ECE 449 - Intelligent Systems Engineering

Lab 4-D41: Fuzzy Logic Decision System

Lab date: *Thursday, November 4, 2021 -- 2:00 - 4:50 PM*
Room: ETLC E5-013
Lab report due: *Saturday, November 14, 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 **ELSE**
2	**IF** (*SOC* **IS** LOW **AND** *power* **IS** $AVERAGE$) **THEN** *duty cycle* **IS** $MEDIUM$ **ELSE**
3	**IF** (*SOC* **IS** LOW **AND** *power* **IS** $ABUNDANT$) **THEN** *duty cycle* **IS** $MEDIUM$ **ELSE**
4	**IF** (*SOC* **IS** $MEDIUM$ **AND** *power* **IS** $SCARCE$) **THEN** *duty cycle* **IS** $MEDIUM$ **ELSE**
5	**IF** (*SOC* **IS** $MEDIUM$ **AND** *power* **IS** $AVERAGE$) **THEN** *duty cycle* **IS** $MEDIUM$ **ELSE**
6	**IF** (*SOC* **IS** $MEDIUM$ **AND** *power* **IS** $ABUNDANT$) **THEN** *duty cycle* **IS** $HIGH$ **ELSE**
7	**IF** (*SOC* **IS** $HIGH$ **AND** *power* **IS** $SCARCE$) **THEN** *duty cycle* **IS** $HIGH$ **ELSE**
8	**IF** (*SOC* **IS** $HIGH$ **AND** *power* **IS** $AVERAGE$) **THEN** *duty cycle* **IS** $HIGH$ **ELSE**
9	**IF** (*SOC* **IS** $HIGH$ **AND** *power* **IS** $ABUNDANT$) **THEN** *duty cycle* **IS** $HIGH$

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 a dictionary
        dataset = sio.loadmat(fileName)
        self.latitude = dataset['latitude']                 # Latitude (°N) of the AWS
        self.longitude = dataset['longitude']               # Longitude (°E) of the AWS
        self.lstMeridian = dataset['lstMeridian']           # Local standard time meridian (°E)
        self.albedo = dataset['albedo']                     # Albedo (measure of reflectance) at the AWS
        self.modules = dataset['modules']                   # Number of discrete generator modules
        self.minSOC = dataset['minSOC']                     # Minimum allowed state of charge (SOC) of the battery
        self.bCapacity = dataset['bCapacity']               # Nominal capacity of the battery
        self.panelArea = dataset['panelArea']               # Area of the solar panel
        self.inclination = dataset['inclination']           # Inclination of the solar panel
        self.azimuth = dataset['azimuth']                   # Azimuth of the solar panel

        self.time = dataset['time']                         # Hour number
        self.normalRadiation = dataset['normalRadiation']   # Direct normal radiation (perpendicular to sunlight r
        self.diffuseRadiation = dataset['diffuseRadiation'] # Diffuse horizontal radiation (scattered by particles
        self.airTemperature = dataset['airTemperature']     # Air temperature (dry bulb)
        self.windSpeed = dataset['windSpeed']               # Wind speed

        self.hours = len(self.time)                         # Total number of hours (8760 in a year)

    ### Calculates the energy generated from a flutter generator for a given hour
    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 of the year (4344 is used to specify t
            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)) - (0.032077 * np.sin(B)) - (0.014615 * np.cos(2
            localStandardTime = hour % 24
            localSolarTime = localStandardTime + 4 / 60 * (self.longitude - self.lstMeridian) + eqnOfTime
            hourAngle = (localSolarTime - 12) * 15
            sunriseHourAngle = np.tan(np.deg2rad(self.latitude)) * np.tan(np.deg2rad(declination))

            # 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.cos(np.deg2rad(declination)) * np.cos(n
            solarAltitudeAngle = np.rad2deg(np.arcsin(solarAltitude))
            incidence = np.cos(np.pi * self.inclination / 180) * solarAltitude + np.sin(np.pi * self.inclination /
            solarZenithAngle = 90 - solarAltitudeAngle
            clearnessRange = np.array([1, 1.065, 1.23, 1.5, 1.95, 2.8, 4.5, 6.2])
            f11 = np.array([-0.008, 0.130, 0.330, 0.568, 0.873, 1.132, 1.060, 0.678])
            f12 = np.array([0.588, 0.683, 0.487, 0.187, -0.392, -1.237, -1.6, -0.327])
            f13 = np.array([-0.062, -0.151, -0.221, -0.295, -0.362, -0.412, -0.359, 0.25])
            f21 = np.array([-0.06, -0.019, 0.055, 0.109, 0.226, 0.288, 0.264, 0.156])
            f22 = np.array([0.072, 0.066, -0.064, -0.152, -0.462, -0.823, -1.127, -1.377])
            f23 = np.array([-0.022, -0.029, -0.026, 0.014, 0.001, 0.056, 0.131, 0.251])

            # Clearness category
```

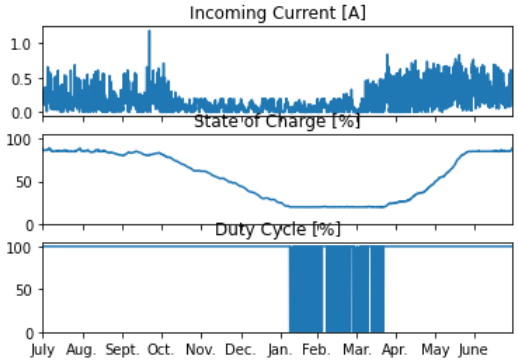
```
            clearnessCategory = 7  # Clear sky
            if (dRadiation > 0):
                clearness = 1 + nRadiation / (dRadiation * (1 + 0.000005535 * solarZenithAngle ** 3))
                while (clearnessRange[clearnessCategory] > clearness):
                    clearnessCategory -= 1
            airMass = 1 / (solarAltitude + 0.5057 * (96.08 - solarZenithAngle) ** (-1.634))
            extraterrestrialNormalIncidenceRadiation = 1367 * (1 + 0.033 * np.cos(np.deg2rad(360 * day / 365)))
            brightness = airMass * dRadiation / extraterrestrialNormalIncidenceRadiation

            # Brightness coefficients
            F1 = max([0, (f11[clearnessCategory] + f12[clearnessCategory] * brightness + f13[clearnessCategory] *
            F2 = f21[clearnessCategory] + f22[clearnessCategory] * brightness + f23[clearnessCategory] * solarZeni
            aB = max([0, incidence]) / max([0.0872, solarAltitude])
            isotropicSkyDiffuseRadiation = max([0, (dRadiation * (1 - F1) * (1 + np.cos(np.deg2rad(self.inclinatio
            circumsolarDiffuseRadiation = max([0, (dRadiation * F1 * aB)])
            horizonDiffuseRadiation = max([0, (dRadiation * F2 * np.sin(np.deg2rad(self.inclination)))])

            # 2. GROUND REFLECTED RADIATION AND DIRECT RADIATION ON INCLINED SURFACE
            directHorizontalRadiation = nRadiation * solarAltitude
            groundReflectedRadiation = self.albedo * (dRadiation + directHorizontalRadiation) * (1 - np.cos(np.deg
            ratioBeamRadiation = aB
            directRadiation = directHorizontalRadiation * ratioBeamRadiation

            # 3. TOTAL RADIATION ON INCLINED SURFACE AND SUN ENERGY
            totalRadiationTilted = directRadiation + isotropicSkyDiffuseRadiation + circumsolarDiffuseRadiation +
            cellTemperature = totalRadiationTilted * (np.exp(-3.56 - 0.075 * wSpeed) + 0.003) + aTemp
            efficiency = 0.155 - 0.0006 * cellTemperature
            efficiency *= np.interp(totalRadiationTilted, [0, 27, 93, 200, 400, 625, 1000, 3000], [0, 0.5, 0.75, 0
            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 filter
    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) * 100)

        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 - satellite * (hour data points)
        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 charge
    def powerManagement(self, prevDutyCycle, prevOutCurrent, incCurrent, hour, fuzzy, fuzzyLookupTable, futureEner
        # 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), (np.round((prevSOC - self.minSOC) /
                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 - maxCapacity
        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], self.diffuseRadiation[hour], self.
            windEnergy[hour] = self.flutterGenerator(self.windSpeed[hour])
            (incomingCurrent[hour], netEnergy[hour]) = self.energy(windEnergy[hour], sunEnergy[hour])

        futureEnergyAvg = self.futureAvailability(netEnergy, 2159)  # 90 day window

        # 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 conditions
        (dutyCycle[0], satellite[0], batteryCurrent[0]) = self.powerManagement(100, 0.129, incomingCurrent[0], 0,
        (SOC[0], excessEnergyAvg[0], storedEnergy[0]) = self.battery(batteryCurrent[0], self.airTemperature[0], 0,
        (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.powerManagement(dutyCycle[hour - 1], c
            (SOC[hour], excessEnergyAvg[hour], storedEnergy[hour]) = self.battery(batteryCurrent[hour], self.airTe
            (outgoingCurrent[hour], dataBuffer[hour]) = self.load(dutyCycle[hour], satellite[hour], dataBuffer[hou

        # 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, 7320, 8040], ('July', 'Aug.', 'Sept.',
        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]:
```
# Create a WeatherStation object for the Resolute data
resolute = WeatherStation('resolutedata.mat')  # Make sure that resolutedata.mat is in your project folder
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 ###
# Value can range from 25-100 and duty cycle will use 101 elements
uniDC = np.linspace(25, 100, num = 101)
print(uniDC)
```

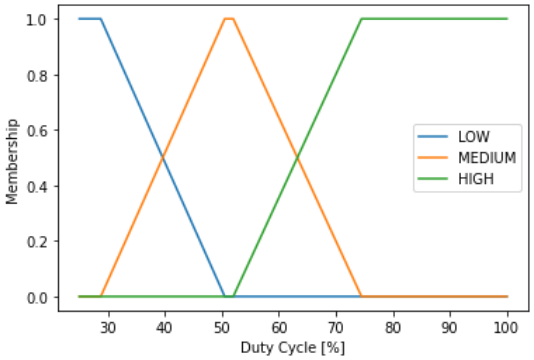
```
[ 25.    25.75  26.5   27.25  28.    28.75  29.5   30.25  31.    31.75
  32.5   33.25  34.    34.75  35.5   36.25  37.    37.75  38.5   39.25
  40.    40.75  41.5   42.25  43.    43.75  44.5   45.25  46.    46.75
  47.5   48.25  49.    49.75  50.5   51.25  52.    52.75  53.5   54.25
  55.    55.75  56.5   57.25  58.    58.75  59.5   60.25  61.    61.75
  62.5   63.25  64.    64.75  65.5   66.25  67.    67.75  68.5   69.25
  70.    70.75  71.5   72.25  73.    73.75  74.5   75.25  76.    76.75
  77.5   78.25  79.    79.75  80.5   81.25  82.    82.75  83.5   84.25
  85.    85.75  86.5   87.25  88.    88.75  89.5   90.25  91.    91.75
  92.5   93.25  94.    94.75  95.5   96.25  97.    97.75  98.5   99.25
 100.  ]
```

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]:

```
lowDC = fuzz.trapmf(uniDC, [25, 25, 28.75, 50.50])
mediumDC = fuzz.trapmf(uniDC, [28.75, 50.50, 52, 74.50])
highDC = fuzz.trapmf(uniDC, [52, 74.50, 100, 100])

plt.xlabel("Duty Cycle [%]")
plt.ylabel("Membership")
plt.plot(uniDC, lowDC, label = "LOW")
plt.plot(uniDC, mediumDC, label = "MEDIUM")
plt.plot(uniDC, highDC, label = "HIGH")
plt.legend()
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.

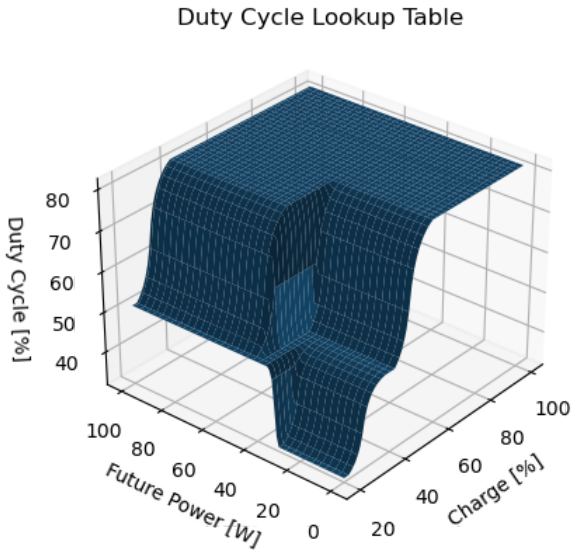
The operations that are used to realize the operators in the fuzzy ruleset can be found below. They are arranged in the order that they should be performed.

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 [23]:

```
# Initialize variables to store values necessary for the fuzzy controller

# define an array of zeros, what should be the size of the array?
# M is the number of elements in SOC universe
M = uniSOC.size
# N is the number of elements in FAP universe
N = uniP.size
# Construct MxN sized look-up table
fuzzyLookupTable = np.zeros((M, N))

# define the rule set as an array
rule_set = [
    (lowSOC, scarceP, lowDC),
    (lowSOC, averageP, mediumDC),
    (lowSOC, abundantP, mediumDC),
    (mediumSOC, scarceP, mediumDC),
    (mediumSOC, averageP, mediumDC),
    (mediumSOC, abundantP, highDC),
    (highSOC, scarceP, highDC),
    (highSOC, averageP, highDC),
    (highSOC, abundantP, highDC),

]

# create arrays for SOC, future average power and duty cycle
# create the structure where you are going to store the duty cycle results
# 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):
        dutycycle = np.zeros((len(rule_set), uniDC.size));
        for r in range(len(rule_set)):
            dutycycle[r] = rule_set[r][0][s] * rule_set[r][1][p] * rule_set[r][2]
        dcMax = dutycycle[0]
        for i in range(len(rule_set)):
            dcMax = np.maximum(dcMax, dutycycle[i])
        fuzzyLookupTable[s,p] = fuzz.defuzz(uniDC, dcMax, 'bisector')

# Plotting 3d plot of fuzzy lookup table
fig = plt.figure()
fig.set_dpi(100)

[gX, gY] = np.meshgrid(uniSOC, uniP, indexing='ij')
ax = fig.add_subplot(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('Duty Cycle Lookup Table')
ax.view_init(azim=220)
plt.tight_layout()
plt.show()
```



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.

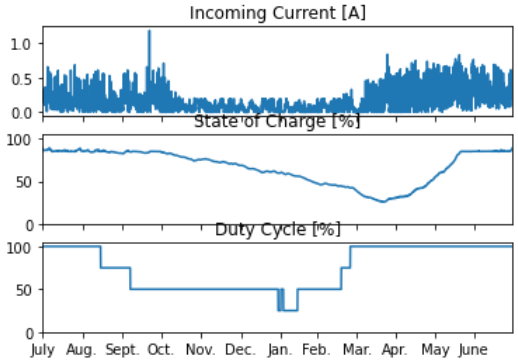
In [24]:
```
# Normalize the table, and set the duty cycle to take values of 25, 50, 75, or 100
fuzzyLookupTable -= np.amin(np.amin(fuzzyLookupTable))
fuzzyLookupTable /= np.amax(np.amax(fuzzyLookupTable))
fuzzyLookupTable = 75 * np.round(fuzzyLookupTable * 3) / 3 + 25

### WRITE CODE BELOW THIS LINE ###
station_states = [
# Note we are normalizing the SOC since the percentage ranges from 20-100
    (20-20, 100),
    (28-20, 23),
    (50-20, 21),
    (88-20, 91)
]

for i in range(len(station_states)):
    print("Station {} duty cycle = {}%".format(i+1, fuzzyLookupTable[station_states[i][0]][station_states[i][1]]))
```

```
Station 1 duty cycle = 50.0%
Station 2 duty cycle = 25.0%
Station 3 duty cycle = 75.0%
Station 4 duty cycle = 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 [25]:
```
resolute.setFuzzyLookupTable(fuzzyLookupTable)
resolute.simulate(1)  # Simulate using the fuzzy algorithm
```



Based on the generated results above, we can see that the employment of a fuzzy controller allows for better utilization of the station's resources. This is because in the previously generated graphs (without using the fuzzy algorithm) the duty cycle would be 0 for extended periods of time during January to April (Winter months), indicating that power outages are occuring. However, the duty cycle when using the fuzzy controller does not reach 0 during the January to April months. Moreover, when not using the fuzzy controller the duty cycle stays at 100% until around mid January where we notice it fluctuate to 0%, indicating improper utilization of power. On the other hand, when using the fuzzy controller, the duty cycle begins to decrease at around mid-August to ty to conserve power for the Winter months. Then, at the start of March the duty cycle goes back to 100%.

Abstract

The purpose of this lab was to build a fuzzy controller to assist a monitoring station in Resolute, Nunavut. During the winters the station is not recording data (duty cycle = 0) for significant periods of time. Through the employment of a fuzzy controller, the station can conserve power during these times and prevent power outages. The fuzzy controller is able to take inputs, proceed to process these inputs according to the programmed rule-set and then output a crispy value. To build this fuzzy controller, we must define the membership functions, the fuzzy rules that make up a rule-set and calculate a fuzzy look-up table based on this rule-set. Before utilizing a fuzzy controller, based on the duty cycle graphs, there were power outages (duty cycle was 0) starting from mid-January up to mid-March. From the last lab, we had already evaluated trapezoidal membership functions for the state of charge (SOC) and future average power (P). The last set of membership functions needed were the duty cycle membership functions (LOW, MEDIUM, HIGH) and they were evaluated and plotted in this lab. Based on these membership functions and the provided rule-set, we evaluated a fuzzy lookup table in the form of a matrix that provides a duty cycle value per state of charge and future average power pair. Once the fuzzy look-up table was obtained, a 3d plot of it was created. Afterwards, using the fuzzy look-up table and data obtained from 4 different stations, the duty cycle that each station should adopt was found. Based on the results, stations 1, 2, 3 and 4 should adopt duty cycles 50.0%, 25.0%, 75.0% and 100.0% respectively. Lastly, we simulated the monitoring station using the fuzzy controller and based on the graphical results, it was clear that the fuzzy controller stopped the duty cycle from going to 0% during the January-April period. The fuzzy controller was able to use the resources better as it started to decrease the duty cycle at around mid-August to conserve power in preparation for the Winter months.

Introduction

In this lab we built and utilized a fuzzy controller to help a monitoring station in Resolute, Nunavut to allocate resources in a better way. A fuzzy controller is able to take inputs from the environment, process the inputs based on a programmed rule-set/knowledge base (collection of IF-THEN rules) and then it will output a crisp value. In this case, the fuzzy controller was built to determine the optimal duty cycle (output variable) that the monitoring station should be running on based on the defineed rule-set (this one consisted of 9 rules) and the two input variables: the state of charge (SOC) and future average power (P). To build a fuzzy controller, the first step is to define all the membership functions (fuzzy sets) for our input and output variables, then define the fuzzy rules that make up the knowledge base/rule-set and lastly, evaluate a fuzzy lookup table based on the rule-set. After constructing all of the trapezoidal membership functions for all 3 variables we were able to construct the fuzzy lookup table using these membership functions and the defined rule-set. This fuzzy lookup table was then used to determine the duty cycle for 4 stations based on each station's input values for state of charge and future average power. A comparison was made for the station's duty cycle graphical results before and after the employment of a fuzzy controller.

Conclusion

In conclusion, this lab served as a great introduction to building a fuzzy controller and putting it into practical use. For this case, the fuzzy controller was designed to assist a monitoring station to make better use of it's resources so that power outages do not occur during the Winter months. The fuzzy controller was built using input and output membership functions (fuzzy sets) and the provided rule-set. This fuzzy controller took two values as inputs, the state of charge (SOC) and the future average power (P) and it returned a output value which was the duty cycle. The input and output variables had 3 trapezoidal membership functions each, each was representing a linguistic value. To generate the fuzzy lookup table (which the fuzzy controller's decision making is based on), we performed these operations in this order: Larsen implication for the AND and IF/THEN, then max operator for ELSE and lastly performed defuzzification to obtain the crisp value using the bisector method. After obtaining the fuzzy lookup table, we made a 3d plot of it. To put this fuzzy controller to the test, we took a set of input values (state of charge and future avearge power) from 4 stations and passed it into the fuzzy controller separately for each station. Based on these inputs from stations 1, 2, 3 and 4, the output duty cycle values were 50.0%, 25.0%, 75.0% and 100.0% respectively. For the last part of this lab, we simulated the monitoring station using this fuzzy controller and compared its graphical results the simulation that did not use a fuzzy controller. For the simulation without the fuzzy controller, the duty cycle was dropping to 0% during the January-April periods mainly because the duty cycle remained at 100% for long periods of time which shows lack of efficiency in terms of power consumption (and resource allocation). On the other hand, when simulating the monitoring station using the fuzzy controller, we saw that the duty cycle started to drop from 100% starting at around mid-August to conserve power for the winter months. During those winter months the duty cycle never dropped to 0% indicating that these power outages were prevented and as a result, the station can continue to collect data. The results from both of these simulations show that the fuzzy controller is functioning well.

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	