PRACTICAL IMPLEMENTATION OF A FUZZY INFERENCE SYSTEM: EXAMPLES

IMPLEMENTATION OF FUZZY INFERENCE

The following issues have to be addressed:

1. Definitions
   ► Linguistic variables
   ► Fuzzy sets as linguistic terms
   ► Rules (representation and storage)
   ► Inference and defuzzification methods

* Fuzzy Systems Toolbox, M. Beale and H. Demuth
IMPLEMENTATION OF FUZZY INFERENC

The following issues have to be addressed

2. Operation
   ▶ Fuzzification of crisp inputs
   ▶ Evaluation of the rule base
   ▶ Defuzzification

FUZZY TOOLBOX EXAMPLE #1

Features:

- MATLAB as programming environment
- Discrete representation of fuzzy sets as vectors
- Table-based specification of the rule base
- Access to LVs and MFs through integer indices

* Fuzzy Systems Toolbox, M. Beale and H. Demuth
Example: Furnace Temperature Control

Goal: Design the fuzzy toolbox that can be used to operate a fuzzy system for furnace temperature control.

- **Inputs**
  - Temperature reading from sensor
  - Furnace Setting
- **Output**
  - Power control to motor

Fuzzy Set: Vector Representation

**Problem 1:** How to represent fuzzy sets in MATLAB?

Two vectors can represent fuzzy discrete sets or fuzzy continuous sets

- Support Vector (universe vector)
- Grade Vector (membership vector)
Example: Vector Representation

Define the concept of “tall” over heights from 5 to 7 feet, using MATLAB

- \( S = [5.00 ~ 5.50 ~ 6.00 ~ 6.50 ~ 7.00] \);
- \( G = [0.00 ~ 0.25 ~ 0.50 ~ 0.75 ~ 1.00] \);

* Fuzzy Systems Toolbox, M. Beale and H. Demuth

DEFINITION OF LINGUISTIC VARIABLES

- \( \text{tempS} = 32:0.5:112; \)
- \( \text{tempG} = \text{triangle}([\text{tempS}, [32 32 52; 32 52 72; 62 72 92; 92 112 112]]; \)

Here is a plot of the universe of discourse for temperature:

- \( \text{tempG} \) – a grade matrix 5x160 (or 160x5), containing definition of all linguistic terms (membership functions) of the linguistic variable “temperature”

Each column (or row) in the matrix contains a discrete representation of a fuzzy set.

* Fuzzy Systems Toolbox, M. Beale and H. Demuth
DEFINITION OF LINGUISTIC VARIABLES

Definition of linguistic variable “Setting”. Note the use of triangle() function to specify triangular fuzzy sets. Similarly, you could come up with trapezoid(), bell(), etc.

* Fuzzy Systems Toolbox, M. Beale and H. Demuth

DEFINITION OF LINGUISTIC VARIABLES

Definition of the output linguistic variable “Power”. MATLAB allocates memory for the grade matrices automatically, in other languages you may have to take care of memory allocations.

* Fuzzy Systems Toolbox, M. Beale and H. Demuth
### IF-THEN rules

**Fuzzy Rules for Furnace control**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cold</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Cool</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Warm</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Hot</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

*Fuzzy Systems Toolbox, M. Beale and H. Demuth*

### Antecedent Table

The antecedent table has 15 rows to represent each unique pair of input conditions. Each row contains a different combination of indices for the two input variables:

\[
A = \begin{bmatrix}
1 & 1 \\
2 & 1 \\
3 & 1 \\
4 & 1 \\
5 & 1 \\
1 & 2 \\
\ldots \text{etc.} \\
3 & 3 \\
4 & 3 \\
5 & 3 \\
\end{bmatrix}
\]

For example, row 3 represents the case where the first variable, temperature, is in its third fuzzy set, “moderate,” and the second variable, power, is in its first fuzzy set “low.” Thus, this row represents the antecedent part of the third rule: *If temperature is moderate and power is low, then...*  

*Fuzzy Systems Toolbox, M. Beale and H. Demuth*
Antecedent Table

In MATLAB:

\[ A = \text{table}(1:5,1:3); \]

- Table generates matrix represents a table of all possible combinations

1 is the index to the first row (or column) of the 160x5 (5x160) matrix containing definitions of MFs for the linguistic variable "Temperature"

* Fuzzy Systems Toolbox, M. Beale and H. Demuth
Consequence Matrix

C = [1 1 1 1 1 2 2 1 1 1 3 3 1 1 1]';

Each element in the column indicates which fuzzy set the output, the fan setting, should fall into, given that its associated input conditions are true. Thus, the consequence for the third rule can be found by looking at the third element. The third element is 1, so the consequence for the third rule

"If temperature is moderate and power is low, then..."

is:

"...set the fan low."

Combining row 3 of the antecedent and consequence tables results in the rule:

"If temperature is moderate and power is low, then set the fan low."

* Fuzzy Systems Toolbox, M. Beale and H. Demuth

Evaluating Rules

The function frule can be used to evaluate a fuzzy system’s output for any input. In the following paragraphs, we will apply frule to the temperature/power/settings system discussed above.

First, we will use the function group to arrange the supports and grades for each input (temperature, power) and output (fan setting) in pairs.

Z = group(tempS,tempG,powerS,powerG,settingS,settingG);

Then we will define the current input vector. For example, if temperature is 70 (moderate) and the power is 2 (low) the input vector x is:

x = [70; 2];

We can now apply frule to this input. (This action tests the third rule of the antecedent consequence table.)

y = frule(A,C,Z,x)

y =

2.333

* Fuzzy Systems Toolbox, M. Beale and H. Demuth
Evaluating Rules

\[ y = \text{FRULE}(A, C, Z, x) \]

**A** - Antecedent table
**C** - Consequence matrix
**Z** – grouped definitions of the LVs
**x** – crisp input
**y** – defuzzified result

In this particular implementation it is not possible to specify the inference and defuzzification procedure to use. You could:
- have \text{FRULEM} and \text{FRULEL} for Mamdani and Larsen inference
- return a vector \( Y \) instead of a scalar value \( y \). Write several different defuzzy(\( Y \)) functions

---

* Fuzzy Systems Toolbox, M. Beale and H. Demuth

---

**An example of Mamdani inference in MATLAB:**

**Rule:** if \( x \) is \( A \) then \( y \) is \( B \)

Function call \( C = \text{ifthen}_\text{min}(A, S, x, B, G) \)

- \( A \) = support vector of \( A \);
- \( S \) = grade vector of \( A \);
- \( B \) = grade vector of \( B \);
- \( x \) = Crisp input \( x \);
- \( C \) = inferred fuzzy set

\[
\text{ifthen}_\text{min}(A, S, x, B, G) \\
\quad [BG\_row, BG\_column] = \text{size}(BG); \\
\quad C = \\
\quad \quad \text{min}(|\text{fuzzify}(A, S, x)|*\text{ones}(1, BG\_column), BG);
\]

---
Evaluating Rules

Rule: if x is A then y is B

Function call m=(fuzzify(AS,AG,x) computes the membership of x in A

- AS = support vector of A;
- AG = grade vector of A;
- x = Crisp input x;
- m = value $\mu_A(x)$

Evaluating Rules

An example of Larsen inference in MATLAB:

Rule: if x is A then y is B

Function call C = ifthen_prod(AS,AG,x,BG)

- AS = support vector of A;
- AG = grade vector of A;
- BG = grade vector of B;
- x = Crisp input x;
- C = inferred fuzzy set

ifthen_min(AS,AG,x,BG)

C = fuzzify(AS,AG,x) * BG;
ALTERNATIVE TABLE REPRESENTATION

Rules in a fuzzy system can be represented by tables of a different kind.

if temperature is hot and humidity is moderate then fan is high

<table>
<thead>
<tr>
<th>Linguistic Variable</th>
<th>Linguistic Variable</th>
<th>Linguistic Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>“temperature”</td>
<td>“humidity”</td>
<td>“fan”</td>
</tr>
<tr>
<td>&quot;cold&quot;</td>
<td>&quot;normal&quot;</td>
<td>&quot;hot&quot;</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FUZZY TOOLBOX EXAMPLE #2

Features:

- Java as programming environment
- Continuous representation of fuzzy sets as trapezoids
- Text string specification of the rule base
- Access to LVs and MFs by linguistic labels (text strings)
Steps to consider:

Fuzzy view:
► Definition of membership functions
► Definition of linguistic variables
► Definition of rules
► Evaluation of rules
► Defuzzification & obtaining results

Steps to consider:

Programming view:
► Implementing OOP approach
► Data representation
► Referring to membership functions
Most membership functions can be represented by a trapezoid:

- An object to represent a member function is needed.
- Each object should contain one trapezoid defined by 4 points.
- Each object should be able to perform a fuzzification operation.
- Reference to an object has to be made by a symbolic name.
Definition of membership functions:

```java
public class MembershipFunction {
    public MembershipFunction(String name_in, double[] range_in) { //Constructor
        // Constructor implementation
    }

    public double fuzzify(double X) { //Fuzzifier
        // Fuzzifier implementation
    }

    public double[] plot(double from, double to, int size) { // Return an array with discrete representation of a membership function
        // Plot implementation
    }
}
```

Definition of Linguistic Variables:

1. A linguistic variable can have several associated membership functions (for each linguistic term, such as temperature={hot,cold, normal}).
2. Those membership functions should be associated with a single linguistic variable object.
3. An LV object should be addressed by its symbolic name.
4. An LV object should provide an opportunity to add/remove membership functions / linguistic terms.
5. An LV variable (object) should store results of rules evaluation and perform defuzzification.
```
public class LinguisticVariable {
    public LinguisticVariable(String name) //Constructor
    public void add(String name, double start, double left_top, double right_top, double end) // Add a membership function
    public double defuzzify() throws NoRulesFiredException //Defuzzify
    public String getLabel() //Get name of the LV
    public MembershipFunction getMembershipFunctionByName(String name)
    public double is(String name) //Fuzzification
    public void reset() //Reset fired rules
    public void set(String name, double value) //Store results of rule evaluation
    public void setInputValue(double value) //Set input value
}
```

### Definition of Linguistic Variables:

Working of a fuzzy inference engine is specified by rules. There are two major approaches: table and symbolic definition of rules.

- Table definition is more convenient for machines, but is bad for humans. A table may have to be fully defined with a huge number of entries.
- Symbolic definition is much better for humans, it generates less errors in knowledge coding. Creates overhead for rule parsing.
Evaluation of Rules:

1. Symbolic rules are evaluated during parsing process.
2. Parsing requires identifying a set of reserved words: if, then, is, not, and, or, (, )," ". Each word carries an associated action.
   - if – starts a rule definition
   - then – separates left and right parts of a rule
   - is – performs fuzzification
   - not – fuzzy ‘not’
   - and – fuzzy ‘and’
   - or – fuzzy ‘or’
   - (,) – nested expressions
   - " " – white space, separator

Evaluation of Rules:

3. Each linguistic variable object should be able to accumulate results of fuzzy evaluation and recall them during defuzzification.
4. Parser should have an idea where is a reserved word, a name reference to a linguistic variable, a name reference to a membership function from certain linguistic variable, i.e. some kind of “language definition”
5. Parser should be able to find and identify errors
Evaluation of Rules:

```java
public class FuzzyEngine {
    public FuzzyEngine() { //Constructor
        public String evaluateRule(String rule) throws RulesParsingException { //Interface for the parser
            private double parseExpression(java.util.StringTokenizer tokens) throws RulesParsingException { //Recursive Parser
                public void register(LinguisticVariable function) { //Add LV to the list of known LV
                    public void reset() { //Reset all fired rules
                }
```

Defuzzification:

1. Defuzzification is performed by a linguistic variable object
2. Defuzzification is performed based on the data accumulated during rule evaluation
3. Accumulated rule evaluation results should be deleted on engine reset
4. If no rules have been fired for a certain linguistic variable, defuzzification should return an error
Running the Fuzzy Engine:

1. Define linguistic variables and membership functions
2. Create an empty fuzzy engine
3. Register linguistic variables with fuzzy engine
4. Reset fuzzy engine
5. Set input values
6. Evaluate a group of rules
7. Invoke defuzzification procedure for an output linguistic variable object and obtain results
8. For the next evaluation, repeat from step 4

An Example

Evaluate alignment of two peaks:
An Example

// STEP 1 – Define membership functions
LinguisticVariable left_edge_distance = new LinguisticVariable("left_edge_distance");
left_edge_distance.add("close",0,0,1,3);
left_edge_distance.add("far",1,3,10,10);

LinguisticVariable right_edge_distance = new LinguisticVariable("right_edge_distance");
right_edge_distance.add("close",0,0,1,3);
right_edge_distance.add("far",1,3,10,10);

LinguisticVariable peak_align = new LinguisticVariable("peak_align");
peak_align.add("aligned",0.3,0.7,1,1);
peak_align.add("not_aligned",0,0,0.3,0.7);

// STEP 2 – Create fuzzy engine
fuzzy.FuzzyEngine engine = new fuzzy.FuzzyEngine();

// STEP 3 – Register Linguistic Variables
engine.register(left_edge_distance);
engine.register(right_edge_distance);
engine.register(peak_align);

// STEP 4 – Reset fuzzy engine
engine.reset();

// STEP 5 – Set input variables
left_edge_distance.setInputValue(dXleft));
right_edge_distance.setInputValue(dXright));
// STEP 6 – Evaluate rules
engine.evaluateRule("if left_edge_distance is close and right_edge_distance is close then peak_align is aligned");

engine.evaluateRule("if left_edge_distance is far or right_edge_distance is far then peak_align is not_aligned");

// STEP 7 – Defuzzify and get results
peak_alignment=peak_align.defuzzify();

An Example

Fuzzy engine diagnostic output:
Mode 1: compare peak at: 15 - 18 with peak in Mode 2 at: 15 - 18
Alignment: 0.73
Confidence: 0.95
Sensitivity: 0.95

Fired: if peak_align is aligned(1.0) and (confidence is high(0.0) or confidence is very_high(1.0)) then damage Probability changed by: 0.96
SUMMARY

1. The software design should account for representation, storage and coding issues.

2. Many different ways to program a fuzzy inference engine.

3. Should account for trade-off between programming and ease-of-use.