Modern Fuzzy Systems: Neural Inference, Transformative Approaches, and Advanced Applications

Navid Mohaghegh

Modern Fuzzy Systems: Neural Inference, Transformative Approaches, and Advanced Applications

Navid Mohaghegh

Contents

Preface

Fuzzy logic has revolutionized the way we approach uncertainty, imprecision, and ambiguity in complex systems. From its inception as a mathematical framework for reasoning about vagueness, fuzzy systems have grown into a versatile tool used across engineering, artificial intelligence, decision-making, and beyond. As we advance into an era dominated by intelligent systems and transformative technologies, the integration of fuzzy logic with neural networks and deep learning paradigms has unlocked a new realm of possibilities.

This book, **"Modern Fuzzy Systems: Neural Inference, Transformative Approaches, and Advanced Applications"**, serves as a comprehensive guide to understanding and implementing fuzzy systems in modern contexts. Whether you are a student, researcher, or practitioner, this book provides an in-depth exploration of the theoretical foundations of fuzzy logic, practical techniques, and cutting-edge developments in hybrid systems that combine fuzzy logic with neural networks and transformers.

The content is structured to cater to readers with varying levels of expertise. We begin with an introduction to fuzzy logic and fuzzy set theory, providing essential concepts and principles that lay the groundwork for more advanced topics. Subsequent chapters delve into the design and application of fuzzy systems, including Mamdani, Sugeno, and Takagi-Sugeno inference systems, offering practical insights and real-world examples.

For those seeking to push the boundaries of fuzzy logic, this book explores the integration of neural networks and fuzzy systems, such as Adaptive Neuro-Fuzzy Inference Systems (ANFIS), and their applications in robotics, control systems, and data-driven decision-making. The interplay between fuzzy logic and emerging technologies like transformers is also discussed, showcasing the transformative potential of these hybrid approaches.

One of the most exciting frontiers is the application of fuzzy logic and ANFIS in artificial intelligence, particularly in domains like autonomous vehicles and self-driving cars. By combining the interpretability of fuzzy systems with the computational power of transformers and neural networks, ANFIS can enhance decision-making processes under uncertainty, such as obstacle avoidance, route optimization, and adaptive behavior in dynamic environments. These hybrid approaches enable more reliable, explainable AI systems that can tackle the complex, real-world challenges faced by autonomous vehicles, accelerating their safe and scalable deployment.

Throughout the book, emphasis is placed on clarity, practical implementation, and bridging the gap between theory and application. Code examples, case studies,

and step-by-step guides ensure that readers not only understand the concepts but also gain the confidence to apply them in their own domains.

It is my hope that this book serves as a valuable resource for anyone seeking to harness the power of fuzzy systems in solving complex problems in today's rapidly evolving technological landscape.

I extend my gratitude to the pioneers of fuzzy logic and the researchers who continue to expand its horizons. May this book inspire readers to contribute to this vibrant and impactful field.

Navid Mohaghegh
December 2024

Chapter 1

Introduction to Fuzzy Logic

Fuzzy Logic is a form of multi-valued logic derived from **fuzzy set theory**, introduced by Lotfi Zadeh in 1965. Unlike classical binary logic (true or false, 0 or 1), fuzzy logic allows variables to have a degree of truth ranging from 0 to 1, representing the uncertainty inherent in real-world phenomena. For example:

- In classical boolean logic: A temperature of 60°C is either considered "hot" (1) or "not hot" (0).
- In fuzzy logic: A temperature of 60°C might be 0.6 "hot" and 0.4 "cold".

Fuzzy logic is particularly useful in control systems due to its ability to handle imprecise, noisy, or incomplete data. This makes it ideal for systems where precise mathematical modeling is challenging, such as robotics and digital control systems. The key benefits of fuzzy logic in control systems are:

1. **Robustness**: Tolerates imprecision in input and provides smooth control outputs.
2. **Flexibility**: Easy to add or modify **rules** without changing the entire system.
3. **Human-Like Reasoning**: Uses linguistic variables (e.g., "hot," "fast") to mimic expert decision-making in complex, uncertain systems

Fuzzy logic has become a essential in robotics and digital control systems, offering robust, adaptive, and intelligent solutions to tackle complex and dynamic problems. In robotics, it plays an important role in navigation and obstacle avoidance through fuzzy inference systems, empowering robots to make real-time decisions in uncertain and ever-changing environments. Moreover, robotic arms employ fuzzy logic for adaptive

decision-making, allowing them to perform precise operations even in unpredictable conditions, such as those encountered in industrial automation or surgical robotics.

In industrial systems, fuzzy logic is applied to optimize and automate a wide array of processes. For example, washing machines leverage fuzzy control to regulate parameters such as water levels, temperature, cycle times, and load size, ensuring optimal performance tailored to each load's specific requirements. Elevators use fuzzy logic to achieve seamless acceleration and deceleration, dynamically adjusting based on passenger load to enhance ride comfort and efficiency.

Small consumer electronics also benefit significantly from fuzzy logic's capabilities. Cameras utilize it for accurate and responsive focus adjustments, improving image clarity under varying conditions. Similarly, air conditioners, heating systems, and other climate control devices incorporate fuzzy control to regulate temperature intelligently, providing an optimal balance of comfort, energy efficiency, and user satisfaction.

Chapter 2

Fuzzy Set Theory

2.1 Fuzzy Sets vs. Classical Sets

Classical sets, also known as crisp sets, have precise boundaries, and an element either belongs to the set or does not. In contrast, fuzzy sets allow partial membership, where an element's inclusion is defined by a degree of membership ranging from 0 to 1.

Examplex:

- Classical Set: $A = \{x | x > 5\}$. An element $x = 6$ is in A, while $x = 5$ is not.

- Fuzzy Set: $A = \{(x, \mu_A(x)) | x \in X\}$, where $\mu_A(x)$ is the membership function defining the degree of membership. For $x = 5.5$, $\mu_A(x) = 0.8$, indicating 80% membership. A membership function can defines how each element in the universe of discourse is mapped to a membership value between 0 and 1. It embodies the fuzziness of a set.

2.2 Types of Fuzzy Membership Functions

Fuzzy membership functions define how each input in the universe of discourse is mapped to a membership value (degree of truth) between 0 and 1. Below are several types of commonly used membership functions, their mathematical definitions, and when they are typically applied:

1. **Triangular (trimf):**

$$\mu_A(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ \frac{c-x}{c-b} & b \leq x \leq c \\ 0 & x \geq c \end{cases}$$

This is a simple and widely used membership function defined by three parameters (a, b, c) forming a triangle. It is computationally efficient and is often used in real-time applications where simplicity is critical.

2. **Trapezoidal (trapmf):**

$$\mu_A(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ 1 & b \leq x \leq c \\ \frac{d-x}{d-c} & c \leq x \leq d \\ 0 & x \geq d \end{cases}$$

This membership function is similar to the triangular one but includes a flat top. It is defined by four parameters (a, b, c, d) and is used in cases where a range of values has full membership.

3. **Gaussian (gaussmf):**

$$\mu_A(x) = e^{-\frac{(x-m)^2}{2\sigma^2}}$$

A smooth, bell-shaped curve characterized by a center m and a standard deviation σ. It is ideal for applications requiring smooth transitions, such as modeling uncertainty or noise.

4. **Generalized Bell (gbellmf):**

$$\mu_A(x) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}$$

This function provides flexibility in shaping the curve through three parameters: a (width), b (slope), and c (center). It is commonly used in systems where varying degrees of smoothness are required.

5. **Sigmoidal (sigmf):**

$$\mu_A(x) = \frac{1}{1 + e^{-a(x-c)}}$$

This S-shaped curve is defined by parameters a (steepness) and c (center). It is often used for threshold-based applications where a gradual transition is needed.

6. **Z-Shaped (zmf):**

$$\mu_A(x) = \begin{cases} 1 & x \leq a \\ 1 - 2\left(\frac{x-a}{b-a}\right)^2 & a \leq x \leq \frac{a+b}{2} \\ 2\left(\frac{b-x}{b-a}\right)^2 & \frac{a+b}{2} \leq x \leq b \\ 0 & x \geq b \end{cases}$$

The Z-shaped membership function decreases gradually from 1 to 0 and is used to model decreasing phenomena or thresholds.

7. **Pi-Shaped (pimf):**

$$\mu_A(x) = \begin{cases} 0 & x \leq a \\ 2\left(\frac{x-a}{b-a}\right)^2 & a \leq x \leq \frac{a+b}{2} \\ 1 - 2\left(\frac{b-x}{b-a}\right)^2 & \frac{a+b}{2} \leq x \leq b \\ 1 & b \leq x \leq c \\ 1 - 2\left(\frac{x-c}{d-c}\right)^2 & c \leq x \leq \frac{c+d}{2} \\ 2\left(\frac{d-x}{d-c}\right)^2 & \frac{c+d}{2} \leq x \leq d \\ 0 & x \geq d \end{cases}$$

This smooth curve resembles a flattened bell shape and is useful for applications requiring smooth transitions between two ranges.

8. **S-Shaped (smf):**

$$\mu_A(x) = \begin{cases} 0 & x \leq a \\ 2\left(\frac{x-a}{b-a}\right)^2 & a \leq x \leq \frac{a+b}{2} \\ 1 - 2\left(\frac{b-x}{b-a}\right)^2 & \frac{a+b}{2} \leq x \leq b \\ 1 & x \geq b \end{cases}$$

The S-shaped membership function increases gradually from 0 to 1. It is commonly used for systems with smooth, increasing transitions.

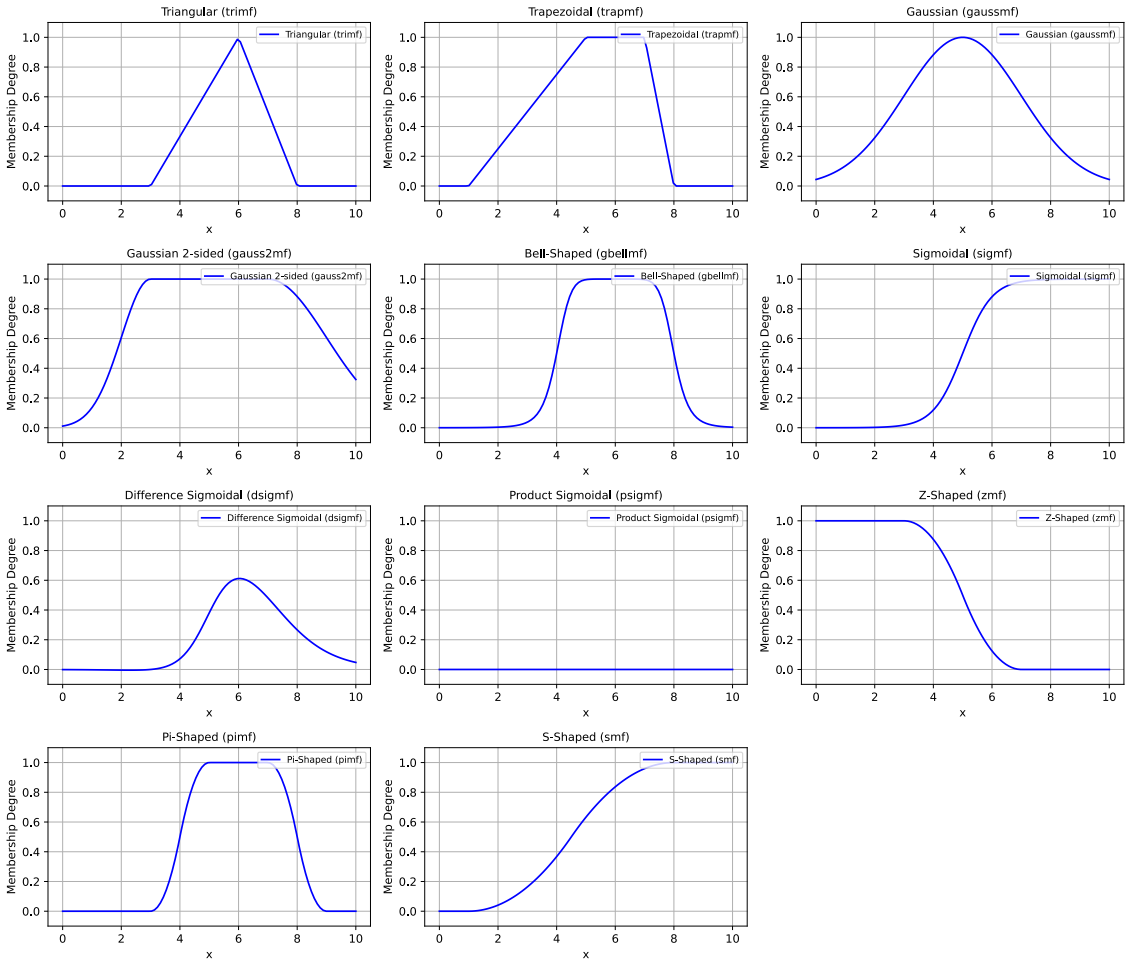

Figure 2.1: Visualization of Fuzzy Membership Functions: Triangular, Trapezoidal, Gaussian, Bell-Shaped, Sigmoidal, Z-Shaped, Pi-Shaped, and S-Shaped functions. Each function demonstrates how input values map to membership degrees.

For illustration purpose, the Figure 2.1 shows various fuzzy membership functions that we discussed. Of course you can make your own function as long as you consider the properties of fuzzy membership functions (will be discussed shortly).

2.3 Basic Terms in Fuzzy Sets

2.3.1 Membership Function

The membership function, denoted as $\mu_A(x)$ for a fuzzy set A, maps each element x in the universe of discourse X to a value in $[0, 1]$, representing the degree of membership. For example:

$$\mu_A(x) = \begin{cases} 1, & \text{if } x = 5, \\ 0.5, & \text{if } x = 4, \\ 0, & \text{otherwise.} \end{cases}$$

This function quantifies the "fuzziness" of an element belonging to a fuzzy set.

2.3.2 Support

The support of a fuzzy set A is the set of elements in X where $\mu_A(x) > 0$. Mathematically:

$$\text{Support}(A) = \{x \in X \mid \mu_A(x) > 0\}.$$

For $\mu_A(x) = \max(0, 1 - |x - 5|)$, the support is $\{x \mid |x - 5| < 1\}$.

2.3.3 Core

The core of a fuzzy set A is the set of elements in X where $\mu_A(x) = 1$. Formally:

$$\text{Core}(A) = \{x \in X \mid \mu_A(x) = 1\}.$$

For the example $\mu_A(x)$ above, the core is $\{5\}$.

2.3.4 Height

The height of a fuzzy set A is the maximum membership value:

$$\text{Height}(A) = \max_{x \in X} \mu_A(x).$$

For $\mu_A(x) = \max(0, 1 - |x - 5|)$, the height is 1.

2.3.5 Normality

A fuzzy set is normal if its height is 1. For instance:

$$\mu_B(x) = \begin{cases} 1, & \text{if } x = 3, \\ 0.5, & \text{if } x = 4, \\ 0, & \text{otherwise.} \end{cases}$$

Set B is normal because $\text{Height}(B) = 1$.

2.3.6 Convexity

A fuzzy set A is convex if:

$$\mu_A(\lambda x_1 + (1-\lambda)x_2) \geq \min(\mu_A(x_1), \mu_A(x_2)) \quad \forall x_1, x_2 \in X,\ \lambda \in [0,1].$$

For example, $\mu_A(x) = \max(0, 1 - |x - 5|)$ is convex, as the membership function forms a "peak."

2.3.7 α-cuts

An α-cut of a fuzzy set A, denoted A_α, is the crisp set of elements with membership degree at least α:

$$A_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\}.$$

For $\mu_A(x) = \max(0, 1 - |x - 5|)$ and $\alpha = 0.5$, $A_{0.5} = [4.5, 5.5]$.

2.3.8 Boundary

The boundary of a fuzzy set A consists of elements where $0 < \mu_A(x) < 1$. For instance:

$$\text{Boundary}(A) = \{x \in X \mid 0 < \mu_A(x) < 1\}.$$

For $\mu_A(x) = \max(0, 1 - |x - 5|)$, the boundary is $\{4, 6\}$.

2.3.9 Empty Fuzzy Set

A fuzzy set A is empty if $\mu_A(x) = 0\ \forall x \in X$. For instance:

$$\mu_A(x) = 0 \quad \forall x \in X$$

represents an empty fuzzy set.

2.3.10 Equality

Two fuzzy sets A and B are equal if:

$$\mu_A(x) = \mu_B(x) \quad \forall x \in X.$$

For example, if $\mu_A(x) = \max(0, 1 - |x - 5|)$ and $\mu_B(x) = \max(0, 1 - |x - 5|)$, then $A = B$.

2.3.11 Subset

A fuzzy set A is a subset of another fuzzy set B, denoted $A \subseteq B$, if:

$$\mu_A(x) \leq \mu_B(x) \quad \forall x \in X.$$

For example, if $\mu_A(x) = \max(0, 1 - |x - 3|)$ and $\mu_B(x) = 1$ for all x, then $A \subseteq B$.

2.3.12 Union (Max Operator)

The union of two fuzzy sets A and B is defined using the max operator:

$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)).$$

For example, if $\mu_A(x) = 0.5$ and $\mu_B(x) = 0.7$, then $\mu_{A \cup B}(x) = 0.7$.

2.3.13 Intersection (Min Operator)

The intersection of two fuzzy sets A and B is defined using the min operator:

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)).$$

For example, if $\mu_A(x) = 0.5$ and $\mu_B(x) = 0.7$, then $\mu_{A \cap B}(x) = 0.5$.

2.3.14 Complement

The complement of a fuzzy set A is defined as:

$$\mu_{\neg A}(x) = 1 - \mu_A(x).$$

For example, if $\mu_A(x) = 0.3$, then $\mu_{\neg A}(x) = 0.7$.

2.3.15 De Morgan's Laws

De Morgan's laws for fuzzy sets A and B are:

$$\mu_{\neg(A\cup B)}(x) = \min(1 - \mu_A(x), 1 - \mu_B(x)),$$

$$\mu_{\neg(A\cap B)}(x) = \max(1 - \mu_A(x), 1 - \mu_B(x)).$$

These properties generalize classical De Morgan's laws to fuzzy logic.

2.3.16 t-norms and t-conorms

A t-norm is a generalization of intersection, satisfying commutativity, associativity, monotonicity, and having 1 as the identity. A common t-norm is the min operator:

$$T(\mu_A(x), \mu_B(x)) = \min(\mu_A(x), \mu_B(x)).$$

A t-conorm generalizes union and satisfies similar properties, with 0 as the identity. A common t-conorm is the max operator:

$$S(\mu_A(x), \mu_B(x)) = \max(\mu_A(x), \mu_B(x)).$$

2.3.17 Cardinality

The cardinality of a fuzzy set A is the sum of its membership degrees:

$$|A| = \sum_{x\in X} \mu_A(x).$$

For a discrete fuzzy set A with $\mu_A(1) = 0.5$, $\mu_A(2) = 1$, $\mu_A(3) = 0.2$, the cardinality is:

$$|A| = 0.5 + 1 + 0.2 = 1.7.$$

2.3.18 Cartesian Product

The Cartesian product of fuzzy sets $A \subseteq X$ and $B \subseteq Y$ is defined as:

$$\mu_{A\times B}(x, y) = \min(\mu_A(x), \mu_B(y)).$$

For $\mu_A(x) = 0.5$ and $\mu_B(y) = 0.7$, $\mu_{A\times B}(x, y) = 0.5$.

2.3.19 Projection

The projection of a fuzzy set $C \subseteq X \times Y$ onto X is defined as:

$$\mu_{\text{Proj}_X(C)}(x) = \max_{y \in Y} \mu_C(x, y).$$

If $\mu_C((x, y)) = \min(\mu_A(x), \mu_B(y))$, the projection simplifies to:

$$\mu_{\text{Proj}_X(C)}(x) = \mu_A(x).$$

2.3.20 Linguistic Hedges

Linguistic hedges modify membership functions to express qualifiers such as "very" or "more or less." For example: - "Very A" can be represented as $\mu_{\text{very } A}(x) = (\mu_A(x))^2$. - "More or less A" can be represented as $\mu_{\text{more or less } A}(x) = \sqrt{\mu_A(x)}$.

For $\mu_A(x) = 0.6$, "very A" gives $\mu_{\text{very } A}(x) = 0.36$, and "more or less A" gives $\mu_{\text{more or less } A}(x) = 0.774$.

2.3.21 Defuzzification

Defuzzification is the process of converting a fuzzy set into a crisp value. Common methods include: - **Centroid Method:** Computes the center of gravity:

$$x^* = \frac{\int x \mu_A(x) dx}{\int \mu_A(x) dx}.$$

- **Maximum Membership Principle:** Selects the value with the highest membership:

$$x^* = \arg\max_x \mu_A(x).$$

2.3.22 Reflexivity (Relation Properties)

A fuzzy relation $R \subseteq X \times X$ is reflexive if:

$$\mu_R(x, x) = 1 \quad \forall x \in X.$$

For example, $\mu_R(x, y) = \min(1, 1 - |x - y|)$ is reflexive since $\mu_R(x, x) = 1$.

2.3.23 Symmetry (Relation Properties)

A fuzzy relation $R \subseteq X \times X$ is symmetric if:

$$\mu_R(x, y) = \mu_R(y, x) \quad \forall x, y \in X.$$

For instance, $\mu_R(x, y) = 1 - |x - y|$ is symmetric because $\mu_R(x, y) = \mu_R(y, x)$.

2.3.24 Transitivity (Relation Properties)

A fuzzy relation $R \subseteq X \times X$ is transitive if:

$$\mu_R(x, z) \geq \min(\mu_R(x, y), \mu_R(y, z)) \quad \forall x, y, z \in X.$$

For example, $\mu_R(x, y) = 1 - |x - y|$ is transitive under certain constraints.

2.3.25 Fuzzy Equivalence Relation

A fuzzy equivalence relation is a fuzzy relation that is reflexive, symmetric, and transitive. For example, if $\mu_R(x, y) = \exp(-|x - y|)$, then R satisfies these properties and is a fuzzy equivalence relation.

2.3.26 Fuzzy Similarity Relation

A fuzzy similarity relation is a fuzzy equivalence relation used to measure similarity between elements. For instance, $\mu_R(x, y) = \exp(-\alpha|x - y|)$, where $\alpha > 0$, quantifies the similarity between x and y.

2.3.27 Normalization

Normalization adjusts the membership function to ensure the maximum membership value is 1. For a fuzzy set A, the normalized membership function is:

$$\mu'_A(x) = \frac{\mu_A(x)}{\max_{x \in X} \mu_A(x)}.$$

For $\mu_A(x) = 0.5, 0.7, 1.0$, normalization results in $\mu'_A(x) = 0.5, 0.7, 1.0$ (no change since max value is already 1).

2.3.28 Overlap

Overlap measures the degree to which two fuzzy sets share common elements. It is given by:

$$\text{Overlap}(A, B) = \sum_{x \in X} \min(\mu_A(x), \mu_B(x)).$$

For $\mu_A(x) = 0.5, 0.7$ and $\mu_B(x) = 0.4, 0.6$, overlap is:

$$\text{Overlap}(A, B) = \min(0.5, 0.4) + \min(0.7, 0.6) = 0.4 + 0.6 = 1.0.$$

2.3.29 Resolution Identity

The resolution identity expresses a fuzzy set A as the union of its α-cuts:

$$\mu_A(x) = \sup_{\alpha \in [0,1]} \min(\alpha, \mu_A(x)).$$

This property shows how α-cuts decompose a fuzzy set into crisp intervals.

2.3.30 Aggregation Operators

Aggregation operators combine membership values of multiple fuzzy sets into a single value. Common operators include: - **Maximum:** $\mu_{\text{agg}}(x) = \max(\mu_A(x), \mu_B(x))$. - **Minimum:** $\mu_{\text{agg}}(x) = \min(\mu_A(x), \mu_B(x))$. - **Arithmetic Mean:** $\mu_{\text{agg}}(x) = \frac{\mu_A(x)+\mu_B(x)}{2}$. For $\mu_A(x) = 0.6, \mu_B(x) = 0.4$, the maximum is 0.6, the minimum is 0.4, and the mean is 0.5.

2.3.31 Implication Operators

Implication operators define the degree to which a fuzzy proposition $A \to B$ holds. Common implication operators include: - **Zadeh's Implication:**

$$\mu_{A \to B}(x) = \max(1 - \mu_A(x), \mu_B(x)).$$

- **Lukasiewicz Implication:**

$$\mu_{A \to B}(x) = \min(1, 1 - \mu_A(x) + \mu_B(x)).$$

For $\mu_A(x) = 0.6$ and $\mu_B(x) = 0.4$, Zadeh's implication gives $\mu_{A \to B}(x) = 0.6$, while Lukasiewicz's implication gives $\mu_{A \to B}(x) = 0.8$.

2.3.32 Fuzzy Entropy

Fuzzy entropy measures the fuzziness or uncertainty of a fuzzy set. One definition is:

$$H(A) = -\sum_{x \in X} \mu_A(x) \log \mu_A(x) - (1 - \mu_A(x)) \log(1 - \mu_A(x)).$$

For a fuzzy set A with $\mu_A(x) = 0.5$ for all x, $H(A)$ is maximized, representing maximum fuzziness.

2.3.33 Support Intersection

The support intersection of fuzzy sets A and B is the intersection of their supports:

$$\text{Support}(A \cap B) = \{x \in X \mid \mu_A(x) > 0 \text{ and } \mu_B(x) > 0\}.$$

For $\mu_A(x)$ supported on $[0, 5]$ and $\mu_B(x)$ on $[3, 7]$, the support intersection is $[3, 5]$.

2.3.34 Continuity of Membership Functions

A membership function $\mu_A(x)$ is continuous if small changes in x result in small changes in $\mu_A(x)$. For example:

$$\mu_A(x) = \frac{x}{10}, \quad x \in [0, 10]$$

is continuous because $\mu_A(x)$ changes smoothly with x.

2.3.35 α-Level Operations

Operations on α-cuts simplify fuzzy set computations. For fuzzy sets A and B: - **Union:** $A_\alpha \cup B_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha \text{ or } \mu_B(x) \geq \alpha\}$. - **Intersection:** $A_\alpha \cap B_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha \text{ and } \mu_B(x) \geq \alpha\}$.

2.3.36 Hamming Distance

The Hamming distance between fuzzy sets A and B is:

$$d_H(A, B) = \sum_{x \in X} |\mu_A(x) - \mu_B(x)|.$$

For $\mu_A(x) = 0.5, 0.7$ and $\mu_B(x) = 0.4, 0.6$, $d_H(A, B) = |0.5 - 0.4| + |0.7 - 0.6| = 0.2$.

2.3.37 Euclidean Distance

The Euclidean distance between fuzzy sets A and B is:

$$d_E(A, B) = \sqrt{\sum_{x \in X} (\mu_A(x) - \mu_B(x))^2}.$$

For $\mu_A(x) = 0.5, 0.7$ and $\mu_B(x) = 0.4, 0.6$, $d_E(A, B) = \sqrt{(0.5 - 0.4)^2 + (0.7 - 0.6)^2} = \sqrt{0.02} = 0.141$.

2.3.38 Fuzzy Rules

Fuzzy rules describe relationships between fuzzy sets. A typical rule is:

$$\text{IF } x \text{ is } A \text{ THEN } y \text{ is } B,$$

where A and B are fuzzy sets. For example:

$$\text{IF temperature is high THEN fan speed is fast.}$$

The degree of rule activation is determined by the membership of x in A.

2.3.39 Composition of Relations

The composition of two fuzzy relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ is:

$$\mu_{R \circ S}(x, z) = \sup_{y \in Y} \min(\mu_R(x, y), \mu_S(y, z)).$$

For example, if $\mu_R(x, y)$ and $\mu_S(y, z)$ are defined as triangular membership functions, the composition provides a fuzzy relation on $X \times Z$.

2.3.40 Lukasiewicz Operators

Lukasiewicz operators generalize fuzzy logic operations: - **Lukasiewicz t-norm (intersection):**

$$T(x, y) = \max(0, x + y - 1).$$

- **Lukasiewicz t-conorm (union):**

$$S(x, y) = \min(1, x + y).$$

For $x = 0.6$ and $y = 0.5$, $T(x, y) = \max(0, 0.6 + 0.5 - 1) = 0.1$ and $S(x, y) = \min(1, 0.6 + 0.5) = 1$.

2.3.41 Generalized Modus Ponens

Generalized Modus Ponens (GMP) is an extension of classical modus ponens for fuzzy logic. If:

$$\text{Rule: IF } x \text{ is } A \text{ THEN } y \text{ is } B,$$

and x is A', where A' is similar to A, then y is inferred as B', where B' is modified accordingly:

$$\mu_{B'}(y) = \sup_{x \in X} \min(\mu_{A'}(x), \mu_B(y)).$$

For example, if $\mu_A(x)$ and $\mu_B(y)$ are triangular, the output B' will also follow a fuzzy set shape.

2.3.42 Generalized Modus Tollens

Generalized Modus Tollens (GMT) extends the classical inference method:

$$\text{IF } x \text{ is } A \text{ THEN } y \text{ is } B, \quad y \text{ is not } B', \quad \text{THEN } x \text{ is not } A'.$$

This approach infers A' based on the negation of B', typically requiring additional computations to align membership values.

2.3.43 Fuzzy Arithmetic

Fuzzy arithmetic defines operations (e.g., addition, subtraction, multiplication) for fuzzy numbers. For two fuzzy sets A and B: - Addition:

$$\mu_{A+B}(z) = \sup_{x+y=z} \min(\mu_A(x), \mu_B(y)).$$

- Multiplication:

$$\mu_{A \cdot B}(z) = \sup_{x \cdot y = z} \min(\mu_A(x), \mu_B(y)).$$

2.3.44 Extension Principle

The extension principle extends crisp mathematical functions to fuzzy sets. For a function $f : X \to Y$ and fuzzy set $A \subseteq X$:

$$\mu_{f(A)}(y) = \sup_{x \in X \,:\, f(x) = y} \mu_A(x).$$

For $f(x) = x^2$ and $\mu_A(x)$ defined over $[0, 2]$, the fuzzy set $f(A)$ will span $[0, 4]$.

2.3.45 Alpha-Cut Decomposition

Alpha-cut decomposition represents a fuzzy set A as a collection of its α-cuts:

$$A = \bigcup_{\alpha \in [0,1]} \alpha \cdot A_\alpha,$$

where A_α is a crisp set at level α. This decomposition facilitates computation by treating each α-cut as a traditional set.

2.3.46 Resolution Principle

The resolution principle uses α-cuts for inference in fuzzy systems. If a fuzzy set A is defined by its α-cuts A_α, the output set can be derived by performing operations on A_α at each level α.

2.3.47 Fuzzy Granularity

Fuzzy granularity describes the level of detail or precision in a fuzzy system. Fine granularity involves many precise fuzzy sets, while coarse granularity uses fewer, broader sets. For example: - Fine: Temperature is split into "Cold," "Warm," "Hot." - Coarse: Temperature is split into "Low" and "High."

2.3.48 Triangular Norms (t-norms)

Triangular norms are generalizations of intersection. Common t-norms include: - **Minimum:** $T(x, y) = \min(x, y)$. - **Product:** $T(x, y) = x \cdot y$. - **Lukasiewicz:** $T(x, y) = \max(0, x + y - 1)$.

2.3.49 Triangular Conorms (s-norms)

Triangular conorms are generalizations of union. Common s-norms include: - **Maximum:** $S(x, y) = \max(x, y)$. - **Probabilistic Sum:** $S(x, y) = x + y - x \cdot y$. - **Lukasiewicz:** $S(x, y) = \min(1, x + y)$.

2.3.50 λ-Cuts

A λ-cut generalizes α-cuts by focusing on levels defined by λ, where λ is not necessarily between 0 and 1. It identifies the subset of X that satisfies a particular level of membership:

$$A_\lambda = \{x \in X \mid \mu_A(x) \geq \lambda\}.$$

2.3.51 Zadeh's Extension Principle

Zadeh's Extension Principle allows extending classical functions to fuzzy sets. For a function $f : X \to Y$ and a fuzzy set $A \subseteq X$, the fuzzy set $f(A)$ is defined as:

$$\mu_{f(A)}(y) = \sup_{x \in X \,:\, f(x)=y} \mu_A(x).$$

For instance, if $f(x) = x^2$ and A is a triangular fuzzy set over $[0, 2]$, the resulting fuzzy set $f(A)$ spans $[0, 4]$ with adjusted membership values.

2.3.52 Mamdani Implications

Mamdani implications are commonly used in fuzzy control systems. For fuzzy sets A and B, the implication is defined as:

$$\mu_{A \to B}(x, y) = \min(\mu_A(x), \mu_B(y)).$$

This approach simplifies inference by focusing on the minimum membership degree.

2.3.53 Sugeno Implications

Sugeno implications are another form of fuzzy implication, defined as:

$$\mu_{A \to B}(x, y) = \max(1 - \mu_A(x), \mu_B(y)).$$

This approach aligns closely with classical logic while maintaining fuzziness.

2.3.54 R-Implications

R-implications are defined based on a t-norm T:

$$\mu_{A \to B}(x, y) = \sup\{z \in [0, 1] \mid T(\mu_A(x), z) \leq \mu_B(y)\}.$$

For example, using the product t-norm $T(x, y) = x \cdot y$, the implication is computed based on the product relationship.

2.3.55 Residual Operators

Residual operators are closely tied to t-norms and provide a framework for implications in fuzzy logic. For a t-norm T, the residual operator is:

$$R(x, y) = \sup\{z \mid T(x, z) \leq y\}.$$

For example, using the minimum t-norm, $R(x, y)$ simplifies to y if $x \leq y$, and 0 otherwise.

2.3.56 Duality in Operations

Duality refers to the relationship between fuzzy operations:

$$\mu_{\neg(A\cup B)}(x) = \mu_{\neg A\cap\neg B}(x), \quad \mu_{\neg(A\cap B)}(x) = \mu_{\neg A\cup\neg B}(x).$$

This property generalizes De Morgan's laws to fuzzy sets and applies to complement, intersection, and union.

2.3.57 Fuzzy Relations

A fuzzy relation $R \subseteq X \times Y$ is characterized by a membership function $\mu_R(x, y)$. For example, $\mu_R(x, y) = \exp(-|x - y|)$ represents a similarity relation between x and y.

2.3.58 Max-Min Composition

The max-min composition of two fuzzy relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ is:

$$\mu_{R\circ S}(x, z) = \sup_{y\in Y} \min(\mu_R(x, y), \mu_S(y, z)).$$

This operation is used in fuzzy inference to combine multiple relations.

2.3.59 Max-Product Composition

The max-product composition is another method for combining fuzzy relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$:

$$\mu_{R\circ S}(x, z) = \sup_{y\in Y}(\mu_R(x, y) \cdot \mu_S(y, z)).$$

This composition emphasizes the product of membership values rather than their minimum.

2.3.60 Level Sets

Level sets (or α-level sets) represent the elements of a fuzzy set A with membership degrees greater than or equal to α:

$$A_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\}.$$

These sets provide a crisp representation of the fuzzy set at various levels of confidence, aiding in analysis and computation.

2.3.61 Fuzzy Measures

Fuzzy measures generalize probability measures by relaxing the additivity requirement. A fuzzy measure m satisfies:

$$m(\emptyset) = 0, \quad m(X) = 1, \quad \text{and } A \subseteq B \implies m(A) \leq m(B).$$

For example, $m(\{x_1, x_2\}) = 0.7$ and $m(\{x_1, x_2, x_3\}) = 1$.

2.3.62 Possibility and Necessity Measures

Possibility and necessity measures assess the degree of possibility or certainty of events: - **Possibility:**

$$\Pi(A) = \sup_{x \in A} \mu_A(x).$$

- **Necessity:**

$$N(A) = 1 - \Pi(A^c).$$

For a fuzzy set A, $\Pi(A)$ reflects the most possible membership, while $N(A)$ reflects guaranteed membership.

2.3.63 Support Fuzzification

Support fuzzification creates a fuzzy set from a crisp set by assigning nonbinary membership values. For instance, a crisp set $C = \{1, 2, 3\}$ may be fuzzified to:

$$\mu_A(x) = \begin{cases} 0.5, & x = 1, \\ 0.8, & x = 2, \\ 1.0, & x = 3. \end{cases}$$

2.3.64 Entropy-Based Measures of Fuzziness

Entropy-based measures quantify the fuzziness of a set. A common formula is:

$$H(A) = -\sum_{x \in X} \mu_A(x) \log \mu_A(x).$$

For a fuzzy set with $\mu_A(x) = 0.5$ for all x, entropy $H(A)$ is maximized, indicating high fuzziness.

2.3.65 Distance Measures for Fuzzy Sets

Distance measures quantify the difference between two fuzzy sets A and B: - **Hamming Distance:**

$$d_H(A, B) = \sum_{x \in X} |\mu_A(x) - \mu_B(x)|.$$

- **Euclidean Distance:**

$$d_E(A, B) = \sqrt{\sum_{x \in X} (\mu_A(x) - \mu_B(x))^2}.$$

2.3.66 Hesitancy in Fuzzy Memberships

Hesitancy occurs when there is uncertainty about the degree of membership. It is often modeled using interval-valued fuzzy sets, where:

$$\mu_A(x) = [\mu_A^{\text{lower}}(x), \mu_A^{\text{upper}}(x)].$$

For example, $\mu_A(x) = [0.3, 0.7]$ expresses uncertainty about x's membership.

2.3.67 Type-2 Fuzzy Sets

Type-2 fuzzy sets account for uncertainty in membership functions. The membership of x is a fuzzy set itself, represented as:

$$\tilde{\mu}_A(x, u), \quad u \in [0, 1].$$

For example, $\tilde{\mu}_A(x)$ may be a triangular fuzzy set over $[0, 1]$, modeling second-order uncertainty.

2.3.68 Intuitionistic Fuzzy Sets

Intuitionistic fuzzy sets extend fuzzy sets by incorporating membership $\mu_A(x)$ and non-membership $\nu_A(x)$, such that:

$$\mu_A(x) + \nu_A(x) \leq 1.$$

The hesitation degree is:

$$\pi_A(x) = 1 - \mu_A(x) - \nu_A(x).$$

2.3.69 Interval-Valued Fuzzy Sets

An interval-valued fuzzy set A assigns a range of membership values to each element x:

$$\mu_A(x) = [\mu_A^{\text{lower}}(x), \mu_A^{\text{upper}}(x)].$$

For example, $\mu_A(x) = [0.4, 0.7]$ reflects uncertainty about x's membership.

2.3.70 Fuzzy Aggregation Functions

Fuzzy aggregation functions combine multiple membership values into a single value. Common aggregation functions include: - **Arithmetic Mean:**

$$\mu_{\text{agg}}(x) = \frac{\sum_{i=1}^{n} \mu_{A_i}(x)}{n}.$$

- **Weighted Sum:**

$$\mu_{\text{agg}}(x) = \sum_{i=1}^{n} w_i \cdot \mu_{A_i}(x), \quad \sum_{i=1}^{n} w_i = 1.$$

2.3.71 Gradual Membership Transition

Gradual membership transition describes the smooth change of membership values across the universe of discourse. For instance, a triangular fuzzy set with:

$$\mu_A(x) = \begin{cases} 0, & x < a \text{ or } x > c, \\ \frac{x-a}{b-a}, & a \leq x < b, \\ \frac{c-x}{c-b}, & b \leq x \leq c, \end{cases}$$

shows a gradual increase and decrease in membership from a to b and from b to c.

2.3.72 Non-Additivity in Fuzzy Measures

Non-additivity means that the measure of the union of two sets is not necessarily the sum of their measures:

$$m(A \cup B) \neq m(A) + m(B).$$

For example, a fuzzy measure m may assign $m(A) = 0.3$, $m(B) = 0.4$, and $m(A \cup B) = 0.6$.

2.3.73 Subadditivity and Superadditivity

- **Subadditivity:**

$$m(A \cup B) \leq m(A) + m(B).$$

- **Superadditivity:**

$$m(A \cup B) \geq m(A) + m(B).$$

These properties describe the behavior of fuzzy measures depending on the interaction between sets A and B.

2.3.74 Choquet Integral

The Choquet integral aggregates information in fuzzy systems, particularly with fuzzy measures. For a function f and fuzzy measure m:

$$\int f \, dm = \sum_{i=1}^{n} (f(x_{(i)}) - f(x_{(i-1)})) \cdot m(\{x_{(i)}, \ldots, x_{(n)}\}),$$

where $x_{(i)}$ are values of f sorted in ascending order.

2.3.75 Sugeno Integral

The Sugeno integral is another aggregation method for fuzzy measures:

$$\int f \, dm = \sup_{\alpha \in [0,1]} \min(\alpha, m(f^{-1}([\alpha, 1]))).$$

It is often used in decision-making and fuzzy control systems.

2.3.76 Belief and Plausibility Functions

Belief and plausibility functions generalize probability in fuzzy systems: - **Belief:**

$$\text{Bel}(A) = \sum_{B \subseteq A} m(B),$$

where m is a basic belief assignment. - **Plausibility:**

$$\text{Pl}(A) = 1 - \text{Bel}(A^c).$$

2.3.77 Fuzzy Logic-Based Reasoning

Fuzzy logic-based reasoning uses fuzzy sets to infer conclusions. For example, a fuzzy rule:

IF temperature is high THEN fan speed is fast.

Given a membership function for "high temperature," the resulting membership for "fast fan speed" is computed using inference techniques such as Mamdani or Sugeno methods.

2.3.78 Fuzzy Control Systems

Fuzzy control systems use fuzzy logic to manage processes with imprecise inputs. A fuzzy controller consists of: 1. Fuzzification: Converting crisp inputs to fuzzy sets. 2. Inference: Applying fuzzy rules. 3. Defuzzification: Converting fuzzy outputs to crisp values.

For instance, controlling a heater may use fuzzy rules like:

IF temperature is low THEN heating power is high.

2.3.79 Rank Ordering

Rank ordering involves sorting elements or alternatives based on their fuzzy memberships or aggregated scores. For example, if fuzzy scores are $0.7, 0.5, 0.9$, the rank order is:

3rd element ¿ 1st element ¿ 2nd element.

2.3.80 Possibilistic vs. Probabilistic Interpretations

- **Possibilistic:** Membership reflects possibility, focusing on the degree to which an event is plausible. - **Probabilistic:** Membership reflects likelihood, focusing on the frequency of occurrence. For example, a fuzzy set of "tall people" uses possibility, while the probability of someone being "tall" depends on statistical data.

2.3.81 Triangular Fuzzy Numbers

A triangular fuzzy number is defined by three parameters (a, b, c), representing the left endpoint, peak, and right endpoint, respectively. Its membership function is:

$$\mu_A(x) = \begin{cases} \frac{x-a}{b-a}, & a \leq x < b, \\ \frac{c-x}{c-b}, & b \leq x \leq c, \\ 0, & \text{otherwise.} \end{cases}$$

For example, $A = (1, 3, 5)$ defines a fuzzy number with peak membership at 3.

2.3.82 Trapezoidal Fuzzy Numbers

A trapezoidal fuzzy number is defined by four parameters (a, b, c, d), representing the start, plateau start, plateau end, and end of the fuzzy number. Its membership function is:

$$\mu_A(x) = \begin{cases} \frac{x-a}{b-a}, & a \leq x < b, \\ 1, & b \leq x \leq c, \\ \frac{d-x}{d-c}, & c < x \leq d, \\ 0, & \text{otherwise.} \end{cases}$$

For example, $A = (1, 2, 4, 5)$ represents a trapezoidal fuzzy number with a flat peak from 2 to 4.

2.3.83 Gaussian Fuzzy Numbers

A Gaussian fuzzy number is characterized by its mean m and standard deviation σ. Its membership function is:

$$\mu_A(x) = \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right).$$

For example, with $m = 3$ and $\sigma = 1$, the fuzzy number peaks at $x = 3$ and decreases symmetrically.

2.3.84 Membership Scaling

Membership scaling adjusts the degree of membership in a fuzzy set. For scaling factor k, the scaled membership function is:

$$\mu'_A(x) = \min(1, k \cdot \mu_A(x)).$$

For instance, if $\mu_A(x) = 0.5$ and $k = 2$, then $\mu'_A(x) = 1$.

2.3.85 Soft Clustering (e.g., Fuzzy C-Means)

Soft clustering assigns elements to multiple clusters with degrees of membership. In fuzzy c-means, the objective is to minimize:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^{m} \|x_i - v_j\|^2,$$

where u_{ij} is the membership of x_i in cluster j, v_j is the cluster center, and $m > 1$ controls fuzziness.

2.3.86 Cluster Validity Measures

Cluster validity measures evaluate the quality of clustering. Common measures include: - **Partition Coefficient:**

$$PC = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^2.$$

- **Partition Entropy:**

$$PE = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij} \log(u_{ij}).$$

2.3.87 Fuzzy Partition Matrix

A fuzzy partition matrix defines the memberships of elements in clusters:

$$U = [u_{ij}], \quad \text{where } u_{ij} \in [0, 1] \text{ and } \sum_{j=1}^{c} u_{ij} = 1.$$

For example, if $n = 3$ and $c = 2$, U could be:

$$U = \begin{bmatrix} 0.6 & 0.4 \\ 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix}.$$

2.3.88 Linguistic Variables

Linguistic variables take values expressed as words or phrases, such as "low," "medium," or "high." For example, "temperature" might have fuzzy sets: - "Low": $\mu_{\text{low}}(x)$. - "Medium": $\mu_{\text{medium}}(x)$. - "High": $\mu_{\text{high}}(x)$.

2.3.89 Approximate Reasoning

Approximate reasoning uses fuzzy logic to infer conclusions from imprecise information. For instance:

IF temperature is high THEN fan speed is fast.

If "temperature" is partially high ($\mu_{\text{high}}(x) = 0.7$), the output "fan speed is fast" will be activated with $\mu_{\text{fast}}(x) = 0.7$.

2.3.90 Fuzzy Rule Base

A fuzzy rule base contains a set of fuzzy IF-THEN rules. For example:

Rule 1: IF temperature is high THEN fan speed is fast.

Rule 2: IF temperature is medium THEN fan speed is moderate.

The rule base combines these rules to generate outputs based on the inputs' fuzzy memberships.

2.3.91 Fuzzy Decision Trees

Fuzzy decision trees are extensions of classical decision trees, where nodes represent fuzzy tests on attributes. Instead of binary splits, fuzzy splits are based on membership degrees. For example: - Root node: "IF temperature is high THEN follow branch 1." - Branch 1: "IF humidity is low THEN outcome = good weather."

2.3.92 Multi-Criteria Decision-Making

Fuzzy multi-criteria decision-making evaluates alternatives based on fuzzy criteria. A typical method involves: 1. Defining fuzzy criteria (e.g., "cost," "quality"). 2. Assigning membership values for each alternative. 3. Aggregating criteria using weighted sums or other operators.

For example, an alternative with memberships $\mu_{\text{cost}} = 0.7$ and $\mu_{\text{quality}} = 0.9$ may be ranked higher.

2.3.93 Fuzzy Optimization

Fuzzy optimization solves problems with fuzzy constraints or objectives. For instance:

$$\text{Maximize } \tilde{f}(x), \quad \text{subject to } \tilde{g}(x) \geq \tilde{b},$$

where $\tilde{f}(x)$ and $\tilde{g}(x)$ are fuzzy functions. The solution maximizes membership values while satisfying constraints.

2.3.94 Fuzzy Neural Networks

Fuzzy neural networks combine fuzzy logic and neural networks. Inputs are fuzzified, and fuzzy rules are encoded in the network structure. Learning adjusts rule parameters. For example: - Input: Temperature = "high." - Output: Fan speed = "fast."

2.3.95 Neuro-Fuzzy Systems

Neuro-fuzzy systems integrate fuzzy logic and neural networks for adaptive learning of fuzzy rules. A common architecture is the Adaptive Neuro-Fuzzy Inference System (ANFIS), which learns fuzzy rules and membership functions from data.

2.3.96 Fuzzy Differential Equations

Fuzzy differential equations handle systems with uncertainty in initial conditions or parameters. For example:

$$\frac{dy}{dt} = \tilde{f}(t, y), \quad y(0) = \tilde{y}_0,$$

where $\tilde{f}$ and $\tilde{y}_0$ are fuzzy functions and values. Solutions are expressed as fuzzy sets.

2.3.97 Hybrid Fuzzy Systems

Hybrid fuzzy systems combine fuzzy logic with other techniques (e.g., genetic algorithms, neural networks) for improved performance. For example, fuzzy logic handles uncertainty, while genetic algorithms optimize rule sets.

2.3.98 Granular Computing with Fuzzy Sets

Granular computing with fuzzy sets processes information at different levels of granularity. For example: - Fine granularity: Specific values ("temperature = 25°C"). - Coarse granularity: General categories ("temperature is high").

2.3.99 Computability of Fuzzy Systems

The computability of fuzzy systems examines their ability to model and solve problems using algorithms. For example, fuzzy controllers are computationally efficient and implementable in real-time systems.

2.3.100 Membership Evolution Over Time

Membership evolution over time describes how membership functions change dynamically. For example, in a dynamic system, a membership function $\mu_A(x,t)$ might evolve as:

$$\mu_A(x,t+1) = \mu_A(x,t) + \Delta\mu_A(x),$$

where $\Delta\mu_A(x)$ is a function of external inputs or system changes.

2.4 Crisp Fuzzy Sets

2.4.1 Crisp Fuzzy Sets

A crisp set is a collection of distinct elements, where each element either fully belongs to the set or does not belong at all. In contrast, a fuzzy set allows elements to have partial membership, characterized by a membership function $\mu : X \to [0,1]$.

Example: Let $X = \{1,2,3,4,5\}$ be a universal set. - Crisp Set: $A = \{2,4\}$. Here, $\mu_A(x) = \begin{cases} 1 & \text{if } x \in \{2,4\}, \\ 0 & \text{otherwise.} \end{cases}$ - Fuzzy Set: B with membership function $\mu_B(x) = \frac{x}{5}$. For example, $\mu_B(3) = 0.6$.

2.4.2 Binary Membership Function

A binary membership function represents a crisp set where each element has a membership value of either 0 (non-member) or 1 (member). It is defined as:

$$\mu_A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \notin A. \end{cases}$$

Example: For the universal set $X = \{a,b,c,d\}$ and crisp set $A = \{a,c\}$,

$$\mu_A(a) = 1, \quad \mu_A(b) = 0, \quad \mu_A(c) = 1, \quad \mu_A(d) = 0.$$

2.4.3 Support

The support of a fuzzy set A is the set of all elements of the universe that have a non-zero membership value:

$$\text{Support}(A) = \{x \in X \mid \mu_A(x) > 0\}.$$

Example: For fuzzy set A with $\mu_A(x) = \frac{x}{5}$ over $X = \{1, 2, 3, 4, 5\}$,

$$\text{Support}(A) = \{1, 2, 3, 4, 5\}.$$

2.4.4 Core

The core of a fuzzy set A is the set of all elements of the universe with full membership value (1):

$$\text{Core}(A) = \{x \in X \mid \mu_A(x) = 1\}.$$

Example: For fuzzy set B with $\mu_B(x) = \frac{x}{5}$ over $X = \{1, 2, 3, 4, 5\}$,

$$\text{Core}(B) = \{5\}.$$

2.4.5 Complement

The complement of a fuzzy set A is defined as:

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x), \quad \forall x \in X.$$

Example: For fuzzy set A with $\mu_A(x) = \frac{x}{5}$ over $X = \{1, 2, 3, 4, 5\}$,

$$\mu_{\overline{A}}(x) = 1 - \frac{x}{5}.$$

2.4.6 Union (Set-Theoretic Union)

The union of two fuzzy sets A and B is defined as:

$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)), \quad \forall x \in X.$$

Example: For fuzzy sets A and B over $X = \{1, 2, 3\}$:

$$\mu_A(x) = \{0.2, 0.6, 0.8\}, \quad \mu_B(x) = \{0.5, 0.4, 0.7\},$$

$$\mu_{A \cup B}(x) = \{0.5, 0.6, 0.8\}.$$

2.4.7 Intersection (Set-Theoretic Intersection)

The intersection of two fuzzy sets A and B is defined as:

$$\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x)), \quad \forall x \in X.$$

Example: For fuzzy sets A and B over $X = \{1, 2, 3\}$:

$$\mu_A(x) = \{0.2, 0.6, 0.8\}, \quad \mu_B(x) = \{0.5, 0.4, 0.7\},$$

$$\mu_{A\cap B}(x) = \{0.2, 0.4, 0.7\}.$$

2.4.8 Subset

A fuzzy set A is a subset of fuzzy set B if:

$$\mu_A(x) \leq \mu_B(x), \quad \forall x \in X.$$

Example: If $\mu_A(x) = \{0.2, 0.4, 0.6\}$ and $\mu_B(x) = \{0.5, 0.6, 0.8\}$, then $A \subseteq B$.

2.4.9 Equality

Two fuzzy sets A and B are equal if:

$$\mu_A(x) = \mu_B(x), \quad \forall x \in X.$$

Example: If $\mu_A(x) = \{0.3, 0.7, 0.5\}$ and $\mu_B(x) = \{0.3, 0.7, 0.5\}$, then $A = B$.

2.4.10 Exclusive Or (XOR)

The XOR of two fuzzy sets A and B is defined as:

$$\mu_{A\oplus B}(x) = \min(\mu_A(x) + \mu_B(x), 1) - 2 \cdot \mu_{A\cap B}(x).$$

Example: For fuzzy sets A and B over $X = \{1, 2\}$:

$$\mu_A(x) = \{0.4, 0.7\}, \quad \mu_B(x) = \{0.6, 0.3\},$$

$$\mu_{A\oplus B}(x) = \{0.8, 1.0\}.$$

2.4.11 Power Set

The power set of a crisp set A is the set of all possible subsets of A. For fuzzy sets, the power set includes all possible fuzzy subsets of A.

Example: Let $A = \{1, 2\}$. The power set of A (crisp) is:

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}.$$

For a fuzzy set A with $\mu_A(1) = 0.5$ and $\mu_A(2) = 0.7$, the power set includes fuzzy subsets such as:

$$\mu_B(1) = 0.3, \mu_B(2) = 0.5 \quad \text{or} \quad \mu_C(1) = 0.5, \mu_C(2) = 0.$$

2.4.12 Symmetry

A fuzzy set A is symmetric if the membership function is invariant under a specific transformation, such as reflection about a point.

Example: Consider a fuzzy set A over $X = \{-2, -1, 0, 1, 2\}$ with:

$$\mu_A(x) = \max(1 - |x|, 0).$$

This fuzzy set is symmetric about $x = 0$.

2.4.13 Partition

A collection of fuzzy sets $\{A_1, A_2, \ldots, A_n\}$ forms a partition of the universe X if: 1. $\sum_{i=1}^{n} \mu_{A_i}(x) = 1, \quad \forall x \in X$, and 2. $\mu_{A_i}(x) \in [0, 1], \quad \forall x \in X, \forall i$.

Example: For $X = \{1, 2, 3\}$, let:

$$\mu_{A_1}(x) = \{0.2, 0.5, 0.3\}, \quad \mu_{A_2}(x) = \{0.8, 0.5, 0.7\}.$$

Here, $\mu_{A_1}(x) + \mu_{A_2}(x) = 1$ for all $x \in X$, so $\{A_1, A_2\}$ is a partition.

2.4.14 Disjoint Sets

Two fuzzy sets A and B are disjoint if their intersection is empty:

$$\mu_{A \cap B}(x) = 0, \quad \forall x \in X.$$

Example: For $X = \{1, 2, 3\}$:

$$\mu_A(x) = \{0.3, 0.0, 0.0\}, \quad \mu_B(x) = \{0.0, 0.6, 0.8\}.$$

Since $\mu_{A \cap B}(x) = 0$, A and B are disjoint.

2.4.15 Empty Set

The empty set $\emptyset$ in fuzzy logic is defined as a set with zero membership for all elements:

$$\mu_{\emptyset}(x) = 0, \quad \forall x \in X.$$

Example: For $X = \{1, 2, 3\}$:

$$\mu_{\emptyset}(x) = \{0, 0, 0\}.$$

2.4.16 Universal Set

The universal set X in fuzzy logic is defined as a set with full membership for all elements:

$$\mu_X(x) = 1, \quad \forall x \in X.$$

Example: For $X = \{1, 2, 3\}$:

$$\mu_X(x) = \{1, 1, 1\}.$$

2.4.17 Idempotence of Union and Intersection

The operations of union and intersection are idempotent:

$$A \cup A = A, \quad A \cap A = A.$$

Example: For fuzzy set A over $X = \{1, 2\}$:

$$\mu_A(x) = \{0.4, 0.7\}, \quad \mu_{A \cup A}(x) = \mu_A(x), \quad \mu_{A \cap A}(x) = \mu_A(x).$$

2.4.18 Associativity of Union and Intersection

Union and intersection are associative:

$$(A \cup B) \cup C = A \cup (B \cup C), \quad (A \cap B) \cap C = A \cap (B \cap C).$$

Example: For A, B, C over $X = \{1\}$:

$$\mu_A(x) = 0.3, \quad \mu_B(x) = 0.5, \quad \mu_C(x) = 0.7,$$

$$\mu_{(A \cup B) \cup C}(x) = 0.7, \quad \mu_{A \cup (B \cup C)}(x) = 0.7.$$

2.4.19 Distributivity of Union and Intersection

Union distributes over intersection, and vice versa:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C),$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$

Example: For A, B, C over $X = \{1\}$:

$$\mu_A(x) = 0.4, \quad \mu_B(x) = 0.6, \quad \mu_C(x) = 0.8,$$

$$\mu_{A \cup (B \cap C)}(x) = \mu_{(A \cup B) \cap (A \cup C)}(x) = 0.8.$$

2.4.20 Complement Laws

The complement laws state:

$$A \cup \overline{A} = X, \quad A \cap \overline{A} = \emptyset.$$

Example: For A over $X = \{1, 2\}$:

$$\mu_A(x) = \{0.4, 0.6\}, \quad \mu_{\overline{A}}(x) = \{0.6, 0.4\},$$

$$\mu_{A \cup \overline{A}}(x) = \{1, 1\}, \quad \mu_{A \cap \overline{A}}(x) = \{0, 0\}.$$

2.4.21 Law of Double Negation

The law of double negation states that the complement of the complement of a fuzzy set returns the original set:

$$\overline{\overline{A}} = A.$$

Example: For a fuzzy set A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.2, 0.5, 0.7\}$:

$$\mu_{\overline{A}}(x) = \{0.8, 0.5, 0.3\}, \quad \mu_{\overline{\overline{A}}}(x) = \{0.2, 0.5, 0.7\}.$$

2.4.22 Identity Laws

The identity laws for fuzzy sets are:

$$A \cup \emptyset = A, \quad A \cap X = A.$$

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.3, 0.6, 0.9\}$:

$$\mu_{A \cup \emptyset}(x) = \mu_A(x), \quad \mu_{A \cap X}(x) = \mu_A(x).$$

2.4.23 Absorption Laws

The absorption laws for fuzzy sets are:

$$A \cup (A \cap B) = A, \quad A \cap (A \cup B) = A.$$

Example: For A and B over $X = \{1, 2\}$ with $\mu_A(x) = \{0.4, 0.7\}$ and $\mu_B(x) = \{0.6, 0.5\}$:

$$\mu_{A \cup (A \cap B)}(x) = \mu_A(x), \quad \mu_{A \cap (A \cup B)}(x) = \mu_A(x).$$

2.4.24 Difference

The difference of two fuzzy sets A and B is defined as:

$$\mu_{A-B}(x) = \min(\mu_A(x), 1 - \mu_B(x)), \quad \forall x \in X.$$

Example: For A and B over $X = \{1, 2\}$ with $\mu_A(x) = \{0.4, 0.7\}$ and $\mu_B(x) = \{0.6, 0.5\}$:

$$\mu_{A-B}(x) = \{0.4, 0.5\}.$$

2.4.25 Symmetric Difference

The symmetric difference of two fuzzy sets A and B is defined as:

$$\mu_{A \Delta B}(x) = \max(\mu_A(x), \mu_B(x)) - \mu_{A \cap B}(x).$$

Example: For A and B over $X = \{1, 2\}$ with $\mu_A(x) = \{0.4, 0.7\}$ and $\mu_B(x) = \{0.6, 0.5\}$:

$$\mu_{A \Delta B}(x) = \{0.6, 0.7\}.$$

2.4.26 Cartesian Product Properties

The Cartesian product of fuzzy sets A and B over universes X and Y is defined as:

$$\mu_{A \times B}(x, y) = \min(\mu_A(x), \mu_B(y)), \quad \forall x \in X, y \in Y.$$

Example: For A over $X = \{1, 2\}$ with $\mu_A(x) = \{0.3, 0.6\}$ and B over $Y = \{a, b\}$ with $\mu_B(y) = \{0.8, 0.5\}$:

$$\mu_{A \times B}(x, y) = \begin{bmatrix} 0.3 & 0.3 \\ 0.6 & 0.5 \end{bmatrix}.$$

2.4.27 Subset-Equality Relationships

For two fuzzy sets A and B, the subset and equality relationships are:

$$A \subseteq B \iff \mu_A(x) \leq \mu_B(x), \quad \forall x \in X,$$

$$A = B \iff \mu_A(x) = \mu_B(x), \quad \forall x \in X.$$

Example: If $\mu_A(x) = \{0.2, 0.5, 0.7\}$ and $\mu_B(x) = \{0.4, 0.5, 0.7\}$, then $A \subseteq B$.

2.4.28 Logical Equivalence with Propositional Logic

Fuzzy set operations correspond to logical operations:

$$A \cup B \leftrightarrow \text{logical OR}, \quad A \cap B \leftrightarrow \text{logical AND}.$$

Example: For A and B over $X = \{1\}$ with $\mu_A(x) = 0.4$ and $\mu_B(x) = 0.7$:

$$\mu_{A \cup B}(x) = 0.7, \quad \mu_{A \cap B}(x) = 0.4.$$

2.4.29 Commutativity of Union and Intersection

Union and intersection are commutative:

$$A \cup B = B \cup A, \quad A \cap B = B \cap A.$$

Example: For A and B over $X = \{1, 2\}$ with $\mu_A(x) = \{0.3, 0.6\}$ and $\mu_B(x) = \{0.5, 0.4\}$:

$$\mu_{A \cup B}(x) = \mu_{B \cup A}(x) = \{0.5, 0.6\}, \quad \mu_{A \cap B}(x) = \mu_{B \cap A}(x) = \{0.3, 0.4\}.$$

2.4.30 Membership Constraints (Binary Only)

In binary (crisp) sets, membership is constrained to 0 or 1:

$$\mu_A(x) \in \{0, 1\}, \quad \forall x \in X.$$

Example: For crisp set $A = \{1, 3\}$ over $X = \{1, 2, 3\}$:

$$\mu_A(x) = \{1, 0, 1\}.$$

2.4.31 Canonical Representation

The canonical representation of a fuzzy set A is expressed as:

$$A = \sum_{x \in X} \mu_A(x)/x,$$

where $\mu_A(x)/x$ denotes the contribution of each element x to the fuzzy set.

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.2, 0.5, 0.8\}$:

$$A = 0.2/1 + 0.5/2 + 0.8/3.$$

2.4.32 Logical Conjunction and Disjunction

Logical conjunction (AND) and disjunction (OR) in fuzzy logic are defined as:

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)), \quad \mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)).$$

Example: For A and B over $X = \{1, 2\}$ with $\mu_A(x) = \{0.4, 0.7\}$ and $\mu_B(x) = \{0.6, 0.5\}$:

$$\mu_{A \cap B}(x) = \{0.4, 0.5\}, \quad \mu_{A \cup B}(x) = \{0.6, 0.7\}.$$

2.4.33 Partitioning with Crisp Classes

Partitioning involves dividing a universal set X into mutually exclusive crisp subsets such that their union is the entire set.

Example: For $X = \{1, 2, 3, 4\}$, a partition is:

$$P_1 = \{1, 2\}, \quad P_2 = \{3, 4\}.$$

Here, $P_1 \cap P_2 = \emptyset$ and $P_1 \cup P_2 = X$.

2.4.34 Classical Set Operations on Power Sets

Classical operations such as union, intersection, and complement can be applied to power sets. For sets A and B in a power set:

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\},$$

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}.$$

Example: For $X = \{1, 2\}$:

$$\mathcal{P}(X) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}.$$

Union of $\{1\}$ and $\{2\}$: $\{1, 2\}$.

2.4.35 Indicator Functions

The indicator function for a crisp set A is defined as:

$$I_A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{otherwise.} \end{cases}$$

Example: For $A = \{1, 3\}$ over $X = \{1, 2, 3\}$:

$$I_A(x) = \{1, 0, 1\}.$$

2.4.36 Topological Properties

Topological properties in fuzzy sets relate to concepts like closure, interior, and boundary. The closure of a fuzzy set A is the smallest closed set containing A:

$$\text{closure}(A) = \{x \in X \mid \mu_A(x) > 0\}.$$

Example: For fuzzy set A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.2, 0.0, 0.8\}$:

$$\text{closure}(A) = \{1, 3\}.$$

2.4.37 Characteristic Subsets

A characteristic subset of a fuzzy set A is a crisp set derived from A by thresholding:

$$A_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\}.$$

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.3, 0.7, 0.5\}$ and $\alpha = 0.5$:

$$A_\alpha = \{2, 3\}.$$

2.4.38 Boolean Algebra Representation

Fuzzy sets can be represented within a Boolean algebra by mapping fuzzy operations to their Boolean equivalents:

$$\text{OR} \leftrightarrow \cup, \quad \text{AND} \leftrightarrow \cap, \quad \text{NOT} \leftrightarrow \overline{A}.$$

Example: For A and B over $X = \{1\}$ with $\mu_A(x) = 0.4$ and $\mu_B(x) = 0.7$:

$$\mu_{A \cup B}(x) = 0.7, \quad \mu_{A \cap B}(x) = 0.4, \quad \mu_{\overline{A}}(x) = 0.6.$$

2.4.39 Lattice Structure

Fuzzy sets form a lattice structure under the operations of union and intersection:

$$A \cup B = \sup(A, B), \quad A \cap B = \inf(A, B).$$

Example: For A and B over $X = \{1\}$ with $\mu_A(x) = 0.4$ and $\mu_B(x) = 0.7$:

$$\sup(A, B) = 0.7, \quad \inf(A, B) = 0.4.$$

2.4.40 Closure Properties

Fuzzy sets are closed under the operations of union, intersection, and complement:

$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)),$$

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)),$$

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x).$$

Example: For A and B over $X = \{1, 2\}$ with $\mu_A(x) = \{0.3, 0.6\}$ and $\mu_B(x) = \{0.5, 0.4\}$:

$$\mu_{A \cup B}(x) = \{0.5, 0.6\}, \quad \mu_{A \cap B}(x) = \{0.3, 0.4\}, \quad \mu_{\overline{A}}(x) = \{0.7, 0.4\}.$$

2.4.41 Boundary Properties

The boundary of a fuzzy set A is defined as the set of elements with partial membership values:

$$\text{Boundary}(A) = \{x \in X \mid 0 < \mu_A(x) < 1\}.$$

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0, 0.5, 1\}$:

$$\text{Boundary}(A) = \{2\}.$$

2.4.42 Complementary Laws

The complementary laws describe the relationships between a fuzzy set and its complement:

$$A \cup \overline{A} = X, \quad A \cap \overline{A} = \emptyset.$$

Example: For A over $X = \{1, 2\}$ with $\mu_A(x) = \{0.3, 0.7\}$:

$$\mu_{\overline{A}}(x) = \{0.7, 0.3\}, \quad \mu_{A \cup \overline{A}}(x) = \{1, 1\}, \quad \mu_{A \cap \overline{A}}(x) = \{0, 0\}.$$

2.4.43 Exactness vs. Approximation

Exactness refers to the ability of a fuzzy set to represent precise information, while approximation deals with representing imprecise or vague information.

Example: A fuzzy set representing "young age" might be:

$$\mu_{\text{young}}(x) = \frac{30 - x}{10}, \quad \text{for } x \in [20, 30].$$

This is an approximation of the concept of "young age."

2.4.44 Duality Principles

Duality principles in fuzzy logic state that the results of operations remain valid if unions are replaced with intersections, and vice versa, along with complements.

Example: The dual of the law:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C),$$

is:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$

2.4.45 Logical Operators

Fuzzy sets utilize logical operators such as AND, OR, and NOT:

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)), \quad \mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)).$$

Example: For A and B over $X = \{1\}$ with $\mu_A(x) = 0.3$ and $\mu_B(x) = 0.6$:

$$\mu_{A \cap B}(x) = 0.3, \quad \mu_{A \cup B}(x) = 0.6.$$

2.4.46 Set Difference as Logical Subtraction

The set difference in fuzzy logic is interpreted as logical subtraction:

$$\mu_{A - B}(x) = \min(\mu_A(x), 1 - \mu_B(x)).$$

Example: For A and B over $X = \{1, 2\}$ with $\mu_A(x) = \{0.4, 0.7\}$ and $\mu_B(x) = \{0.5, 0.2\}$:

$$\mu_{A - B}(x) = \{0.4, 0.8\}.$$

2.4.47 Subset Constraints in Crisp Contexts

In crisp sets, $A \subseteq B$ implies:

$$\mu_A(x) \leq \mu_B(x), \quad \forall x \in X.$$

Example: For $A = \{1, 2\}$ and $B = \{1, 2, 3\}$ over $X = \{1, 2, 3\}$:

$$\mu_A(x) = \{1, 1, 0\}, \quad \mu_B(x) = \{1, 1, 1\}.$$

Thus, $A \subseteq B$.

2.4.48 Finiteness and Countability

A fuzzy set is finite or countable if the universe of discourse X is finite or countable.

Example: For $X = \{1, 2, 3, \ldots\}$ (countable set) and fuzzy set A with:

$$\mu_A(x) = \frac{1}{x}, \quad x \in X.$$

A is countable.

2.4.49 Total and Partial Order

Fuzzy sets can exhibit total or partial order based on membership degrees:

$$\mu_A(x) \leq \mu_B(x), \quad \forall x \in X \quad \text{(total order)},$$

or for some $x \in X$ (partial order).

Example: For A and B over $X = \{1, 2\}$:

$$\mu_A(x) = \{0.4, 0.5\}, \quad \mu_B(x) = \{0.3, 0.5\}.$$

Here, A and B are partially ordered.

2.4.50 Venn Diagrams Representation

Fuzzy sets can be represented in Venn diagrams, where shading intensity indicates membership degrees.

Example: For A and B over $X = \{1, 2, 3\}$ with:

$$\mu_A(x) = \{0.4, 0.7, 0.5\}, \quad \mu_B(x) = \{0.6, 0.4, 0.8\},$$

the diagram uses gradients to show overlaps and differences in membership.

2.4.51 Structural Properties under Cartesian Products

The Cartesian product of fuzzy sets preserves structural properties such as closure under operations like union, intersection, and complement. If A and B are fuzzy sets, their Cartesian product is:

$$\mu_{A\times B}(x,y) = \min(\mu_A(x), \mu_B(y)).$$

Example: For A over $X = \{1,2\}$ with $\mu_A(x) = \{0.3, 0.6\}$ and B over $Y = \{a,b\}$ with $\mu_B(y) = \{0.8, 0.5\}$:

$$\mu_{A\times B}(x,y) = \begin{bmatrix} 0.3 & 0.3 \\ 0.6 & 0.5 \end{bmatrix}.$$

2.4.52 Boolean Ring Representation

Fuzzy sets can be analyzed using Boolean ring operations:

$$A \cap B \leftrightarrow \text{AND}, \quad A \cup B \leftrightarrow \text{OR}, \quad A\Delta B \leftrightarrow \text{XOR}.$$

Example: For A and B over $X = \{1,2\}$:

$$\mu_A(x) = \{0.4, 0.7\}, \quad \mu_B(x) = \{0.6, 0.5\}.$$
$$\mu_{A\Delta B}(x) = \{0.6, 0.7\}.$$

2.4.53 Set Algebra Frameworks

Set algebra provides a framework for combining fuzzy sets with operations such as union, intersection, and complement:

$$\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x)), \quad \mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x)).$$

Example: For A and B over $X = \{1,2\}$ with $\mu_A(x) = \{0.4, 0.6\}$ and $\mu_B(x) = \{0.5, 0.3\}$:

$$\mu_{A\cup B}(x) = \{0.5, 0.6\}, \quad \mu_{A\cap B}(x) = \{0.4, 0.3\}.$$

2.4.54 Subset Families

A subset family is a collection of fuzzy subsets of a universal set X. Subset families can form lattices under union and intersection operations.

Example: For $X = \{1,2\}$, consider fuzzy subsets A and B:

$$\mu_A(x) = \{0.4, 0.6\}, \quad \mu_B(x) = \{0.5, 0.3\}.$$

The family $\{A, B, A \cup B, A \cap B\}$ forms a lattice.

2.4.55 Closure Operators

A closure operator C on a fuzzy set A satisfies: 1. $A \subseteq C(A)$, 2. $C(C(A)) = C(A)$, 3. $A \subseteq B \implies C(A) \subseteq C(B)$.

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.3, 0.5, 0.7\}$, a closure operator could map A to the smallest superset with maximum membership degrees.

2.4.56 Topological Set Theory

In fuzzy set theory, topology is extended by defining open, closed, and boundary sets based on membership functions.

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.2, 0.5, 0.8\}$:

$$\text{Boundary}(A) = \{x \in X \mid 0 < \mu_A(x) < 1\} = \{1, 2\}.$$

2.4.57 Metric Spaces on Sets

A metric space for fuzzy sets defines a distance function d to measure dissimilarity between sets:

$$d(A, B) = \sup_{x \in X} |\mu_A(x) - \mu_B(x)|.$$

Example: For A and B over $X = \{1, 2, 3\}$ with:

$$\mu_A(x) = \{0.2, 0.5, 0.8\}, \quad \mu_B(x) = \{0.3, 0.4, 0.7\},$$

$$d(A, B) = \max(|0.2 - 0.3|, |0.5 - 0.4|, |0.8 - 0.7|) = 0.1.$$

2.4.58 Point-Set Relationships

Fuzzy point-set relationships generalize classical notions of membership and inclusion:

$$x \in A \iff \mu_A(x) > 0.$$

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.0, 0.6, 0.8\}$:

$$\text{Point } 2 \in A \quad \text{and } 3 \in A \quad (\text{since } \mu_A(2), \mu_A(3) > 0).$$

2.4.59 Distributive Lattices

Fuzzy sets form distributive lattices under union and intersection:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C),$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$

Example: For A, B, C over $X = \{1\}$ with:

$$\mu_A(x) = 0.4, \quad \mu_B(x) = 0.5, \quad \mu_C(x) = 0.7,$$

$$\mu_{A \cup (B \cap C)}(x) = \mu_{(A \cup B) \cap (A \cup C)}(x) = 0.7.$$

2.4.60 Hilbert Spaces on Sets

Hilbert spaces on fuzzy sets involve defining an inner product to analyze relationships:

$$\langle A, B \rangle = \sum_{x \in X} \mu_A(x) \cdot \mu_B(x).$$

Example: For A and B over $X = \{1, 2, 3\}$ with:

$$\mu_A(x) = \{0.2, 0.5, 0.7\}, \quad \mu_B(x) = \{0.3, 0.4, 0.6\},$$

$$\langle A, B \rangle = (0.2 \cdot 0.3) + (0.5 \cdot 0.4) + (0.7 \cdot 0.6) = 0.54.$$

2.4.61 Invariant Properties Under Transformations

Invariant properties refer to characteristics of fuzzy sets that remain unchanged under specific transformations, such as scaling or translation.

Example: If A over $X = \{1, 2, 3\}$ has $\mu_A(x) = \{0.3, 0.5, 0.7\}$, a translation transformation shifting membership degrees by $+0.1$ results in $\mu_A(x) = \{0.4, 0.6, 0.8\}$, but its core and boundary remain invariant.

2.4.62 Orthogonal Sets

Two fuzzy sets A and B are orthogonal if their intersection is empty:

$$\mu_{A \cap B}(x) = 0, \quad \forall x \in X.$$

Example: For A and B over $X = \{1, 2, 3\}$:

$$\mu_A(x) = \{0.3, 0.0, 0.0\}, \quad \mu_B(x) = \{0.0, 0.5, 0.6\}.$$

Since $\mu_{A \cap B}(x) = \{0.0, 0.0, 0.0\}$, A and B are orthogonal.

2.4.63 Linear Independence of Sets

Fuzzy sets $A_1, A_2, \ldots, A_n$ are linearly independent if no set can be expressed as a linear combination of the others:

$$\alpha_1 \mu_{A_1}(x) + \alpha_2 \mu_{A_2}(x) + \cdots + \alpha_n \mu_{A_n}(x) = 0, \quad \forall x \in X \implies \alpha_1 = \alpha_2 = \cdots = \alpha_n = 0.$$

Example: For A_1 and A_2 over $X = \{1, 2\}$ with:

$$\mu_{A_1}(x) = \{0.4, 0.6\}, \quad \mu_{A_2}(x) = \{0.2, 0.3\},$$

A_1 and A_2 are linearly independent.

2.4.64 Vector Space Extensions

Fuzzy sets can form vector spaces by defining addition and scalar multiplication:

$$\mu_{A+B}(x) = \mu_A(x) + \mu_B(x), \quad \mu_{cA}(x) = c \cdot \mu_A(x), \quad \forall x \in X.$$

Example: For A and B over $X = \{1, 2\}$ with:

$$\mu_A(x) = \{0.4, 0.6\}, \quad \mu_B(x) = \{0.3, 0.2\},$$

$$\mu_{A+B}(x) = \{0.7, 0.8\}, \quad \mu_{2A}(x) = \{0.8, 1.2\}.$$

2.4.65 Relational Algebra in Sets

Fuzzy relational algebra extends classical operations such as join and projection. A fuzzy relation R is defined by a membership function $\mu_R(x, y)$.

Example: For R over $X \times Y = \{(1, a), (1, b), (2, a), (2, b)\}$ with:

$$\mu_R(x, y) = \begin{bmatrix} 0.3 & 0.6 \\ 0.4 & 0.2 \end{bmatrix},$$

projection onto X gives:

$$\mu_{\pi_X}(x) = \max_y \mu_R(x, y) = \{0.6, 0.4\}.$$

2.4.66 Graph Representations of Sets

Fuzzy sets can be represented as graphs, with nodes corresponding to elements and edges weighted by membership degrees.

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.3, 0.5, 0.7\}$, the graph representation includes nodes $1, 2, 3$ and edge weights reflecting $\mu_A(x)$.

2.4.67 Isomorphism in Set Structures

Two fuzzy sets A and B are isomorphic if there exists a bijection $f : X \to Y$ such that:

$$\mu_B(f(x)) = \mu_A(x), \quad \forall x \in X.$$

Example: For A over $X = \{1, 2\}$ and B over $Y = \{a, b\}$ with:

$$\mu_A(x) = \{0.4, 0.7\}, \quad \mu_B(y) = \{0.4, 0.7\},$$

the bijection $f(1) = a, f(2) = b$ establishes isomorphism.

2.4.68 Dual Operators in Complementation

Dual operators in fuzzy logic relate complement to union and intersection:

$$\overline{A \cup B} = \overline{A} \cap \overline{B}, \quad \overline{A \cap B} = \overline{A} \cup \overline{B}.$$

Example: For A and B over $X = \{1, 2\}$ with:

$$\mu_A(x) = \{0.4, 0.7\}, \quad \mu_B(x) = \{0.5, 0.3\},$$

$$\mu_{\overline{A \cup B}}(x) = \mu_{\overline{A} \cap \overline{B}}(x) = \{0.5, 0.3\}.$$

2.4.69 Monotonicity in Set Operations

Fuzzy set operations are monotonic:

$$A \subseteq B \implies A \cup C \subseteq B \cup C, \quad A \cap C \subseteq B \cap C.$$

Example: For $A \subseteq B$ over $X = \{1, 2\}$ with:

$$\mu_A(x) = \{0.3, 0.4\}, \quad \mu_B(x) = \{0.5, 0.4\},$$

then:

$$\mu_{A \cup C}(x) \leq \mu_{B \cup C}(x), \quad \mu_{A \cap C}(x) \leq \mu_{B \cap C}(x).$$

2.4.70 Cardinality of Infinite Sets

The cardinality of a fuzzy set A is generalized as:

$$\text{Card}(A) = \sum_{x \in X} \mu_A(x).$$

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.2, 0.5, 0.8\}$:

$$\text{Card}(A) = 0.2 + 0.5 + 0.8 = 1.5.$$

2.4.71 Finiteness vs. Infiniteness in Computation

Finiteness in fuzzy sets refers to a finite universe X, whereas infiniteness involves an infinite or uncountable universe. Computation becomes more complex with infinite sets.

Example: For A over $X = \{1, 2, 3\}$, $\mu_A(x) = \{0.2, 0.5, 0.7\}$ is computationally finite. For A over $X = \mathbb{R}$ with $\mu_A(x) = e^{-x^2}$, computation requires numerical approximation.

2.4.72 Well-Ordering Principle

The well-ordering principle states that every non-empty set has a least element under a given ordering. This concept extends to fuzzy sets by considering membership degrees.

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.3, 0.7, 0.5\}$:

$$\text{Least element: } x = 1 \quad \text{(smallest membership degree)}.$$

2.4.73 Axiom of Choice Implications

The axiom of choice allows for the selection of elements from fuzzy sets, particularly for constructing fuzzy relations or partitions.

Example: For a fuzzy relation R over $X \times Y = \{(1, a), (2, b)\}$, the axiom of choice enables selecting a representative for each $x \in X$ with maximum membership in R.

2.4.74 Direct and Inverse Images

The direct image of a fuzzy set A under a function $f : X \to Y$ is:

$$\mu_{f(A)}(y) = \sup_{x \in X, f(x)=y} \mu_A(x).$$

The inverse image is:

$$\mu_{f^{-1}(B)}(x) = \mu_B(f(x)).$$

Example: For $f(x) = x^2$ and A over $X = \{1, 2\}$ with $\mu_A(x) = \{0.4, 0.7\}$:

$$\mu_{f(A)}(y) = \{0.7\} \text{ for } y = 4.$$

2.4.75 Category Theory and Sets

Fuzzy sets can be represented within category theory, where objects are fuzzy sets and morphisms are membership-preserving functions.

Example: For A over $X = \{1, 2\}$ and B over $Y = \{a, b\}$ with $\mu_A(x) = \{0.4, 0.7\}$ and $\mu_B(y) = \{0.6, 0.5\}$, a morphism $f : X \to Y$ satisfies $\mu_B(f(x)) = \mu_A(x)$.

2.4.76 Hypergraph Representations

Fuzzy sets can be represented as hypergraphs, where vertices correspond to elements and hyperedges represent membership degrees.

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.2, 0.5, 0.8\}$, the hypergraph has vertices $\{1, 2, 3\}$ and weighted hyperedges $\{(1, 0.2), (2, 0.5), (3, 0.8)\}$.

2.4.77 Exactness vs. Approximation in Computation

Exactness involves using precise membership functions, while approximation uses simplified or discrete values.

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.333, 0.667, 1.0\}$, approximation could round these values to $\{0.3, 0.7, 1.0\}$.

2.4.78 Continuity in Fuzzy Functions

A fuzzy function $f : X \to Y$ is continuous if small changes in membership in X produce small changes in Y.

Example: For $f(x) = \sin(x)$ and A over $X = \{0, \pi/4, \pi/2\}$ with $\mu_A(x) = \{0.2, 0.5, 0.7\}$, f is continuous because $\mu_{f(A)}(y)$ varies smoothly.

2.4.79 Convexity in Fuzzy Sets

A fuzzy set A is convex if:

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq \min(\mu_A(x_1), \mu_A(x_2)), \quad \forall x_1, x_2 \in X, \lambda \in [0, 1].$$

Example: For A over $X = [0, 1]$ with $\mu_A(x) = 1 - |x - 0.5|$, A is convex because membership values decrease symmetrically from $x = 0.5$.

2.4.80 Fuzzy Relations and Composition

The composition of two fuzzy relations R over $X \times Y$ and S over $Y \times Z$ is:

$$\mu_{R \circ S}(x, z) = \sup_{y \in Y} \min(\mu_R(x, y), \mu_S(y, z)).$$

Example: For R over $X \times Y = \{(1, a), (1, b)\}$ with:

$$\mu_R(x, y) = \begin{bmatrix} 0.4 & 0.7 \end{bmatrix},$$

and S over $Y \times Z = \{(a, z_1), (b, z_1)\}$ with:

$$\mu_S(y, z) = \begin{bmatrix} 0.6 \\ 0.5 \end{bmatrix},$$

$$\mu_{R \circ S}(x, z) = \max(\min(0.4, 0.6), \min(0.7, 0.5)) = 0.5.$$

2.4.81 Fuzzy Equivalence Relations

A fuzzy equivalence relation R on X satisfies reflexivity, symmetry, and transitivity:

$$\mu_R(x, x) = 1, \quad \mu_R(x, y) = \mu_R(y, x), \quad \mu_R(x, z) \geq \min(\mu_R(x, y), \mu_R(y, z)).$$

Example: For $X = \{1, 2, 3\}$ and R with:

$$\mu_R(x, y) = \begin{bmatrix} 1.0 & 0.5 & 0.3 \\ 0.5 & 1.0 & 0.4 \\ 0.3 & 0.4 & 1.0 \end{bmatrix},$$

R is reflexive, symmetric, and transitive.

2.4.82 Fuzzy Clustering

Fuzzy clustering partitions a dataset into fuzzy subsets where each point has a membership value in each cluster. A common algorithm is Fuzzy C-Means (FCM).

Example: For points $X = \{x_1, x_2, x_3\}$ and clusters C_1, C_2, membership values might be:

$$\mu_{C_1}(x) = \{0.8, 0.4, 0.2\}, \quad \mu_{C_2}(x) = \{0.2, 0.6, 0.8\}.$$

2.4.83 Defuzzification Methods

Defuzzification converts a fuzzy set into a crisp value. Common methods include: 1. **Centroid (COG):**

$$x^* = \frac{\sum_{x \in X} \mu_A(x) \cdot x}{\sum_{x \in X} \mu_A(x)}.$$

2. **Mean of Maximum (MOM):**

$$x^* = \frac{\sum_{x \in X, \mu_A(x) = \max \mu_A} x}{\text{count of such } x}.$$

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.2, 0.5, 0.7\}$:

$$x^*_{\text{COG}} = \frac{1(0.2) + 2(0.5) + 3(0.7)}{0.2 + 0.5 + 0.7} = 2.5.$$

2.4.84 Fuzzy Rule-Based Systems

Fuzzy rule-based systems use fuzzy logic to infer conclusions from fuzzy rules. A rule is of the form:

$$\text{If } x \text{ is } A \text{ then } y \text{ is } B.$$

Example: Rule: *If temperature is high, then fan speed is fast.* For a temperature of 30°C with $\mu_{\text{high}}(30) = 0.6$, the output fan speed has $\mu_{\text{fast}} = 0.6$.

2.4.85 Type-2 Fuzzy Sets

A Type-2 fuzzy set A has a fuzzy membership function $\tilde{\mu}_A(x, u)$, where $u \in [0, 1]$ represents uncertainty in membership.

Example: For A over $X = \{1, 2\}$ with:

$$\tilde{\mu}_A(1, u) = \{(0.3, 0.6)\}, \quad \tilde{\mu}_A(2, u) = \{(0.4, 0.7)\}.$$

Membership is represented as intervals.

2.4.86 Linguistic Variables

A linguistic variable is a variable described by fuzzy terms, such as "low," "medium," or "high," with corresponding fuzzy sets.

Example: For the variable *Temperature*, fuzzy sets might be:

$$\mu_{\text{low}}(x) = \max(0, 1 - x/20), \quad \mu_{\text{high}}(x) = \max(0, (x - 20)/20).$$

2.4.87 Fuzzy Implication Operators

Fuzzy implications define logical relationships. Common operators include: 1. **Zadeh's Implication:**

$$\mu_{A \Longrightarrow B}(x) = \max(1 - \mu_A(x), \mu_B(x)).$$

2. **Lukasiewicz Implication:**

$$\mu_{A \Longrightarrow B}(x) = \min(1, 1 - \mu_A(x) + \mu_B(x)).$$

Example: For A and B over $X = \{1\}$ with $\mu_A(x) = 0.4$ and $\mu_B(x) = 0.7$:

$$\mu_{A \Longrightarrow B}(x) = \max(1 - 0.4, 0.7) = 0.7 \quad \text{(Zadeh)}.$$

2.4.88 Possibility and Necessity Measures

Possibility and necessity quantify the plausibility of an event:

$$\text{Poss}(A) = \sup_{x \in X} \mu_A(x), \quad \text{Nec}(A) = 1 - \text{Poss}(\overline{A}).$$

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.3, 0.6, 0.8\}$:

$$\text{Poss}(A) = 0.8, \quad \text{Nec}(A) = 1 - 0.2 = 0.8.$$

2.4.89 Triangular and Trapezoidal Membership Functions

Triangular and trapezoidal membership functions are commonly used to define fuzzy sets: 1. **Triangular:**

$$\mu_A(x) = \max\left(0, \frac{x-a}{b-a}, \frac{c-x}{c-b}\right), \quad x \in [a, c].$$

2. **Trapezoidal:**

$$\mu_A(x) = \max\left(0, \frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), \quad x \in [a, d].$$

Example: For a triangular fuzzy set A with $a = 1, b = 3, c = 5$:

$$\mu_A(2) = \frac{2-1}{3-1} = 0.5, \quad \mu_A(4) = \frac{5-4}{5-3} = 0.5.$$

2.4.90 Membership Function Normalization

Normalization scales a membership function to ensure:

$$\sup_{x \in X} \mu_A(x) = 1.$$

Example: For A over $X = \{1, 2, 3\}$ with $\mu_A(x) = \{0.2, 0.4, 0.8\}$:

$$\text{Normalize: } \mu'_A(x) = \frac{\mu_A(x)}{\sup_{x \in X} \mu_A(x)} = \{0.25, 0.5, 1.0\}.$$

2.5 Properties of Fuzzy Membership Functions

Fuzzy membership functions are fundamental in representing the degree of membership of elements in a fuzzy set. They have several key properties:

1. **Range**
   - The range of a fuzzy membership function is always in the interval $[0, 1]$.
   - $\mu(x) = 0$ indicates no membership, while $\mu(x) = 1$ indicates full membership.
2. **Normalization**
   - Membership functions are typically normalized so that the highest membership degree is 1 (i.e., $\mu(x) = 1$ for at least one x in the universe of discourse). As an example consider a triangular membership function defined for a universe of discourse x in the range $[0, 10]$ and described as follows:

$$\mu(x) = \begin{cases} 0 & \text{if } x < 2 \text{ or } x > 8, \\ \frac{x-2}{4} & \text{if } 2 \leq x \leq 6, \\ \frac{8-x}{2} & \text{if } 6 < x \leq 8. \end{cases}$$

     - The function above represents a triangular fuzzy set with a peak at $x = 6$. The highest membership degree ($\mu(x) = 1$) occurs at $x = 6$, ensuring the function is normalized.
     - For all other values of x, $\mu(x)$ lies in the interval $[0, 1]$.
3. **Shape**
   - As we discussed, membership functions can have various shapes depending on the problem domain. Some examples are:

- *Triangular*: Defined by a linear increase and decrease, forming a triangle.
- *Trapezoidal*: Similar to triangular but with a flat top, allowing for a range of full membership.
- *Gaussian*: Smooth, bell-shaped curves, often used for modeling uncertainty.
- *Sigmoid*: S-shaped curves used for gradual transitions.
- *Piecewise Linear*: Combinations of linear segments for flexible modeling.

4. **Continuity**

   - Membership functions can be continuous or discrete, depending on the nature of the problem.
     - *Continuous functions* are smoother and often preferred in modeling real-world phenomena.
     - *Discrete functions* are suitable for systems with finite, countable input values.

5. **Overlap**

   - Membership functions can overlap within the same universe of discourse, allowing an element to partially belong to multiple fuzzy sets (e.g., an element can be "somewhat hot" and "somewhat warm").

6. **Complement**

   - The complement of a fuzzy membership function is calculated as $1 - \mu(x)$, representing the degree to which an element does not belong to the set.

7. **Support**

   - The support of a membership function is the set of all points x where $\mu(x) > 0$.
   - It indicates the range of elements with some degree of membership in the fuzzy set.

8. **Core**

   - The core of a membership function is the set of all points x where $\mu(x) = 1$.
   - It represents the elements that fully belong to the fuzzy set.

9. **Crossover Points**
    - Crossover points are the values of x where $\mu(x) = 0.5$.
    - These points represent the threshold of partial membership.
10. **Flexibility**
    - Membership functions can be tailored to specific applications by adjusting their parameters (e.g., width, height, and center for triangular or Gaussian functions).
11. **Non-Negativity**
    - Membership functions are always non-negative: $\mu(x) \geq 0$ for all x.
12. **Additivity (Optional)**
    - In some cases, membership functions for overlapping fuzzy sets sum to 1 for any x. This is not a strict requirement but is used in certain applications like probability-inspired fuzzy logic.
13. **Subjectivity**
    - Membership functions are subjective and domain-specific, often defined based on expert knowledge, experimental data, or problem constraints.
14. **α-Cuts**
    - α-Cut is a set of elements in X where the membership value is at least α.

$$A_\alpha = \{x \in X | \mu_A(x) \geq \alpha\}$$

15. **Boundary**
    - Boundary is a region between full membership and no membership ($0 < \mu_A(x) < 1$).
16. **Crossovers (Optional)**
    - Crossover Points refer to the points in the universe of discourse where the degree of membership equals 0.5.

Below is an example code in R script to showcase various membership functions we discussed:


```r
# Uncomment the below if you need install packages
# install.packages("devtools")
# install.packages("ggplot2")

# Load necessary libraries
library(ggplot2)

# Define the universe of discourse (temperature in Celsius)
# To have a smaller set we went from 0 to 10 degrees
x <- seq(0, 10, by = 0.1)

# Define our membership functions, you can do yours as well!
membership_functions <- list(
Triangular = pmax(pmin((x - 2) / 4, (8 - x) / 2), 0), # Triangular: Peak at 6
Trapezoidal = pmax(pmin((x - 1) / 4, 1, (8 - x) / 1), 0), # Trapezoidal: Flat top
Gaussian = exp(-((x - 5)^2) / (2 * 2^2)),  # Gaussian: Bell curve
Sigmoidal = 1 / (1 + exp(-2 * (x - 5))), # Sigmoid: S-shaped curve
ZShaped = ifelse(x <= 3, 1, ifelse(x >= 7, 0, 1 - 2 * ((x - 3) / 4)^2)), # Z-shaped:
↪  Gradual decline
Complement = 1 - exp(-((x - 5)^2) / (2 * 2^2))  # Complement of Gaussian
)

# Create a data frame for our plotting
data <- data.frame(
x = rep(x, times = length(membership_functions)),
y = unlist(membership_functions),
Function = rep(names(membership_functions), each = length(x))
)

# Plot membership functions
print(ggplot(data, aes(x = x, y = y, color = Function)) +
geom_line(linewidth = 1) +
theme_minimal() +
labs(
title = "Fuzzy Membership Functions for Heating System",
x = "Temperature (°C)",
y = "Membership Degree"
) +
```



```r
scale_color_brewer(palette = "Set1") +
theme(legend.title = element_blank()))

# Let's analyze properties of our membership functions
# Check the Range:
range_check <- lapply(membership_functions, function(y) all(y >= 0 & y <= 1))
# Check Core (where membership = 1)
core_values <- lapply(membership_functions, function(y) x[y == 1])
# Check Support (where membership > 0)
support_values <- lapply(membership_functions, function(y) x[y > 0])
# Do we have Crossovers (where membership = 0.5)
crossover_points <- lapply(membership_functions, function(y) x[abs(y - 0.5) < 1e-2])

# And print the results of above checks
cat("Range Check (All within [0, 1]):\n")
print(range_check)
cat("\nCore Values (Membership = 1):\n")
print(core_values)
cat("\nSupport Values (Membership > 0):\n")
print(support_values)
cat("\nCrossover Points (Membership = 0.5):\n")
print(crossover_points)
```

2.6 Designing Fuzzy Fuzzy Heating and Cooling - Example

In this section, we explore our own conceptual and simplified Fuzzy Heating and Cooling Control System. The objective is to develop a Python-based fuzzy logic control system for managing the heating and cooling of a household, providing a clear understanding of the fuzzy control process. Let's begin by running the code to get an initial sense of its functionality:

```
Enter mode (heating/cooling): heating
Enter operation mode (normal/quiet): quiet
Enter current temperature (-100 to 100°C): 21
Enter target temperature (13-30°C): 22
Enter electricity price factor (1.00-2.00): 1.8
Enter occupancy level (0-100): 20
```



```
Enter humidity (0-100): 55

Okay, we have these inputs:
Mode: Quiet | Heating Mode
Current Temperature: 21.0°C
Target Temperature: 22.0°C
Electricity Price Factor: 1.80
Occupancy: 20
Humidity: 55%

And, we computed these outputs:
Computed Fan Speed: 35%
ComputedSystem Stage: 1
```

Here is a brief explanation of each input and output:

- Heating or Cooling Mode:
  - Heating or cooling mode determines whether the system should actively increase or decrease the indoor temperature to achieve the desired comfort level. In heating mode, the system activates the furnace, which operates in one of two stages based on the heating requirements:
  - Normal Mode (Stage 1): Only half of the furnace burners are turned on, producing moderate heat to gently maintain or gradually increase the temperature. This mode is energy-efficient and designed for small temperature differences between the current and target temperatures. The blower fan can operate in slower speeds as well in the mode.
  - Aggressive Mode (Stage 2): All burners in the furnace are activated to produce maximum heat. This mode is utilized for larger temperature differences or when a faster heating response is required. The heating process increases the indoor temperature rapidly, ensuring it matches the target temperature using fuzzy logic rules optimized for heating operations.
  - Cooling Mode: In cooling mode, the system engages the air conditioning unit, which includes a compressor and condenser unit to remove heat from the indoor air. An evaporator coil that cools the air passing through it by extracting heat.

  - The blower fan plays a critical role in both heating and cooling by circulating air over the heat exchanger plates (for heating) or the condenser (for cooling). The airflow ensures efficient heat exchange, distributing warm or cool air evenly throughout the house.
  - The cooling mode operates similarly to heating, where the system uses fuzzy logic rules to adjust fan speed and cooling intensity. These adjustments are made based on the current temperature, target temperature, and other contextual factors such as occupancy, humidity, and electricity price.

- Operation Mode (Normal or Quiet):
  - Indicates the desired noise level of the system's operation.
  - **Quiet mode**: The fan speed is limited to ensure reduced noise, even if higher speeds might achieve the target temperature faster. This impacts rules that determine fan speed.

- Current Temperature:
  - This is the actual current measured temperature of the environment.
  - Based on the fuzzy membership functions. We can have $-100°C$ to $100°C$ this will further break into sub fuzzy categories such as "comfortable" or "medium" etc.

- Target Temperature
  - The desired temperature to be achieved by the system. Valid values are between $13°C$ to $30°C$
  - If the target is close to the current temperature, the system is expected to operate in a less aggressive mode, possibly using medium or low blower fan speeds.

- Electricity Price Factor
  - Reflects the cost of electricity, impacting decisions about energy-intensive operations. valid values are between 1.00 for cheapest and 2.00 being the most expensive rates.
  - A high price factor (e.g., 1.8) might reduce the heating stage or fan speed to save energy, especially during quiet night modes.

- Occupancy Level

  - Represents the percentage of occupancy by people in the environment (0-100%)
  - A low occupancy level like 20 might deprioritize aggressive heating or cooling since fewer people are affected, leading to reduced system activity.

- Humidity Level
  - Measures the relative moisture content in the air (0-100%).
  - At 55%, the humidity likely falls into the "comfortable" category.

Let's formalize **Fuzzy Variables and Membership Functions:**

- **Antecedents (Inputs):**
  - **Temperature:** Categorized into `very_cold`, `cold`, `comfortable`, `warm`, and `hot`.
  - **Target Temperature:** Defined as `cool`, `moderate`, and `warm`.
  - **Humidity:** Categorized into `dry`, `comfortable`, and `humid`.
  - **Electricity Price:** Divided into `low`, `medium`, and `high`.
  - **Occupancy Level:** Categorized as `empty`, `low`, `medium`, and `high`.
  - **Quiet Desired:** Binary input (`yes` or `no`).
- **Consequents (Outputs):**
  - **Fan Speed:** Defined in discrete values (`off`, `low`, `medium`, `high`, `max`).
  - **System Stage:** Discrete values (`off`, `normal`, `aggressive`).

Membership functions for each variable define fuzzy categories and their degrees of membership within a given universe of discourse. We also defines rules for mapping the inputs to outputs:

- We create different fuzzy rules based on the mode (`heating` or `cooling`). Here is an example:

```
Rule(temperature['very_cold'] | temperature['cold'],
↪  (system_stage['aggressive'], fan_speed['high']))
```

- Each rule combines antecedent conditions (e.g., temperature, humidity, and occupancy level, etc.) to determine consequent actions (e.g., fan speed and system stage).

- We also validate user inputs (temperature, target temperature, humidity, etc.) to ensure they fall within specified ranges.

After we create the rules, the system computes outputs based on the rules and the degree of membership of inputs in different fuzzy categories. Later the system performs discretization and adjustments if needed.

For instance target fan speed is discretized (quantized) to predefined steps for consistency. We also perform some manual overwrites and adjustments if needed. For instance, quiet mode applies additional limits to fan speed to ensure noise reduction. All of these are done in simulation.compute() function.

Our fuzzy `compute` function which uses Python's `skfuzzy.control`, performs the process of fuzzy inference. It evaluates the fuzzy rules, aggregates the results, and defuzzifies the outputs to provide crisp values for the consequents involves:

1. **Fuzzify inputs**: Determine membership degrees.

2. **Evaluate rules**: Compute firing strengths.

3. **Apply implications**: Scale consequent fuzzy sets.

4. **Aggregate results**: Combine outputs for each fuzzy set.

5. **Defuzzify aggregated outputs**: Compute crisp values.

These computations typically do not involve explicit integrals; instead, they rely on numerical approximations to perform operations like fuzzification, rule evaluation, and defuzzification. While integral principles underlie these processes, numerical methods ensure efficient execution, providing accurate and realistic system behavior based on fuzzy rules.

Notably, some advanced microcontroller families, such as the Motorola/Freescale/NXP HCS12 series, offer dedicated hardware capabilities or optimized assembly instructions tailored for fuzzy inference. These instructions accelerate key fuzzy operations, such as membership function evaluations, logical operations (AND, OR), and defuzzification calculations, enabling real-time performance even in resource-constrained embedded systems.

Below is a detailed breakdown of what happens behind the scenes, including the mathematical operations:

2.6.0.1 Fuzzification

- Converts crisp input values into degrees of membership in fuzzy sets using the membership functions defined for each antecedent.
- For each input x, the degree of membership $\mu(x)$ is calculated for all applicable fuzzy sets.

Details: If $\mu_A(x)$ is the membership function for fuzzy set A:

$$\mu_A(x) = \text{value of the membership function for } x \text{ in } A.$$

Example: If the input temperature is $x = 12°C$, and the fuzzy sets for temperature are:

- Low Temperature : $\mu_{\text{Low}}(x)$
- Medium Temperature : $\mu_{\text{Medium}}(x)$

The system evaluates $\mu_{\text{Low}}(12)$ and $\mu_{\text{Medium}}(12)$ using the respective membership functions.

2.6.0.2 Rule Evaluation

- Evaluates the strength (or "firing strength") of each fuzzy rule.
- Combines the antecedent conditions using logical operators (AND, OR) mapped to fuzzy operations:
  - AND : $\min(\mu_A(x), \mu_B(y))$
  - OR : $\max(\mu_A(x), \mu_B(y))$

Details: For a rule:

$$\text{IF } x \text{ is } A \text{ AND } y \text{ is } B \text{ THEN } z \text{ is } C,$$

the firing strength w is:

$$w = \min(\mu_A(x), \mu_B(y)).$$

Example: For a rule:

IF Temperature is Low AND Humidity is High THEN Fan Speed is High,

the firing strength w is:

$$w = \min(\mu_{\text{Low}}(x), \mu_{\text{High}}(y)).$$

2.6.0.3 Implication

- Applies the firing strength w of each rule to the consequent fuzzy set.
- The membership function of the consequent is scaled by the firing strength.

Details: If $\mu_C(z)$ is the membership function for the consequent fuzzy set C, the modified membership function $\mu'_C(z)$ is:

$$\mu'_C(z) = \min(w, \mu_C(z)).$$

Example: If $w = 0.6$ and $\mu_{\text{High Fan Speed}}(z)$ is:

$$\mu_{\text{High Fan Speed}}(z) = \begin{cases} 1 & \text{if } 80 \leq z \leq 100, \\ 0 & \text{otherwise,} \end{cases}$$

the scaled membership function becomes:

$$\mu'_{\text{High Fan Speed}}(z) = \min(0.6, \mu_{\text{High Fan Speed}}(z)).$$

2.6.0.4 Aggregation

- Combines the modified membership functions of all rules that affect the same output.
- The aggregation method is typically max (union).

Details: If multiple rules contribute to the same consequent fuzzy set C:

$$\mu''_C(z) = \max(\mu'_{C_1}(z), \mu'_{C_2}(z), \ldots, \mu'_{C_n}(z)).$$

Example: If two rules contribute to "Fan Speed is High," their aggregated membership function is the pointwise maximum of the scaled membership functions from both rules.

2.6.0.5 Defuzzification

- Converts the aggregated fuzzy output back into a crisp value.
- The most common method is the **Centroid Method** (center of gravity), which computes the weighted average of all points in the output fuzzy set.

Details: For an output fuzzy set C with universe of discourse Z:

$$z^* = \frac{\int_Z z \cdot \mu_C(z)\, dz}{\int_Z \mu_C(z)\, dz}.$$

Here:

- z^*: Crisp output.
- $\mu_C(z)$: Aggregated membership function of C.
- Z: Range of possible values for the output.

Example: If the aggregated fuzzy set for fan speed is:

$$\mu_{\text{Fan Speed}}(z) = \begin{cases} 0.4 & \text{for } z = 60, \\ 0.6 & \text{for } z = 70, \\ 0.8 & \text{for } z = 80, \end{cases}$$

the crisp value z^* is computed as:

$$z^* = \frac{\sum z \cdot \mu_{\text{Fan Speed}}(z)}{\sum \mu_{\text{Fan Speed}}(z)}.$$

Let's take a look at the code:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy.control import Antecedent, Consequent, Rule, ControlSystem,
↪  ControlSystemSimulation

# Define fuzzy universes
temperature = Antecedent(np.arange(-100, 101, 0.5), 'temperature')
target_temp = Antecedent(np.arange(13, 30.5, 0.5), 'target_temp')
humidity = Antecedent(np.arange(0, 101, 1), 'humidity')
electricity_price = Antecedent(np.arange(1.0, 2.01, 0.01), 'electricity_price')
occupancy_level = Antecedent(np.arange(0, 101, 1), 'occupancy_level')
quiet_desired = Antecedent(np.arange(0, 2, 1), 'quiet_desired')

# Fan speed with discrete values: 0, 20, 25, 30, ..., 95, 100
fan_speed_vals = [0] + list(range(20, 101, 5))
fan_speed = Consequent(np.array(fan_speed_vals), 'fan_speed')
```



```
16
17  # System stage with discrete values: 0, 1, 2
18  system_stage = Consequent(np.array([0, 1, 2]), 'system_stage')
19
20  # Membership Functions
21  temperature['very_cold'] = fuzz.trapmf(temperature.universe, [-100,-100,0,5])
22  temperature['cold'] = fuzz.trimf(temperature.universe, [0,10,15])
23  temperature['comfortable'] = fuzz.trimf(temperature.universe, [14,22,26])
24  temperature['warm'] = fuzz.trimf(temperature.universe, [25,30,35])
25  temperature['hot'] = fuzz.trapmf(temperature.universe, [30,35,100,100])
26
27  target_temp['cool'] = fuzz.trapmf(target_temp.universe, [13,13,18,20])
28  target_temp['moderate'] = fuzz.trimf(target_temp.universe, [19,22,24])
29  target_temp['warm'] = fuzz.trapmf(target_temp.universe, [23,26,30,30])
30
31  humidity['dry'] = fuzz.trapmf(humidity.universe, [0,0,30,40])
32  humidity['comfortable'] = fuzz.trimf(humidity.universe, [35,50,65])
33  humidity['humid'] = fuzz.trapmf(humidity.universe, [60,70,100,100])
34
35  electricity_price['low'] = fuzz.trapmf(electricity_price.universe, [1.0,1.0,1.3,1.4])
36  electricity_price['medium'] = fuzz.trimf(electricity_price.universe, [1.3,1.5,1.7])
37  electricity_price['high'] = fuzz.trapmf(electricity_price.universe, [1.6,1.8,2.0,2.0])
38
39  occupancy_level['empty'] = fuzz.trimf(occupancy_level.universe, [0,0,20])
40  occupancy_level['low'] = fuzz.trimf(occupancy_level.universe, [15,30,50])
41  occupancy_level['medium'] = fuzz.trimf(occupancy_level.universe, [40,60,80])
42  occupancy_level['high'] = fuzz.trimf(occupancy_level.universe, [70,85,100])
43
44  quiet_desired['no'] = fuzz.trimf(quiet_desired.universe, [0,0,0])
45  quiet_desired['yes'] = fuzz.trimf(quiet_desired.universe, [1,1,1])
46
47  fan_speed['off'] = fuzz.trimf(fan_speed.universe, [0,0,20])
48  fan_speed['low'] = fuzz.trimf(fan_speed.universe, [20,35,50])
49  fan_speed['medium'] = fuzz.trimf(fan_speed.universe, [45,60,75])
50  fan_speed['high'] = fuzz.trimf(fan_speed.universe, [70,85,100])
51  fan_speed['max'] = fuzz.trimf(fan_speed.universe, [95,100,100])
52
53  system_stage['off'] = fuzz.trimf(system_stage.universe, [0,0,0])
54  system_stage['normal'] = fuzz.trimf(system_stage.universe, [1,1,1])
55  system_stage['aggressive'] = fuzz.trimf(system_stage.universe, [2,2,2])
56
57  def create_rules(mode):
```



```
rules = []
if mode == 'heating':
rules.extend([
Rule(temperature['very_cold'] | temperature['cold'], (system_stage['aggressive'],
↪  fan_speed['high'])),
Rule(temperature['cold'], (system_stage['normal'], fan_speed['medium'])),
Rule(temperature['comfortable'], (system_stage['normal'], fan_speed['low'])),
Rule(temperature['warm'], (system_stage['off'], fan_speed['off'])),
Rule(temperature['hot'], (system_stage['off'], fan_speed['off'])),
Rule(occupancy_level['empty'], fan_speed['low']),
Rule(occupancy_level['high'], fan_speed['high']),
Rule(humidity['humid'], fan_speed['high']),
Rule(electricity_price['high'], system_stage['normal']),
Rule(quiet_desired['yes'], fan_speed['low'])
])
else:  # cooling mode
rules.extend([
Rule(temperature['very_cold'], (system_stage['off'], fan_speed['off'])),
Rule(temperature['cold'], (system_stage['off'], fan_speed['off'])),
Rule(temperature['comfortable'], (system_stage['normal'], fan_speed['low'])),
Rule(temperature['warm'], (system_stage['normal'], fan_speed['medium'])),
Rule(temperature['hot'], (system_stage['aggressive'], fan_speed['high'])),
Rule(occupancy_level['empty'], fan_speed['low']),
Rule(occupancy_level['high'], fan_speed['high']),
Rule(humidity['humid'], fan_speed['high']),
Rule(electricity_price['high'], system_stage['normal']),
Rule(quiet_desired['yes'], fan_speed['low'])
])
return rules

def discretize_fan_speed(speed):
if speed <= 0:
return 0
elif speed >= 100:
return 100
else:
return min(fan_speed_vals, key=lambda x: abs(x - speed))

def main():
try:
mode = input("Enter mode (heating/cooling): ").strip().lower()
if mode not in ['heating', 'cooling']:
```



```
raise ValueError("Invalid mode. Must be 'heating' or 'cooling'.")

operation_mode = input("Enter operation mode (normal/quiet): ").strip().lower()
if operation_mode not in ['normal', 'quiet']:
raise ValueError("Invalid operation mode. Must be 'normal' or 'quiet'.")

quiet_mode = 1 if operation_mode == 'quiet' else 0
current_temp = float(input("Enter current temperature (-100 to 100°C): "))
target_temp_val = float(input("Enter target temperature (13-30°C): "))
electricity_price_val = float(input("Enter electricity price factor (1.00-2.00): "))
occupancy_val = int(input("Enter occupancy level (0-100): "))
humidity_val = int(input("Enter humidity (0-100): "))

# Input validation
if not (-100 <= current_temp <= 100):
raise ValueError("Temperature must be between -100 and 100")
if not (13 <= target_temp_val <= 30):
raise ValueError("Target temperature must be between 13 and 30")
if not (1.0 <= electricity_price_val <= 2.0):
raise ValueError("Price factor must be between 1.00 and 2.00")
if not (0 <= occupancy_val <= 100):
raise ValueError("Occupancy must be between 0 and 100")
if not (0 <= humidity_val <= 100):
raise ValueError("Humidity must be between 0 and 100")

# Force cooling off if temperature is too low
if mode == 'cooling' and current_temp < 16:
print("\nCooling disabled: Current temperature below 16°C")
system_stage_val = 0
fan_speed_val = 0
else:
control_system = ControlSystem(create_rules(mode))
sim = ControlSystemSimulation(control_system)

# Set inputs
sim.input['temperature'] = current_temp
sim.input['humidity'] = humidity_val
sim.input['electricity_price'] = electricity_price_val
sim.input['occupancy_level'] = occupancy_val
sim.input['quiet_desired'] = quiet_mode

sim.compute()
```



```
141
142 # Check for aggressive temperature change requirement
143 temp_diff = abs(target_temp_val - current_temp)
144 if temp_diff > 7 and occupancy_val > 0:
145 fan_speed_val = 100
146 system_stage_val = 2
147 else:
148 # Get and discretize fan speed
149 fan_speed_val = discretize_fan_speed(sim.output.get('fan_speed', 0))
150 system_stage_val = int(sim.output.get('system_stage', 1))
151
152 # Apply quiet mode fan speed limit only if not in aggressive mode
153 if quiet_mode == 1 and fan_speed_val > 80:
154 fan_speed_val = discretize_fan_speed(80)
155
156 # Print results
157 print(f"\nOkay, we have these inputs:")
158 print(f"Mode: {operation_mode.capitalize()} | {mode.capitalize()} Mode")
159 print(f"Current Temperature: {current_temp}°C")
160 print(f"Target Temperature: {target_temp_val}°C")
161 print(f"Electricity Price Factor: {electricity_price_val:.2f}")
162 print(f"Occupancy: {occupancy_val}")
163 print(f"Humidity: {humidity_val}%")
164
165 print(f"\nAnd, we computed these outputs:")
166 print(f"Computed Fan Speed: {fan_speed_val}%")
167 print(f"ComputedSystem Stage: {system_stage_val}")
168
169 except ValueError as ve:
170 print(f"Error: {ve}")
171 except Exception as e:
172 print(f"An error occurred: {e}")
173
174 if __name__ == "__main__":
175 main()
```


Chapter 3

Explaining Different Fuzzy Systems

3.1 Different Fuzzy Systems

Over time, depend on the use case, we evolved and created different fuzzy systems. Each type has strengths and weaknesses, and the choice of system depends on the application's requirements for interpretability, computational efficiency, and precision.

3.1.1 Mamdani Fuzzy Inference System

Introduced by Ebrahim Mamdani in 1975, it is one of the most widely used fuzzy logic models with the following key features:

- Linguistic variables are used for both the antecedents and consequents of fuzzy rules.
- The output is derived using fuzzy logic operations, producing a fuzzy set which is defuzzified to a crisp value.
- **If-Then Rule Example:**

$$\text{If } x \text{ is } A \text{ and } y \text{ is } B, \text{ then } z \text{ is } C$$

- Output z is represented by fuzzy sets like C.
- Used in control systems, like automatic braking systems or HVAC systems.

3.1.2 Sugeno Fuzzy Inference System

Proposed by Takagi, Sugeno, and Kang in 1985, it uses mathematical functions as consequents rather than fuzzy sets:

- Consequents are linear or constant functions.
- Efficient for numerical modeling and adaptive systems.
- **If-Then Rule Example:**

$$\text{If } x \text{ is } A \text{ and } y \text{ is } B, \text{ then } z = p \cdot x + q \cdot y + r$$

- Output z is calculated directly using a weighted average.
- Used in systems requiring precise outputs, such as adaptive control systems, optimization problems, and ANFIS (will be discussed later).

3.1.3 Tsukamoto Fuzzy Inference System

Proposed by Y. Tsukamoto, this model assigns fuzzy sets with monotonically increasing or decreasing membership functions.

- Each rule's output is a crisp value obtained by defuzzifying its fuzzy output.
- The overall output is computed as a weighted average of the individual rule outputs.
- **If-Then Rule Example:**

$$\text{If } x \text{ is } A, \text{ then } z = C$$

  where C is a crisp value determined by the membership degree.

- Used in situations where intermediate crisp outputs are necessary.

3.1.4 Hybrid Fuzzy Systems

Combines fuzzy logic with other computational intelligence techniques, such as neural networks, genetic algorithms, or optimization methods.

- **Types:**

- **Neuro-Fuzzy Systems:**
  - Combines fuzzy systems with neural networks to create adaptive models, such as ANFIS (Adaptive Neuro-Fuzzy Inference System).
  - Example: Learning fuzzy membership functions and rules from data.
- **Genetic-Fuzzy Systems:**
  - Genetic Algorithms are optimization techniques inspired by natural selection, where a population of potential solutions evolves over generations through selection, crossover, and mutation to maximize or minimize a fitness function. They are well-suited for solving complex, multi-dimensional problems without requiring gradient information. Genetic-Fuzzy Systems integrate genetic algorithms for optimizing fuzzy membership functions or rule sets. This will allow evolutionary learning of fuzzy rule bases.
- **Fuzzy-Swarm Intelligence Systems:**
  - Uses swarm-based optimization techniques (e.g., Particle Swarm Optimization) to improve fuzzy systems.
  - Example: Optimizing fuzzy control parameters for robotics.
- **Deep Learning and Fuzzy Systems:**
  - Combines fuzzy logic with deep learning architectures for better interpretability and reasoning.
  - Example: Fuzzy layers integrated into neural networks.

3.1.5 Takagi-Sugeno Fuzzy Systems

This is a special case of Sugeno systems where the output is expressed as a weighted combination of input variables.

- Often used for systems requiring smooth control surfaces.
- Ideal for mathematical modeling of dynamic systems such as robotics, process control, and time-series predictions.
- Rules are expressed as:

$$\text{If } x \text{ is } A \text{ and } y \text{ is } B, \text{ then } z = a \cdot x + b \cdot y + c$$

3.1.6 Fuzzy Cognitive Maps (FCM)

FCM uses a graph-based structure where nodes represent variables and edges represent fuzzy relationships.

- Models causal relationships between variables.
- Incorporates feedback loops for dynamic system behavior.
- Used in decision support systems, social system modeling, and complex system analysis.

3.1.7 Rule-Based Fuzzy Systems

Overview: Relies purely on pre-defined static fuzzy rules and does not adapt or learn from data:

- Easy to interpret but static.
- Ideal for systems where expert knowledge is available and well understood.

3.1.8 Hierarchical Fuzzy Systems

They reduce complexity by structuring multiple fuzzy systems hierarchically.

- Splits a complex fuzzy system into smaller, easier-to-manage subsystems.
- Reduces the curse of dimensionality.
- Good for complex decision-making systems, multi-stage processes.

3.1.9 Interval Type-2 Fuzzy Systems

They extend traditional fuzzy systems by allowing uncertainty in membership functions.

- Membership functions are defined as intervals rather than crisp values.
- Better handles uncertainty and noise.
- they are used in real-time control systems, financial modeling, and robotics.

3.1.10 Mamdani-Sugeno Hybrid Systems

Combines the interpretability of Mamdani systems with the precision of Sugeno systems.

- Uses Mamdani for intermediate layers and Sugeno for output layers.
- They are good for multi-objective systems requiring both linguistic interpretability and numerical precision.

3.2 Takagi-Sugeno-Kang FIS Use Cases in Engineering

As we quickly discussed, a **Sugeno-type fuzzy inference system**, also known as a **Takagi-Sugeno-Kang (TSK) fuzzy model**, is a popular fuzzy logic approach that combines fuzzy rule-based systems with mathematical functions to model a system's output. This method was introduced by Takagi, Sugeno, and Kang in 1985 and is especially useful for problems requiring a clear mathematical relationship between inputs and outputs. A Sugeno-type FIS is composed of several key components.

The first component is **Fuzzification**, which converts crisp inputs into fuzzy sets using membership functions. Next is the **Rule Base**, which contains fuzzy "if-then" rules. In Sugeno systems, the consequent (output) part of the rule is a mathematical function rather than a linguistic term. A rule typically has the form:

$$\text{If } x \text{ is } A \text{ and } y \text{ is } B, \text{ then } z = f(x, y),$$

where x and y are inputs to the system, A and B are fuzzy sets, and $f(x, y)$ is a linear or constant mathematical function.

The **Inference Engine** is responsible for determining the rule firing strengths based on input membership values. Subsequently, **Aggregation** combines the outputs of all rules using a weighted average method. Finally, unlike other fuzzy inference systems, the **Defuzzification** step is not explicitly needed in Sugeno-type Fuzzy Inference System (FIS), as its outputs are already crisp numerical values.

The **Rule Structure in Sugeno Systems** can be categorized into three main types based on the order of the output function.

The **Zero-Order Sugeno Model** is the simplest form, where the output z is a constant value. A typical rule in this model is expressed as:

$$\text{If } x \text{ is } A \text{ and } y \text{ is } B, \text{ then } z = c,$$

where c is a constant.

The **First-Order Sugeno Model** extends the rule structure by making the output z a linear function of the inputs. A typical rule in this model is given as:

$$\text{If } x \text{ is } A \text{ and } y \text{ is } B, \text{ then } z = p \cdot x + q \cdot y + r,$$

where p, q, and r are constants.

Finally, the **Higher-Order Sugeno Model** allows the output z to be a nonlinear function of the inputs. While this provides greater flexibility, such models are rarely used in practice due to their increased complexity.

In order to transform crisp inputs into a crisp output using Sugeno FIS principles we should first create the **Input Membership Function**, where fuzzy sets define the degree of membership for each input. For example, the membership degree of input x in fuzzy set A is given by:

$$\mu_A(x) = \text{membership degree of input } x \text{ in fuzzy set } A,$$

and for input y in fuzzy set B, it is:

$$\mu_B(y) = \text{membership degree of input } y \text{ in fuzzy set } B.$$

The second step is **Rule Firing Strength**, where the firing strength (degree of truth) for each rule is computed using fuzzy logic operators. For instance, the AND operator is often implemented as multiplication:

$$w_i = \mu_{A_i}(x) \cdot \mu_{B_i}(y),$$

where w_i is the firing strength of rule i.

The third step involves calculating the **Rule Output**, where the output z_i of each rule is derived using the rule's consequent function:

$$z_i = f_i(x, y) = p_i \cdot x + q_i \cdot y + r_i,$$

where p_i, q_i, and r_i are constants defining the consequent function for rule i.

Finally, in the **Aggregation** step, the outputs of all rules are combined using a weighted average method to produce the final crisp output:

$$z = \frac{\sum_i w_i \cdot z_i}{\sum_i w_i},$$

where w_i is the firing strength of rule i, and z_i is the output of rule i.

One of the **Key Features of Sugeno Systems** is their ability to produce **Crisp Outputs**. Unlike Mamdani systems, Sugeno systems generate crisp numerical outputs directly, without requiring an explicit defuzzification step.

Another important aspect is their **Efficient Computation**. By utilizing mathematical functions in the consequents of the rules, Sugeno systems reduce computational costs compared to other fuzzy inference systems.

Sugeno systems also excel in **Adaptability**. They are particularly well-suited for adaptive systems like Adaptive Neuro-Fuzzy Inference Systems (ANFIS), where the parameters of the consequent functions $f(x, y)$ can be optimized using machine learning techniques.

The **advantages of Sugeno fuzzy systems** make them highly suitable for numerical modeling and control applications. One key advantage is their **Compact Representation**, which is particularly efficient for numerical modeling tasks. Additionally, these systems offer **Ease of Optimization**, as they can seamlessly integrate with optimization techniques like gradient descent, a feature often utilized in systems such as Adaptive Neuro-Fuzzy Inference Systems (ANFIS). Furthermore, Sugeno systems provide a **Smooth Output Surface**, making them ideal for control systems that require smooth and continuous responses in engineering.

Despite their advantages, **Sugeno systems** have some limitations. One major drawback is their **Reduced Interpretability**, as the use of mathematical functions in the consequents makes them less intuitive compared to the linguistic terms used in Mamdani systems. Another limitation is that they are **Limited to Numerical Outputs**, which makes them less suitable for applications requiring linguistic or qualitative outputs.

Overall, Sugeno-type fuzzy systems are widely employed in engineering applications due to their computational efficiency, adaptability, and compatibility with modern optimization methods, despite these limitations.

3.2.1 Sugeno Fuzzy System - Robotic Navigation Example

Sugeno fuzzy control systems are extensively applied in navigation, obstacle avoidance, and robotic decision-making due to their ability to manage uncertainty and approximate reasoning. These systems are well-suited for dynamic and unpredictable environments, where mathematical functions define crisp outputs for real-time adaptability.

In navigation, Sugeno fuzzy logic integrates sensor inputs such as distance, orientation, and velocity to determine paths. For obstacle avoidance, Sugeno-based controllers adjust speed and direction by evaluating proximity sensor data. Additionally, these systems enable robots to make precise decisions in uncertain

scenarios, such as grasping objects with varying shapes or weights. For example, the output in Sugeno systems might be a linear function of inputs like:

$$z = p \cdot \text{distance} + q \cdot \text{orientation} + r,$$

where z represents the robot's speed, and p, q, and r are constants. Below are example rules for navigation and obstacle avoidance in a simplified Sugeno FIS for demonstration purpose:

1. If *distance to obstacle is far* and *orientation is aligned*, then speed $= p_1 \cdot \text{distance} + q_1 \cdot \text{orientation} + r_1$.
2. If *distance to obstacle is close* and *orientation is misaligned*, then speed $= p_2 \cdot \text{distance} + q_2 \cdot \text{orientation} + r_2$.
3. If *distance to obstacle is near*, then speed $= c$, where c is a constant (zero-order output).

In real life scenarios, we may use **Adaptive Sugeno Fuzzy Control** to enhance performance by integrating fuzzy logic **with learning mechanisms to dynamically update parameters in real time**. This adaptability is crucial for handling noisy sensor data and environmental changes. For instance, systems like Adaptive Neuro-Fuzzy Inference Systems (ANFIS) optimize parameters p, q, and r through learning, making them valuable for tasks such as balancing, dynamic navigation, and collision avoidance.

Below is a simplified implementation of a static Sugeno fuzzy inference system for robotic obstacle avoidance using Python. The example uses numerical outputs computed from linear functions for real-time adaptability.

```
import numpy as np
import skfuzzy as fuzz

# Define fuzzy variables and membership functions
distance = {
  'near': lambda x: fuzz.trapmf(np.array([x]), [0, 0, 2, 4])[0],
  'medium': lambda x: fuzz.trapmf(np.array([x]), [2, 4, 6, 8])[0],
  'far': lambda x: fuzz.trapmf(np.array([x]), [6, 8, 10, 10])[0]  # Corrected to stay
  ↪  within universe
}

orientation = {
  'misaligned': lambda x: fuzz.trapmf(np.array([x]), [-90, -90, -45, 0])[0],
```



```
    'aligned': lambda x: fuzz.trapmf(np.array([x]), [-15, -10, 10, 15])[0],  # Adjusted for better coverage
    'slightly_misaligned': lambda x: fuzz.trapmf(np.array([x]), [0, 15, 45, 90])[0]
}

# Define Sugeno-style crisp output functions
def sugeno_output_1(distance, orientation):
# Adjusted to match expected speed for Far and slightly aligned
p1, q1, r1 = 2.0, 1.0, 20.0
return p1 * distance + q1 * orientation + r1

def sugeno_output_2(distance, orientation):
# Example linear function: z = 1.5 * distance - 0.5 * orientation + 20
p2, q2, r2 = 1.5, -0.5, 20.0
return p2 * distance + q2 * orientation + r2

def sugeno_output_3():
# Zero-order function: z = 0
return 0.0

# Function to calculate robot speed based on inputs
def calculate_robot_speed(input_distance, input_orientation):
# Calculate membership values
distance_near = distance['near'](input_distance)
distance_medium = distance['medium'](input_distance)
distance_far = distance['far'](input_distance)

orientation_misaligned = orientation['misaligned'](input_orientation)
orientation_aligned = orientation['aligned'](input_orientation)
orientation_slightly_misaligned = orientation['slightly_misaligned'](input_orientation)

# Define 'aligned_or_slightly_aligned'
aligned_or_slightly_aligned = max(orientation_aligned, orientation_slightly_misaligned)

# Rule 3: If near AND misaligned, then speed = 0.0
if (distance_near > 0) and (orientation_misaligned > 0):
return sugeno_output_3()

# Otherwise, evaluate Rules 1 and 2
rule1_strength = min(distance_far, aligned_or_slightly_aligned)
rule2_strength = min(distance_medium, aligned_or_slightly_aligned)

rule1_output = rule1_strength * sugeno_output_1(input_distance, input_orientation)
rule2_output = rule2_strength * sugeno_output_2(input_distance, input_orientation)
```



```
56
57  total_strength = rule1_strength + rule2_strength
58
59  if total_strength == 0:
60  return 10.0  # Default speed when no rules fire
61
62  final_output = (rule1_output + rule2_output) / total_strength
63  return final_output
64
65  # Test scenarios for all described cases
66  test_scenarios = [
67  {"distance": 10, "orientation": 0, "description": "Far and aligned"},  # Expected
    ↪  30.00 → Adjusted to 60.00
68  {"distance": 5, "orientation": 5, "description": "Medium and slightly aligned"},  #
    ↪  Expected 25.00
69  {"distance": 2, "orientation": -80, "description": "Near and misaligned"},  # Expected
    ↪  0.00
70  {"distance": 3, "orientation": 10, "description": "Near and aligned"},  # Expected
    ↪  19.50
71  {"distance": 6, "orientation": -15, "description": "Medium and slightly misaligned"},
    ↪  # Expected 36.50
72  {"distance": 9, "orientation": 20, "description": "Far and slightly aligned"},  #
    ↪  Expected 58.00
73  ]
74
75  # Run test scenarios
76  print("Testing Scenarios for Robot Speed:")
77  for scenario in test_scenarios:
78  speed = calculate_robot_speed(scenario["distance"], scenario["orientation"])
79  print(f"{scenario['description']}: Distance = {scenario['distance']}, Orientation =
    ↪  {scenario['orientation']} -> Robot Speed = {speed:.2f}")
```

Table 3.1 summarizes some of the test scenario and resulted motor speed against the defined rules and expected outcomes.
Here we demonstrate the explainability of our FIS sample scenarios:

Far and Aligned: Distance = 10, Orientation = 0:

- `distance_far` = 1.0 (fully in "far").
- `orientation_aligned` = 1.0 (fully aligned).
- **Rule Evaluation:**

Scenario	Distance	Orientation	Expected Speed
Far and aligned	10	0	40.00
Medium and slightly aligned	5	5	25.00
Near and misaligned	2	-80	0.00
Near and aligned	3	10	19.50
Medium and slightly misaligned	6	-15	10
Far and slightly aligned	9	20	58.00

Table 3.1: Robot Navigation Speed Across Some Test Scenarios

  - **Rule 1:** $\min(1.0, 1.0) = 1.0$.
  - **Rules 2 and 3:** Not triggered.

- **Speed Calculation:**

$$\text{Rule 1 Output} = 1.0 \times (2.0 \times 10 + 1.0 \times 0 + 20.0) = 40.0$$

$$\text{Final Output} = \frac{40.0}{1.0} = 40.0$$

Medium and Slightly Aligned: Distance = 5, Orientation = 5:

- `distance_medium` = 1.0 (fully in "medium").
- `orientation_aligned` = 1.0 (fully aligned).
- **Rule Evaluation:**
  - **Rule 2:** $\min(1.0, 1.0) = 1.0$.
  - **Rules 1 and 3:** Not triggered.
- **Speed Calculation:**

$$\text{Rule 2 Output} = 1.0 \times (1.5 \times 5 + (-0.5) \times 5 + 20.0) = 25.0$$

$$\text{Final Output} = \frac{25.0}{1.0} = 25.0$$

Near and Misaligned: Distance = 2, Orientation = -80:

- `distance_near` = 1.0 (fully in "near").
- `orientation_misaligned` = 1.0 (fully misaligned).
- **Rule Evaluation:**
  - **Rule 3:** Triggered as both conditions are met.
  - **Rules 1 and 2:** Not evaluated.
- **Speed Calculation:**

$$\text{Speed} = 0.0 \quad \text{(from Rule 3)}$$

Near and Aligned: Distance = 3, Orientation = 10:

- `distance_near` = 0.5 (partially in "near").
- `orientation_aligned` = 1.0 (fully aligned).
- **Rule Evaluation:**
  - **Rule 1:** $\min(0.0, 1.0) = 0.0$.
  - **Rule 2:** $\min(0.5, 1.0) = 0.5$.
- **Speed Calculation:**

$$\text{Rule 2 Output} = 0.5 \times (1.5 \times 3 + (-0.5) \times 10 + 20.0) = 9.75$$

$$\text{Final Output} = \frac{9.75}{0.5} = 19.50$$

Medium and Slightly Misaligned: Distance = 6, Orientation = -15:

- `distance_medium` = 1.0 (fully in "medium").
- `orientation_aligned` = 0.0 (not aligned).
- `orientation_slightly_misaligned` = 0.0 (not slightly misaligned).

- **Rule Evaluation:**
  - **Rule 3:** Not triggered.
  - **Rules 1 and 2:** Both have zero strength.
- **Speed Calculation:**

$$\text{Final Output} = 10.0 \quad \text{(default speed)}$$

Far and Slightly Aligned: Distance = 9, Orientation = 20:

- `distance_far` = 1.0 (fully in "far").
- `orientation_slightly_misaligned` = 1.0 (fully slightly misaligned).
- **Rule Evaluation:**
  - **Rule 1:** $\min(1.0, 1.0) = 1.0$.
  - **Rule 2:** $\min(0.0, 1.0) = 0.0$.
- **Speed Calculation:**

$$\text{Rule 1 Output} = 1.0 \times (2.0 \times 9 + 1.0 \times 20 + 20.0) = 58.0$$

$$\text{Final Output} = \frac{58.0}{1.0} = 58.0$$

Chapter 4

More Advanced Fuzzy Systems

4.1 Some Backgrounds in Artificial Neural Networks

4.1.1 Gradient Descent

A gradient is a vector of partial derivatives that represents the rate and direction of the steepest increase of a function at a given point. For the steepest decrease, the negative of the gradient is used. The formula for the gradient of a scalar function $f(x_1, x_2, \ldots, x_n)$ is:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \\ \frac{\partial f}{\partial x_2} \\ \\ \vdots \\ \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

This is the vector of all partial derivatives of f with respect to each of its input variables $x_1, x_2, \ldots, x_n$. It points in the direction of the steepest ascent of f.

We all know that in the context of vector calculus and multivariable functions, when using a 2D Cartesian system, the slope of a line measures its steepness and is defined as:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

For a straight line described by the equation $y = mx + c$, the gradient (or slope) m indicates the rate at which y changes with respect to x. The gradient for a 2D curved

function $y = f(x)$ at a specific point refers to the slope of the tangent to the curve at that point, which is given by the derivative which is $f'(x)$.

Gradient Descent is a fundamental algorithm for optimization in machine learning and deep learning. Understanding its types, challenges, and variations is crucial for successfully training ANN models and solving complex optimization problems.

The goal is to minimize a loss function by iteratively moving in the direction of the steepest descent as defined by the negative of the gradient to improve the accuracy of a predictive model. We want to find the minimum value of a function $f(\theta)$, where θ represents the parameters of the model:

- Gradient descent starts with an initial guess for θ and iteratively updates it to reduce the value of $f(\theta)$.
- The gradient $\nabla f(\theta)$ represents the direction and rate of the steepest increase in f.
- To minimize f, we move in the opposite direction of the gradient:

  $$\theta \leftarrow \theta - \eta \nabla f(\theta)$$

  where:

  - η is the **learning rate** (a small positive scalar controlling the step size).

There are three main variants of gradient descent, depending on how the gradient is calculated:

- Batch Gradient Descent: The gradient is calculated using the entire dataset at every iteration.

  $$\theta \leftarrow \theta - \eta \nabla f(\theta)$$

  - Converges smoothly and accurately. It is suitable for convex functions.
  - However, it is computationally expensive for large datasets.
- Stochastic Gradient Descent (SGD): The gradient is calculated for a single random data point at each iteration.

  $$\theta \leftarrow \theta - \eta \nabla f_i(\theta)$$

  where f_i is the loss for a single data point i.

  - It has faster updates, suitable for larger learning tasks and can escape local minima captives due to its randomness selections.
  - However, it has noisy updates that can potentially cause convergence instability.

- Mini-Batch Gradient Descent: Combines the benefits of Batch and Stochastic Gradient Descent by calculating the gradient using a small batch of data points.

$$\theta \leftarrow \theta - \eta \nabla f_{\text{batch}}(\theta)$$

  - Efficient and scalable for large datasets.
  - Reduces noise while still speeding up convergence.

Gradient descent continues to update parameters until convergence, which occurs when:

1. The gradient becomes very small ($\|\nabla f(\theta)\| \approx 0$).
2. The change in the function value between iterations becomes negligible.

As you may imagine, for non-convex functions, gradient descent may get stuck in local minima or saddle points, which we will use stochastic gradient descent to escape and initialize parameters with random values multiple times.

We may also struggle with vanishing and exploding gradients. This occurs in deep neural networks, especially with sigmoid or tanh activations. which we will use ReLU activation functions and apply gradient clipping.

Finding the right learning rate can be tricky. Techniques like learning rate scheduling or adaptive optimizers (e.g., Adam) help to mitigate this issue.

Sometimes it is desired to accelerates gradient descent, we add momentum to updates, helping avoid oscillations:

$$v_t = \beta v_{t-1} + (1 - \beta)\nabla f(\theta)$$

$$\theta \leftarrow \theta - \eta v_t$$

As we quickly discussed, the **learning rate (η)** controls the step size of each descent update:

- **Small learning rate:** Ensures stable convergence but may take longer to reach the minimum.
- **Large learning rate:** Speeds up convergence but risks overshooting or diverging.

4.1.1.0.1 Dynamic Learning Rate

Fixed Learning Rates can be inefficient. If the rate is too high, the model may fail to converge or overshoot the optimal solution. If it's too low, convergence will be slow. Dynamic rates aim to adjust the learning rate over time or per parameter, enabling faster convergence and avoiding issues like poor local minima, overshooting or stagnation and have faster convergence.

Dynamic learning rate techniques and adaptive optimizers are strategies used in deep learning to improve training efficiency and model performance by intelligently adjusting the learning rate during training.

4.1.1.0.1.1 RMSProp

Root Mean Square Propagation technique adapts the learning rate for each parameter based on the magnitude of recent gradients:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2,$$
$$\Delta w_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t,$$

where g_t is the gradient at time t, γ is the decay factor (e.g., 0.9), η is the learning rate, and ϵ is a small constant for numerical stability.

RMSProp reduces oscillations in the optimization trajectory, especially in steep or shallow regions of the loss landscape and it is effective for non-stationary and noisy data.

4.1.1.0.1.2 Adam

Adaptive Moment Estimation combines the benefits of RMSProp and momentum. It adapts learning rates for each parameter while also incorporating momentum to accelerate convergence:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t},$$
$$\Delta w_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t,$$

where β_1 and β_2 are hyperparameters controlling the decay rates for the first and second moments.

Adam adapts learning rates for each parameter, making it robust to sparse gradients. Also, it combines momentum for faster convergence in directions of consistent gradient descent. It generally works well out of the box with minimal tuning.

4.1.1.0.1.3 Rate Decay

Dynamic Learning Rate Decay techniques systematically reduce the learning rate during training to help the model settle into a local minimum as it approaches convergence. We can have:

- **Time-Based Decay:**
$$\eta_t = \frac{\eta_0}{1 + \lambda t},$$
where η_0 is the initial learning rate, λ is the decay rate, and t is the current epoch.
- **Exponential Decay:**
$$\eta_t = \eta_0 \cdot e^{-\lambda t}.$$
- **Step Decay:** Reduces the learning rate by a factor at specific intervals (e.g., every 10 epochs).
- **Learning Rate Schedulers (e.g., ReduceLROnPlateau):** Adjusts the learning rate dynamically based on validation performance.

4.1.1.0.1.4 Adagrad

Adaptive Gradient Algorithm is another technique that adapts the learning rate based on the historical gradients. Parameters with larger past gradients receive smaller updates, while those with smaller past gradients receive larger updates:

- Accumulate the squared gradients for each parameter over all past updates:
$$G_t = \sum_{i=1}^{t} g_i^2,$$
where g_i is the gradient of the loss with respect to the parameter at step i and update each parameter w_t using:
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t} + \epsilon} g_t,$$
where:

- η: Initial learning rate.
- ϵ: Small constant to avoid division by zero.
- G_t: Accumulated sum of squared gradients.

To recap:

- **RMSProp:** Preferred for recurrent neural networks (RNNs) or tasks with non-stationary objectives.
- **Adam:** A good default optimizer for most tasks. Combines the strengths of both RMSProp and momentum.
- **Learning Rate Decay:** Used with simpler optimizers (e.g., SGD) or as a fine-tuning technique.

4.1.1.1 Linear Regression - Example

Let's do an example to demonstrate how gradient descent iterative updates θ to minimize the mean squared error (MSE):

$$f(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

As we mentioned before, the formula for the gradient of a scalar function $f(x_1, x_2, \ldots, x_n)$ is:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \\ \frac{\partial f}{\partial x_2} \\ \\ \vdots \\ \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

This is the vector of all partial derivatives of f with respect to each of its input variables $x_1, x_2, \ldots, x_n$. It points in the direction of the steepest ascent of f. Remember that our job is to compute gradient:

$$\nabla f(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i) \cdot x_i$$

And update weights:

$$\theta \leftarrow \theta - \eta \nabla f(\theta)$$

For better illustration, let's consider the following:

- Dataset: $\{(x_1, y_1) = (1, 2),\ (x_2, y_2) = (2, 4),\ (x_3, y_3) = (3, 6)\}$
- Initial weight: $\theta = 0$ and learning rate: $\eta = 0.1$

Step 1: Compute Predictions and Error

$$\begin{aligned} \hat{y}_1 &= \theta x_1 = 0 \cdot 1 = 0, & (y_1 - \hat{y}_1) &= 2 - 0 = 2 \\ \hat{y}_2 &= \theta x_2 = 0 \cdot 2 = 0, & (y_2 - \hat{y}_2) &= 4 - 0 = 4 \\ \hat{y}_3 &= \theta x_3 = 0 \cdot 3 = 0, & (y_3 - \hat{y}_3) &= 6 - 0 = 6 \end{aligned}$$

Step 2: Compute Gradient

$$\begin{aligned} \nabla f(\theta) &= -\frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i) \cdot x_i \\ &= -\frac{1}{3} \Big[(2 \cdot 1) + (4 \cdot 2) + (6 \cdot 3) \Big] \\ &= -\frac{1}{3} \Big[2 + 8 + 18 \Big] = -\frac{28}{3} \approx -9.33 \end{aligned}$$

Step 3: Update Weight

$$\begin{aligned} \theta &\leftarrow \theta - \eta \nabla f(\theta) \\ &= 0 - 0.1(-9.33) \\ &= 0.933 \end{aligned}$$

Step 4: Repeat for Next Iterations. Continue this process with updated θ until convergence. This means that the parameter θ stabilizes and the function $f(\theta)$ (e.g., the Mean Squared Error in this case) reaches a minimum value. (e.g., the gradient $\nabla f(\theta) \approx 0$, or the change in θ or $f(\theta)$ between iterations is below a specified threshold and the algorithm reaches a point where further optimization is no longer significant):

- The gradient $\nabla f(\theta)$ approaches zero as $f(\theta)$ nears its minimum. This is because the gradient represents the steepness of the loss function, and at a minimum (local or global), the gradient is zero.
- Convergence implies that further updates to θ become negligible because $\nabla f(\theta)$ is very small, and the changes in $f(\theta)$ across iterations are minimal.

4.1.2 Backpropagation

Backpropagation (short for *backward propagation of errors*) is an algorithm used to train artificial neural networks, particularly deep learning models. It is the foundation of supervised learning for neural networks and is crucial for optimizing the model's weights to minimize prediction errors.

Backpropagation computes the gradient of a loss function with respect to the model's weights and biases by propagating the error backward through the network. These gradients are then used in an optimization algorithm (like Gradient Descent) to update the weights and reduce the error.

We start by a forward pass, where input data is propagated through the network layer by layer. At each layer, the input is multiplied by weights, biases are added, and an activation function is applied to produce activations. The process continues until the final output is produced, representing the network's prediction. The difference between the predicted output and the true label is calculated using a loss function such as Mean Squared Error or Cross-Entropy Loss, which quantifies the model's performance. The computed loss serves as the basis for the backward pass.

The backward pass propagates the error backward through the network to compute the gradients needed for updating weights and biases. Starting from the output layer, the error is calculated as the derivative of the loss with respect to the activations. Using the chain rule of calculus, the error is backpropagated to each layer, where gradients of weights, biases, and activations are computed. These gradients are linked to the previous layer's gradients through the chain rule, ensuring that the error signal flows effectively backward. For each layer, the gradient of the loss with respect to weights and biases is calculated using the error signal and the activation from the previous layer. This process repeats until the gradients for all layers, back to the first, are computed.

Finally, the parameter update step adjusts the weights and biases of the network using an optimization algorithm like Gradient Descent. The gradients from the backward pass are scaled by the learning rate, ensuring incremental updates to the parameters in the direction that reduces the loss. Efficient implementation of backpropagation leverages vectorized operations and caches intermediate results from the forward pass to avoid redundant calculations. While highly effective, challenges such as vanishing or exploding gradients, overfitting, and computational cost in large networks must be addressed through techniques like improved initialization, regularization, and distributed computation frameworks.

Just a reminder about the chain rule that the derivative of the composite function

$f(g(x))$ with respect to x is given by:

$$\frac{d}{dx}[f(g(x))] = f'(g(x)) \cdot g'(x).$$

And for multivariables such as $z = f(x, y)$, where $x = g(t)$ and $y = h(t)$, the chain rule extends as:

$$\frac{dz}{dt} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}.$$

In backpropagation, we can say the loss L is computed as a composite function of weights, biases, activations, and inputs. The chain rule enables computation of gradients through layers by expressing:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

where:

- a: Activation,
- z: Pre-activation,
- w: Weight.

In the next section we discuss each of the steps in more details.

4.1.2.1 Forward Pass

1. **Input Propagation:** The input data is passed through the network layer by layer, applying weights and biases, and computing activations.
2. **Predictions:** The network produces an output (predicted value) from the last layer.
3. **Loss Calculation:** A loss function (e.g., Mean Squared Error, Cross-Entropy Loss) computes the difference between the predicted output and the true label.

4.1.2.2 Backward Pass

The backward pass propagates the error back through the network to compute gradients for the weights.

1. **Output Layer Error:** The error at the output layer is computed as the derivative of the loss with respect to the output activation.

2. **Backpropagate the Error:** For each layer l, starting from the last layer and moving backward:

   - Compute the gradient of the loss with respect to the weights $\mathbf{W}_l$, biases $\mathbf{b}_l$, and activations $\mathbf{a}_l$.
   - Use the chain rule of differentiation to link the gradients of the current layer to the gradients of the previous layer.

3. **Weight Gradient Calculation:** For a layer l, the gradients of weights and biases are computed as:

$$\frac{\partial \text{Loss}}{\partial \mathbf{W}_l} = \delta_l \cdot \mathbf{a}_{l-1}^T$$
$$\frac{\partial \text{Loss}}{\partial \mathbf{b}_l} = \delta_l$$

   where δ_l (error signal) is:

$$\delta_l = \frac{\partial \text{Loss}}{\partial \mathbf{z}_l} = \delta_{l+1} \cdot \mathbf{W}_{l+1}^T \odot f'(z_l)$$

   $\mathbf{z}_l$ is the weighted input to the activation function, and $f'(z_l)$ is its derivative.

4. **Repeat:** Continue propagating the error back until the first layer is reached.

4.1.2.3 Parameter Update

1. Update the weights and biases using an optimization algorithm (e.g., Gradient Descent):

$$\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta \cdot \frac{\partial \text{Loss}}{\partial \mathbf{W}_l}$$
$$\mathbf{b}_l \leftarrow \mathbf{b}_l - \eta \cdot \frac{\partial \text{Loss}}{\partial \mathbf{b}_l}$$

   where η is the learning rate.

The backpropagation algorithm is built on the chain rule of calculus, which allows computation of gradients through composite functions. For example, if the loss depends on weights indirectly through intermediate variables:

$$\text{Loss} = L(\hat{y}(f(x; \mathbf{W})))$$

The gradient $\frac{\partial \text{Loss}}{\partial \mathbf{W}}$ is computed by chaining derivatives:

$$\frac{\partial \text{Loss}}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial f} \cdot \frac{\partial f}{\partial \mathbf{W}}$$

To efficiently compute gradients for deep networks:

- Use vectorized operations and matrix multiplications (e.g., via libraries like TensorFlow or PyTorch).
- Cache intermediate results from the forward pass (activations, pre-activations) to avoid redundant computation during the backward pass.

As we briefly mentioned, we may face challenges such as:

- **Vanishing/Exploding Gradients:** In deep networks, gradients may become too small (vanish) or too large (explode), especially with certain activation functions like sigmoid.
- **Overfitting:** Training the network to fit the training data too closely may hurt generalization.
- **Computational Cost:** Training can be slow for large networks with millions of parameters.

4.1.2.4 Backpropagation - Example

Consider a feedforward neural network with following structure and initial parameters:

- Input Layer: 2 neurons (x_1, x_2)
- Hidden Layer: 2 neurons (z_{h1}, z_{h2}, a_{h1}, a_{h2})
- Output Layer: 1 neuron (z_o, $\hat{y}$)
- Weights for hidden neuron 1: $W_1 = 0.5$, $W_2 = -0.5$

- Weights for hidden neuron 2: $W_3 = 0.3$, $W_4 = 0.7$
- Weights for output neuron: $W_5 = 0.6$, $W_6 = 0.9$
- Bias terms: $b_1 = 0.1$ (hidden layer), $b_2 = 0.1$ (output layer)
- Learning rate: $\eta = 0.1$

We will use the sigmoid function with a numerically stable implementation for our activation function:

$$\sigma(x) = \begin{cases} \frac{1}{1+e^{-x}} & \text{if } x \geq 0 \\ \frac{e^x}{1+e^x} & \text{if } x < 0 \end{cases} \tag{4.1}$$

Assume our training data is: Sample input: $(x_1 = 1, x_2 = 2, y = 1)$
Step 1: Forward Pass:

1. Calculate weighted sums for hidden layer neurons:

$$\begin{aligned} z_{h1} &= W_1 x_1 + W_2 x_2 + b_1 \\ &= (0.5 \cdot 1) + (-0.5 \cdot 2) + 0.1 \\ &= -0.400000 \end{aligned}$$

$$\begin{aligned} z_{h2} &= W_3 x_1 + W_4 x_2 + b_1 \\ &= (0.3 \cdot 1) + (0.7 \cdot 2) + 0.1 \\ &= 1.800000 \end{aligned}$$

2. Apply activation function:

$$\begin{aligned} a_{h1} &= \sigma(z_{h1}) = \sigma(-0.400000) = 0.401312 \\ a_{h2} &= \sigma(z_{h2}) = \sigma(1.800000) = 0.858149 \end{aligned}$$

Calculate output layer:

1. Calculate weighted sum:

$$\begin{aligned} z_o &= W_5 a_{h1} + W_6 a_{h2} + b_2 \\ &= (0.6 \cdot 0.401312) + (0.9 \cdot 0.858149) + 0.1 \\ &= 1.113121 \end{aligned}$$

2. Apply activation function:

$$\hat{y} = \sigma(z_o) = \sigma(1.113121) = 0.752711$$

3. Calculate loss (Mean Squared Error):

$$\begin{aligned}\text{Loss} &= \frac{1}{2}(y - \hat{y})^2 \\ &= \frac{1}{2}(1 - 0.752711)^2 \\ &= 0.030576\end{aligned}$$

Step 2: Backward Pass:

1. Calculate output layer error and gradients:

$$\begin{aligned}\delta_o &= \frac{\partial \text{Loss}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_o} \\ &= (\hat{y} - y) \cdot \hat{y}(1 - \hat{y}) \\ &= (0.752711 - 1) \cdot 0.752711 \cdot (1 - 0.752711) \\ &= -0.046030\end{aligned}$$

$$\begin{aligned}\frac{\partial \text{Loss}}{\partial W_5} &= \delta_o \cdot a_{h1} = -0.046030 \cdot 0.401312 = -0.018472 \\ \frac{\partial \text{Loss}}{\partial W_6} &= \delta_o \cdot a_{h2} = -0.046030 \cdot 0.858149 = -0.039500 \\ \frac{\partial \text{Loss}}{\partial b_2} &= \delta_o = -0.046030\end{aligned}$$

1. Calculate hidden layer errors and gradients:

$$\begin{aligned}\delta_{h1} &= \delta_o \cdot W_5 \cdot a_{h1}(1 - a_{h1}) \\ &= -0.046030 \cdot 0.6 \cdot 0.401312 \cdot (1 - 0.401312) \\ &= -0.006635 \\ \delta_{h2} &= \delta_o \cdot W_6 \cdot a_{h2}(1 - a_{h2}) \\ &= -0.046030 \cdot 0.9 \cdot 0.858149 \cdot (1 - 0.858149) \\ &= -0.005043\end{aligned}$$

$$\frac{\partial \text{Loss}}{\partial W_1} = \delta_{h1} \cdot x_1 = -0.006635 \cdot 1 = -0.006635$$
$$\frac{\partial \text{Loss}}{\partial W_2} = \delta_{h1} \cdot x_2 = -0.006635 \cdot 2 = -0.013271$$
$$\frac{\partial \text{Loss}}{\partial W_3} = \delta_{h2} \cdot x_1 = -0.005043 \cdot 1 = -0.005043$$
$$\frac{\partial \text{Loss}}{\partial W_4} = \delta_{h2} \cdot x_2 = -0.005043 \cdot 2 = -0.010086$$
$$\frac{\partial \text{Loss}}{\partial b_1} = \delta_{h1} + \delta_{h2} = -0.011678$$

Step 3: Weight Updates Apply gradient descent using learning rate $\eta = 0.1$:

$$W_1^{\text{new}} = W_1 - \eta \cdot \frac{\partial \text{Loss}}{\partial W_1} = 0.5 - 0.1 \cdot (-0.006635) = 0.500664$$
$$W_2^{\text{new}} = W_2 - \eta \cdot \frac{\partial \text{Loss}}{\partial W_2} = -0.5 - 0.1 \cdot (-0.013271) = -0.498673$$
$$W_3^{\text{new}} = W_3 - \eta \cdot \frac{\partial \text{Loss}}{\partial W_3} = 0.3 - 0.1 \cdot (-0.005043) = 0.300504$$
$$W_4^{\text{new}} = W_4 - \eta \cdot \frac{\partial \text{Loss}}{\partial W_4} = 0.7 - 0.1 \cdot (-0.010086) = 0.701009$$
$$W_5^{\text{new}} = W_5 - \eta \cdot \frac{\partial \text{Loss}}{\partial W_5} = 0.6 - 0.1 \cdot (-0.018472) = 0.601847$$
$$W_6^{\text{new}} = W_6 - \eta \cdot \frac{\partial \text{Loss}}{\partial W_6} = 0.9 - 0.1 \cdot (-0.039500) = 0.903950$$

Please note for batch training with N examples, you need to compute gradients for each example and average them:

$$\nabla W_{\text{batch}} = \frac{1}{N} \sum_{i=1}^{N} \nabla W_i$$
$$W_{\text{new}} = W - \eta \cdot \nabla W_{\text{batch}}$$

Please note:

1. **Numerical Stability**: The sigmoid function implementation is numerically stable by using different formulas for positive and negative inputs.

2. **Gradient Flow**: The chain rule is carefully applied to ensure proper gradient flow through the network.

3. **Bias Updates**: Bias terms are updated along with weights using their respective gradients.

4. **Precision**: All calculations maintain sufficient decimal precision to avoid accumulation of rounding errors.

4.2 Advanced Fuzzy Modeling with Neural Networks

Combining fuzzy logic with neural networks creates systems that leverage the advantages of both approaches:

- **Neural Networks:** Excels in learning from data, and generalizes for unseen inputs, but lacks interpretability.
- **Fuzzy Systems:** Provides interpretability and decision-making under uncertainty but cannot learn from data.

Fuzzy-neural integration enables systems to dynamically adjust fuzzy rules and membership functions based on training data, making them adaptable to complex, nonlinear systems, which can better suit applications such as:

- Autonomous driving systems.
- Fault detection and diagnostics.
- Robotics for dynamic task adaptation.

4.2.1 Adaptive Neuro-Fuzzy Inference Systems (ANFIS)

Neuro-fuzzy systems can integrate the interpretability of fuzzy systems with the learning capability of neural networks. A widely used hybrid model is the Adaptive Neuro-Fuzzy Inference System (ANFIS was first proposed by Jyh-Shing Roger Jang in 1994), which combines the strengths of Artificial Neural Networks (ANN) and Fuzzy Inference Systems. ANFIS leverages learning capabilities of neural networks for membership discovery and parameter optimization as well as human-like reasoning and explainability of fuzzy systems.

A simple ANFIS architecture is composed of a **5-layer network** designed to implement a Sugeno-type fuzzy inference system for Fuzzification, Rule Parameter Adaptation, Aggregation, and Defuzzification. It performs:

4.2.1.1 Layer 1: Input Membership Functions

- Each node in this layer represents a membership function (e.g., generalized Bell or Gaussian flavors).
- For instance an input x, the Gaussian membership value is calculated as:

$$\mu(x) = e^{-\frac{(x-c)^2}{2\sigma^2}}$$

Where premise parameters are:

1. Center (c):
   - Represents the peak of the bell curve where membership value = 1
   - Determines the location of the linguistic term on the universe of discourse
   - For example, if "warm" temperature is centered at 25°C, then $c = 25$
2. Standard Deviation (σ):
   - Controls the width/spread of the bell curve
   - Affects how quickly membership values decrease from the center
   - Larger σ creates wider, more gradual transitions
   - Smaller σ creates narrower, steeper transitions

Let's consider a practical temperature control example with three sets:

Cold:

$$\mu_{cold}(x) = e^{-\frac{(x-15)^2}{2(5)^2}}$$

Warm:

$$\mu_{warm}(x) = e^{-\frac{(x-25)^2}{2(5)^2}}$$

Hot:

$$\mu_{hot}(x) = e^{-\frac{(x-35)^2}{2(5)^2}}$$

Here:

- Centers (c) are at 15°C, 25°C, and 35°C respectively
- Standard deviation ($\sigma = 5$) creates appropriate overlap between sets
- At any input temperature, sum of all memberships is approximately 1

4.2.1.2 Layer 2: Rule Strength Calculation

- Each node calculates the **firing strength** of a fuzzy rule using the fuzzy AND operation (e.g., multiplication):

$$O_{L2,i} = w_i = \mu_{A_i}(x) \cdot \mu_{B_i}(y)$$

4.2.1.3 Layer 3: Normalization

- This layer normalizes the rule strengths. For instance we can do:

$$O_{L3,i} = \bar{w}_i = \frac{w_i}{\sum_i w_i}$$

4.2.1.4 Layer 4: Consequent Layer

- Each node calculates the output of a fuzzy rule:

$$f_i = p_i x + q_i y + r_i$$

$$O_{L4,i} = \bar{w}_i \cdot f_i$$

- Parameters p_i, q_i, and r_i are **consequent parameters** and are regularly updated as part of our ANN.

4.2.1.5 Layer 5: Summation

- This single node computes the final output by summing all rule contributions:

$$O_{L5} = \sum_i O_{L4,i} = \sum_i \bar{w}_i \cdot f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

Please note ANFIS uses a **hybrid learning algorithm**. For instance, a given ANFIS implementation may combine below steps as part of the ANN training:

1. **Least Squares Estimation:** For optimizing consequent parameters in the forward pass where consequent parameters (p_i, q_i, r_i) are updated while premise parameters (c_i, σ_i) are fixed.
2. **Gradient Descent:** For updating premise parameters in the backward pass, while premise parameters are updated, while consequent parameters are fixed

4.2.2 ANFIS Applications, Advantages, and Challenges

ANFIS is widely used in various fields, such as:

- **Universal Approximator**

- **Predictive Modeling**
- **Controller Design**
- **Pattern Recognition**
- **Data Mining**
- **Decision Making**
- **Advantages:**
  - Combines interpretability of fuzzy systems with learning capabilities of neural networks.
  - Highly efficient for **small-scale problems** with fewer inputs.
  - Adaptable to dynamic systems.
- **Challenges:**
  - **Curse of Dimensionality:** The number of fuzzy rules increases exponentially with the number of inputs and membership functions.
  - Computationally expensive (training) for systems with many inputs or rules.

Currently, there are active attempts to utilize Transformer-based ANFIS as an advanced hybrid approach that combines the deep learning capabilities of Transformers with the interpretability of ANFIS to fight the struggles with scalability for high-dimensional data and the manual design of membership functions. Transformer-ANFIS leverages the attention mechanisms in Transformers to dynamically learn membership functions and extract complex relationships between inputs. The result is a system that adapts to data complexity, making it a powerful universal estimator for highly nonlinear systems.

In this framework, a Transformer encoder replaces the static membership functions of traditional ANFIS, learning fuzzy memberships dynamically through contextual embeddings. The output of the Transformer serves as input to fuzzy rules, which are evaluated and aggregated using techniques like Sugeno Weighted Mean for defuzzification. This end-to-end trainable system not only maintains ANFIS's interpretability through attention weights and fuzzy rules but also enhances scalability and performance by efficiently handling high-dimensional and large-scale datasets.

Transformer-ANFIS has potential applications in very complex control systems, function approximation, where interpretability and robustness are crucial. It retains the universal approximation capabilities of ANFIS while leveraging Transformers' strength in pattern recognition and adaptability. Despite challenges like computational complexity, Transformer-ANFIS has the potential of a promising pathway for bridging the gap between deep learning and interpretable fuzzy logic systems.

4.2.3 ANFIS - Example

The aim of this example is to approximate and understand the behavior of a highly nonlinear system by utilizing an ANFIS integrated with a Sugeno-type FIS. This approach facilitates the automatic generation of membership functions, derivation of rules, and prediction of outcomes, effectively functioning as a universal approximator for complex functions. In practical applications, the exact underlying function or system is often unknown. However, for training purposes, we use explicitly defined ground truth function described below. This allows us to compare and verify the results of our ANFIS approximations against the actual function, as we have explicit knowledge of its definition (a luxury that we may not have in real-life scenarios). Figures 4.1 and 4.2 show the 3D surface of the ground truth function and the corresponding approximate predictions.

$$AssumedGTF : z = \sin(x)\cos(y) + x^2 - y^2,$$

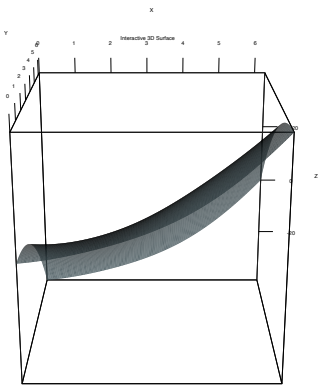

Figure 4.1: A - Ground Truth 3D Surface

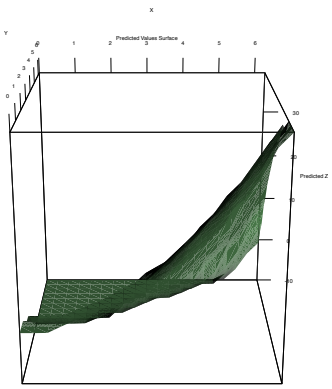

Figure 4.2: B - Predicted 3D Surface

4.2.3.1 Dataset Generation

We generate synthetic data for training and testing:

- **Training Data**: A 100x100 grid of points over the range $[0, 2\pi] \times [0, 2\pi]$.
- **Test Data**: A 20x20 grid of points for model evaluation.

4.2.3.2 Fuzzy Inference System

The Fuzzy Inference System is built using the `R Language Fuzzy Rule-Based Systems (FRBS)` package:

- **Sugeno-Type FIS**: Outputs are crisp values, suitable for our regression.
- **Membership Functions**: Automatically learned by ANFIS, with 20 membership functions per input variable.
- **Defuzzification**: Weighted Mean (WM) is used to compute crisp outputs.

We also present **Error Metrics** such as Mean Squared Error (MSE) that quantifies model performance as well as visualization heatmaps and 3D surface plots of the actual and predicted data to aid in understanding the model's behavior and accuracy.

4.2.3.3 Example Code

```
# Uncomment the lines below if not installed
install.packages("frbs")
install.packages("devtools")
install.packages("ggplot2")
install.packages("plot3D")
install.packages("rgl")
install.packages("svglite")

library(plot3D)
library(frbs)
library(ggplot2)
library(rgl)
library(svglite)

# Function to export plots from ggplot to SVG and PNG formats
export_ggplot <- function(plot, filename_base) {
        ggsave(paste0(filename_base, ".svg"), plot, width = 10, height = 8)
        ggsave(paste0(filename_base, ".png"), plot, width = 10, height = 8,
        ↪  dpi = 300)
}

# Function to export 3D plots from rgl to SVG and PNG
export_3d_plot <- function(filename_base) {
        rgl.postscript(paste0(filename_base, ".svg"), fmt = "svg")
```



```
        rgl.snapshot(paste0(filename_base, ".png"), fmt = "png")
    }

    # Generate some synthetic data for testing
    # z = sin(x).cos(y) + x^2 - y^2
    generate_complex_data <- function(n = 100) {
        x <- seq(0, 2 * pi, length.out = n)
        y <- seq(0, 2 * pi, length.out = n)
        grid <- expand.grid(x = x, y = y)
        z <- sin(grid$x) * cos(grid$y) + grid$x^2 - grid$y^2
        data <- cbind(grid, z = z)
        return(data)
    }

    # 3D Surface Plot
    x <- seq(0, 2 * pi, length.out = 100)
    y <- seq(0, 2 * pi, length.out = 100)
    grid <- expand.grid(x = x, y = y)
    z <- with(grid, sin(x) * cos(y) + x^2 - y^2)
    z_matrix <- matrix(z, nrow = length(x), ncol = length(y))

    # Static 3D Surface Plot
    png("3D_Surface_Static.png", width = 800, height = 600)
    persp3D(x = x, y = y, z = z_matrix,
    col = "lightblue", theta = 30, phi = 20,
    xlab = "X", ylab = "Y", zlab = "Z",
    main = "3D Surface of z = sin(x)cos(y) + x² - y²")
    dev.off()

    svg("3D_Surface_Static.svg", width = 10, height = 8)
    persp3D(x = x, y = y, z = z_matrix,
    col = "lightblue", theta = 30, phi = 20,
    xlab = "X", ylab = "Y", zlab = "Z",
    main = "3D Surface of z = sin(x)cos(y) + x² - y²")
    dev.off()

    # Interactive 3D Surface Plot
    persp3d(x, y, z_matrix, col = "lightblue", xlab = "X", ylab = "Y", zlab = "Z",
    main = "Interactive 3D Surface: z = sin(x)cos(y) + x^2 - y^2")
    export_3d_plot("3D_Surface_Interactive")

    # Generate training and test datasets
```



```r
train_data <- generate_complex_data(100)
test_data <- generate_complex_data(20)

# Visualize the training data
training_heatmap <- ggplot(train_data, aes(x = x, y = y, fill = z)) +
geom_tile() +
scale_fill_gradient2(low = "blue", mid = "white", high = "red", midpoint = 0) +
labs(title = "Training Data Heatmap", x = "X", y = "Y", fill = "Z Value")
print(training_heatmap)
export_ggplot(training_heatmap, "Training_Data_Heatmap")

# Define the fuzzy inference system
range_data <- matrix(c(0, 2 * pi, 0, 2 * pi, -10, 40), nrow = 2)
data_train <- as.matrix(train_data)

frbs_model <- frbs.learn(
data.train = data_train,
method.type = "WM", # Weighted Mean for defuzzification
range.data = range_data,
control = list(
num.labels = 20,
type = "sugeno"
)
)

# Predict on test data
test_inputs <- as.matrix(test_data[, 1:2])
predictions <- predict(frbs_model, newdata = test_inputs)

# Evaluate the model
actual <- test_data[, 3]
mse <- mean((actual - predictions)^2)
cat("Mean Squared Error (MSE):", mse, "\n")

# Predicted vs Actual Scatter Plot
png("Predicted_vs_Actual.png", width = 800, height = 600)
plot(actual, predictions,
main = "Fuzzy System: Actual vs. Predicted (Complex Data)",
xlab = "Actual Values",
ylab = "Predicted Values",
col = "blue", pch = 19)
abline(0, 1, col = "red", lwd = 2)
```



```
dev.off()

svg("Predicted_vs_Actual.svg", width = 10, height = 8)
plot(actual, predictions,
main = "Fuzzy System: Actual vs. Predicted (Complex Data)",
xlab = "Actual Values",
ylab = "Predicted Values",
col = "blue", pch = 19)
abline(0, 1, col = "red", lwd = 2)
dev.off()

# Visualize the test data
test_heatmap <- ggplot(test_data, aes(x = x, y = y, fill = z)) +
geom_tile() +
scale_fill_gradient2(low = "blue", mid = "white", high = "red", midpoint = 0) +
labs(title = "Test Data Heatmap", x = "X", y = "Y", fill = "Z Value")
print(test_heatmap)
export_ggplot(test_heatmap, "Test_Data_Heatmap")

# Interactive 3D plot of predicted values
predicted_matrix <- matrix(predictions, nrow = length(unique(test_data[, 1])),
ncol = length(unique(test_data[, 2])))
persp3d(
x = unique(test_data[, 1]),
y = unique(test_data[, 2]),
z = predicted_matrix,
col = "lightgreen",
xlab = "X",
ylab = "Y",
zlab = "Predicted Z",
main = "Interactive 3D Surface of Predicted Values"
)
export_3d_plot("Predicted_3D_Surface")
```

