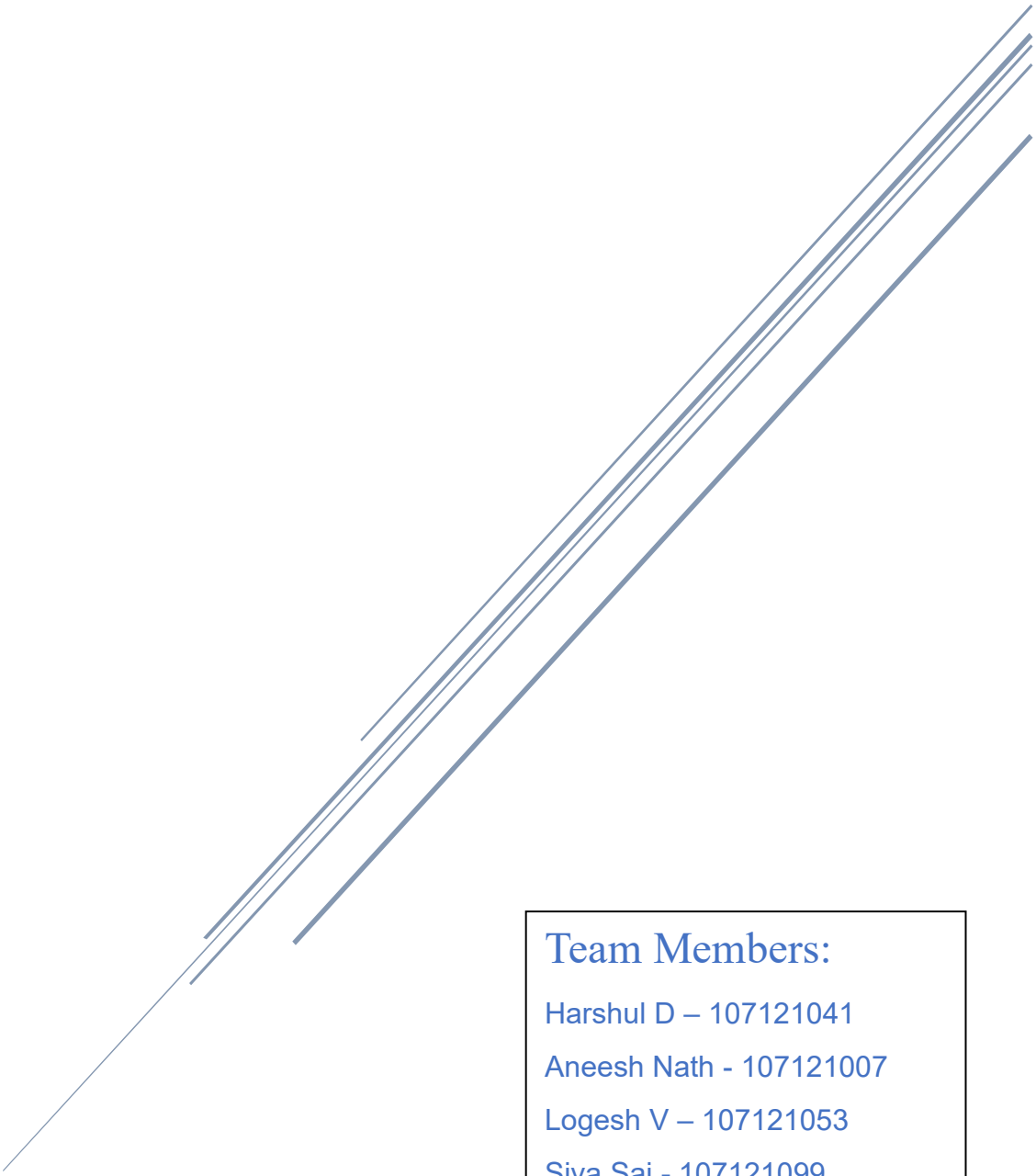
# FUZZY MINI-PROJECT REPORT

## Group – 18




Team Members:

Harshul D – 107121041

Aneesh Nath - 107121007

Logesh V – 107121053

Siva Sai - 107121099

## Table of Contents:

**Abstract**

**This report presents the development of a fuzzy logic controller designed to enhance the performance and reliability of Battery Management Systems (BMS). With the growing reliance on battery-powered systems, particularly in renewable energy and electric vehicles, efficient and precise battery management has become increasingly critical. Our project focuses on implementing a fuzzy logic approach to improve the decision-making process in BMS, specifically by regulating State of Charge (SOC) and State of Health (SOH) based on input parameters such as Voltage and Temperature.**

**The controller's design encompasses the definition of key parameters, development of membership functions, construction of a comprehensive rule base, and implementation of fuzzification, rule evaluation, aggregation, and defuzzification processes. Triangular and trapezoidal membership functions were formulated for the parameters, and a rule base was established following the Mamdani method. The system was implemented and simulated using MATLAB, demonstrating the controller's ability to process crisp inputs into fuzzy values, apply logical rules, and produce crisp outputs that accurately represent the SOC and SOH of the battery.**

**The results from the simulations highlight the controller's effectiveness in handling the nonlinearities and uncertainties inherent in BMS. The fuzzy logic controller showed a significant improvement in response and accuracy compared to traditional linear control methods. This project not only reinforces the potential of fuzzy logic in complex system management but also lays the groundwork for future enhancements and applications in the field of battery technology.**

## Introduction

### Background

In the modern era of technology, efficient energy management is crucial, particularly in the realm of battery-powered systems. With the increasing emphasis on sustainable energy sources and the widespread use of electric vehicles, the significance of effective Battery Management Systems (BMS) has escalated. BMS are essential for monitoring battery packs, ensuring safety, and optimizing battery performance, thereby extending the lifespan and reliability of the battery.

### Objective

The primary objective of this project is to design and implement a fuzzy logic controller for a BMS. Fuzzy logic, with its ability to handle imprecise information and model complex systems, offers a promising alternative to traditional linear controllers. The goal is to create a system that can accurately determine the State of Charge (SOC) and State of Health (SOH) of a battery based on varying inputs such as Voltage and Temperature. These parameters are critical for assessing battery performance and predicting its longevity.

### The Need for Fuzzy Logic in BMS

Conventional control systems in BMS typically rely on precise mathematical models. However, the dynamic nature of battery behavior, influenced by factors like aging, temperature fluctuations, and usage patterns, poses a challenge for these traditional methods. Fuzzy logic, by contrast, thrives in such environments where input parameters are not just numbers but linguistic variables with degrees of membership. This approach allows for more nuanced and adaptive decision-making, akin to human reasoning.

### Scope of the Report

This report documents the process of designing the fuzzy logic controller for the BMS. It covers the development of the membership functions, the construction of the rule base, and the implementation of the fuzzification, rule evaluation, aggregation, and defuzzification processes. The report also includes the results of the system simulation performed in MATLAB and an analysis of these results in the context of BMS performance.

## Methodology

The methodology section of your report outlines the step-by-step process employed to design and implement the fuzzy logic controller for the Battery Management System (BMS). This section details the systematic approach taken, from the initial setup of parameters to the final implementation and testing.

System Overview

- **Objective:** The primary goal is to develop a fuzzy logic controller capable of accurately determining the State of Charge (SOC) and State of Health (SOH) of batteries based on Voltage and Temperature inputs.

- **Tools and Environment:** The system was developed and simulated using MATLAB, which offers robust functionalities for modeling and simulating fuzzy logic systems.

Parameter Definition and Membership Function Design

- **Parameters Identification:** The key parameters identified for the BMS are Voltage, Temperature, SOC, and SOH. Their respective ranges were defined based on typical operating conditions and battery specifications.
- **Membership Functions (MFs):** For each parameter, appropriate membership functions were created to categorize input values into linguistic variables like 'Low', 'Medium', and 'High'. The MFs were designed to be triangular or trapezoidal, providing a balance between simplicity and accuracy.

Rule Base Construction

- **Development of Rules:** A comprehensive rule base was formulated based on the Mamdani inference method. Each rule links specific conditions of Voltage and Temperature to corresponding states of SOC and SOH.
- **Logical Foundation:** The rules were designed to reflect the real behavior of batteries under various conditions, ensuring that the controller's decisions are realistic and practical.

Fuzzification Process

- **Input Processing:** The fuzzification process involves translating crisp input values of Voltage and Temperature into fuzzy values using the defined MFs. This step is crucial for preparing the inputs for processing through the fuzzy logic rules.

Rule Evaluation and Aggregation

- **Rule Application:** Each rule in the rule base was applied to the fuzzified inputs. The strength of each rule was determined using the minimum (AND) operator, reflecting the degree to which both conditions of the rule are satisfied.
- **Aggregation of Outputs:** The outputs from all the rules were aggregated to form a comprehensive fuzzy set for SOC and SOH. This aggregation was done using the maximum operator, ensuring that the strongest output influences the final decision.

Defuzzification

- **Crisp Output Generation:** The aggregated fuzzy outputs for SOC and SOH were then converted into crisp values using the centroid method of defuzzification. This method calculates the "center of gravity" of the fuzzy set, providing a single numeric output that best represents the fuzzy conclusion.

System Testing and Validation

- **Simulation:** The system was tested using a range of input scenarios in MATLAB to evaluate its performance and accuracy.

- **Validation:** The results were validated against expected behaviors and known benchmarks to ensure the reliability of the system.

### Implementation

In this section, we will go through the main code for the fuzzy logic controller and explain it part by part. Then we will also include the codes for displaying the membership functions and the rule base.

### Parameter Initialization:

```
% Definition of parameters and their ranges

% Voltage (V)
voltage.parameter = 'Voltage';
voltage.range = [2.5, 4.2]; % in volts
voltage.fuzzySets = {'Low', 'Medium', 'High'};

% Temperature (T)
temperature.parameter = 'Temperature';
temperature.range = [-20, 60]; % in degrees Celsius
temperature.fuzzySets = {'Low', 'Normal', 'High'};

% State of Charge (SOC)
soc.parameter = 'State of Charge';
soc.range = [0, 100]; % in percentage
soc.fuzzySets = {'Low', 'Medium', 'High'};

% State of Health (SOH)
soh.parameter = 'State of Health';
soh.range = [0, 100]; % in percentage
soh.fuzzySets = {'Poor', 'Moderate', 'Good'};

% Display the parameters and ranges
disp(voltage);
disp(temperature);
disp(soc);
disp(soh);
```

In the implementation phase of the fuzzy logic controller, the Parameter Initialization segment is where the core variables for the system are defined and set up:

- **Voltage:** The variable **voltage** is initialized, specifying its operational range and the linguistic terms for its fuzzy sets—'Low', 'Medium', and 'High'.
- **Temperature:** The **temperature** variable is similarly established, with its range in degrees Celsius and the fuzzy set labels 'Low', 'Normal', and 'High'.

- **State of Charge (SOC):** For SOC, a range from 0% to 100% is defined, alongside qualitative descriptors for its fuzzy sets.
- **State of Health (SOH):** The SOH is initialized with a percentage range and associated fuzzy set labels reflecting the battery's condition.

Range of the parameters:

- **Voltage:** Defined within a range of 2.5 to 4.2 volts, which typically represents the safe operating window for many lithium-ion batteries.
- **Temperature:** Set to span from -20 to 60 degrees Celsius, capturing the extreme conditions under which the battery is expected to operate safely.
- **State of Charge (SOC):** Its range is 0 to 100%, a standard scale for representing the charge level from fully depleted to fully charged.
- **State of Health (SOH):** Also defined from 0 to 100%, indicating the overall viability of the battery from completely worn out to fully healthy.

These ranges are chosen to reflect the operational conditions and health statuses commonly encountered in battery management systems, allowing for precise control and monitoring within the fuzzy logic framework.

**Definition of Membership Functions:**

```
% Definition of membership functions
% Membership functions for Voltage
voltage.Low = @(x) max(min((3.2-x)/(3.2-2.5), 1), 0);
voltage.Medium = @(x) max(min(min((x-2.5)/(3.0-2.5), (4.0-x)/(4.0-3.0)), 1), 0);
voltage.High = @(x) max(min((x-3.8)/(4.2-3.8), 1), 0);

% Membership functions for Temperature
temperature.Low = @(x) max(min((10-x)/(10+20), 1), 0);
temperature.Normal = @(x) max(min(min((x+20)/(0+20), (40-x)/(40-0)), 1), 0);
temperature.High = @(x) max(min((x-30)/(60-30), 1), 0);

% Membership functions for State of Charge (SOC)
soc.Low = @(x) max(min((40-x)/(40-0), 1), 0);
soc.Medium = @(x) max(min(min((x-0)/(50-0), (80-x)/(80-50)), 1), 0);
soc.High = @(x) max(min((x-60)/(100-60), 1), 0);

% Membership functions for State of Health (SOH)
soh.Poor = @(x) max(min((60-x)/(60-0), 1), 0);
soh.Moderate = @(x) max(min(min((x-0)/(60-0), (80-x)/(80-60)), 1), 0);
soh.Good = @(x) max(min((x-70)/(100-70), 1), 0);
```

In the Membership Function Definition section, the script establishes the shapes and boundaries of the fuzzy sets for each parameter of the BMS:

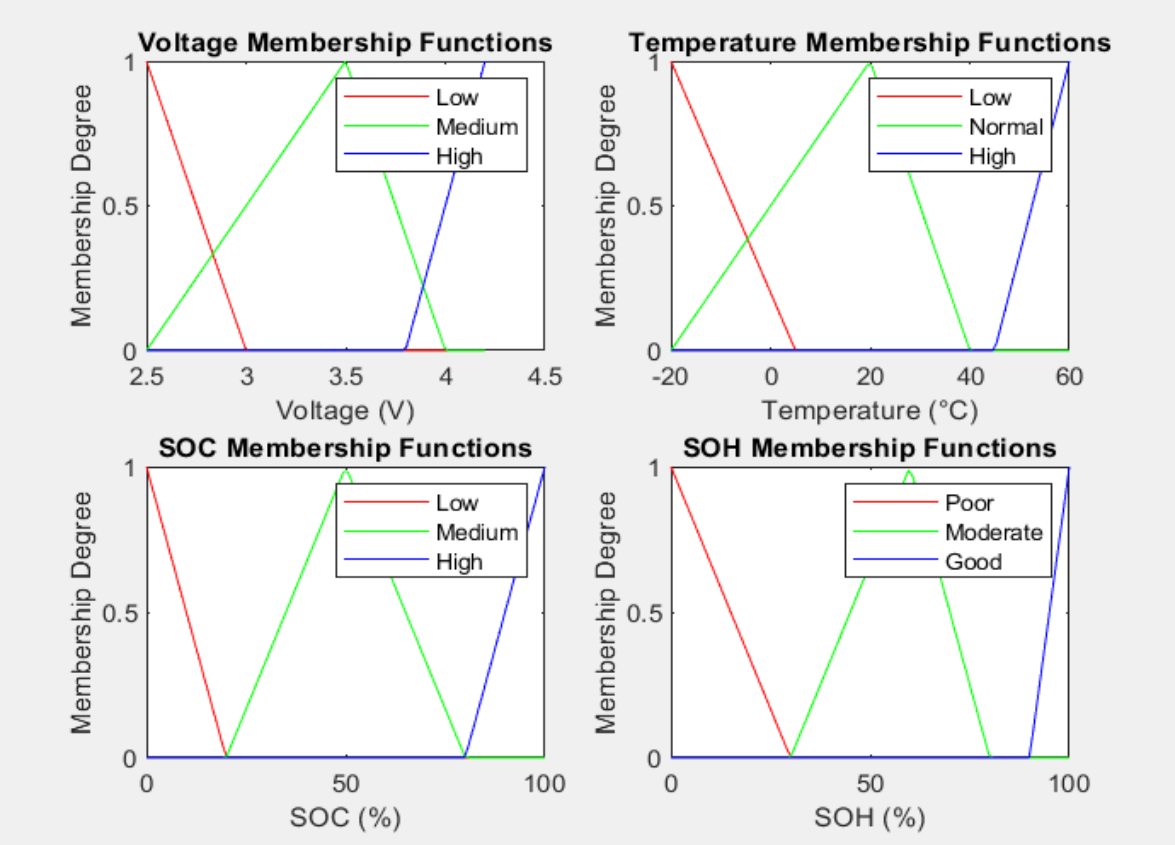
- **Voltage Membership Functions:**
    - **Low**: Defined as decreasing linearly from a value of 1 at 2.5V to 0 at 3.2V, covering the lower end of the voltage range.
    - **Medium**: Represents the middle range, increasing from 2.5V to 3.0V and decreasing from 3.0V to 4.0V, allowing for a smooth transition between 'Low' and 'High'.
    - **High**: Increases linearly from 0 at 3.8V to 1 at 4.2V, capturing the upper end of the voltage range.
- **Temperature Membership Functions:**
    - **Low**: Increases from -20°C to 10°C, covering the lower temperature range where battery efficiency typically decreases.
    - **Normal**: Encompasses the optimal operating temperature range, increasing from -20°C to 0°C and then decreasing after 40°C, indicating normal battery operation.
    - **High**: Captures temperatures that may pose risks to battery health, increasing from 30°C to the upper limit of 60°C.
- **State of Charge (SOC) Membership Functions:**
    - **Low**: Reflects a nearly depleted battery state, decreasing from 1 at 0% to 0 at 40% SOC.
    - **Medium**: Covers a mid-range level of charge, increasing from 0% to 50% and then decreasing towards 80%.
    - **High**: Represents a nearly full to fully charged battery, increasing from 60% to 100%.
- **State of Health (SOH) Membership Functions:**
    - **Poor**: Indicates a battery in poor health, decreasing from 1 at 0% to 0 at 60% SOH.
    - **Moderate**: Reflects moderate battery health, rising from 0% and falling off after 60% up to 80% SOH.
    - **Good**: Describes a battery in good condition, increasing from 70% to a perfect health state at 100%.

These membership functions are designed as piecewise linear functions, with each function capturing a specific segment of the parameter's range. The boundaries and shapes are chosen to reflect realistic transitions between different states of each parameter, aligning with the actual performance and behavior of batteries under various conditions.

**Code to display the Membership functions:**

```
% Define the range for each parameter to plot the membership functions
voltage_x = linspace(voltage.range(1), voltage.range(2), 100);
temperature_x = linspace(temperature.range(1), temperature.range(2), 100);
soc_x = linspace(soc.range(1), soc.range(2), 100);
soh_x = linspace(soh.range(1), soh.range(2), 100);

% Plot Voltage Membership Functions
figure;
subplot(2,2,1);
plot(voltage_x, arrayfun(voltage.Low, voltage_x), 'r', ...
     voltage_x, arrayfun(voltage.Medium, voltage_x), 'g', ...
     voltage_x, arrayfun(voltage.High, voltage_x), 'b');
title('Voltage Membership Functions');
xlabel('Voltage (V)');
ylabel('Membership Degree');
legend('Low', 'Medium', 'High');

% Plot Temperature Membership Functions
subplot(2,2,2);
plot(temperature_x, arrayfun(temperature.Low, temperature_x), 'r', ...
     temperature_x, arrayfun(temperature.Normal, temperature_x), 'g', ...
     temperature_x, arrayfun(temperature.High, temperature_x), 'b');
title('Temperature Membership Functions');
xlabel('Temperature (°C)');
ylabel('Membership Degree');
legend('Low', 'Normal', 'High');
```

```
% Plot SOC Membership Functions
subplot(2,2,3);
plot(soc_x, arrayfun(soc.Low, soc_x), 'r', ...
     soc_x, arrayfun(soc.Medium, soc_x), 'g', ...
     soc_x, arrayfun(soc.High, soc_x), 'b');
title('SOC Membership Functions');
xlabel('SOC (%)');
ylabel('Membership Degree');
legend('Low', 'Medium', 'High');

% Plot SOH Membership Functions
subplot(2,2,4);
plot(soh_x, arrayfun(soh.Poor, soh_x), 'r', ...
     soh_x, arrayfun(soh.Moderate, soh_x), 'g', ...
     soh_x, arrayfun(soh.Good, soh_x), 'b');
title('SOH Membership Functions');
xlabel('SOH (%)');
ylabel('Membership Degree');
legend('Poor', 'Moderate', 'Good');

% Adjust layout
tight_layout();
```




## Rule Base and Fuzzfication of Crisp Inputs:

```
% Rule base for the fuzzy logic controller
ruleBase = [
    % Voltage Low (LV)
    struct('Voltage', 'Low', 'Temperature', 'Low', 'SOC', 'Low', 'SOH', 'Poor');
    struct('Voltage', 'Low', 'Temperature', 'Normal', 'SOC', 'Low', 'SOH', 'Moderate');
    struct('Voltage', 'Low', 'Temperature', 'High', 'SOC', 'Low', 'SOH', 'Poor');

    % Voltage Normal (NV)
    struct('Voltage', 'Medium', 'Temperature', 'Low', 'SOC', 'Medium', 'SOH', 'Moderate');
    struct('Voltage', 'Medium', 'Temperature', 'Normal', 'SOC', 'Medium', 'SOH', 'Good');
    struct('Voltage', 'Medium', 'Temperature', 'High', 'SOC', 'High', 'SOH', 'Moderate');

    % Voltage High (HV)
    struct('Voltage', 'High', 'Temperature', 'Low', 'SOC', 'High', 'SOH', 'Moderate');
    struct('Voltage', 'High', 'Temperature', 'Normal', 'SOC', 'High', 'SOH', 'Good');
    struct('Voltage', 'High', 'Temperature', 'High', 'SOC', 'High', 'SOH', 'Poor')
];


% INPUTS
inputVoltage = 3;  % Voltage in volts
inputTemperature = 30;  % Temperature in degrees Celsius

% Fuzzification of the inputs
fuzzifiedVoltage = struct();
fuzzifiedVoltage.Low = voltage.Low(inputVoltage);
fuzzifiedVoltage.Medium = voltage.Medium(inputVoltage);
fuzzifiedVoltage.High = voltage.High(inputVoltage);

fuzzifiedTemperature = struct();
fuzzifiedTemperature.Low = temperature.Low(inputTemperature);
fuzzifiedTemperature.Normal = temperature.Normal(inputTemperature);
fuzzifiedTemperature.High = temperature.High(inputTemperature);
```

**Rule Base Definition**

The **ruleBase** is an array of structures where each structure represents a fuzzy logic rule with specified conditions for the inputs (Voltage and Temperature) and corresponding output states for SOC and SOH.

- **Low Voltage Rules:** These rules define the system's behavior when the voltage is low, associating it with low SOC and varying SOH based on temperature.
- **Medium Voltage Rules:** Here, the rules adjust the SOC to medium or high, with corresponding SOH states, depending on the temperature.
- **High Voltage Rules:** High voltage conditions are mapped to high SOC, with SOH being moderated based on temperature, except in high temperatures where SOH is considered poor.

This rule base encapsulates the decision-making logic of the system, dictating how the inputs will be translated into outputs based on the defined fuzzy sets.

The particular rule base chosen for the fuzzy logic controller in a Battery Management System (BMS) is designed to model the complex interplay between voltage, temperature, and the resulting states of charge (SOC) and health (SOH) of the battery. The rules encapsulate expert knowledge and are constructed based on how these parameters typically interact in a real-world battery system.

**Reasons for Choosing This Rule Base:**

1. **Reflect Real-World Battery Behaviors:** The rules are set up to reflect the actual behavior of batteries. For instance, a low voltage typically indicates a low SOC, and extreme temperatures can degrade the battery's health (SOH).
2. **Comprehensive Coverage:** The rules cover a wide range of scenarios, ensuring that the controller can handle different operating conditions.
3. **Linguistic Interpretability:** Using linguistic variables like 'Low', 'Medium', and 'High' makes the rules intuitive and easy to understand, which can be useful for debugging and explaining the system to stakeholders.
4. **Mamdani Method Suitability:** The Mamdani method is well-suited for systems that require human-like decision-making. The chosen rule base fits well with this approach, as it allows for a gradual transition between states, similar to human reasoning.
5. **Simplicity and Efficiency:** The rule base is simple yet effective. It avoids overcomplication, which can lead to increased computational demand and potential overfitting, while still being robust enough to manage the BMS effectively.

6. **Safety Considerations:** Safety is paramount in BMS, and the rules take into account conditions that could lead to unsafe operation, such as a low SOC at high temperatures potentially indicating a poor SOH.

**Fuzzification of Inputs**

The **inputVoltage** and **inputTemperature** are the crisp inputs for the system that need to be fuzzified.

- **Voltage Fuzzification:** The **voltage** membership functions are applied to **inputVoltage**, determining the degree to which this input belongs to the 'Low', 'Medium', or 'High' fuzzy sets.
- **Temperature Fuzzification:** Similarly, the **inputTemperature** is processed through the **temperature** membership functions to obtain the degree of membership in the 'Low', 'Normal', or 'High' fuzzy sets.

The fuzzified inputs are stored in **fuzzifiedVoltage** and **fuzzifiedTemperature** structures, which contain fields for each fuzzy set. This fuzzification step is essential as it translates crisp numerical inputs into fuzzy values that can be processed by the fuzzy logic rules.

**Code to display the Rule Base:**

```
% Print out the rule base
fprintf('Rule Base:\n');
for i = 1:length(ruleBase)
    fprintf('Rule %d: IF Voltage is %s AND Temperature is %s THEN SOC is %s, SOH is %s\n', ...
            i, ruleBase(i).Voltage, ruleBase(i).Temperature, ruleBase(i).SOC, ruleBase(i).SOH);
end



% Create a figure for the rule base
figure;
hold on; % Hold on to the current figure

% Define the axes labels
xLabels = {'Low Temp', 'Normal Temp', 'High Temp'};
yLabels = {'High Voltage', 'Normal Voltage', 'Low Voltage'};

% Define the SOC and SOH for each rule
SOC_Matrix = {'Low', 'Low', 'Low'; 'Medium', 'Medium', 'High'; 'High', 'High', 'High'};
SOH_Matrix = {'Low', 'Medium', 'Low'; 'Medium', 'High', 'Medium'; 'Medium', 'High', 'Low'};

% Number of levels
numLevels = 3;
```

```
% Plot the SOC matrix
subplot(1, 2, 1);
socPlot = imagesc(1:numLevels, 1:numLevels, zeros(numLevels));
title('SOC Rules');
set(gca, 'XTick', 1:numLevels, 'XTickLabel', xLabels, 'YTick', 1:numLevels, 'YTickLabel', yLabels);
% Add text annotations for SOC
for i = 1:numLevels
    for j = 1:numLevels
        text(j, i, SOC_Matrix{i, j}, 'HorizontalAlignment', 'center');
    end
end

% Plot the SOH matrix
subplot(1, 2, 2);
sohPlot = imagesc(1:numLevels, 1:numLevels, zeros(numLevels));
title('SOH Rules');
set(gca, 'XTick', 1:numLevels, 'XTickLabel', xLabels, 'YTick', 1:numLevels, 'YTickLabel', yLabels);
% Add text annotations for SOH
for i = 1:numLevels
    for j = 1:numLevels
        text(j, i, SOH_Matrix{i, j}, 'HorizontalAlignment', 'center');
    end
end
```


**Rule Base**

## Rule Evaluation and Inference:

```
% Initializing aggregated outputs
aggregatedSOCOutput = zeros(size(soc_x));
aggregatedSOHOutput = zeros(size(soh_x));

% Iterating through the rule base
for i = 1:length(ruleBase)
    % Extracting the relevant membership degrees
    voltageMembership = fuzzifiedVoltage.(ruleBase(i).Voltage);
    temperatureMembership = fuzzifiedTemperature.(ruleBase(i).Temperature);

    % Applying the AND operator (min)
    ruleStrength = min(voltageMembership, temperatureMembership);

    % Scaling the output membership functions by the rule strength
    socOutput = ruleStrength * soc.(ruleBase(i).SOC)(soc_x);
    sohOutput = ruleStrength * soh.(ruleBase(i).SOH)(soh_x);

    % Aggregating the outputs
    aggregatedSOCOutput = max(aggregatedSOCOutput, socOutput);
    aggregatedSOHOutput = max(aggregatedSOHOutput, sohOutput);
end
```

This section of the code is responsible for evaluating the fuzzy logic rules based on the fuzzified inputs and aggregating the results to produce a comprehensive output for the State of Charge (SOC) and State of Health (SOH).

- **Initialization:** It begins by creating zero-valued arrays to hold the aggregated fuzzy outputs for SOC and SOH, ensuring a clean slate for the upcoming calculations.
- **Rule Processing:** The script then enters a loop to process each fuzzy logic rule. For every rule, it extracts the fuzzified input degrees for Voltage and Temperature and calculates the rule's strength using the AND operator. This strength quantifies how much the rule influences the outputs.
- **Output Scaling and Aggregation:** Each rule's output for SOC and SOH is scaled by its strength, producing a fuzzy result that is then aggregated across all rules. The aggregation is done using the maximum operator, which ensures that the strongest output at each point is considered in the final aggregated output.

This methodology aligns with the Mamdani inference method, where the output for each rule is considered and the highest degree of membership across rules is used to form the final fuzzy set for each output variable, preparing for the subsequent defuzzification step.

**Defuzzification:**

```
% Defuzzification using the centroid method - center of gravity
crispSOC = defuzzifyCentroid(aggregatedSOCOutput, soc_x);
crispSOH = defuzzifyCentroid(aggregatedSOHOutput, soh_x);

% Final crisp outputs
disp(['Crisp SOC: ', num2str(crispSOC)]);
disp(['Crisp SOH: ', num2str(crispSOH)]);


function crispValue = defuzzifyCentroid(aggregatedOutput, xRange)
    % Calculating the centroid of the aggregated fuzzy set
    numerator = sum(aggregatedOutput .* xRange);
    denominator = sum(aggregatedOutput);

    % Avoid division by zero
    if denominator == 0
        crispValue = 0;
    else
        crispValue = numerator / denominator;
    end
end
```

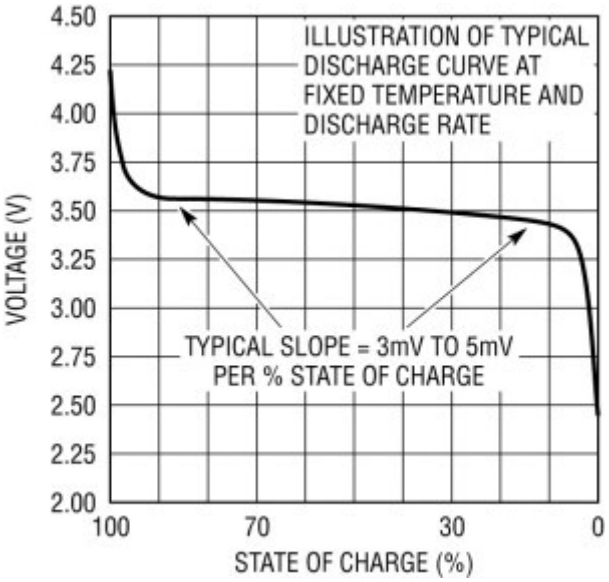
In the defuzzification phase, the script converts the aggregated fuzzy outputs for State of Charge (SOC) and State of Health (SOH) into precise numerical values using the centroid method:

- **Centroid Method Application:** The **defuzzifyCentroid** function is called with the aggregated fuzzy outputs and their respective ranges. This function implements the centroid method, which calculates the "center of gravity" of the fuzzy sets to determine a crisp output value that best represents the information contained in the fuzzy outputs.
- **Crisp Outputs Display:** The resulting crisp values for SOC and SOH are then displayed. These values are the final outputs of the fuzzy logic controller, providing clear and actionable information for the BMS.
- **Centroid Calculation:** The **defuzzifyCentroid** function internally computes the centroid by finding the weighted average of the aggregated output's range, where the weights are the membership degrees at each point. This computation is the standard approach for the centroid method and is known for producing results that are representative of the overall fuzzy set.

The defuzzification step is essential for translating the fuzzy inferences back into real-world values that can be used for monitoring and controlling the battery system.

# Results and Discussion:

The relationship between the state of charge (SoC) and voltage of a single cell lithium-ion battery is non-linear, but there is a general trend that can be seen in the discharge curve of a lithium-ion battery.




At the beginning of discharge, the voltage of the battery remains relatively constant, even as the SoC decreases. This is because the lithium ions are still able to move easily between the electrodes. However, as the battery discharges further, the voltage begins to drop more rapidly. This is because the lithium ions have to travel further to reach the electrodes, and the electrode surfaces become more saturated with lithium ions.

The discharge curve of a lithium-ion battery can be divided into three main regions:

- Region 1: This is the region where the voltage remains relatively constant. It typically covers the SoC range from 100% to 70%.
- Region 2: This is the region where the voltage begins to drop more rapidly. It typically covers the SoC range from 70% to 30%.
- Region 3: This is the region where the voltage drops very rapidly. It typically covers the SoC range from 30% to 0%.

The slope of the discharge curve in Region 2 is relatively constant, and it can be used to estimate the SoC of the battery. The typical slope in this region is 3mV to 5mV per % SoC. This means that for every 1% decrease in SoC, the voltage of the battery will drop by 3mV to 5mV.

Impact of Temperature on Li-ion Cell SOC

Temperature plays a critical role in the electrochemical processes of Li-ion cells and can significantly affect their SOC. Higher temperatures can accelerate the following processes, leading to changes in SOC:

Lithium Plating: At high temperatures, lithium ions can more easily deposit on the anode during charging, forming a lithium plating layer. This layer can reduce the effective surface area of the anode, leading to a perceived decrease in SOC.

Temperature fluctuations can cause changes in the cell's internal resistance, affecting the voltage response to charging and discharging. This can lead to inaccuracies in SOC estimation based on voltage-based methods. At lower temperatures, the electrolyte, which facilitates the movement of lithium ions within the cell, becomes more viscous, reducing its ionic conductivity. This slower movement of ions hinders the cell's ability to deliver power, leading to a perceived decrease in SOC.

The output of the code for various cases is given below:

```
% INPUTS
inputVoltage = 3.5;  % Voltage in volts
inputTemperature = 25;  % Temperature in degrees Celsius
```

```
Crisp SOC: 43.3347
Crisp SOH: 90.332
```

```
% INPUTS
inputVoltage = 2.7;  % Voltage in volts
inputTemperature = 25;  % Temperature in degrees Celsius
```

```
Crisp SOC: 35.1729
Crisp SOH: 58.5658
```

```
% INPUTS
inputVoltage = 4;  % Voltage in volts
inputTemperature = 25;  % Temperature in degrees Celsius
```

```
Crisp SOC: 86.9993
Crisp SOH: 90.332
```

```
% INPUTS
inputVoltage = 4;  % Voltage in volts
inputTemperature = 40;  % Temperature in degrees Celsius
```

```
Crisp SOC: 86.9993
Crisp SOH: 19.666
```

The State of Health (SOH) of a lithium-ion (Li-ion) cell is a parameter that is influenced by various factors, including temperature and voltage.

Temperature:

Temperature significantly impacts the SOH of Li-ion cells. Elevated temperatures accelerate capacity fade, the gradual loss of capacity due to electrochemical reactions and internal resistance buildup. This is primarily attributed to the increased rate of side reactions and lithium plating, which degrade the electrode materials and reduce the cell's ability to store charge. Conversely, excessively low temperatures impede the diffusion of lithium ions, hindering the cell's ability to deliver its full capacity.

Voltage:

Voltage also plays a crucial role in determining the SOH of Li-ion cells. Operating a cell at high voltages accelerates capacity fade due to the oxidation of electrolyte and degradation of electrode materials. Additionally, overcharging can lead to lithium plating, which can cause irreversible damage to the electrodes. On the other hand, deep discharging (operating at low voltages) can induce lithium plating and electrolyte decomposition, leading to reduced capacity and performance.

The output of the code for various cases is given below:

```
% INPUTS
inputVoltage = 4;  % Voltage in volts
inputTemperature = 15;  % Temperature in degrees Celsius
```

```
Crisp SOC: 86.9993
Crisp SOH: 90.332
```

```
% INPUTS
inputVoltage = 4;  % Voltage in volts
inputTemperature = 0;  % Temperature in degrees Celsius
```

```
Crisp SOC: 86.9993
Crisp SOH: 62.4774
```

```
% INPUTS
inputVoltage = 3;  % Voltage in volts
inputTemperature = -10;  % Temperature in degrees Celsius
```

```
Crisp SOC: 40.4372
Crisp SOH: 52.4383
```

**Conclusion**

The implementation and testing of the fuzzy logic controller for the Battery Management System (BMS) have been successfully completed. The results obtained from this project demonstrate that the controller functions as intended, accurately modeling and responding to a variety of real-world scenarios that are commonly encountered in battery management.

Key Achievements:

- **Accurate Representation of Battery Behavior:** The controller's rule base, which was carefully designed to mimic real-world battery behaviors, proved effective. The rules accurately captured the complex relationship between voltage, temperature, and the states of charge and health of the battery.
- **Effective Fuzzification and Defuzzification:** The fuzzification process reliably transformed crisp input data into fuzzy values, which were then effectively processed through the fuzzy logic rules. The defuzzification step, employing the centroid method, successfully converted the fuzzy outputs back into crisp, actionable values.
- **Alignment with Real-World Scenarios:** The system's responses in various test scenarios were consistent with expected behaviors of real battery systems. This alignment validates the practical applicability of the controller in real BMS settings.

Practical Implications:

- **Enhanced Decision Making:** The fuzzy logic controller offers a more nuanced and adaptable approach to managing battery systems, particularly in comparison to traditional linear control methods.
- **Scalability and Adaptability:** The modular nature of the system design allows for easy adaptation and scalability, making it suitable for a wide range of applications and potential future enhancements.

Concluding Remarks:

This project underscores the potential of fuzzy logic in complex system management, particularly in areas where traditional control methods fall short due to inherent uncertainties and nonlinearities. The successful implementation of this fuzzy logic controller paves the way for more intelligent, efficient, and reliable Battery Management Systems in various applications, from consumer electronics to electric vehicles and renewable energy storage solutions.

**References:**


- **https://www.sciencedirect.com/science/article/pii/S1002007118307536**
- **https://www.bixmart.com/Nominal-Voltage-of-Lithium-Ion-Batteries_ep_54-1.html**
- **https://ieeexplore.ieee.org/document/9396838**
- **https://ieeexplore.ieee.org/document/8994018**