Practical_3

Fuzzy Logic is a mathematical system that deals with reasoning that is **approximate rather than fixed and exact**, unlike traditional Boolean logic which only works with **true (1)** or **false (0)**.

In fuzzy logic, truth values range **between 0 and 1**.
This allows it to model **real-world situations** where things are **not just black or white**, but **somewhere in between**.

📊 **Example**

Let's say you're designing a system to determine if a room is "hot".

- **Boolean logic**:
  - If temperature > 30°C, then it's hot → 1 (true)
  - Else → 0 (false)
- **Fuzzy logic**:
  - If temperature = 25°C → hot = 0.4
  - If temperature = 30°C → hot = 0.7
  - If temperature = 35°C → hot = 0.9
    (Here, "hot" is not yes/no, but a degree of truth.)

What is a fuzzy set?	A fuzzy set allows partial membership, meaning elements can belong to the set with degrees between 0 and 1.
What is the union of two fuzzy sets?	It is the maximum of the membership values for each element: µAUB(x) = max(µA(x), µB(x)).
What is the intersection of two fuzzy sets?	It is the minimum of the membership values: µA∩B(x) = min(µA(x), µB(x)).
How is the complement of a fuzzy set calculated?	By subtracting each membership value from 1: µA'(x) = 1 - µA(x).
How is the difference of two fuzzy sets calculated?	µA-B(x) = min(µA(x), 1 - µB(x))
What is the Cartesian product in fuzzy sets?	It generates a fuzzy relation with membership values: µ((a, b)) = min(µA(a), µB(b)).
What is a fuzzy relation?	A fuzzy relation is a set of ordered pairs with associated degrees of relation (between 0 and 1).

What is max-min composition?	It's used to compose two fuzzy relations: $\mu R \circ S(x, z) = \max(\min(\mu R(x, y), \mu S(y, z)))$ over all y.
Where is fuzzy logic applied?	It's used in control systems like ACs, washing machines, robotics, AI, and decision-making systems.
Why do we use fuzzy logic instead of classical logic?	Fuzzy logic allows handling of uncertainty and partial truths, unlike classical logic which is binary.
What is a fuzzy set?	A fuzzy set allows partial membership, meaning elements can belong to the set with degrees between 0 and 1.
What is the union of two fuzzy sets?	It is the maximum of the membership values for each element: $\mu A U B(x) = \max(\mu A(x), \mu B(x))$.

$\mu A\text{-}B(x)$ is equal to $\mu B\text{-}A(x)$

NO,Because the **difference operation in fuzzy sets is not symmetric**. It depends on the specific membership values in **A** and **B**.

in fuzzy set theory, the **Cartesian product** of two fuzzy sets A and B results in a new fuzzy relation that contains all possible ordered pairs between the elements of A and B.

For each pair (a, b), the membership value is calculated using the **minimum** of the individual membership values of 'a' in set A and 'b' in set B.

Mathematically, it is given by:

$\mu A \times B(a,b)=\min(\mu A(a),\mu B(b))$

Max-Min Composition is used to combine two fuzzy relations. If we have two fuzzy relations:

- $R \subseteq X \times Y R$
- $S \subseteq Y \times Z S$

Then their composition $R \circ S$ is a new fuzzy relation from **X to Z**, calculated as:

This means we look at all intermediate elements y in set Y, take the **minimum** of $\mu R(x,y)$ and $\mu S(y,z)$ then take the **maximum** of all these min values."

For each pair (x,z) we:

1. Look at all intermediate elements y in set Y.
2. Calculate $\min(\mu R(x,y),\mu S(y,z))$
3. Take the maximum of these minimums.

Code :

```
for a_key, a_val in A.items():
    for b_key, b_val in B.items():
```

The outer for loop iterates over each element of fuzzy set A. Here, a_key represents the element (e.g., x1, x2), and a_val represents the membership value of that element.

```
X = {x for x, _ in R.keys()}
Y = {y for _, y in R.keys()}
Z = {z for _, z in S.keys()}
```

X: A set of all first elements of the pairs in the relation R (i.e., all x values from the relation R).

Y: A set of all second elements of the pairs in R (i.e., all y values from the relation R).

Z: A set of all second elements of the pairs in S (i.e., all z values from the relation S).

For each y in the set Y, the function retrieves:

- **r_val**: The value of the fuzzy relation R(x, y) (membership value of x and y), using R.get((x, y), 0) to handle cases where the key doesn't exist (it defaults to 0).
- **s_val**: The value of the fuzzy relation S(y, z) (membership value of y and z), similarly using S.get((y, z), 0).

RPC:

Remote Procedure Call (RPC) is a protocol that allows one program (the **client**) to execute a **procedure/function on another computer** (the **server**) as if it were a local function call

A **local function call** is when a function is called **within the same program** or on the **same machine**, and the **execution happens directly in memory**.

Load balancing :

Theory: Load Balancing

Load Balancing is the process of distributing network or application traffic across multiple servers. It helps:

- Prevent server overload
- Improve responsiveness
- Ensure high availability

Genetic

 Parents: These are the current solutions in the population that are selected to create new solutions (children). They represent good solutions based on their fitness scores.

 Children (Offspring): These are new solutions created by combining the parents' characteristics (through crossover) and sometimes introducing randomness (mutation). The goal is to improve upon the parents over successive generations.

 Genetic Algorithm (GA):

- A **Genetic Algorithm** is a search heuristic that mimics the process of natural selection. It's used to find approximate solutions to optimization and search problems.
- **GA components** include:
    - **Population**: A set of candidate solutions to the problem.

    - **Fitness Function**: Measures how well a candidate solution solves the problem.
    - **Selection**: Selects the best candidates (fittest individuals) to form the next generation.
    - **Crossover (Recombination)**: Combines parts of two or more parent solutions to create new solutions.
    - **Mutation**: Introduces small random changes to a solution to maintain diversity in the population.
- The goal of the GA is to **optimize** parameters or find the **best solution** to a problem, based on predefined criteria or constraints.

⍰ **Neural Network (NN):**

- A **Neural Network** is a machine learning model inspired by the human brain's neural structure. It's made of layers of interconnected nodes (neurons), which learn to approximate complex functions or patterns from data.
- Neural networks are widely used for regression, classification, and time-series prediction tasks.
- In the context of **spray drying**, an NN might be used to model the relationship between process parameters (e.g., temperature, airflow, spray speed) and output quality (e.g., moisture content, particle size).

⍰ **Hybrid GA-NN Model:**

- A **Hybrid GA-NN** model combines the strengths of **Genetic Algorithms** (for global optimization) and **Neural Networks** (for prediction). This approach is useful when solving complex problems that involve:
    - Finding the optimal parameters (using GA).
    - Modeling the system or process (using NN).
- **Application to Spray Drying of Coconut Milk**: In this specific application, the goal would be to optimize the parameters of the spray drying process, such as the drying air temperature, spray flow rate, and nozzle size, using a **Genetic Algorithm**. Then, a **Neural Network** would model the spray drying process and predict the final product quality based on those parameters.

⍰ **Optimization of GA Parameters**:

- The optimization refers to finding the best parameters for the **Genetic Algorithm** itself (like population size, mutation rate, crossover rate, etc.) that allow it to more efficiently find the optimal process parameters.

- **Optimized GA Parameters**: This could mean adjusting the mutation and crossover rates to get faster convergence to an optimal solution in a shorter time

Purpose of Mutation: Mutation helps maintain **diversity** in the population, preventing the algorithm from converging too early on suboptimal solutions. It ensures that the GA doesn't get "stuck" in a local optimum and continues exploring the solution space.

 Selection:

- From the population, two solutions are selected as **parents** based on their fitness. These solutions might be good at predicting spray drying performance based on the input parameters.

 Crossover:

- The selected parents undergo **crossover**, where their solutions are combined. A new solution (the **child**) is created, inheriting features from both parents.

 Mutation:

- The child is then subjected to **mutation**, where small, random changes are made to its solution to explore new possibilities.

 New Generation:

- The new child solution is added to the population, and this process is repeated for multiple generations.

Clonal

The **Clonal Selection Algorithm** is inspired by the biological immune system. It mimics how the body fights antigens by selecting the most effective antibodies and cloning them, with small mutations.

In optimization:

- Each **antibody** = a potential solution.
- **Fitness** = how good the solution is (the lower, the better in minimization problems).
- Best antibodies are cloned and mutated to search for better solutions.
- Diversity is preserved by adding new random antibodies

Ant Colony Optimization is a **probabilistic optimization technique** inspired by the **foraging behavior of real ants**.

In nature, ants:

- Wander randomly in search of food.
- Upon finding food, return to the colony while laying **pheromone trails**.
- Other ants follow these pheromone trails.
- Shorter paths receive more pheromones faster, so more ants follow them.
- Over time, the **shortest path** gets reinforced and becomes the dominant route.

This behavior is simulated in ACO to solve complex problems like TSP.

🧭 Traveling Salesman Problem (TSP):

"A salesman must visit **N cities exactly once** and return to the starting city. The goal is to find the **shortest possible route** that satisfies this."

TSP is a **combinatorial optimization problem**, meaning that as the number of cities increases, the number of possible routes increases **factorially** ((n-1)!/2).

ASI

What is Artificial Immune System (AIS)?

An **Artificial Immune System (AIS)** is a computational system inspired by the principles and processes of the **biological immune system**. It is used to solve problems in areas like **pattern recognition**, **anomaly detection**, **classification**, and **optimization**.

🏗 Structural Damage Classification – Overview

Structural Health Monitoring (SHM) is the process of detecting and diagnosing damage in structures such as bridges, buildings, and machinery.

The goal of **structural damage classification** is to detect **whether a structure is damaged**, **where it is damaged**, and **how severe the damage is**, based on **input features** (like vibration, frequency, strain data).

🧠 Why Use AIS for Damage Classification?

The biological immune system is:

- Capable of **learning** (adapting to new pathogens).
- Able to **recognize self vs. non-self** (normal vs. abnormal).
- Efficient in **memory and pattern matching**.

This makes AIS **ideal for detecting abnormal patterns** (e.g., damage) in complex systems, including engineering structures.

🧪 Key Concepts of AIS Applied to Classification:

AIS Term	Structural Damage Analogy
Antigen	New input signal (e.g., sensor data) from the structure
Antibody	Trained classifier (or memory detector) representing known patterns (damaged/healthy)
Affinity	Similarity between input and known pattern
Cloning	Copying best-matching antibodies for learning
Mutation	Slightly altering antibodies to improve performance
Selection	Keeping only the most effective antibodie

📚 Theory: Implementing DEAP (Distributed Evolutionary Algorithms in Python)

🔍 What is DEAP?

DEAP (Distributed Evolutionary Algorithms in Python) is an **open-source framework** designed to help you **quickly implement evolutionary algorithms**, such as:

- Genetic Algorithms (GA)
- Genetic Programming (GP)
- Evolution Strategies
- Particle Swarm Optimization (PSO)
- and more...

It is **modular, flexible, and powerful**, allowing researchers and developers to **focus on the algorithm logic** rather than reinventing components like selection, crossover, or mutation.

🧬 Why use DEAP?

- ✅ Simplifies implementation of bio-inspired algorithms.
- ✅ Supports parallel/distributed computation (great for large problems).
- ✅ Clean and well-structured API using Pythonic objects like Toolbox, Creator, and Algorithms.
- ✅ Includes tools for statistics, logging, and visualization.

⚙️ Core Concepts in DEAP

DEAP Component	Description
creator	Defines custom classes (e.g., Fitness, Individual) with desired properties.
base.Toolbox	Toolbox for registering operators: create individuals, evaluate fitness, mate, mutate, etc.
algorithms	Contains standard algorithms like eaSimple, eaMuPlusLambda to run evolution.
tools	Utility functions: selection, crossover, mutation, statistics collection.