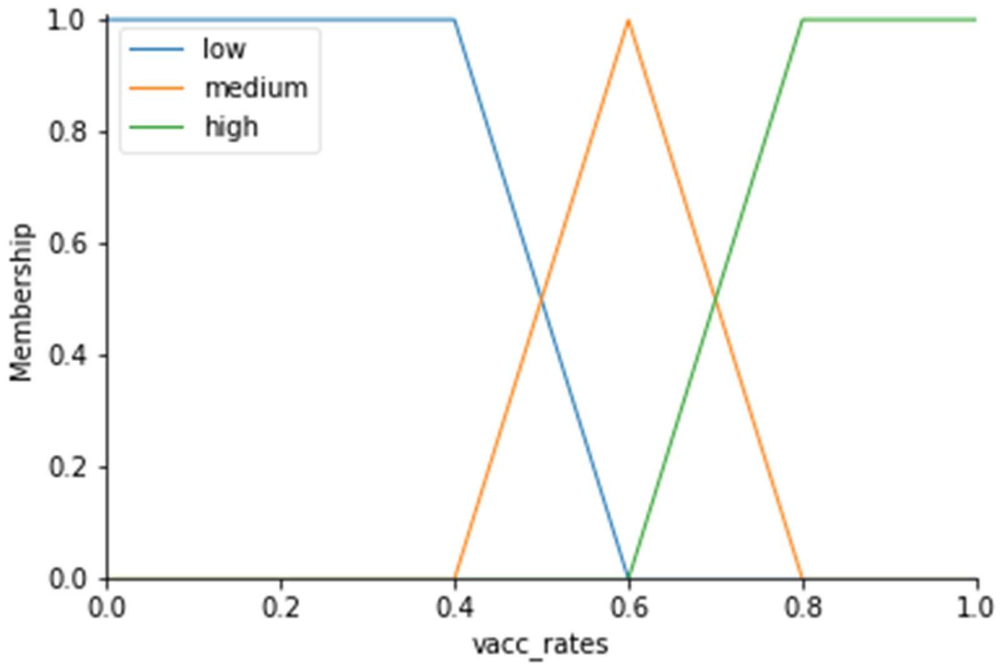


Figure 6: Vaccination Rate Partitioning

The center of medium should be 0.6, which is desired output. The control should give 0 when only vaccination rate reaches 60%. For medium and high rates, I changed initial parameters to see their effects. As high rate starting point to increase decreases (or low rate starting point to decrease increases), it is converging in a faster way. The reason is that the system labels a vaccination rate as medium in earlier stage. Therefore, it is output control will be smaller. If we want to get flawless result at the point 0.6, we can change these points (0.4 and 0.8) to wider points (0.3 and 0.9). However, I chose to converge them as soon as possible. Therefore, I gave their parameters as like that.

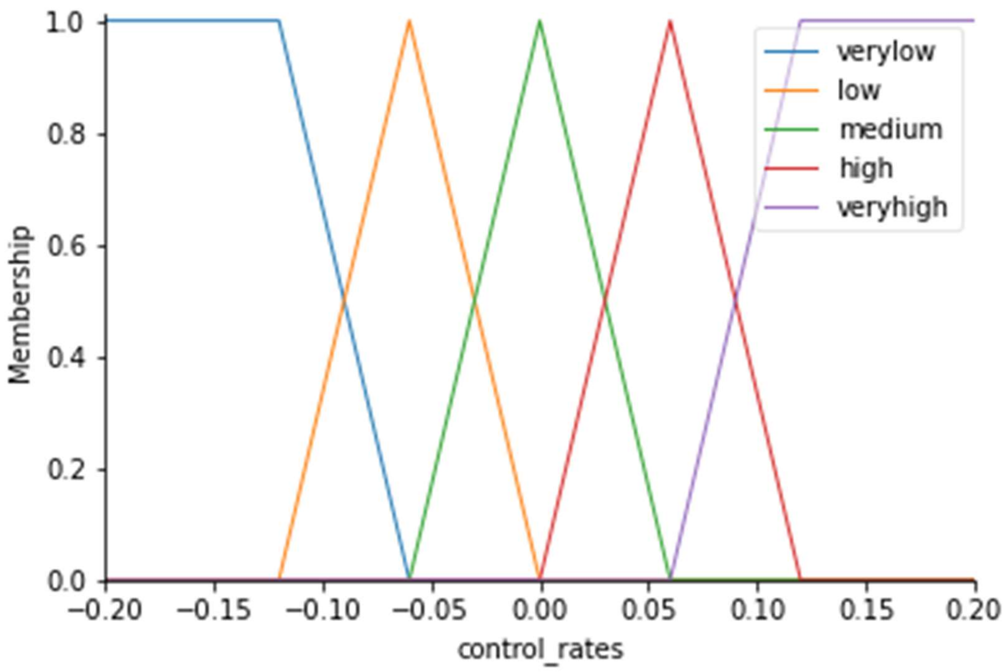


Figure 7: Control Rate Partitioning

The center of medium should be 0.0, which is desired output. The control should give 0 when only vaccination rate reaches 60%. For medium and high rates, I changed initial parameters to see their effects. The results were similar to previous one. As the medium one range increases, convergence time increases also since system response is much larger. Therefore, I gave these parameters like that.

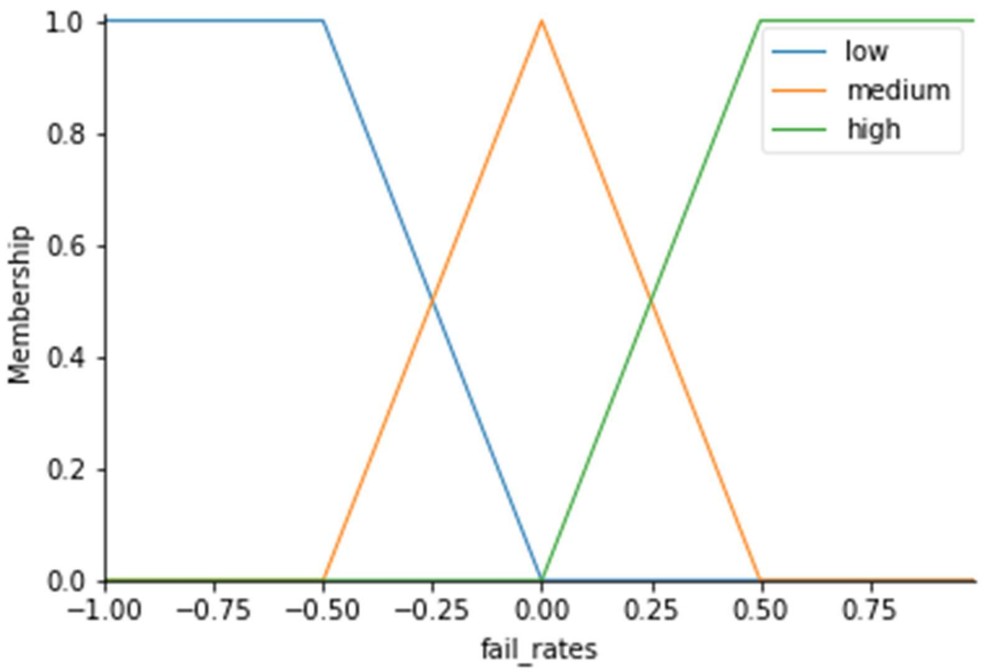


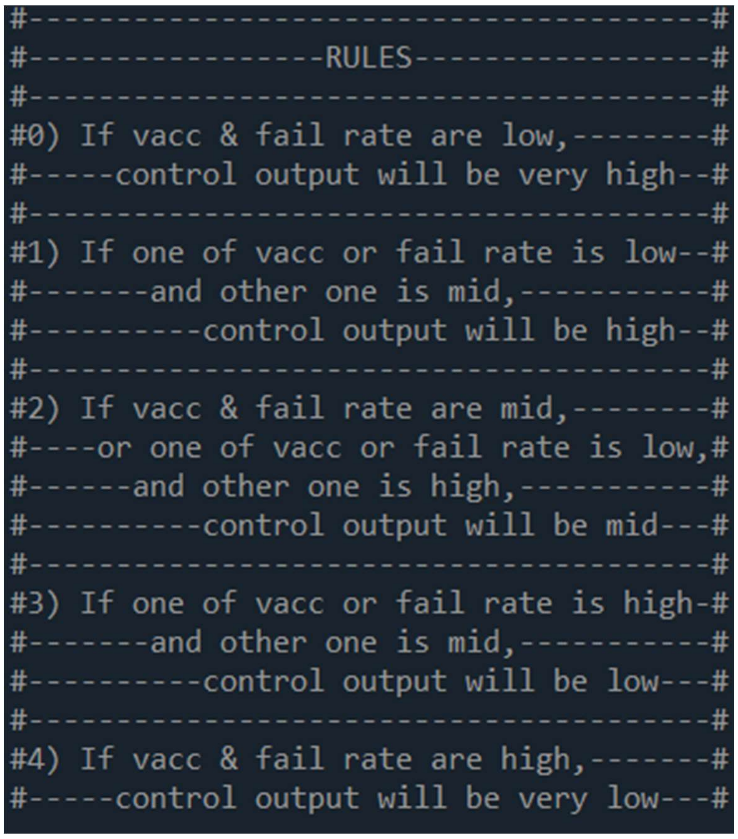
Figure 8: Failure Rate Partitioning

The center of medium should be 0.0, which is desired output. The control should give 0 when only vaccination rate reaches 60%. For medium and high rates, I changed initial parameters to see their effects. The results were similar to previous one. As the medium one range increases, convergence time (settle time) increases also since system response is much larger. However, as it increases, overshoot problem decreases. Moreover, at some point it starts to be underdamped situation rather than overshooting situation.

For all input sets, I used trimf and trampf methods in the library of skfuzzy. (See the code in the appendix).

2.2 Fuzzy Control Rules

- *Considering your partitions, list your control rules. You may experiment on different set of rules by examining the performances. Decide your favorite set of rules and list them. Explain how you decide that rule set.*



```
#------------------------------------#
#----------------RULES---------------#
#------------------------------------#
#0) If vacc & fail rate are low,--------#
#-----control output will be very high--#
#------------------------------------#
#1) If one of vacc or fail rate is low--#
#-------and other one is mid,----------#
#---------control output will be high--#
#------------------------------------#
#2) If vacc & fail rate are mid,--------#
#----or one of vacc or fail rate is low,#
#------and other one is high,----------#
#---------control output will be mid---#
#------------------------------------#
#3) If one of vacc or fail rate is high-#
#-------and other one is mid,----------#
#---------control output will be low---#
#------------------------------------#
#4) If vacc & fail rate are high,-------#
#-----control output will be very low---#
```

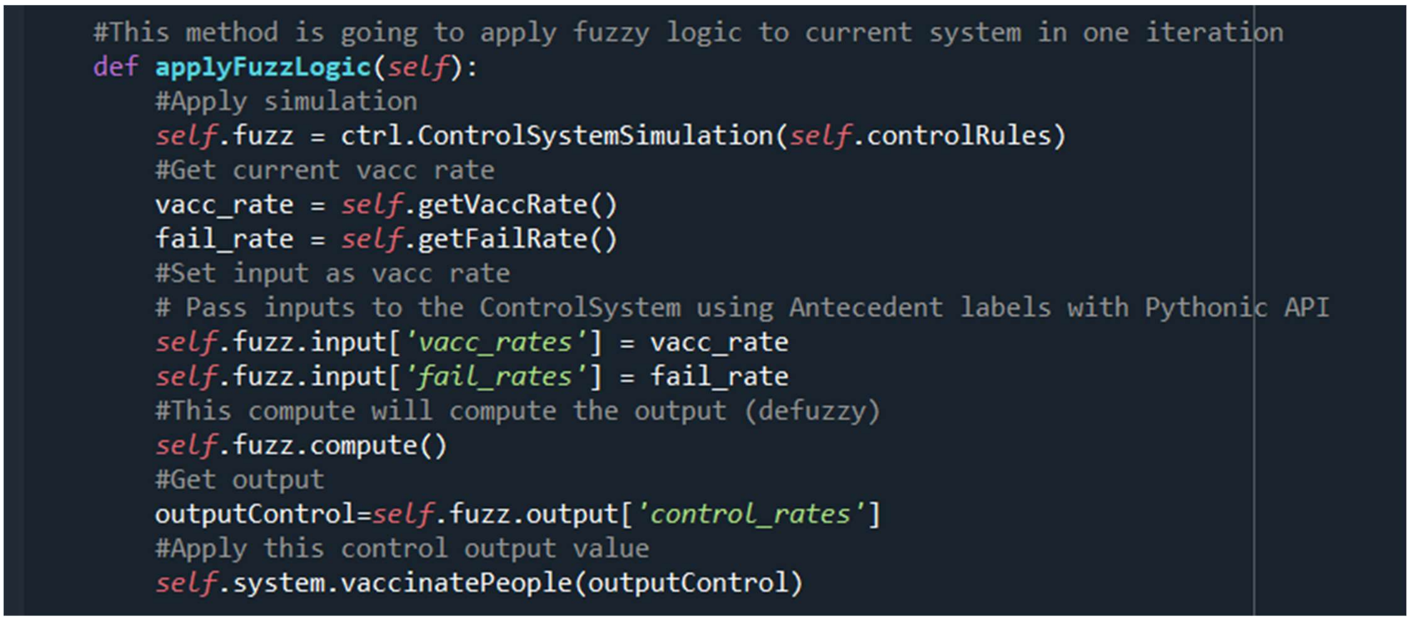
Figure 9: Control Rules

We know that if vaccination rate and failure rate are fully mid, then we got 60% vaccination rate. Also, if we get vaccination rate of high and failure rate of low, we can achieve this rate. For example, if we have vaccination rate of 61% and failure rate of %1 then we will achieve real vaccination rate of %60. Also, we can achieve it with having vaccination rate of low and failure rate of high. Therefore, control output should be middle when these situations occur (Rule 2).

Rules 0 and 4 state that, when both rates have same partitioning, control should be very low or high. It is because as vaccination & failure rates are high (or low), control should be very low (or high) to decrease (increase) vaccination rate.

Rules 1 and 3 state that when one of them is middle, which is desired situation, other one is low or high, then we should give high or low to get balanced.

2.3 Implementation and Simulation



```
    #This method is going to apply fuzzy logic to current system in one iteration
    def applyFuzzLogic(self):
        #Apply simulation
        self.fuzz = ctrl.ControlSystemSimulation(self.controlRules)
        #Get current vacc rate
        vacc_rate = self.getVaccRate()
        fail_rate = self.getFailRate()
        #Set input as vacc rate
        # Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
        self.fuzz.input['vacc_rates'] = vacc_rate
        self.fuzz.input['fail_rates'] = fail_rate
        #This compute will compute the output (defuzzy)
        self.fuzz.compute()
        #Get output
        outputControl=self.fuzz.output['control_rates']
        #Apply this control output value
        self.system.vaccinatePeople(outputControl)
```

Figure 10: Fuzzification and Defuzzification Interface

It is same with the previous part expect for input size. Failure rate increases,

Firstly, I defined control rules as explained in previous part. Then, assigned it to a member of a class, called as controlRules(it is a list of rule objects in skfuzzy library). Then, I imported control class from skfuzzy library. It has a method called as ControlSystemSimulation with an argument of rules. That's how my fuzzification works.

For defuzzification, I used compute method for control system simulation variable in the skfuzzy library. Then, I got output by 'control_rates' key of the dictionary. As a result, I gave this control output to system.

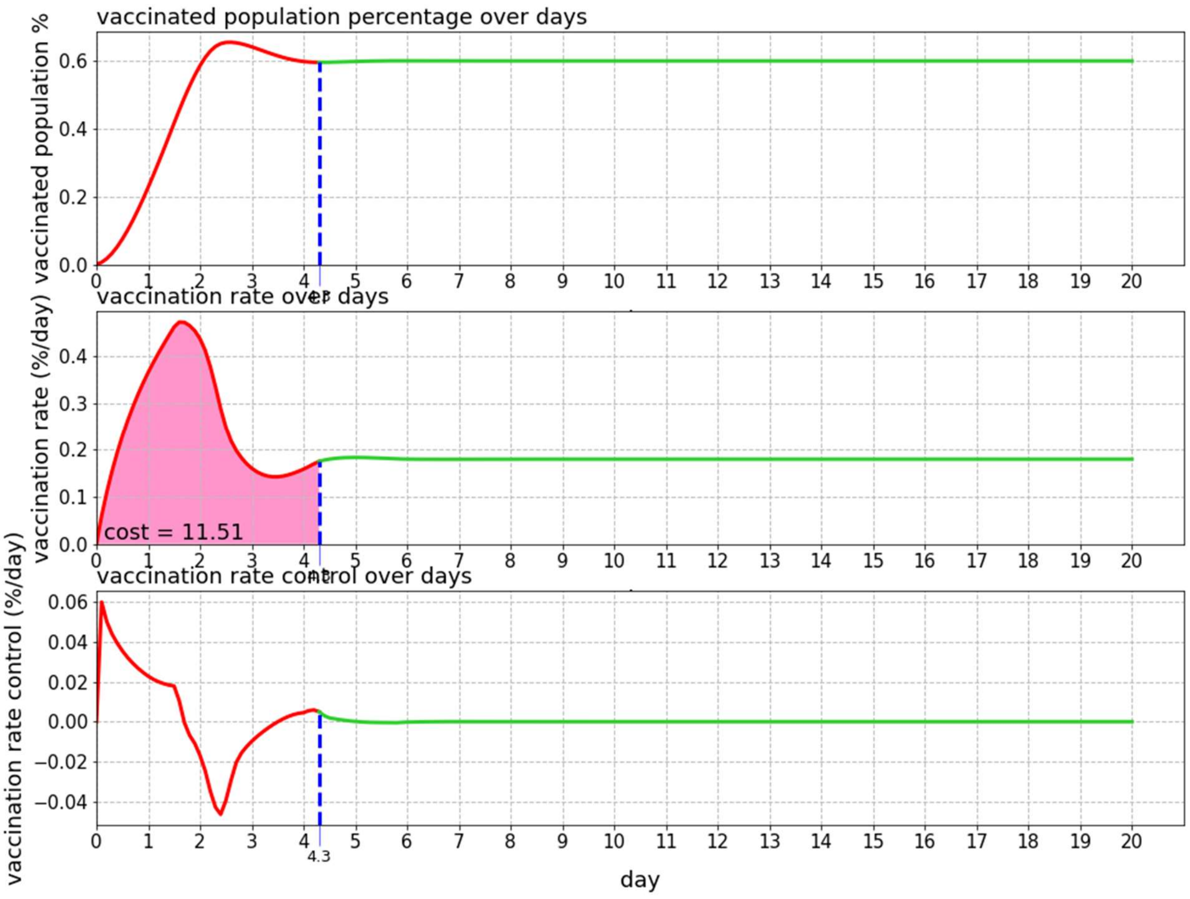


Figure 11: Simulation Result of Vaccination-2

2.4 Comparison

Main difference is overshooting problems. For vacc-1 version, convergence time is very long, and cost is quite high since the system has not enough input parameters. However, for vacc-2 version, convergence time is shorter, and cost is lower than vacc-1 version. As system gets proper inputs, results are getting better as this experiment show. Therefore, when we are going to simulate, firstly problem should be described in detail, so vacc-2 version is better vacc-1 version.

Vacc-1 version have an advantage of being simple and calculation time. When system needs more iteration and has negligible failure rate, rather than having vacc-2 can be chosen since it is easy to build and faster to compute.

3. Appendix

3.1 Vaccination v1 Code

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from vaccination import Vaccination
class FuzzSystem():
    #Constructor Method
    def __init__(self):
        #Inspiried from these websites:

#https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_tipping_problem_newapi.html

#https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_control_system_advanced.html#exam
ple-plot-control-system-advanced-py
        # New Antecedent/Consequent objects hold universe variables and membership
        # functions
        self.vacc = ctrl.Antecedent(np.arange(0,1.01, 0.01), 'vacc_rates')

        self.control = ctrl.Consequent(np.arange(-0.2,0.21, 0.05), 'control_rates')

        # Custom membership functions can be built interactively with a familiar,
        # Pythonic API
        #  Vacc Set Partioning
        self.vacc['low'] = fuzz.trapmf(self.vacc.universe, [0, 0, 0.4, 0.6])
        self.vacc['medium'] = fuzz.trimf(self.vacc.universe,[0.4, 0.6, 0.8])
        self.vacc['high'] =  fuzz.trapmf(self.vacc.universe, [0.6, 0.8, 1, 1])
        # Printing set partioning
        self.vacc.view()
        # Custom membership functions can be built interactively with a familiar,
        # Pythonic API
        # Control Set Partioning
        self.control['low'] = fuzz.trapmf(self.control.universe, [-0.2, -0.2, -0.1,0])
        self.control['medium'] = fuzz.trimf(self.control.universe, [-0.1, 0, 0.1])
        self.control['high'] = fuzz.trapmf(self.control.universe,[0, 0.1, 0.2,0.2])
        # Printing set partioning
        self.control.view()

        #----------------------------------------#
        #-----------------RULES------------------#
        #----------------------------------------#
        #0) If vacc rate is low,----------------#
        #-------control output will be high-----#
        #----------------------------------------#
        #1) If vacc rate is mid,----------------#
        #-------control output will be mid------#
        #----------------------------------------#
        #2) If vacc rate is high,---------------#
        #-------control output will be low------#
        #----------------------------------------#
        rule0 = ctrl.Rule(antecedent=self.vacc['low'] ,
                          consequent=self.control['high'], label='rule high')

        rule1 = ctrl.Rule(antecedent= self.vacc['medium'],
                          consequent=self.control['medium'], label='rule mid')

        rule2 = ctrl.Rule(antecedent= self.vacc['high'],
                          consequent=self.control['low'], label='rule low')

        #Apply Rules
        self.controlRules = ctrl.ControlSystem(rules=[rule0, rule1, rule2])
        #Initialize system
        self.system=Vaccination()

    #This method is going to apply fuzzy logic to current system in one iteration
    def applyFuzzLogic(self):
        #Apply simulation
        self.fuzz = ctrl.ControlSystemSimulation(self.controlRules)
        #Get current vacc rate
        vacc_rate = self.getVaccRate()
        #Set input as vacc rate
        # Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
        self.fuzz.input['vacc_rates'] = vacc_rate
        #This compute will compute the output (defuzzy)
        self.fuzz.compute()
        #Get output
        outputControl=self.fuzz.output['control_rates']
        #Apply this control output value
        self.system.vaccinatePeople(outputControl)

    #Getting the last element of vacc rate curve
    def getLastRate(self):
        return self.system.vaccination_rate_curve_[-1]
    #Getting the current vacc rate
    def getVaccRate(self):
        return self.system.checkVaccinationStatus()[0]
    #Getting the current fail rate
    def getFailRate(self):
        return self.system.checkVaccinationStatus()[1]



berkay=FuzzSystem() #Main system
flag=True #Flag will be True until equilibrium point
cost=0      #Initial Cost value
stopDiff=0.00001 #Stopper diff
for slot in range(200):
    prev_vacc_rate=berkay.getVaccRate() #Get previous vacc rate
    berkay.applyFuzzLogic() #Apply the control by calling the method
    vacc_rate =berkay.getVaccRate() #Get current vacc rate
    if(flag==True):
        cost+=berkay.getLastRate()#Sum all costs until equilibrium point
    currentDiff=abs(vacc_rate-prev_vacc_rate) #Calculate the diff between percentages of 
consc two iteration
    if(currentDiff< stopDiff ) and flag==True: #Check if the difference is enough small
        flag=False #Flag will be Flase when equilibrium point is reached
        point_ss=slot #record the time when there is a equibilirium

berkay.system.viewVaccination(point_ss = point_ss, vaccination_cost = cost, 
filename='vacc-v1')
```

3.2 Vaccination v2 Code

```python
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from vaccination import Vaccination
class FuzzSystem():
    #Constructor Method
    def __init__(self):
        #Inspiried from these websites:

#https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_tipping_problem_newapi.html

#https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_control_system_advanced.html#exam
ple-plot-control-system-advanced-py
        # New Antecedent/Consequent objects hold universe variables and membership
        # functions
        self.vacc = ctrl.Antecedent(np.arange(0,1.01, 0.01), 'vacc_rates')
        self.fail = ctrl.Antecedent( np.arange(-1,1, 0.01), 'fail_rates')
        self.control = ctrl.Consequent(np.arange(-0.20,0.21, 0.02), 'control_rates')


        # Custom membership functions can be built interactively with a familiar,
        # Pythonic API
        #  Vacc Set Partioning
        self.vacc['low'] = fuzz.trapmf(self.vacc.universe, [0, 0, 0.4, 0.6])
        self.vacc['medium'] = fuzz.trimf(self.vacc.universe,[0.4, 0.6, 0.8])
        self.vacc['high'] =  fuzz.trapmf(self.vacc.universe, [0.6, 0.8, 1, 1])
        # Printing set partioning
        self.vacc.view()
        # Custom membership functions can be built interactively with a familiar,
        # Pythonic API
        # Failure Rate Set Partioning
        self.fail['low'] = fuzz.trapmf(self.fail.universe, [-1, -1, -0.5, 0.0])
        self.fail['medium'] = fuzz.trimf(self.fail.universe,[-0.5, 0.0, 0.5])
        self.fail['high'] =  fuzz.trapmf(self.fail.universe, [0.0, 0.5, 1.0, 1.0])
        # Printing set partioning
        self.fail.view()
        # Custom membership functions can be built interactively with a familiar,
        # Pythonic API
        # Control Set Partioning
        self.control['verylow'] = fuzz.trapmf(self.control.universe, [-0.2, -0.2,
-0.12,-0.06])
        self.control['low'] = fuzz.trimf(self.control.universe, [-0.12,-0.06,0])
        self.control['medium'] = fuzz.trimf(self.control.universe, [-0.06, 0, 0.06])
        self.control['high'] = fuzz.trimf(self.control.universe,[0,0.06, 0.12])
        self.control['veryhigh'] = fuzz.trapmf(self.control.universe,[0.06, 0.12, 0.2,0.2])
        # Printing set partioning
        self.control.view()
        #----------------------------------------#
        #-----------------RULES------------------#
        #----------------------------------------#
        #0) If vacc & fail rate are low,--------#
        #-----control output will be very high--#
        #----------------------------------------#
        #1) If one of vacc or fail rate is low--#
        #-------and other one is mid,-----------#
        #----------control output will be high--#
        #----------------------------------------#
        #2) If vacc & fail rate are mid,--------#
        #----or one of vacc or fail rate is low,#
        #------and other one is high,-----------#
        #----------control output will be mid---#
        #----------------------------------------#
        #3) If one of vacc or fail rate is high-#
        #-------and other one is mid,-----------#
        #----------control output will be low---#
        #----------------------------------------#
        #4) If vacc & fail rate are high,-------#
        #-----control output will be very low---#

        rule0 = ctrl.Rule(antecedent=(self.vacc['low'] & self.fail['low']),
                          consequent=self.control['veryhigh'], label='rule very high')


        rule1 = ctrl.Rule(antecedent=( (self.vacc['low'] & self.fail['medium']) |
                                        (self.vacc['medium'] & self.fail['low']) ),
                            consequent=self.control['high'], label='rule high')

        rule2 = ctrl.Rule(antecedent=( (self.vacc['medium'] & self.fail['medium']) |
                                        (self.vacc['high'] & self.fail['low']) |
                                        (self.vacc['low'] & self.fail['high']) ),
                            consequent=self.control['medium'], label='rule medium')

        rule3 =  ctrl.Rule(antecedent=( (self.vacc['high'] & self.fail['medium']) |
                                        (self.vacc['medium'] & self.fail['high']) ),
                            consequent=self.control['low'], label='rule low')

        rule4 = ctrl.Rule(antecedent=(self.vacc['high'] & self.fail['high']),
                            consequent=self.control['verylow'], label='rule very low')
        #Apply Rules
        self.controlRules = ctrl.ControlSystem(rules=[rule0, rule1, rule2, rule3, rule4])
        #Initialize system
        self.system=Vaccination()

    #This method is going to apply fuzzy logic to current system in one iteration
    def applyFuzzLogic(self):
        #Apply simulation
        self.fuzz = ctrl.ControlSystemSimulation(self.controlRules)
        #Get current vacc rate
        vacc_rate = self.getVaccRate()
        fail_rate = self.getFailRate()
        #Set input as vacc rate
        # Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
        self.fuzz.input['vacc_rates'] = vacc_rate
        self.fuzz.input['fail_rates'] = fail_rate
        #This compute will compute the output (defuzzy)
        self.fuzz.compute()
        #Get output
        outputControl=self.fuzz.output['control_rates']
        #Apply this control output value
        self.system.vaccinatePeople(outputControl)

    #Getting the last element of vacc rate curve
    def getLastRate(self):
        return self.system.vaccination_rate_curve_[-1]
    #Getting the current vacc rate
    def getVaccRate(self):
        return self.system.checkVaccinationStatus()[0]
    #Getting the current fail rate
    def getFailRate(self):
        return self.system.checkVaccinationStatus()[1]


berkay=FuzzSystem() #Main system
flag=True #Flag will be True until equilibrium point
cost=0      #Initial Cost value
stopDiff=0.0001 #Stopper diff
for slot in range(200):
    prev_vacc_rate=berkay.getVaccRate() #Get previous vacc rate
    berkay.applyFuzzLogic() #Apply the control by calling the method
    vacc_rate =berkay.getVaccRate() #Get current vacc rate
    fail_rate = berkay.getFailRate() #Get current failure rate
    if(flag==True):
        cost+=berkay.getLastRate()#Sum all costs until equilibrium point
    currentDiff=abs(vacc_rate-prev_vacc_rate) #Calculate the diff between percentages of
consec two iteration
    if(currentDiff< stopDiff ) and flag==True: #Check if the difference is enough small
        flag=False #Flag will be Flase when equilibrium point is reached
        point_ss=slot #record the time when there is a equibilirium

berkay.system.viewVaccination(point_ss = point_ss, vaccination_cost = cost,
filename='vacc-v2')
```