```
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
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-v2')
```