ECE 449 - Intelligent Systems Engineering

Lab 4-D42: Fuzzy Logic Decision System

Lab date: *Thursday, November 18, 2021 -- 2:00 - 4:50 PM*
Room: ETLC E5-013
Lab report due: *Saturday, November 28, 2021 -- 3:50 PM*

---

1. Objectives

The objective of this lab is to build a fuzzy controller, based on the concepts developed in the first lab. This controller will decide what duty cycle should be used, based on the *state of charge* and *future average power*. The concepts involved in building a fuzzy controller include:

- defining membership functions
- defining fuzzy rules to build a ruleset
- calculating a fuzzy lookup table, based on the ruleset

2. Expectations

Complete the pre-lab, and hand it in before the lab starts. A formal lab report is required for this lab, which will be the completed version of this notebook. There is a marking guide at the end of the lab manual. If figures are required, label the axes and provide a legend when appropriate. An abstract, introduction, and conclusion are required as well, for which cells are provided at the end of the notebook. The abstract should be a brief description of the topic, the introduction a description of the goals of the lab, and the conclusion a summary of what you learned, what you found difficult, and your own ideas and observations.

3. Pre-lab

1. Describe what a fuzzy lookup table is and how you would use it. Given that you know all of the fuzzy rules and sets, explain how you would calculate an element of a fuzzy lookup table.

4. Introduction

A *fuzzy controller* takes inputs, processes these inputs according to a programmed ruleset, and outputs the decision as a crisp value. A fuzzy rule is written in the following form:

IF U_1 **IS** A_1 **AND** U_2 **IS** A_2 **THEN** V **IS** B
U_1, U_2 - inputs; linguistic variables with elements $x_1 \in X_1$ and $x_2 \in X_2$
V - output; linguistic variable with elements $y \in Y$
$A_1(x_1)$, $A_2(x_2)$, $B(y)$ - linguistic values represented by fuzzy sets

Assuming that U_1 and U_2 are given as crisp inputs, evaluating a fuzzy rule is performed with the following procedure:

1. Obtain the memberships of U_1 and U_2 in sets A_1 and A_2 (fuzzification). This will result in two numbers, the membership values.
2. Perform the appropriate operation for the operator that joins the inputs of the rule. For example, the **AND** operator could correspond to a minimum function. This will result in a single number, the minimimum of the two fuzzified inputs.
3. Perform the implication (**IF/THEN** operator) between the number obtained from step 2 and the output membership function C. For example, if Larsen implication were used, C would be multiplied by the scalar value obtained from step 2. This outputs a membership function of the output variable.
   Remember, an implication proposition is a relation which can take different forms depending on the combination of the input fuzzy sets, Larsen is one of those combinations.

There are usually multiple rules that describe a fuzzy controller. These rules would be in the following form:

IF case 1 **THEN** *V* **IS** B_1 **ELSE**
IF case 2 **THEN** *V* **IS** B_2 **ELSE**
IF case 3 **THEN** *V* **IS** B_3

In order to combine these rules mathematically, each rule would be evaluated separately, following the procedure previously described, to obtain three membership functions defined on the universe set of variable *V*. The final result is obtained by combining these membership functions together with the operation described by the **ELSE** operator. This could be performed using the maximum operation, yielding a single membership function that can be defuzzified in order to obtain the crisp value or decision.

5. Background

A team of climatologists has noticed that the monitoring station in Resolute, Nunavut is not recording data (duty cycle of 0) for a significant period of time during the winter. Based on typical meteorological data of Resolute, the team has built a simulation model to calculate the state of charge and future average power. They determined that the power outages can be attributed to the scarcity of renewable resources during the winter. Therefore, they are designing a fuzzy controller to manage the station's power consumption. The fuzzy ruleset that they plan to use is:

Rule #	*Rule*
1	**IF** (*SOC* **IS** LOW **AND** *power* **IS** $SCARCE$) **THEN** *duty cycle* **IS** LOW **ELSI**
2	**IF** (*SOC* **IS** LOW **AND** *power* **IS** $AVERAGE$) **THEN** *duty cycle* **IS** $MEDIUI$
3	**IF** (*SOC* **IS** LOW **AND** *power* **IS** $ABUNDANT$) **THEN** *duty cycle* **IS** $MEDI$
4	**IF** (*SOC* **IS** $MEDIUM$ **AND** *power* **IS** $SCARCE$) **THEN** *duty cycle* **IS** MED
5	**IF** (*SOC* **IS** $MEDIUM$ **AND** *power* **IS** $AVERAGE$) **THEN** *duty cycle* **IS** ME
6	**IF** (*SOC* **IS** $MEDIUM$ **AND** *power* **IS** $ABUNDANT$) **THEN** *duty cycle* **IS** H
7	**IF** (*SOC* **IS** $HIGH$ **AND** *power* **IS** $SCARCE$) **THEN** *duty cycle* **IS** $HIGH$ **EI**
8	**IF** (*SOC* **IS** $HIGH$ **AND** *power* **IS** $AVERAGE$) **THEN** *duty cycle* **IS** $HIGH$ **I**
9	**IF** (*SOC* **IS** $HIGH$ **AND** *power* **IS** $ABUNDANT$) **THEN** *duty cycle* **IS** $HIGI$

6. Experimental Procedure

Run the cell below to import the libraries and the simulation model required to complete this lab. Also, ensure that the Resolute dataset is placed in the same directory as this Jupyter notebook.

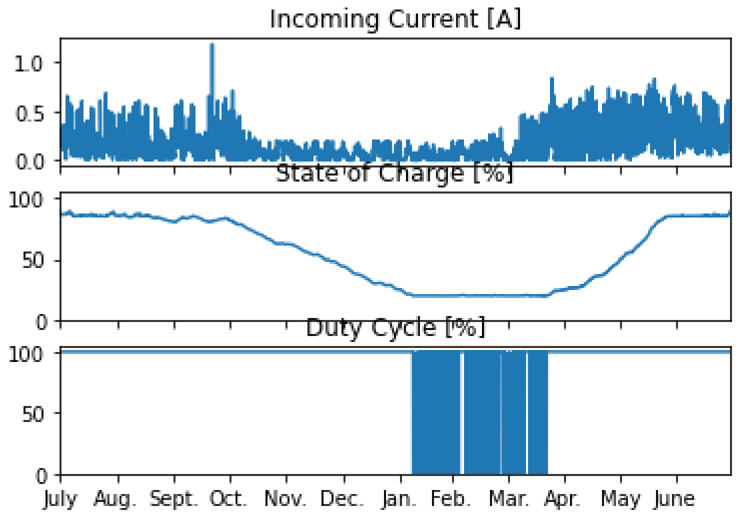
In [1]:

```
%matplotlib inline

import numpy as np                                  # General math operations
import scipy.io as sio                              # Loads .mat variables
import matplotlib.pyplot as plt                     # Data visualization
from mpl_toolkits.mplot3d import Axes3D             # 3D data visualization
import skfuzzy as fuzz                              # Fuzzy toolbox

# Represents an arctic weather monitoring station
class WeatherStation(object):
    fuzzyLookupTable = 0

    def __init__(self, fileName):
        # Load dataset of the weather for typical meteorological months into
        dataset = sio.loadmat(fileName)
        self.latitude = dataset['latitude']                    # Latitude (°N) c
        self.longitude = dataset['longitude']                  # Longitude (°E)
        self.lstMeridian = dataset['lstMeridian']              # Local standard
        self.albedo = dataset['albedo']                        # Albedo (measure
        self.modules = dataset['modules']                      # Number of discr
        self.minSOC = dataset['minSOC']                        # Minimum allowed
        self.bCapacity = dataset['bCapacity']                  # Nominal capacit
        self.panelArea = dataset['panelArea']                  # Area of the sol
        self.inclination = dataset['inclination']              # Inclination of
        self.azimuth = dataset['azimuth']                      # Azimuth of the

        self.time = dataset['time']                            # Hour number
        self.normalRadiation = dataset['normalRadiation']      # Direct normal r
        self.diffuseRadiation = dataset['diffuseRadiation']    # Diffuse horizon
        self.airTemperature = dataset['airTemperature']        # Air temperature
        self.windSpeed = dataset['windSpeed']                  # Wind speed

        self.hours = len(self.time)                            # Total number of

    ### Calculates the energy generated from a flutter generator for a given
    def flutterGenerator(self, wSpeed):
        if (wSpeed >= 2.3):
            wEnergy = 25 * wSpeed - 56
        else:
            wEnergy = 0

        # Limit wind energy to 319mW
        if (wEnergy > 319):
            wEnergy = 319

        wEnergy = self.modules * wEnergy / 1000

        return wEnergy

    ### Calculates the energy generated from a solar panel for a given hour
    def solarPanel(self, hour, nRadiation, dRadiation, wSpeed, aTemp):
        if (dRadiation == 0):
            sEnergy = 0
        else:
            # 1. DIFFUSE RADIATION ON INCLINED SURFACE
            day = (np.floor(((hour + 4344) / 24)) % 365) + 1  # Current day o
```

```
declination = 23.442 * np.sin(np.deg2rad((360 / 365) * (284 + day
B = np.deg2rad((day - 1) * (360 / 365))
eqnOfTime = 229.2 * (0.000075 + (0.001868 * np.cos(B)) - (0.03207
localStandardTime = hour % 24
localSolarTime = localStandardTime + 4 / 60 * (self.longitude - s
hourAngle = (localSolarTime - 12) * 15
sunriseHourAngle = np.tan(np.deg2rad(self.latitude)) * np.tan(np.

# Midpoint hour at sunrise
if (sunriseHourAngle > hourAngle - 15):
    hourAngle = (hourAngle + sunriseHourAngle) / 2
# Midpoint hour at sunset
elif (-sunriseHourAngle < hourAngle):
    hourAngle = (hourAngle - 15 - sunriseHourAngle) / 2
# Midpoint hour between sunrise and sunset
else:
    hourAngle -= 7.5

solarAltitude = max([0, np.cos(np.deg2rad(self.latitude)) * np.co
solarAltitudeAngle = np.rad2deg(np.arcsin(solarAltitude))
incidence = np.cos(np.pi * self.inclination / 180) * solarAltitud
solarZenithAngle = 90 - solarAltitudeAngle
clearnessRange = np.array([1, 1.065, 1.23, 1.5, 1.95, 2.8, 4.5, 6
f11 = np.array([-0.008, 0.130, 0.330, 0.568, 0.873, 1.132, 1.060,
f12 = np.array([0.588, 0.683, 0.487, 0.187, -0.392, -1.237, -1.6,
f13 = np.array([-0.062, -0.151, -0.221, -0.295, -0.362, -0.412, -
f21 = np.array([-0.06, -0.019, 0.055, 0.109, 0.226, 0.288, 0.264,
f22 = np.array([0.072, 0.066, -0.064, -0.152, -0.462, -0.823, -1.
f23 = np.array([-0.022, -0.029, -0.026, 0.014, 0.001, 0.056, 0.13

# Clearness category
clearnessCategory = 7  # Clear sky
if (dRadiation > 0):
    clearness = 1 + nRadiation / (dRadiation * (1 + 0.000005535 *
    while (clearnessRange[clearnessCategory] > clearness):
        clearnessCategory -= 1
airMass = 1 / (solarAltitude + 0.5057 * (96.08 - solarZenithAngle
extraterrestrialNormalIncidenceRadiation = 1367 * (1 + 0.033 * np
brightness = airMass * dRadiation / extraterrestrialNormalInciden

# Brightness coefficients
F1 = max([0, (f11[clearnessCategory] + f12[clearnessCategory] * b
F2 = f21[clearnessCategory] + f22[clearnessCategory] * brightness
aB = max([0, incidence]) / max([0.0872, solarAltitude])
isotropicSkyDiffuseRadiation = max([0, (dRadiation * (1 - F1) * (
circumsolarDiffuseRadiation = max([0, (dRadiation * F1 * aB)])
horizonDiffuseRadiation = max([0, (dRadiation * F2 * np.sin(np.de

# 2. GROUND REFLECTED RADIATION AND DIRECT RADIATION ON INCLINED 
directHorizontalRadiation = nRadiation * solarAltitude
groundReflectedRadiation = self.albedo * (dRadiation + directHori
ratioBeamRadiation = aB
directRadiation = directHorizontalRadiation * ratioBeamRadiation 

# 3. TOTAL RADIATION ON INCLINED SURFACE AND SUN ENERGY
totalRadiationTilted = directRadiation + isotropicSkyDiffuseRadia
cellTemperature = totalRadiationTilted * (np.exp(-3.56 - 0.075 * 
            efficiency = 0.155 - 0.0006 * cellTemperature
            efficiency *= np.interp(totalRadiationTilted, [0, 27, 93, 200, 40
            sEnergy = totalRadiationTilted * self.panelArea * efficiency
        return sEnergy

    ### Calculates the net current and energy generated
    def energy(self, wEnergy, sEnergy):
        # 90% efficiency
        netEnergy = (sEnergy + wEnergy) * 0.9

        # 12V system
        incCurrent = netEnergy / 12

        return (incCurrent, netEnergy)

    ### Calculates the future energy availability by using a moving average f
    def futureAvailability(self, netEnergy, hourWindow):
        initHours = len(netEnergy)
        futureEnergyAvg = np.zeros((initHours, 1))
        # Augment netEnergy to account for the hour window
        np.resize(netEnergy, (initHours + hourWindow, 1))

        for hour in range(initHours):
            futureEnergyAvg[hour] = np.mean(netEnergy[hour:hour + hourWindow]

        # Scale values from 0 to 100
        futureEnergyAvg = futureEnergyAvg - np.min(futureEnergyAvg)
        futureEnergyAvg = np.round(futureEnergyAvg / np.max(futureEnergyAvg)

        return futureEnergyAvg

    ### Calculates the current consumed by the load
    def load(self, dutyCycle, satellite, prevDataBuffer):
        satellite = min(satellite, prevDataBuffer * 38.55)

        # Consumption: 0.039A - data logger; 0.036A - sensors; 0.054A - satel
        outCurrent = dutyCycle / 100 * (0.039 + 0.036) + satellite * 0.054

        # Data buffer capacity is 1MB or 3855 hours of recordings
        dBuffer = prevDataBuffer + (1 - satellite) / 38.55
        return (outCurrent, dBuffer)

    ### Determines the state of the monitoring station based on the state of
    def powerManagement(self, prevDutyCycle, prevOutCurrent, incCurrent, hour
        # FUZZY ENERGY MANAGEMENT
        if (fuzzy == 1):
            if (prevSOC >= self.minSOC):
                # Update at midnight only to avoid oscillations
                if (hour % 24 == 0):
                    dutyCycle = fuzzyLookupTable[futureEnergyAvg.astype(int),
                else:
                    dutyCycle = prevDutyCycle
                satellite = 1  # Transmit data every hour
            else:
                dutyCycle = 0
                satellite = 0
        # SIMPLE ENERGY MANAGEMENT
        else:
            if (prevSOC >= self.minSOC):
                dutyCycle = 100
                satellite = 1
            else:
                dutyCycle = 0
                satellite = 0
        bCurrent = incCurrent - prevOutCurrent
        return (dutyCycle, satellite, bCurrent)

    ### Calculates the state of charge of the battery
    def battery(self, bCurrent, aTemp, prevExcessEnergyAvg, prevStoredEnergy)
        # Hourly self-discharge
        storedEnergy = prevStoredEnergy * 0.99997

        # Add incoming current
        if (bCurrent >= 0):
            storedEnergy += bCurrent * 0.9  # Charging efficiency of 90%
        else:
            storedEnergy += bCurrent / 0.95  # Discharging efficiency of 95%

        # Temperature effect
        if (aTemp >= 25):
            maxCapacity = self.bCapacity
        elif (aTemp < 0):
            maxCapacity = self.bCapacity * 0.85
        else:
            maxCapacity = self.bCapacity * (0.6 * aTemp + 85) / 100

        if storedEnergy > maxCapacity:
            excessEnergyAvg = prevExcessEnergyAvg + storedEnergy - maxCapacit
        else:
            excessEnergyAvg = prevExcessEnergyAvg

        # Stored energy saturation
        storedEnergy = max([0, min([maxCapacity, storedEnergy])])
        SOC = storedEnergy / self.bCapacity * 100
        return (SOC, excessEnergyAvg, storedEnergy)

    ### Sets the fuzzy lookup table to be used in the simulation
    def setFuzzyLookupTable(self, fuzzyLookupTable):
        self.fuzzyLookupTable = fuzzyLookupTable

    ### Performs a simulation using the typical meteorological year dataset
    def simulate(self, fuzzy):
        # Pre-allocate arrays with zeros
        windEnergy = np.zeros((self.hours, 1))          # Wind energy
        sunEnergy = np.zeros((self.hours, 1))           # Sun energy
        incomingCurrent = np.zeros((self.hours, 1))     # Incoming current
        netEnergy = np.zeros((self.hours, 1))           # Net energy

        # Compute energy obtained from weather conditions
        for hour in self.time:
            sunEnergy[hour] = self.solarPanel(hour, self.normalRadiation[hour
            windEnergy[hour] = self.flutterGenerator(self.windSpeed[hour])
            (incomingCurrent[hour], netEnergy[hour]) = self.energy(windEnergy
        futureEnergyAvg = self.futureAvailability(netEnergy, 2159)  # 90 day

        # Pre-allocate arrays with zeros
        outgoingCurrent = np.zeros((self.hours, 1))
        dataBuffer = np.zeros((self.hours, 1))
        dutyCycle = np.zeros((self.hours, 1))
        satellite = np.zeros((self.hours, 1))
        batteryCurrent = np.zeros((self.hours, 1))
        SOC = np.zeros((self.hours, 1))
        excessEnergyAvg = np.zeros((self.hours, 1))
        storedEnergy = np.zeros((self.hours, 1))

        # Determine the first element of each array based on the initial cond
        (dutyCycle[0], satellite[0], batteryCurrent[0]) = self.powerManagemen
        (SOC[0], excessEnergyAvg[0], storedEnergy[0]) = self.battery(batteryC
        (outgoingCurrent[0], dataBuffer[0]) = self.load(100, satellite[0], 0)

        # Calculate the rest of the elements in the arrays
        for hour in self.time[1:]:
            (dutyCycle[hour], satellite[hour], batteryCurrent[hour]) = self.p
            (SOC[hour], excessEnergyAvg[hour], storedEnergy[hour]) = self.bat
            (outgoingCurrent[hour], dataBuffer[hour]) = self.load(dutyCycle[h

        # Plot the results
        f, axarr = plt.subplots(3, sharex = True)
        axarr[0].plot(self.time, incomingCurrent)
        axarr[0].set_title('Incoming Current [A]')
        axarr[1].plot(self.time, SOC)
        axarr[1].set_title('State of Charge [%]')
        axarr[1].set_ylim([0, 105])
        axarr[2].plot(self.time, dutyCycle)
        axarr[2].set_title('Duty Cycle [%]')
        axarr[2].set_xlim([0, self.hours])
        axarr[2].set_ylim([0, 105])
        plt.xticks([0, 744, 1488, 2208, 2952, 3696, 4440, 5112, 5856, 6576, 7
        plt.show()
```

Run the cell below to verify that power outages do occur, and that the duty cycle of the station becomes 0 for extended periods from January to April.

In [2]:
```python
# Create a WeatherStation object for the Resolute data
resolute = WeatherStation('resolutedata.mat')  # Make sure that resolutedata.
resolute.simulate(0)  # Simulate using the simple algorithm
```



Exercise 1: Membership functions
The membership functions for state of charge and future average power are the same as last lab, and have been provided for you in the following cell. However, the duty cycle membership functions must be defined. This value can range from 25% to 100%.

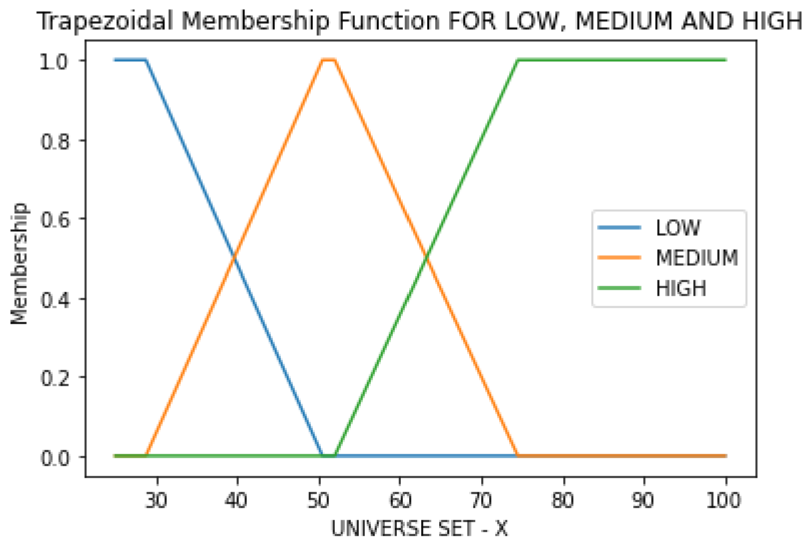
1. Define the universe set for duty cycle, using 101 elements.

In [3]:
```
# Define membership functions for state of charge
uniSOC = np.linspace(20, 100, num = 81)
lowSOC = fuzz.trapmf(uniSOC, [20, 20, 22, 38])
mediumSOC = fuzz.trapmf(uniSOC, [22, 38, 42, 58])
highSOC = fuzz.trapmf(uniSOC, [42, 58, 100, 100])

# Define membership functions for future average power
uniP = np.linspace(0, 100, num = 101)
scarceP = fuzz.trapmf(uniP, [0, 0, 30, 35])
averageP = fuzz.trapmf(uniP, [30, 35, 40 ,45])
abundantP = fuzz.trapmf(uniP, [40, 45, 100, 100])

### WRITE CODE BELOW THIS LINE ###
X = np.linspace(25,100, num = 101)      #Universe Set for Duty Cycle
```

2. Plot the trapezoidal membership functions, LOW, MEDIUM, and HIGH, on one figure according to the parameters given below.

Fuzzy set	*a*	*b*	*c*	*d*
LOW	25	25	28.75	50.50
MEDIUM	28.75	50.50	52	74.50
HIGH	52	74.50	100	100

In [4]:
```
LOW = fuzz.trapmf(X, [25,25,28.75,50.50])
MEDIUM = fuzz.trapmf(X, [28.75,50.50,52,74.50])
HIGH = fuzz.trapmf(X, [52,74.50,100,100])

#Plotting MF FOR LOW
lowHandle ,= plt.plot(X,LOW, label = 'LOW')
#Plotting MF FOR MEDIUM
mediumHandle ,= plt.plot(X,MEDIUM, label = 'MEDIUM')
#Plotting MF FOR HIGH
highHandle ,= plt.plot(X,HIGH, label = 'HIGH')

plt.xlabel('UNIVERSE SET - X')
plt.ylabel('Membership')
plt.title('Trapezoidal Membership Function FOR LOW, MEDIUM AND HIGH')
plt.legend(handles = [lowHandle, mediumHandle, highHandle])
plt.show()
```



Exercise 2: Fuzzy lookup table
Determine the fuzzy lookup table, *fuzzyLookupTable*, that provides the values of duty cycle for each combination of state of charge and future average power. The matrix should be of dimension $M \times N$, where M is the number of elements in the state of charge universe set and N is the number of elements in the future average power universe set. This means that the element at $(0, 0)$ should yield the duty cycle if the state of charge were at 20% and the future average power were 0W.
The operations that are used to realize the operators in the fuzzy ruleset can be found below. They are arranged in the order that they should be performed.

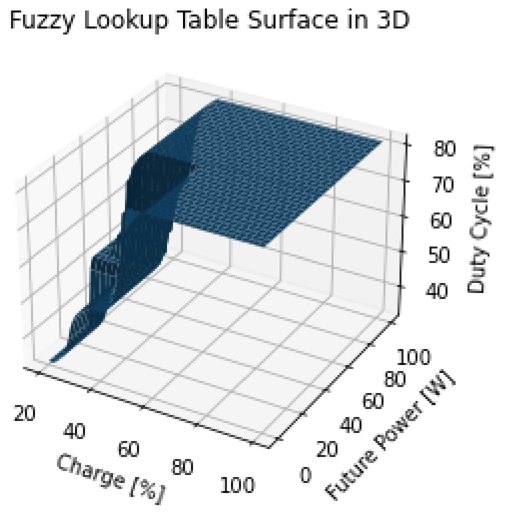
i. **AND** - *Larsen*
ii. **IF/THEN** - *Larsen*
iii. **ELSE** - *maximum*
iv. Defuzzification - *bisector*

1. Once all of the elements are calculated, plot the fuzzy lookup table.

In [32]:
```
# Initialize variables to store values necessary for the fuzzy controller
M = 81   # Number of elements in SOC universe set
N = 101   # Number of elements in the future average power universe set

# define an array of zeros, what should be the size of the array?
fuzzyLookupTable = np.zeros((M,N))   # A fuzzy look up table with size MxN

# define the rule set as an array
rules = np.array([[0,(lowSOC, scarceP, LOW)],[1,(lowSOC, averageP, MEDIUM)],
                  [2,(lowSOC, abundantP, MEDIUM)],[3,(mediumSOC, scarceP, MED
                  [4,(mediumSOC, averageP, MEDIUM)],[5,(mediumSOC, abundantP,
                  [6, (highSOC, scarceP, HIGH)],[7,(highSOC, averageP, HIGH)]
                  [8,(highSOC, abundantP, HIGH)]])


# create arrays for SOC, future average power
#and duty cycle ---- Reference: Lab4 Slides, slide 15
arr_SOC = np.zeros(81)
arr_FuturePower = np.zeros(101)
arr_DutyCycle = np.zeros(101)

# create the structure where you are going to store the duty cycle results
dutyCycle = np.zeros((9,101))

# Determine fuzzy lookup table using product operation
#for AND/THEN and maximum operation for OR
for s in range(M):
    for p in range(N):
        for r, (arr_SOC, arr_FuturePower, arr_DutyCycle) in rules:
            dutyCycle[r] = arr_SOC[s] * arr_FuturePower[p] * arr_DutyCycle
        dcMax = dutyCycle[0]   # Structure for ELSE - maximum
        #Additional for loop added to the lab pseudo code
        #so that X and dcMax have same no of elements in order
        #to use fuzz.defuzz else an error was generated
        for i in range(0,8):
            # Reference:
            #https://numpy.org/doc/stable/reference/generated/numpy.maximum.h
            dcMax = np.maximum(dcMax, dutyCycle[i+1])
        fuzzyLookupTable[s,p] = fuzz.defuzz(X, dcMax, 'bisector')

# Plot the fuzzy lookup table surface in 3D
fig = plt.figure()
[gX, gY]= np.meshgrid(uniSOC, uniP, indexing = 'ij')
ax = fig.gca(projection = '3d')
ax.plot_surface(gX, gY, fuzzyLookupTable)
ax.set_xlabel('Charge [%]')
ax.set_ylabel('Future Power [W]')
ax.set_zlabel('Duty Cycle [%]')
ax.set_title('Fuzzy Lookup Table Surface in 3D')
plt.show()
```

```
/tmp/ipykernel_92/1062759336.py:10: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you
meant to do this, you must specify 'dtype=object' when creating the ndarra
y.
```

```
  rules = np.array([[0,(lowSOC, scarceP, LOW)],[1,(lowSOC, averageP, MEDIU
M)],[2,(lowSOC, abundantP, MEDIUM)],
/tmp/ipykernel_92/1062759336.py:39: MatplotlibDeprecationWarning: Calling g
ca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two
minor releases later, gca() will take no keyword arguments. The gca() funct
ion should only be used to get the current axes, or if no axes exist, creat
e new axes with default keyword arguments. To create a new axes with non-de
fault arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection = '3d')
```



Exercise 3: Making decisions
Four monitoring stations are checked at their current states and found to have the parameters listed in the table below.

$Station\ \#$	$State\ of\ charge\ [\%]$	$Future\ average\ power\ [W]$
1	20	100
2	28	23
3	50	21
4	88	91

The initial code provided in the cell below rounds all duty cycle values to 25%, 50%, 75%, or 100% to make the simulation results more intuitive and easier to read.

1. Determine what duty cycle each station should adopt, based on the new fuzzy lookup table, and print each result.

In [33]:

```
# Normalize the table, and set the duty cycle to take values of 25, 50, 75, o
fuzzyLookupTable -= np.amin(np.amin(fuzzyLookupTable))
fuzzyLookupTable /= np.amax(np.amax(fuzzyLookupTable))
fuzzyLookupTable = 75 * np.round(fuzzyLookupTable * 3) / 3 + 25

### WRITE CODE BELOW THIS LINE ###
# Duty cycle for station 1 using the fuzzy lookup table
station1_SOC = 20-20   # Subtracted by 20 to prevent an out of
                       # bounds value for axis 0 with size 81
station1_FuturePower = 100
station1_DutyCycle = fuzzyLookupTable[station1_SOC][station1_FuturePower]
print("Duty Cycle that station 1 should adopt: {dc}%".
      format(dc = station1_DutyCycle))

# Duty cycle for station 2 using the fuzzy lookup table
station2_SOC = 28-20   # Subtracted by 20 to prevent an out of
                       # bounds value for axis 0 with size 81
station2_FuturePower = 23
station2_DutyCycle = fuzzyLookupTable[station2_SOC][station2_FuturePower]
print("Duty Cycle that station 2 should adopt: {dc}%".
      format(dc = station2_DutyCycle))

# Duty cycle for station 3 using the fuzzy lookup table
station3_SOC = 50-20   # Subtracted by 20 to prevent an out of
                       # bounds value for axis 0 with size 81
station3_FuturePower = 21
station3_DutyCycle = fuzzyLookupTable[station3_SOC][station3_FuturePower]
print("Duty Cycle that station 3 should adopt: {dc}%".
      format(dc = station3_DutyCycle))

# Duty cycle for station 4 using the fuzzy lookup table
station4_SOC =88-20   # Subtracted by 20 to prevent an out of
                      # bounds value for axis 0 with size 81
station4_FuturePower = 91
station4_DutyCycle = fuzzyLookupTable[station4_SOC][station4_FuturePower]
print("Duty Cycle that station 4 should adopt: {dc}%".
      format(dc = station4_DutyCycle))
```

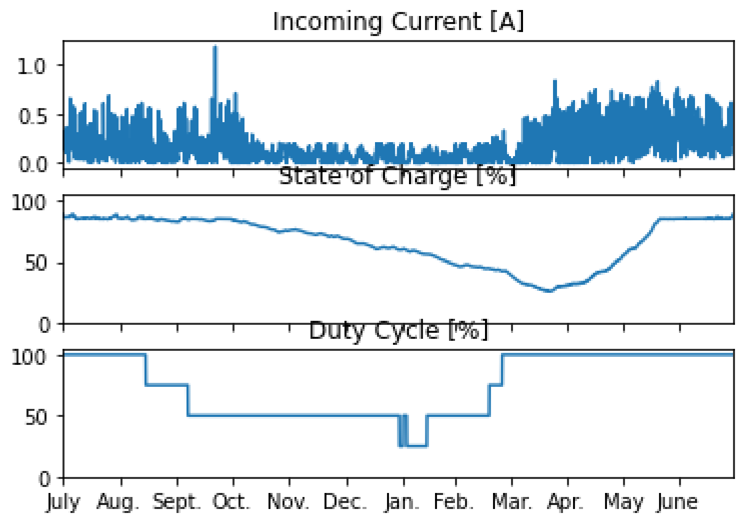
```
Duty Cycle that station 1 should adopt: 50.0%
Duty Cycle that station 2 should adopt: 25.0%
Duty Cycle that station 3 should adopt: 75.0%
Duty Cycle that station 4 should adopt: 100.0%
```

Exercise 4: Fuzzy simulation

Finally, run the cell below to simulate the monitoring station using the new fuzzy controller.

1. Discuss any changes that were noted and how the fuzzy controller managed the station's power consumption relative to the incoming current, generated from the renewable resources. Include the plot of your final results in the report.

In [34]:
```python
resolute.setFuzzyLookupTable(fuzzyLookupTable)
resolute.simulate(1)  # Simulate using the fuzzy algorithm
```



It was observed that initially the monitoring station had a duty cycle of 0 for extended periods from January to April. However, after implementing the simulation model/fuzzy controller using fuzzy rulesets and fuzzy lookup table, we observed that the value of duty cycle changed i.e., it became non zero for extended periods from January to April when there is a scarcity of renewable resources during the winter. We also observed that the fuzzy controller managed the station's power consumption relative to the incoming current, generated from the renewable resources in a sense that it would lower down the value of duty cycle around August until around March unlike previously. Thus, the controller is now saving the station’s power consumption relative to the incoming current generated from the renewable resources until winter, so that the duty cycle does not become zero in winter. We observe that the value of duty cycle drops from 100 to around 50 in September and then it drops further in January and then starts increasing rising again around March.

Abstract

A fuzzy logic controller describes a control protocol by means of if-then rules. It comprises of some major components like a fuzzifier, fuzzy knowledge base, a fuzzy rule base and a defuzzifier.A fuzzy controller works by taking inputs, then processing them according to a programmed ruleset, and finally producing a decision in the form of a crisp value.

A fuzzy rule is written in the form: IF U1 IS A1 AND U2 IS A2 THEN V IS B where, U1, and U2 are the inputs (given to the controller as crisp values) i.e., linguistic variables with elements x1∈ X1 and x2 ∈ X2, V is the output i.e., linguistic variable with elements y ∈ Y and A1(x1), A2(x2), B(y) are the linguistic values represented by fuzzy sets. A fuzzy rule is performed in three steps - firstly, by fuzzification i.e., obtaining the memberships of U1 and U2 in sets A1 and A2 which results in two numbers - the membership values. Then, an appropriate operation (like AND) is performed for the operator that joins the inputs of the rule. Finally, an implication i.e., IF/THEN operator is performed between the value obtained after the operation and the output membership function.

Fuzzy rules in a fuzzy controller are usually in the form:
 IF case 1 THEN V IS B1 ELSE
 IF case 2 THEN V IS B2 ELSE
 IF case 3 THEN V IS B3
These rules are evaluated separately and the final result is obtained using the ELSE operator to combine the membership functions. An ELSE operation for instance, can be performed using the maximum operation that will give us a single membership function which will be defuzzified to obtain a decision in the form of a crisp value.

Introduction

In this lab, our aim was to build a fuzzy controller using the concepts from the previous lab. The controller was designed to decide what duty cycle should be used by the monitoring station based on the state of charge and future average power. We used the concepts of defining membership functions, fuzzy rules to build a ruleset and calculating a fuzzy lookup table based on the ruleset in order to build the controller.

The problem addressed in this lab was that a team of climatologists noticed the monitoring station in Resolute, Nunavut recording a duty cycle of 0 for a significant period of time during the winter due to the scarcity of renewable resources during the winter. Thus, we built a fuzzy controller to manage the station's power consumption. This was done by using a fuzzy ruleset with 9 rules and a fuzzy lookup table. We created a fuzzy lookup table of dimensions MxN (here, M is the number of elements in the state of charge universe set and N is the number of elements in the future average power universe set) which provides the values of duty cycle for each combination of state of charge and future average power. The following operations were performed in an orderly fashion on the fuzzy ruleset - AND - Larsen, IF/THEN - Larsen, ELSE - maximum and Defuzzification - bisector.

We then made a decision for four stations using the fuzzy lookup table to decide which duty cycle each station should adopt. We finally observed the working of the controller by checking for non-zero duty cycle values in extended periods of winter by running fuzzy simulations. We also observed how the fuzzy controller managed the station's power consumption relative to the incoming current, generated from the renewable resources.

Conclusion

```
A fuzzy controller was designed in this lab to decide what duty cycle should
be used by the monitoring station based on the state of charge and future
average power. This was done by using concepts of defining membership
functions, fuzzy rules to build a ruleset and calculating a fuzzy lookup
table based on the ruleset in order to build the controller. We used a fuzzy
ruleset with 9 rules and a fuzzy lookup table.

A fuzzy lookup table of dimensions MxN (here, M is the number of elements in
the state of charge universe set and N is the number of elements in the
future average power universe set) was designed in this lab. The table
provided the values of duty cycle for each combination of state of charge and
future average power by performing the operations of :  AND - Larsen,
IF/THEN - Larsen, ELSE - maximum and Defuzzification - bisector on the fuzzy
ruleset. This table was also used to make a decision for four stations to
determine which duty cycle each station should adopt.

After designing the fuzzy controller, we observed that the value duty cycle
changed to a non zero value for extended periods from January to April when
there is a scarcity of renewable resources during the winter.However, it was
initially observed that the monitoring station had a duty cycle of 0 for
extended periods from January to April. The fuzzy controller also managed the
station's power consumption relative to the incoming current, generated from
the renewable resources by lowering the value of duty cycle until winter. In
conclusion, the controller saves the station’s power consumption relative to
the incoming current generated from the renewable resources until winter to
avoid a zero duty cycle value in winter.
```

Lab 2 Marking Guide

Exercise	Item	Total Marks	Earned Marks
	Pre − lab	10	
	Abstract	3	
	Introduction	3	
	Conclusion	4	
1	*Membership functions*	10	
2	*Algorithm*	30	
3	*Fuzzy lookup table*	10	
4	*Making decisions*	20	
5	*Fuzzy simulation*	10	
	TOTAL	100	