# EE449-Computational Intelligence

# Vaccinating People Against COVID-19 with Fuzzy Control

## Vaccination v1

### 1.1 Set Partitioning

Input ‘π’ that is the vaccinated population percentage and output rate ‘δ’ universes are separated into three fuzzy sets. These sets are called ‘low’, ‘medium’ and ‘high’ as shown in fig1 and fig2.

Chart, line chart

Description automatically generated

Figure 1. Currently Vaccinated Set Partition

Line chart

Description automatically generated

Figure 2. Output Control Rate Set Partition

For the input vaccinated percentage universe, we know that 0.6 is to be equilibrium point. Hence, we adjust medium to peak while low and high decays to 0 at this point. Accordingly, medium set is chosen to be a triangular function while low and high sets are trapezoids since we’d like to avoid highest and lowest areas.

For the output control rate universe, we pick a symmetrical function since when equilibrium is reached, control rate will converge to 0. Low and high sets are again trapezoids to make sure at high and low vaccination percentage cases, stable behavior is reached faster.

Exact points for trapezoid and triangular function edges are picked with considering preconditions stated above and experimenting.

### 1.2. Fuzzy Control Rules

1. If vaccination percentage is low, set control rate high.
2. If vaccination percentage is medium, set control rate medium.
3. If vaccination percentage is high, set control rate low.

### 1.3. Fuzzification and Defuzzification Interface

Fuzzification is implemented in python according to the sets shown in fig.1 & fig.2. Skfuzzy package is used for interface which presents required built-in functions that are very easy to implement.

# New Antecedent/Consequent objects hold universe variables and membership

# functions

#antecedent for input consequent for output

cur\_vac = ctrl.Antecedent(np.arange(0, 1.1, 0.1), 'Currently Vaccinated')

out\_rate = ctrl.Consequent(np.arange(-0.2, 0.25, 0.05), 'Output Rate')

# Custom membership functions can be built interactively with a familiar,

# Pythonic API

#trapmf for trapezoid functions 4 edges provide

#trimf for triangular functions 3 edges provided

cur\_vac['low'] = fuzz.trapmf(cur\_vac.universe, [0, 0, 0.3, 0.6])

cur\_vac['medium'] = fuzz.trimf(cur\_vac.universe, [0.3, 0.6, 0.9])

cur\_vac['high'] = fuzz.trapmf(cur\_vac.universe, [0.6, 0.8, 1.0, 1.0])

out\_rate['low'] = fuzz.trapmf(out\_rate.universe, [-0.2,-0.2, -0.15, 0])

out\_rate['medium'] = fuzz.trimf(out\_rate.universe, [-0.1, 0, 0.1])

out\_rate['high'] = fuzz.trapmf(out\_rate.universe, [0, 0.15, 0.20,0.20])

Defuzzification is also implemented using Skfuzzy package. Center of Gravity method is used to compute output control rate value.

### 1.4. Simulation

Equilibrium point is roughly estimated to be at 125th iteration from the vaccinated population percentage curve. Each 1% vaccination rate costs 1 unit, hence, cost is computed by summing vaccination rate over the first 125 iterations and multiplying the number by 100. Behavior of our control system is shown with the provided graphs in fig.3.

Chart, line chart

Description automatically generated

Figure 3 Control System Behavior Graphs

## Vaccination v2

### 2.1 Set Partitioning

Inputs ‘π’ that is the vaccinated population percentage, ‘ π˙ ’that is the current vaccination rate, and output rate ‘δ’ universes are separated into three fuzzy sets. These sets are called ‘low’, ‘medium’ and ‘high’ as shown in fig4 fig.5 and fig.6.

Chart, line chart

Description automatically generated

Figure 4 Currently Vaccinated Set Partition

Chart, line chart

Description automatically generated

Figure 5 Effective Vaccination Set Partition

Line chart

Description automatically generated

Figure 6 Output Control Rate Set Partition

Sets ‘π’ and ‘δ’ are partitioned exactly the same as in part 1 for the same reasons. Effective vaccination rate universe is partitioned symmetrically, and medium is chosen to peak at 0 since effective rate will converge to 0 at equilibrium. Also edges of the set borders are chosen as symmetrically and evenly as possible to make sure they are interpreted accurately enough. We’ll make use of effective rate value to get rid of the oscillations in the previous graph (fig.3). It is inferred from the graph that oscillation occur when π is at medium or π low π\_dot high or π high π\_dot low since they are the points where failures play the most critical role. Hence the rules are chosen accordingly.

### 2.2. Fuzzy Control Rules

1. If vaccination percentage is low and effective rate is low, set control rate high.
2. If vaccination percentage is low and effective rate is medium, set control rate high.
3. If vaccination percentage is low and effective rate is high, set control rate medium.
4. If vaccination percentage is medium and effective rate is low, set control rate high.
5. If vaccination percentage is medium and effective rate is medium, set control rate medium.
6. If vaccination percentage is medium and effective rate is high, set control rate low.
7. If vaccination percentage is high and effective rate is low, set control rate medium.
8. If vaccination percentage is high and effective rate is medium, set control rate low.
9. If vaccination percentage is high and effective rate is high, set control rate low.

### 2.3. Fuzzification and Defuzzification Interface

Python implementation is done with the help of Skfuzzy package. Same as stated in 1.3.

### 1.4. Simulation

Chart

Description automatically generated

Figure 7. Control System Behavior Graphs

Equilibrium point is roughly estimated to be at 30th iteration from the vaccinated population percentage curve. Each 1% vaccination rate costs 1 unit, hence, cost is computed by summing vaccination rate over the first 30 iterations and multiplying the number by 100. Behavior of our control system is shown with the provided graphs in fig.7.

### 2.4. Comparison

Comparing two distinct systems for the same application with behavior analyses as given in fig.3. and fig7., we conclude that system 2 demonstrates a much-improved performance. The main reason behind this result is the advantage of 2 control inputs provided for the second system, where the first system only uses vaccinated percentage variable which is misleading at times due to failure rates.

## Vaccination v1 Codes

# part of the code from: https://pythonhosted.org/scikit-fuzzy/auto\_examples/plot\_tipping\_problem\_newapi.html

import os

from matplotlib import pyplot as plt

from matplotlib import ticker

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

# class object to simulate vaccinating people

class Vaccination(object):

def \_\_init\_\_(self):

self.vaccinated\_percentage\_curve\_ = [0.] # percentage of the vaccinated people

self.vaccination\_rate\_curve\_ = [0.] # percentage / day

self.vaccination\_control\_curve\_ = [0.] # percentage / day

def \_vaccinationFailureRate(self):

p = self.vaccinated\_percentage\_curve\_[-1]

return .5 \* p \* p

# method to apply the control

def vaccinatePeople(self, vaccination\_control):

"""

applies the control signal to vaccinate people and updates the status curves

Arguments:

----------

vaccination\_control: float, vaccination rate to be added to the current vaccination rate

"""

# update vaccination rate according to control signal

curr\_vaccination\_rate = self.vaccination\_rate\_curve\_[-1]

vaccination\_rate = max(0., min(.6, curr\_vaccination\_rate + vaccination\_control))

effective\_vaccination\_rate = vaccination\_rate - self.\_vaccinationFailureRate()

# update the vaccinated percentage after 0.1 Day of vaccination with the current rate

vaccination\_percentage = \

min(1., self.vaccinated\_percentage\_curve\_[-1] + effective\_vaccination\_rate \* .1)

# update status curves

self.vaccinated\_percentage\_curve\_.append(vaccination\_percentage)

self.vaccination\_rate\_curve\_.append(vaccination\_rate)

self.vaccination\_control\_curve\_.append(vaccination\_control)

# method to obtain measurements

def checkVaccinationStatus(self):

"""

returns the current vaccinated percentage and vaccination rate as a two-tuple

(vaccinated\_percentage, vaccination\_rate)

Returns:

----------

(vaccinated\_percentage, effective\_vaccination\_rate): (float, float) tuple,

vaccination percentage and rate to be used by the controller

"""

vaccinated\_percentage = self.vaccinated\_percentage\_curve\_[-1]

effective\_vaccination\_rate = \

self.vaccination\_rate\_curve\_[-1] - self.\_vaccinationFailureRate()

return (vaccinated\_percentage, effective\_vaccination\_rate)

# method to visualize the results for the homework

def viewVaccination(self, point\_ss, vaccination\_cost, save\_dir='', filename='vaccination', show\_plot=True):

"""

plots multiple curves for the vaccination and

saves the resultant plot as a png image

Arguments:

----------

point\_ss: int, the estimated iteration index at which the system is at steady state

vaccination\_cost: float, the estimated cost of the vaccination until the steady state

save\_dir: string, indicating the path to directory where the plot image is to be saved

filename: string, indicating the name of the image file. Note that .png will be automatically

appended to the filename.

show\_plot: bool, whether the figure is to be shown

Example:

--------

visualizing the results of the vaccination

# assume many control signals have been consecutively applied to vaccine people

>>> my\_vaccine = Vaccination()

>>> my\_vaccine.vaccinatePeople(vaccination\_control) # assume this has been repeated many times

>>> # assume state state index and the vaccination cost have been computed

>>> # as point\_ss=k and vaccination\_cost=c

>>> my\_vaccine.viewVaccination(point\_ss=k, vaccination\_cost=c,

>>> save\_dir='some\location\to\save', filename='vaccination')

"""

color\_list = ['#ff0000', '#32CD32', '#0000ff', '#d2691e', '#ff00ff', '#000000', '#373788']

style\_list = ['-', '--']

num\_plots = 3

plot\_curve\_args = [{'c': color\_list[k],

'linestyle': style\_list[0],

'linewidth': 3} for k in range(num\_plots)]

plot\_vert\_args = [{'c': color\_list[k],

'linestyle': style\_list[1],

'linewidth': 3} for k in range(num\_plots)]

font\_size = 18

fig, axes = plt.subplots(3, 1, figsize=(16, 12))

day\_x = [i \* .1 for i in range(len(self.vaccinated\_percentage\_curve\_))]

x\_ticks = day\_x[::10]

# vaccinated population

ax = axes[0]

ax.set\_title('vaccinated population percentage over days', loc='left', fontsize=font\_size)

ax.plot(day\_x[:point\_ss+1], self.vaccinated\_percentage\_curve\_[:point\_ss+1], \*\*plot\_curve\_args[0])

ax.plot(day\_x[point\_ss:], self.vaccinated\_percentage\_curve\_[point\_ss:], \*\*plot\_curve\_args[1])

ax.plot([day\_x[point\_ss]] \* 2, [0, self.vaccinated\_percentage\_curve\_[point\_ss]], \*\*plot\_vert\_args[2])

ax.set\_xlabel(xlabel='day', fontsize=font\_size)

ax.set\_ylabel(ylabel='vaccinated population %', fontsize=font\_size)

ax.set\_xticks(x\_ticks)

ax.xaxis.set\_minor\_locator(ticker.FixedLocator([day\_x[point\_ss]]))

ax.xaxis.set\_minor\_formatter(ticker.ScalarFormatter())

ax.tick\_params(which='minor', length=17, color='b', labelsize=13)

ax.tick\_params(labelsize=15)

ax.set\_ylim(bottom=0)

ax.set\_xlim(left=0)

ax.grid(True, lw = 1, ls = '--', c = '.75')

# vaccination rate

ax = axes[1]

ax.set\_title('vaccination rate over days', loc='left', fontsize=font\_size)

ax.plot(day\_x[:point\_ss + 1], self.vaccination\_rate\_curve\_[:point\_ss + 1],

\*\*plot\_curve\_args[0])

ax.plot(day\_x[point\_ss:], self.vaccination\_rate\_curve\_[point\_ss:],

\*\*plot\_curve\_args[1])

ax.plot([day\_x[point\_ss]] \* 2, [0, self.vaccination\_rate\_curve\_[point\_ss]],

\*\*plot\_vert\_args[2])

ax.fill\_between(day\_x[:point\_ss + 1], 0, self.vaccination\_rate\_curve\_[:point\_ss + 1],

facecolor='#FF69B4', alpha=0.7)

ax.text(1.5, .01, 'cost = %.2f'%vaccination\_cost,

horizontalalignment='center', fontsize=font\_size)

ax.set\_xlabel(xlabel='day', fontsize=font\_size)

ax.set\_ylabel(ylabel='vaccination rate (%/day)', fontsize=font\_size)

ax.set\_xticks(x\_ticks)

ax.xaxis.set\_minor\_locator(ticker.FixedLocator([day\_x[point\_ss]]))

ax.xaxis.set\_minor\_formatter(ticker.ScalarFormatter())

ax.tick\_params(which='minor', length=17, color='b', labelsize=13)

ax.tick\_params(labelsize=15)

ax.set\_ylim(bottom=0)

ax.set\_xlim(left=0)

ax.grid(True, lw=1, ls='--', c='.75')

# vaccination rate control

ax = axes[2]

ax.set\_title('vaccination rate control over days', loc='left', fontsize=font\_size)

ax.plot(day\_x[:point\_ss + 1], self.vaccination\_control\_curve\_[:point\_ss + 1],

\*\*plot\_curve\_args[0])

ax.plot(day\_x[point\_ss:], self.vaccination\_control\_curve\_[point\_ss:],

\*\*plot\_curve\_args[1])

y\_min = ax.get\_ylim()[0]

ax.plot([day\_x[point\_ss]] \* 2,

[y\_min, self.vaccination\_control\_curve\_[point\_ss]],

\*\*plot\_vert\_args[2])

ax.set\_xlabel(xlabel='day', fontsize=font\_size)

ax.set\_ylabel(ylabel='vaccination rate control (%/day)', fontsize=font\_size)

ax.set\_xticks(x\_ticks)

ax.xaxis.set\_minor\_locator(ticker.FixedLocator([day\_x[point\_ss]]))

ax.xaxis.set\_minor\_formatter(ticker.ScalarFormatter())

ax.tick\_params(which='minor', length=17, color='b', labelsize=13)

ax.tick\_params(labelsize=15)

ax.set\_xlim(left=0)

ax.set\_ylim(bottom=y\_min)

ax.grid(True, lw=1, ls='--', c='.75')

if show\_plot:

plt.show()

fig.savefig(os.path.join(save\_dir, filename + '.png'))

#create vaccination class object and check current status to start vaccination process

vaccination=Vaccination()

(pi,pi\_dot)= vaccination.checkVaccinationStatus()

# New Antecedent/Consequent objects hold universe variables and membership

# functions

#antecedent for input consequent for output

cur\_vac = ctrl.Antecedent(np.arange(0, 1.1, 0.1), 'Currently Vaccinated')

out\_rate = ctrl.Consequent(np.arange(-0.2, 0.25, 0.05), 'Output Rate')

# Custom membership functions can be built interactively with a familiar,

# Pythonic API

#trapmf for trapezoid functions 4 edges provide

#trimf for triangular functions 3 edges provided

cur\_vac['low'] = fuzz.trapmf(cur\_vac.universe, [0, 0, 0.3, 0.6])

cur\_vac['medium'] = fuzz.trimf(cur\_vac.universe, [0.3, 0.6, 0.9])

cur\_vac['high'] = fuzz.trapmf(cur\_vac.universe, [0.6, 0.8, 1.0, 1.0])

out\_rate['low'] = fuzz.trapmf(out\_rate.universe, [-0.2,-0.2, -0.15, 0])

out\_rate['medium'] = fuzz.trimf(out\_rate.universe, [-0.1, 0, 0.1])

out\_rate['high'] = fuzz.trapmf(out\_rate.universe, [0, 0.15, 0.20,0.20])

# View membership functions

cur\_vac.view()

out\_rate.view()

#set rules

rule1 = ctrl.Rule(cur\_vac['low'] , out\_rate['high'])

rule2 = ctrl.Rule(cur\_vac['medium'], out\_rate['medium'])

rule3 = ctrl.Rule(cur\_vac['high'] , out\_rate['low'])

#rule1.view()

#create a control system to simulate

vaccination\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

vaccinate = ctrl.ControlSystemSimulation(vaccination\_ctrl)

rate=0

effective= []

effective.append(0)

# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API

# Note: if you like passing many inputs all at once, use .inputs(dict\_of\_data)

for i in range(200):

# print(vaccination.vaccination\_rate\_curve\_)

vaccinate.input['Currently Vaccinated'] = pi

# Crunch the numbers

vaccinate.compute()

#viusalize output

#print (vaccinate.output['Output Rate'])

#out\_rate.view(sim=vaccinate)

vaccination.vaccinatePeople(vaccinate.output['Output Rate'])

(pi,pi\_dot)= vaccination.checkVaccinationStatus()

#effective.append(pi\_dot)

#plot to see vaccinated percentage curve

plt.figure()

x=np.arange(0,201)

plt.plot(x,vaccination.vaccinated\_percentage\_curve\_)

plt.xlim([0, 200])

plt.title( "Vaccinated Percentage Curve")

plt.show()

#plot to see Effective Vaccination Rate Curve

# plt.plot(x,effective)

# plt.title("Effective Vaccination Rate Curve")

# plt.show()

#plot to see Vaccination Rate Curve

x=np.arange(0,201)

plt.plot(x,vaccination.vaccination\_rate\_curve\_)

plt.xlim([0, 200])

plt.title( "Vaccination Rate Curve")

plt.show()

#compute cost from vaccination rate curve

vac\_cost=vaccination.vaccination\_rate\_curve\_[0:125]

cost=np.sum(vac\_cost) \*100

# print(cost\*100)

#view vaccination process

vaccination.viewVaccination(125, cost, save\_dir='/Users/gulceonder/Desktop/webpage/webdevelopment', filename='vsc', show\_plot=True)

Vaccination v2 Codes

# part of the code from: https://pythonhosted.org/scikit-fuzzy/auto\_examples/plot\_tipping\_problem\_newapi.html

import os

from matplotlib import pyplot as plt

from matplotlib import ticker

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

# class object to simulate vaccinating people

class Vaccination(object):

def \_\_init\_\_(self):

self.vaccinated\_percentage\_curve\_ = [0.] # percentage of the vaccinated people

self.vaccination\_rate\_curve\_ = [0.] # percentage / day

self.vaccination\_control\_curve\_ = [0.] # percentage / day

def \_vaccinationFailureRate(self):

p = self.vaccinated\_percentage\_curve\_[-1]

return .5 \* p \* p

# method to apply the control

def vaccinatePeople(self, vaccination\_control):

"""

applies the control signal to vaccinate people and updates the status curves

Arguments:

----------

vaccination\_control: float, vaccination rate to be added to the current vaccination rate

"""

# update vaccination rate according to control signal

curr\_vaccination\_rate = self.vaccination\_rate\_curve\_[-1]

vaccination\_rate = max(0., min(.6, curr\_vaccination\_rate + vaccination\_control))

effective\_vaccination\_rate = vaccination\_rate - self.\_vaccinationFailureRate()

# update the vaccinated percentage after 0.1 Day of vaccination with the current rate

vaccination\_percentage = \

min(1., self.vaccinated\_percentage\_curve\_[-1] + effective\_vaccination\_rate \* .1)

# update status curves

self.vaccinated\_percentage\_curve\_.append(vaccination\_percentage)

self.vaccination\_rate\_curve\_.append(vaccination\_rate)

self.vaccination\_control\_curve\_.append(vaccination\_control)

# method to obtain measurements

def checkVaccinationStatus(self):

"""

returns the current vaccinated percentage and vaccination rate as a two-tuple

(vaccinated\_percentage, vaccination\_rate)

Returns:

----------

(vaccinated\_percentage, effective\_vaccination\_rate): (float, float) tuple,

vaccination percentage and rate to be used by the controller

"""

vaccinated\_percentage = self.vaccinated\_percentage\_curve\_[-1]

effective\_vaccination\_rate = \

self.vaccination\_rate\_curve\_[-1] - self.\_vaccinationFailureRate()

return (vaccinated\_percentage, effective\_vaccination\_rate)

# method to visualize the results for the homework

def viewVaccination(self, point\_ss, vaccination\_cost, save\_dir='', filename='vaccination', show\_plot=True):

"""

plots multiple curves for the vaccination and

saves the resultant plot as a png image

Arguments:

----------

point\_ss: int, the estimated iteration index at which the system is at steady state

vaccination\_cost: float, the estimated cost of the vaccination until the steady state

save\_dir: string, indicating the path to directory where the plot image is to be saved

filename: string, indicating the name of the image file. Note that .png will be automatically

appended to the filename.

show\_plot: bool, whether the figure is to be shown

Example:

--------

visualizing the results of the vaccination

# assume many control signals have been consecutively applied to vaccine people

>>> my\_vaccine = Vaccination()

>>> my\_vaccine.vaccinatePeople(vaccination\_control) # assume this has been repeated many times

>>> # assume state state index and the vaccination cost have been computed

>>> # as point\_ss=k and vaccination\_cost=c

>>> my\_vaccine.viewVaccination(point\_ss=k, vaccination\_cost=c,

>>> save\_dir='some\location\to\save', filename='vaccination')

"""

color\_list = ['#ff0000', '#32CD32', '#0000ff', '#d2691e', '#ff00ff', '#000000', '#373788']

style\_list = ['-', '--']

num\_plots = 3

plot\_curve\_args = [{'c': color\_list[k],

'linestyle': style\_list[0],

'linewidth': 3} for k in range(num\_plots)]

plot\_vert\_args = [{'c': color\_list[k],

'linestyle': style\_list[1],

'linewidth': 3} for k in range(num\_plots)]

font\_size = 18

fig, axes = plt.subplots(3, 1, figsize=(16, 12))

day\_x = [i \* .1 for i in range(len(self.vaccinated\_percentage\_curve\_))]

x\_ticks = day\_x[::10]

# vaccinated population

ax = axes[0]

ax.set\_title('vaccinated population percentage over days', loc='left', fontsize=font\_size)

ax.plot(day\_x[:point\_ss+1], self.vaccinated\_percentage\_curve\_[:point\_ss+1], \*\*plot\_curve\_args[0])

ax.plot(day\_x[point\_ss:], self.vaccinated\_percentage\_curve\_[point\_ss:], \*\*plot\_curve\_args[1])

ax.plot([day\_x[point\_ss]] \* 2, [0, self.vaccinated\_percentage\_curve\_[point\_ss]], \*\*plot\_vert\_args[2])

ax.set\_xlabel(xlabel='day', fontsize=font\_size)

ax.set\_ylabel(ylabel='vaccinated population %', fontsize=font\_size)

ax.set\_xticks(x\_ticks)

ax.xaxis.set\_minor\_locator(ticker.FixedLocator([day\_x[point\_ss]]))

ax.xaxis.set\_minor\_formatter(ticker.ScalarFormatter())

ax.tick\_params(which='minor', length=17, color='b', labelsize=13)

ax.tick\_params(labelsize=15)

ax.set\_ylim(bottom=0)

ax.set\_xlim(left=0)

ax.grid(True, lw = 1, ls = '--', c = '.75')

# vaccination rate

ax = axes[1]

ax.set\_title('vaccination rate over days', loc='left', fontsize=font\_size)

ax.plot(day\_x[:point\_ss + 1], self.vaccination\_rate\_curve\_[:point\_ss + 1],

\*\*plot\_curve\_args[0])

ax.plot(day\_x[point\_ss:], self.vaccination\_rate\_curve\_[point\_ss:],

\*\*plot\_curve\_args[1])

ax.plot([day\_x[point\_ss]] \* 2, [0, self.vaccination\_rate\_curve\_[point\_ss]],

\*\*plot\_vert\_args[2])

ax.fill\_between(day\_x[:point\_ss + 1], 0, self.vaccination\_rate\_curve\_[:point\_ss + 1],

facecolor='#FF69B4', alpha=0.7)

ax.text(1.5, .01, 'cost = %.2f'%vaccination\_cost,

horizontalalignment='center', fontsize=font\_size)

ax.set\_xlabel(xlabel='day', fontsize=font\_size)

ax.set\_ylabel(ylabel='vaccination rate (%/day)', fontsize=font\_size)

ax.set\_xticks(x\_ticks)

ax.xaxis.set\_minor\_locator(ticker.FixedLocator([day\_x[point\_ss]]))

ax.xaxis.set\_minor\_formatter(ticker.ScalarFormatter())

ax.tick\_params(which='minor', length=17, color='b', labelsize=13)

ax.tick\_params(labelsize=15)

ax.set\_ylim(bottom=0)

ax.set\_xlim(left=0)

ax.grid(True, lw=1, ls='--', c='.75')

# vaccination rate control

ax = axes[2]

ax.set\_title('vaccination rate control over days', loc='left', fontsize=font\_size)

ax.plot(day\_x[:point\_ss + 1], self.vaccination\_control\_curve\_[:point\_ss + 1],

\*\*plot\_curve\_args[0])

ax.plot(day\_x[point\_ss:], self.vaccination\_control\_curve\_[point\_ss:],

\*\*plot\_curve\_args[1])

y\_min = ax.get\_ylim()[0]

ax.plot([day\_x[point\_ss]] \* 2,

[y\_min, self.vaccination\_control\_curve\_[point\_ss]],

\*\*plot\_vert\_args[2])

ax.set\_xlabel(xlabel='day', fontsize=font\_size)

ax.set\_ylabel(ylabel='vaccination rate control (%/day)', fontsize=font\_size)

ax.set\_xticks(x\_ticks)

ax.xaxis.set\_minor\_locator(ticker.FixedLocator([day\_x[point\_ss]]))

ax.xaxis.set\_minor\_formatter(ticker.ScalarFormatter())

ax.tick\_params(which='minor', length=17, color='b', labelsize=13)

ax.tick\_params(labelsize=15)

ax.set\_xlim(left=0)

ax.set\_ylim(bottom=y\_min)

ax.grid(True, lw=1, ls='--', c='.75')

if show\_plot:

plt.show()

fig.savefig(os.path.join(save\_dir, filename + '.png'))

# New Antecedent/Consequent objects hold universe variables and membership

# functions

#antecedent for input consequent for output

cur\_vac = ctrl.Antecedent(np.arange(0, 1.1, 0.1), 'Currently Vaccinated')

eff\_vac=ctrl.Antecedent(np.arange(-1.0, 1.1, 0.1), 'Effective Vaccination')

out\_rate = ctrl.Consequent(np.arange(-0.2, 0.25, 0.05), 'Output Rate')

# Custom membership functions can be built interactively with a familiar,

# Pythonic API

#trapmf for trapezoid functions 4 edges provide

#trimf for triangular functions 3 edges provided

cur\_vac['low'] = fuzz.trapmf(cur\_vac.universe, [0, 0, 0.2, 0.6])

cur\_vac['medium'] = fuzz.trimf(cur\_vac.universe, [0.3, 0.6, 0.9])

cur\_vac['high'] = fuzz.trapmf(cur\_vac.universe, [0.6, 0.9, 1.0, 1.0])

eff\_vac['low'] = fuzz.trapmf(eff\_vac.universe, [-1., -1., -0.5, 0.])

eff\_vac['medium'] = fuzz.trimf(eff\_vac.universe, [-0.5, 0., 0.5])

eff\_vac['high'] = fuzz.trapmf(eff\_vac.universe, [0., 0.5, 1.0, 1.0])

out\_rate['low'] = fuzz.trapmf(out\_rate.universe, [-0.2,-0.2, -0.15, 0])

out\_rate['medium'] = fuzz.trimf(out\_rate.universe, [-0.1, 0, 0.1])

out\_rate['high'] = fuzz.trapmf(out\_rate.universe, [0, 0.15, 0.20,0.20])

# View membership functions

cur\_vac.view()

eff\_vac.view()

out\_rate.view()

#set rules

rule1 = ctrl.Rule(cur\_vac['low'] & eff\_vac['low'] , out\_rate['high'])

rule2 = ctrl.Rule(cur\_vac['low'] & eff\_vac['medium'] , out\_rate['high'])

rule3 = ctrl.Rule(cur\_vac['low'] & eff\_vac['high'] , out\_rate['medium'])

rule4 = ctrl.Rule(cur\_vac['medium']& eff\_vac['low'], out\_rate['high'])

rule5 = ctrl.Rule(cur\_vac['medium']& eff\_vac['high'], out\_rate['low'])

rule6 = ctrl.Rule(cur\_vac['medium']& eff\_vac['medium'], out\_rate['medium'])

rule7 = ctrl.Rule(cur\_vac['high']& eff\_vac['low'] , out\_rate['medium'])

rule8 = ctrl.Rule(cur\_vac['high']& eff\_vac['high'] , out\_rate['low'])

rule9 = ctrl.Rule(cur\_vac['high']& eff\_vac['medium'] , out\_rate['low'])

#rule1.view()

#create a control system to simulate

vaccination\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3,rule4,rule5,rule6,rule7,rule8,rule9])

vaccinate = ctrl.ControlSystemSimulation(vaccination\_ctrl)

# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API

# Note: if you like passing many inputs all at once, use .inputs(dict\_of\_data)

vaccination=Vaccination()

(pi,pi\_dot)= vaccination.checkVaccinationStatus()

# effective= []

# effective.append(pi\_dot)

for i in range(200):

# print(vaccination.vaccination\_rate\_curve\_)

vaccinate.input['Currently Vaccinated'] = pi

vaccinate.input['Effective Vaccination'] = pi\_dot

# Crunch the numbers

vaccinate.compute()

#viusalize output

#print (vaccinate.output['Output Rate'])

#out\_rate.view(sim=vaccinate)

vaccination.vaccinatePeople(vaccinate.output['Output Rate'])

(pi,pi\_dot)= vaccination.checkVaccinationStatus()

# effective.append(pi\_dot)

plt.figure()

#plot to see vaccinated percentage curve

x=np.arange(0,201)

plt.plot(x,vaccination.vaccinated\_percentage\_curve\_)

plt.xlim([0, 200])

plt.title( "Vaccinated Percentage Curve")

plt.show()

#plot to see Vaccination Rate Curve

plt.plot(x,vaccination.vaccination\_rate\_curve\_)

plt.xlim([0, 200])

plt.title( "Vaccination Rate Curve")

plt.show()

vac\_cost=vaccination.vaccination\_rate\_curve\_[0:30]

cost=np.sum(vac\_cost)\*100

vaccination.viewVaccination(30, cost, save\_dir='/Users/gulceonder/Desktop/webpage/webdevelopment', filename='vsc', show\_plot=True)

#plot effective vacc rate curve

# plt.plot(x,effective)

plt.show()