EE449 HW3 FUZZY CONTROL OF VACCINATION

Kutay Uğurlu 2232841

June 20, 2021

1 Plague v1

1.1 Set Partitioning

The vaccination rate to achieve the herd immunity is given as 0.6. Therefore, the vaccination rate should converge to this value in the end. Hence,in the proximity of this value, a lower δ in magnitude should be used and the π values values around 0.6 can be considered as the members of **Optimal Vaccination Rate** set. For the π values lower than 0.6, the effective vaccination rate should be increased, hence this values should have higher memberships in the **Low Vaccination Rate** set. On the other hand, since vaccination rates higher than 0.6 increase the probability of failure, the infection rates should be negative to have less infection. Using the same approach, vaccination rates in this range should have high membership in the **High Vaccination Rate** set. The margin around **Optimal Vaccination Rate** is determined experimentally, and the remainings are set accordingly.

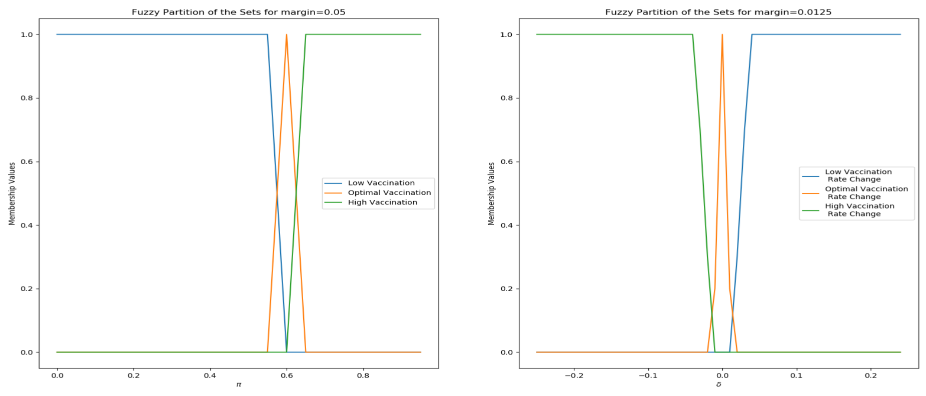

Figure 1: Set Partitioning for π and δ

1.2 Fuzzy Control Rules

The fuzzy control rules used to stabilize the vaccination is given below in the list:

- If vaccination rate is lower than the desired rate of %60, then the output control variable vaccination rate change is positive, to increase the vaccination rate per day so that vaccination rate rises.
- If vaccination rate is higher than the desired rate of %60, then the output control variable vaccination rate change is negative, to decrease the vaccination rate per day so that vaccination rate falls.
- If vaccination rate is around %60, the change δ will be approximately zero, to make the vaccination rate fluctuate around the desired value with decreasing steps.

The partitioning corresponding to the given rules above, are represented with blue, green, orange curves in *Figure 1* respectively.

1.3 Fuzzification and Deffuzification Interface

The controller is implemented as follows:

- The membership values are calculated using the vaccination rate at each iteration.
- The output control variable, change in vaccination rate per day, are calculated by using the maximum membership value, and the corresponding set. The α-cut corresponding to the membership is found, then using **Maximum Criterion Approach**, a random number belonging to this interval is returned.

As a side note, the method is not deterministic, resulting from the randomized control variable selection in the Maximum Criterion method. So, results may change at each run. However, having partitions that span relatively small range of real numbers did not create any problems in terms of convergence.

1.4 Simulation

The result obtained via vaccination.viewVaccination() is presented below. The point where the curve reached the equilibrium is found by checking if vaccination rate remains constant for a determined value of iterations.

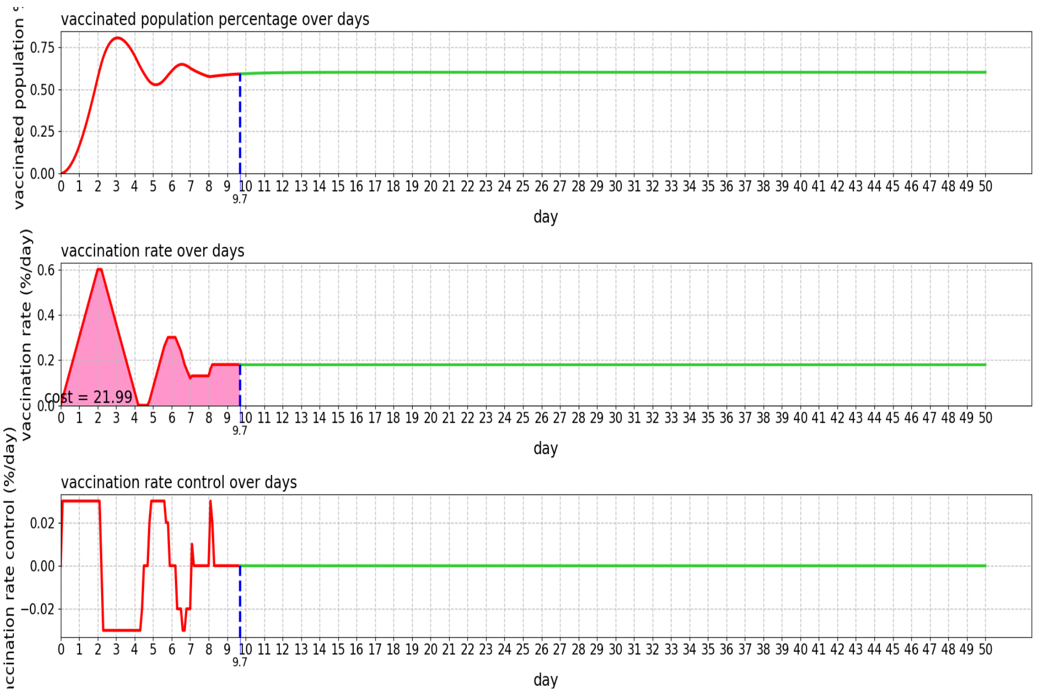

Figure 2: Vaccination v1 vaccination rate tendencies

2 Vaccination v2

To decrease the overshoot and to increase the convergence rate, the number of inputs are increased in this section, namely from solely π to π and $\dot{\pi}$.

2.1 Set Partitioning

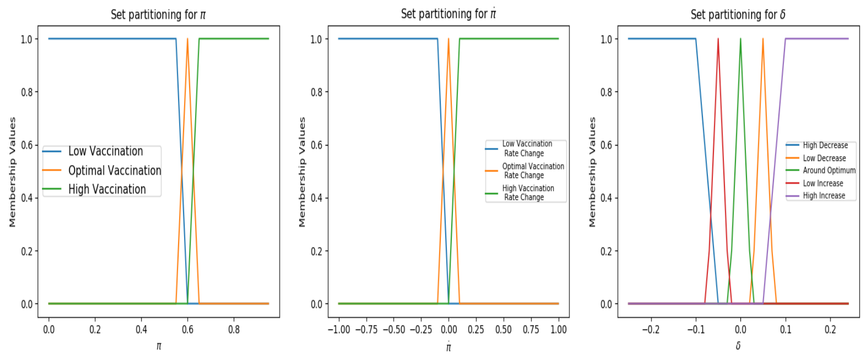

Figure 3: Vaccination v2 Set partitioning

The above partitioning was determined as follows:

- The partition for π is determined as the same as in part v1. The proximity of 0.6 is determined to be the optimal partition, with triangular membership function, *i.e.* decreasing membership around 0.6 with a margin of 0.05.
- The same approach was also utilized to partition $\dot{\pi}$. However, a margin of 0.1 and center 0 is used to partition the vaccination rate per day as **positive**, **negative** and **around zero**.
- The δ partitioning was slightly different from that in part v1. In addition to the fact that delta is partitioned to 5 sets in this case, the "edge cases" were not partitioned using triangular membership function, but using the one in *Figure 3*. This helps model returning higher δ values for these "extreme" cases and returning vaccinated population rate to equilibrium faster.

As a final note, the margins of set partitions are determined experimentally to provide the best performance.

2.2 Fuzzy Control Rules

The controller has 9 fuzzy control rules. The main approach that was used to determine the rules was as follows:

- The Vaccination Rate per day should be increased if the vaccinated percentage is higher than 0.6.
- The Vaccination Rate per day should be decreased if the vaccinated percentage is lower than 0.6.
- If Vaccination Rate per day is going to be increased, the amount of increase depend on effective vaccination, $\dot{\pi}$. If $\dot{\pi}$ is negative or around zero, the increase will be high, otherwise a low amount of increase is applied.
- If Vaccination Rate per day is going to be decreased, the amount of decrease depend on effective vaccination, $\dot{\pi}$. If $\dot{\pi}$ is positive or around zero, the decrease will be high, otherwise a low amount of decrease is applied.

However, it was observed in the experiments that optimal vaccinated population percentage and positive vaccination rate per day requires a strong decrease in π, to decrease the overshoot and the oscillation around 0.6. The resultant rule of tables according to the above rules are given in below:

Table 1: Fuzzy Controller Rules Table

π \ $\dot{\pi}$	Low Rate	Optimal Rate	High Rate
Low Rate	Increase High	Increase High	Increase Low
Optimal Rate	Optimal	Optimal	Decrease High
High Rate	Decrease Low	Decrease High	Decrease High

2.3 Fuzzification and Defuzzification Interface

The controller is implemented as follows:

- The membership values are calculated using the vaccination rate at each iteration.
- The output control variable, change in vaccination rate per day, are calculated by using the maximum membership values of the π and $\dot{\pi}$ input variables. Once the maximum memberships are found, the corresponding output set partitioning are determined using the fuzzy control rules given in *Table 1.* Using that set partitioning, the α-cut corresponding to the membership value of π is found, and using **Maximum Criterion** approach, an output control variable is selected.

As a side note, the method is not deterministic, resulting from the randomized control variable selection in the Maximum Criterion method. So, results may change at each run. However, having partitions that span relatively small range of real numbers did not create any problems in terms of convergence.

2.4 Simulation

The result obtained via vaccination.viewVaccination() is presented below. The point where the curve reached the equilibrium is found by checking if vaccination rate remains constant for a determined value of iterations, with an error margin of 0.005.

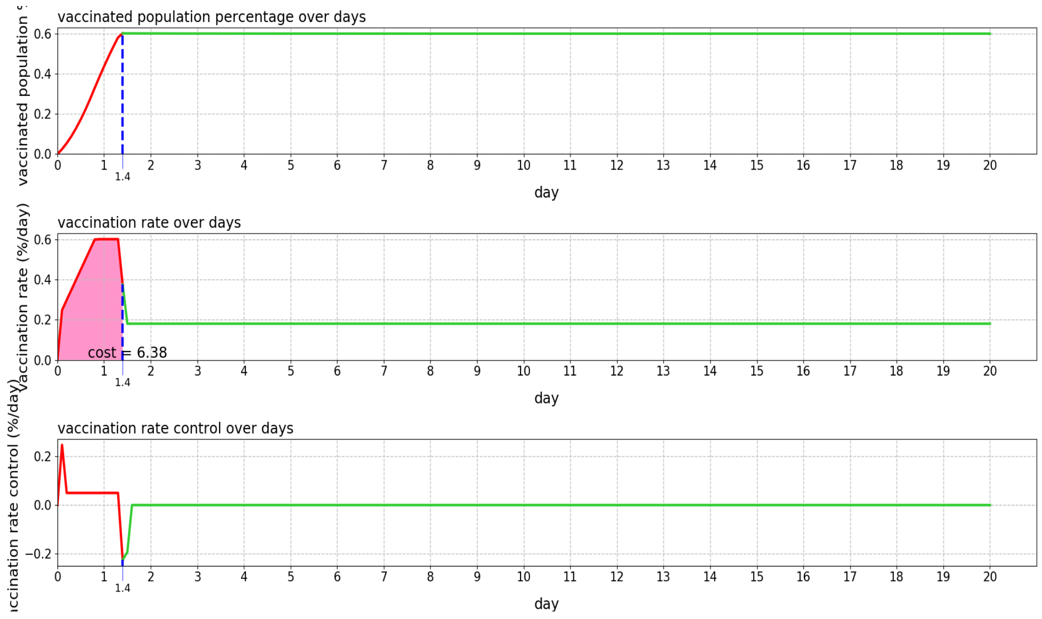

Figure 4: Vaccination v2 vaccination rate tendencies

3 Comparison and Discussion

For the both cases, the fuzzy controller was able to fix the vaccinated population rate π at 0.6 and $\dot{\pi}$ at 0.18, which is the failure rate defined in vaccination.py as $\frac{\pi_{final}^2}{2} = \frac{0.6^2}{2}$. However, in terms of convergence rate, overshoot and hence the cost, there are some differences between the cases.

For the case v1, the convergence day of vaccinated population to 0.6 was calculated to be 9.5, whereas it was calculated as 1.4 for case v2, resulting in a significantly increased convergence rate. Another observation that should be mentioned is the overshoot occurring in the first plot of the vaccination figure for v1 case. In the first case, the vaccinated population needed to fluctuate around the final value to converge it, overshooting 0.6 by 33% with a value of approximately 0.2, and reaching 0.8, due to the fact that only input parameter was itself. However, in the second case, almost no overshoot is observed thanks to the introduction of effective vaccination rate as an input parameter to the system. Both the increased convergence rate and decreased overshooting resulted in significantly decreased the cost by 70.99% from 21.99 to 6.38, which is a measurement of vaccinated people before the herd immunity is achieved.

Finally, the experiments showed that, introduction of $\dot{\pi}$, effective vaccination rate as an input to the fuzzy controller, increased the success of the vaccination significantly by increasing the convergence and decreasing the overshoot, resulting in decreased cost.

4 APPENDIX

4.1 v1 Partitions

```
from matplotlib import pyplot as plt
from vaccination import Vaccination
import numpy as np

# margin is a hyperparameter to be determined with the experiments
pi_margin = 0.05
delta_margin = 0.0125
low_center = 2*delta_margin
high_center = -low_center


####### Vaccination Rate Set Partitioning #######################
def low_vac_rate(x):
    if x < 0.6 - pi_margin:
        return 1
    elif x > 0.6:
        return 0
    else:
        return -1/pi_margin * x + 0.6/pi_margin


def high_vac_rate(x):
    if x < 0.6:
        return 0
    elif x > 0.6 + pi_margin:
        return 1
    else:
        return 1/pi_margin * x - 0.6/pi_margin


def opt_vac_rate(x):
    if x < 0.6-pi_margin or x > 0.6+pi_margin:
        return 0
    else:
        return np.sign(x-0.6) * ((0.6-x)/pi_margin) + 1

############# Vaccination Rate Change Set Partitioning
                                       #######################


def low_vac_rate_change(x, low_center):
    if x < low_center-delta_margin or x > low_center+delta_margin:
        return 0
    else:
        return np.sign(x-low_center) * ((low_center-x)/delta_margin) + 1


def high_vac_rate_change(x, high_center):
    if x < high_center-delta_margin or x > high_center+delta_margin:
        return 0
    else:
        return np.sign(x-high_center) * ((high_center-x)/delta_margin) +
                                        1


def opt_vac_rate_change(x):
    if x < 0-delta_margin or x > 0+delta_margin:
        return 0
    else:
        return np.sign(x-0) * ((0-x)/delta_margin) + 1


def increase_high(x, high_center):
    if x > high_center+delta_margin:
        return 1
    elif x < high_center - delta_margin:
        return 0
    else:
        return 1/(2*delta_margin) * x - (high_center-delta_margin)/(2*
                                        delta_margin)


def decrease_high(x, low_center):
    if x < low_center - delta_margin:
        return 1
    elif x > low_center + delta_margin:
        return 0
    else:
        return -1/(2*delta_margin) * x + (low_center+delta_margin)/(2*
                                        delta_margin)


plt.subplot(1, 2, 1)
pi = np.arange(0, 1, 0.05)
low_rate = list(map(low_vac_rate, pi))
high_rate = list(map(high_vac_rate, pi))
opt_rate = list(map(opt_vac_rate, pi))
plt.plot(pi, low_rate)
plt.plot(pi, opt_rate)
plt.plot(pi, high_rate)
plt.title("Fuzzy Partition of the Sets for margin="+str(pi_margin))
plt.xlabel("$\pi$")
plt.ylabel("Membership Values")
plt.legend(["Low Vaccination", "Optimal Vaccination", "High Vaccination"
                                        ],)


plt.subplot(1, 2, 2)
delta = np.arange(-0.25, 0.25, 0.01)
low_rate = list(map(lambda item: increase_high(item, low_center), delta)
                                        )
high_rate = list(
    map(lambda item: decrease_high(item, high_center), delta))
opt_rate = list(map(opt_vac_rate_change, delta))
plt.plot(delta, low_rate)
```

```
    plt.plot(delta, opt_rate)
    plt.plot(delta, high_rate)
    plt.title("Fuzzy Partition of the Sets for margin="+str(delta_margin))
    plt.xlabel("$\delta$")
    plt.ylabel("Membership Values")
    plt.legend(["Low Vaccination \n Rate Change",
                "Optimal Vaccination \n Rate Change", "High Vaccination \n
                                                      Rate Change"],)
    plt.show()
```

4.2 Vaccination v1

```
# Vaccination v1
# imports
from matplotlib import pyplot as plt
from vaccination import Vaccination
import numpy as np
from random import uniform
from math import ceil

# margin is a hyperparameter to be determined with the experiments
pi_margin = 0.05
delta_margin = 0.0125
low_center = 2*delta_margin
high_center = -low_center


############# Vaccination Rate Set Partitioning #######################
def low_vac_rate(x):
    if x < 0.6 - pi_margin:
        return 1
    elif x > 0.6:
        return 0
    else:
        return -1/pi_margin * x + 0.6/pi_margin


def high_vac_rate(x):
    if x < 0.6:
        return 0
    elif x > 0.6 + pi_margin:
        return 1
    else:
        return 1/pi_margin * x - 0.6/pi_margin


def opt_vac_rate(x):
    if x < 0.6-pi_margin or x > 0.6+pi_margin:
        return 0
    else:
        return np.sign(x-0.6) * ((0.6-x)/pi_margin) + 1

############# Vaccination Rate Change Set Partitioning ################
```

```
def low_vac_rate_change(x, low_center):
    if x < low_center-delta_margin or x > low_center+delta_margin:
        return 0
    else:
        return np.sign(x-low_center) * ((low_center-x)/delta_margin) + 1


def high_vac_rate_change(x, high_center):
    if x < high_center-delta_margin or x > high_center+delta_margin:
        return 0
    else:
        return np.sign(x-high_center) * ((high_center-x)/delta_margin) + 1


def opt_vac_rate_change(x):
    if x < 0-delta_margin or x > 0+delta_margin:
        return 0
    else:
        return np.sign(x-0) * ((0-x)/delta_margin) + 1

############################# HELPERS #################################


def maximum_membership(vaccination_rate):
    # Returns the class of the current vaccination rate status as an index
    memberships = [low_vac_rate(vaccination_rate), opt_vac_rate(
        vaccination_rate), high_vac_rate(vaccination_rate)]
    idx = memberships.index(max(memberships))
    membership = memberships[idx]
    return idx, membership


def MaxCriterion(index, membership):
    # Throws a random Vaccination rate change depending on the current
    #  vaccinating rate membership value to the corresponding set
    if index == 0:   # Low Vac Rate
        left = (membership-1)*delta_margin + low_center
        right = low_center - (membership-1) * delta_margin
        # left = delta_margin*(2*membership-1) + low_center
        # right = 0.25
    elif index == 1:   # Opt Vac Rate
        left = (membership-1)*delta_margin + 0
        right = 0 - (membership-1) * delta_margin
    else:   # High Vac Rate
        left = (membership-1)*delta_margin + high_center
        right = high_center - (membership-1) * delta_margin
        # left = -0.25
        # right = high_center + delta_margin*(1-2*membership)
    return round(uniform(left, right), 2)


def round_up_to_nearest_ten(x):
    # Just to be used in .viewVaccination()
    return int(ceil(x/10))*10


def find_stabilized_point(l):
    # Finds convergence point from vaccinated_percentage_curve_ attribute
    error = 0.01
    left = 0.6-error
    right = 0.6+error
    counter = 0
    found = False
    for i in range(len(l)):
        if left < l[i] < right:
            if not found:
                first_point = i
            found = True
            counter += 1
            if counter >= 30:
                return first_point
        else:
            found = False
            counter = 0
    return False


############################# VACCINATION ###############################

def Vaccinationv1():
    vaccination = Vaccination()
    days = 500
    convergence_day = 100

    for day in range(days):
        # First get pi
        pi, _ = vaccination.checkVaccinationStatus()
        # calculate membership and return set and the membership
        idx, membership = maximum_membership(pi)
        # calculate output control variable from membership using
                                              MaxCriterion
        delta = MaxCriterion(idx, membership)
        # Update pi_dot
        vaccination.vaccinatePeople(delta)

    # Once the loop finishes, get convergence day
    convergence_day = find_stabilized_point(
        vaccination.vaccinated_percentage_curve_)

    return vaccination, convergence_day


vaccination, convergence_day = Vaccinationv1()

# Visualize results
vaccination.viewVaccination(convergence_day, np.sum(
    vaccination.vaccination_rate_curve_[1:convergence_day]), save_dir="
                                          FIGSv1\\denemeler")
```

4.3 v2 partitions

```
from matplotlib import pyplot as plt
from vaccination import Vaccination
import numpy as np

# margin is a hyperparameter to be determined with the experiments
pi_margin = 0.05
pi_dot_margin = 0.1
delta_margin = 0.05
low_center = 2*delta_margin
high_center = -low_center


####### Vaccination Rate Set Partitioning ######################
def low_vac_rate(x):
    if x < 0.6 - pi_margin:
        return 1
    elif x > 0.6:
        return 0
    else:
        return -1/pi_margin * x + 0.6/pi_margin


def high_vac_rate(x):
    if x < 0.6:
        return 0
    elif x > 0.6 + pi_margin:
        return 1
    else:
        return 1/pi_margin * x - 0.6/pi_margin


def opt_vac_rate(x):
    if x < 0.6-pi_margin or x > 0.6+pi_margin:
        return 0
    else:
        return np.sign(x-0.6) * ((0.6-x)/pi_margin) + 1

####### Input Vaccination Rate Change Set Partitioning
                                      ######################


def low_vac_rate_input(x):
    if x < 0 - pi_dot_margin:
        return 1
    elif x > 0:
        return 0
    else:
        return -1/pi_dot_margin * x + 0/pi_dot_margin


def high_vac_rate_input(x):
    if x < 0:
        return 0
    elif x > 0 + pi_dot_margin:
        return 1
    else:
        return 1/pi_dot_margin * x - 0/pi_dot_margin


def opt_vac_rate_input(x):
    if x < 0-pi_dot_margin or x > 0+pi_dot_margin:
        return 0
    else:
        return np.sign(x-0) * ((0-x)/pi_dot_margin) + 1

############# Output Vaccination Rate Change Set Partitioning
                                        #######################


def increase_low(x, low_center):
    if x < low_center-delta_margin or x > low_center+delta_margin:
        return 0
    else:
        return np.sign(x-low_center) * ((low_center-x)/delta_margin) + 1


def decrease_low(x, high_center):
    if x < high_center-delta_margin or x > high_center+delta_margin:
        return 0
    else:
        return np.sign(x-high_center) * ((high_center-x)/delta_margin) + 1


def increase_high(x, high_center):
    if x > high_center+delta_margin:
        return 1
    elif x < high_center - delta_margin:
        return 0
    else:
        return 1/(2*delta_margin) * x - (high_center-delta_margin)/(2*
                                                    delta_margin)


def decrease_high(x, low_center):
    if x < low_center - delta_margin:
        return 1
    elif x > low_center + delta_margin:
        return 0
    else:
        return -1/(2*delta_margin) * x + (low_center+delta_margin)/(2*
                                                    delta_margin)


def opt_vac_rate_change(x):
    if x < 0-delta_margin or x > 0+delta_margin:
        return 0
    else:
        return np.sign(x-0) * ((0-x)/delta_margin) + 1
```

```
##########################################################################
plt.figure(figsize=(75, 5))

plt.subplot(1, 3, 1)
pi = np.arange(0, 1, 0.05)
low_rate = list(map(low_vac_rate, pi))
high_rate = list(map(high_vac_rate, pi))
opt_rate = list(map(opt_vac_rate, pi))
plt.plot(pi, low_rate)
plt.plot(pi, opt_rate)
plt.plot(pi, high_rate)
plt.title("Set partitioning for $\pi$")
plt.xlabel("$\pi$")
plt.ylabel("Membership Values")
plt.legend(["Low Vaccination", "Optimal Vaccination",
            "High Vaccination"], borderpad=0, fontsize=12, loc="center left"
                                                   )


plt.subplot(1, 3, 2)
delta = np.arange(-1, 1, 0.001)
low_rate = list(map(lambda item: low_vac_rate_input(item), delta))
high_rate = list(
    map(lambda item: high_vac_rate_input(item), delta))
opt_rate = list(map(opt_vac_rate_input, delta))
plt.plot(delta, low_rate)
plt.plot(delta, opt_rate)
plt.plot(delta, high_rate)
plt.title("Set partitioning for $\dot{\pi}$")
plt.xlabel("$\dot{\pi}$")
plt.ylabel("Membership Values")
plt.legend(["Low Vaccination \n Rate Change",
            "Optimal Vaccination \n Rate Change", "High Vaccination \n Rate
                                                   Change"], borderpad=0,
                                                   fontsize=8)

plt.subplot(1, 3, 3)
delta = np.arange(-0.25, 0.25, 0.01)
dh = list(map(lambda item: decrease_high(
    item, high_center-delta_margin), delta))
ih = list(
    map(lambda item: increase_high(item, low_center+delta_margin), delta))
dl = list(map(lambda item: decrease_low(item, low_center), delta))
il = list(
    map(lambda item: increase_low(item, high_center), delta))

opt_rate = list(map(opt_vac_rate_change, delta))
plt.plot(delta, dh)
plt.plot(delta, dl)
plt.plot(delta, opt_rate)
plt.plot(delta, il)
plt.plot(delta, ih)
plt.title("Set partitioning for $\delta$")
plt.xlabel("$\delta$")
plt.ylabel("Membership Values")
plt.legend(["High Decrease", "Low Decrease", "Around Optimum",
            "Low Increase", "High Increase"], borderpad=0, fontsize=8)

fig = plt.gcf()
fig.set_size_inches(25.5, 5)

plt.show()
```

4.4 Vaccination v2

```
    # Vaccination v2
from matplotlib import pyplot as plt
from vaccination import Vaccination
from random import uniform
import numpy as np
from math import ceil

# margin is a hyperparameter to be determined with the experiments
pi_margin = 0.05
pi_dot_margin = 0.1
delta_margin = 0.025
low_center = 2*delta_margin
high_center = -low_center


####### Vaccination Rate Set Partitioning #######################
def low_vac_rate(x):
    if x < 0.6 - pi_margin:
        return 1
    elif x > 0.6:
        return 0
    else:
        return -1/pi_margin * x + 0.6/pi_margin


def high_vac_rate(x):
    if x < 0.6:
        return 0
    elif x > 0.6 + pi_margin:
        return 1
    else:
        return 1/pi_margin * x - 0.6/pi_margin


def opt_vac_rate(x):
    if x < 0.6-pi_margin or x > 0.6+pi_margin:
        return 0
    else:
        return np.sign(x-0.6) * ((0.6-x)/pi_margin) + 1
```

```
####### Input Vaccination Rate Change Set Partitioning
                                              #######################


def low_vac_rate_input(x):
    if x < 0 - pi_dot_margin:
        return 1
    elif x > 0:
        return 0
    else:
        return -1/pi_dot_margin * x + 0/pi_dot_margin


def high_vac_rate_input(x):
    if x < 0:
        return 0
    elif x > 0 + pi_dot_margin:
        return 1
    else:
        return 1/pi_dot_margin * x - 0/pi_dot_margin


def opt_vac_rate_input(x):
    if x < 0-pi_dot_margin or x > 0+pi_dot_margin:
        return 0
    else:
        return np.sign(x-0) * ((0-x)/pi_dot_margin) + 1

############# Output Vaccination Rate Change Set Partitioning
                                              #######################


def increase_low(x, low_center):
    if x < low_center-delta_margin or x > low_center+delta_margin:
        return 0
    else:
        return np.sign(x-low_center) * ((low_center-x)/delta_margin) + 1


def decrease_low(x, high_center):
    if x < high_center-delta_margin or x > high_center+delta_margin:
        return 0
    else:
        return np.sign(x-high_center) * ((high_center-x)/delta_margin) + 1


def increase_high(x, high_center):
    if x > high_center+delta_margin:
        return 1
    elif x < high_center - delta_margin:
        return 0
    else:
        return 1/(2*delta_margin) * x - (high_center-delta_margin)/(2*
                                                     delta_margin)
```

```
def decrease_high(x, low_center):
    if x < low_center - delta_margin:
        return 1
    elif x > low_center + delta_margin:
        return 0
    else:
        return -1/(2*delta_margin) * x + (low_center+delta_margin)/(2*
                                                delta_margin)


def opt_vac_rate_change(x):
    if x < 0-delta_margin or x > 0+delta_margin:
        return 0
    else:
        return np.sign(x-0) * ((0-x)/delta_margin) + 1

############################# HELPERS #################################


def maximum_membership_pi(vaccination_rate):
    # Returns the class of the current vaccination rate status as an index
    memberships = [low_vac_rate(vaccination_rate), opt_vac_rate(
        vaccination_rate), high_vac_rate(vaccination_rate)]
    idx = memberships.index(max(memberships))
    membership = memberships[idx]
    return idx, membership


def maximum_membership_pi_dot(vaccination_rate):
    # Returns the class of the current vaccination rate change status as an
                                            index
    memberships = [low_vac_rate_input(vaccination_rate), opt_vac_rate_input(
        vaccination_rate), high_vac_rate_input(vaccination_rate)]
    idx = memberships.index(max(memberships))
    membership = memberships[idx]
    return idx, membership


def MaxCriterion(pi_idx, pi_dot_idx, membership):
    # Throws a random Vaccination rate per day change depending on the
                                            current
    #  vaccinating rate membership value to the corresponding set
    if pi_idx == 0:  # Low Vac Rate
        if pi_dot_idx == 0 or pi_dot_idx == 1:  # Low OR Opt
            # INCREASE HIGH
            left = delta_margin*(2*membership-1) + low_center
            right = 0.25
        else:
            # INCREASE LOW
            left = (membership-1)*delta_margin + low_center
            right = low_center - (membership-1) * delta_margin

    elif pi_idx == 2:  # High Vac Rate
        if pi_dot_idx == 1 or pi_dot_idx == 2:  # High OR Opt
```

```
            # DECREASE HIGH
            left = -0.25
            right = high_center + delta_margin*(1-2*membership)
        else:
            # DECREASE LOW
            left = (membership-1)*delta_margin + low_center
            right = low_center - (membership-1) * delta_margin
    else:  # Opt Vac Rate
        if pi_dot_idx == 0 or pi_dot_idx == 1:  # Low OR Opt
            # OPTIMAL CHANGE
            left = (membership-1)*delta_margin
            right = -(membership-1) * delta_margin
        else:
            # DECREASE HIGH
            left = -0.25
            right = high_center + delta_margin*(1-2*membership)

    # Return random number from alpha cut
    return round(uniform(left, right), 3)


def round_up_to_nearest_ten(x):
    return int(ceil(x/10))*10


def find_stabilized_point(l):
    # Finds convergence point from vaccinated_percentage_curve_ attribute
    error = 0.005
    left = 0.6-error
    right = 0.6+error
    counter = 0
    found = False
    for i in range(len(l)):
        if left < l[i] < right:
            if not found:
                first_point = i
            found = True
            counter += 1
            if counter >= 60:
                return first_point
        else:
            found = False
            counter = 0
    return False


############################# VACCINATION ###############################

def vaccinationv2():
    vaccination = Vaccination()

    days = 200
    convergence_day = 200

    for day in range(days):
        # First get pi and pi_dot
        current, effective = vaccination.checkVaccinationStatus()
        # calculate memberships and return sets and the memberships
        pi_dot_idx, _ = maximum_membership_pi_dot(effective)
        pi_idx, membership = maximum_membership_pi(current)
        # calculate output control variable from pi and pi_dot indices and
                                              pi membership using
                                              MaxCriterion
        delta = MaxCriterion(pi_idx, pi_dot_idx, membership)
        # Update pi_dot
        vaccination.vaccinatePeople(delta)

    # Once the loop finishes, get convergence day
    convergence_day = find_stabilized_point(
        vaccination.vaccinated_percentage_curve_)
    return convergence_day, vaccination


convergence_day, vaccination = vaccinationv2()


vaccination.viewVaccination(convergence_day, np.sum(
    vaccination.vaccination_rate_curve_[1:convergence_day]), save_dir="
                                         FIGSv2\\denemeler\\")
```