Fuzzy Logic Quick Reference Cheatsheet

Core Concepts

Fuzzy Set

A set where elements have degrees of membership between 0 and 1

- **Crisp Set**: {0, 1} - binary membership
- **Fuzzy Set**: [0, 1] - partial membership

Membership Function μ(x)

Maps elements to membership degrees: μ: X → [0, 1]

Python (scikit-fuzzy) Quick Reference

Installation

```
pip install scikit-fuzzy numpy matplotlib
```

Basic Membership Functions

```
import numpy as np
import skfuzzy as fuzz

x = np.arange(0, 11, 1)

# Triangular: trimf(x, [a, b, c])
triangular = fuzz.trimf(x, [0, 5, 10])

# Trapezoidal: trapmf(x, [a, b, c, d])
trapezoidal = fuzz.trapmf(x, [0, 2, 8, 10])

# Gaussian: gaussmf(x, mean, sigma)
gaussian = fuzz.gaussmf(x, 5, 1.5)

# Sigmoid: sigmf(x, b, c)
sigmoid = fuzz.sigmf(x, 5, 1)

# Generalized Bell: gbellmf(x, a, b, c)
gbell = fuzz.gbellmf(x, 2, 4, 5)
```

Fuzzy Operations

```
# Union (OR) - Maximum
union = fuzz.fuzzy_or(x, mf1, x, mf2)[1]

# Intersection (AND) - Minimum
intersection = fuzz.fuzzy_and(x, mf1, x, mf2)[1]

# Complement (NOT)
complement = fuzz.fuzzy_not(mf1)
```

Defuzzification Methods

```
# Centroid (Center of Area)
crisp = fuzz.defuzz(x, mf, 'centroid')

# Bisector (divides area in half)
crisp = fuzz.defuzz(x, mf, 'bisector')

# Mean of Maximum
crisp = fuzz.defuzz(x, mf, 'mom')

# Smallest of Maximum
crisp = fuzz.defuzz(x, mf, 'som')

# Largest of Maximum
crisp = fuzz.defuzz(x, mf, 'lom')
```

Fuzzy Inference System

```
from skfuzzy import control as ctrl

# Define variables
temperature = ctrl.Antecedent(np.arange(0, 41, 1), 'temperature')
power = ctrl.Consequent(np.arange(0, 101, 1), 'power')

# Auto-generate membership functions
temperature.automf(3)  # Creates poor, average, good

# Or define custom membership functions
temperature['cold'] = fuzz.trimf(temperature.universe, [0, 0, 20])
temperature['warm'] = fuzz.trimf(temperature.universe, [15, 25, 35])
temperature['hot'] = fuzz.trimf(temperature.universe, [30, 40, 40])

power['low'] = fuzz.trimf(power.universe, [0, 0, 50])
power['high'] = fuzz.trimf(power.universe, [50, 100, 100])

# Define rules
rule1 = ctrl.Rule(temperature['cold'], power['high'])
rule2 = ctrl.Rule(temperature['hot'], power['low'])
```

```
# Create control system
control_system = ctrl.ControlSystem([rule1, rule2])
controller = ctrl.ControlSystemSimulation(control_system)

# Use the system
controller.input['temperature'] = 15
controller.compute()
output = controller.output['power']

# Visualize
temperature.view()
power.view()
```

Get Membership Value

```
# Interpolate membership at specific point
membership_value = fuzz.interp_membership(x, mf, 5.5)
```

Rust Quick Reference

Setup in Cargo.toml

```
[dependencies]
num-traits = "0.2"
```

Basic Membership Functions

```
use fuzzy_logic_study::membership::MembershipFunction;

// Triangular
let triangular = MembershipFunction::Triangular {
    a: 0.0,
    b: 5.0,
    c: 10.0,
};

// Trapezoidal
let trapezoidal = MembershipFunction::Trapezoidal {
    a: 0.0,
    b: 2.0,
    c: 8.0,
    d: 10.0,
};
```

```
// Gaussian
let gaussian = MembershipFunction::Gaussian {
    mean: 5.0,
    sigma: 1.5,
};

// Sigmoid
let sigmoid = MembershipFunction::Sigmoid {
    a: 1.0,
    c: 5.0,
};

// Evaluate
let membership = triangular.evaluate(5.0);  // Returns 1.0
```

Fuzzy Sets

```
use fuzzy_logic_study::FuzzySet;

// Create fuzzy set
let cold = FuzzySet::new(
    "Cold",
    Box::new(move |x| triangular.evaluate(x))
);

// Get membership
let degree = cold.membership(5.0);

// Get multiple memberships
let x_values = vec![0.0, 2.5, 5.0, 7.5, 10.0];
let memberships = cold.membership_vec(&x_values);
```

Fuzzy Operations

```
use fuzzy_logic_study::operations::{union, intersection, complement};

// Union (OR)
let union_set = union(&set1, &set2, "Set1 OR Set2");

// Intersection (AND)
let intersection_set = intersection(&set1, &set2, "Set1 AND Set2");

// Complement (NOT)
let not_set = complement(&set1, "NOT Set1");
```

T-norms and S-norms

```rust
use fuzzy_logic_study::operations::{tnorms, snorms};

let a = 0.7;
let b = 0.3;

// T-norms (AND operations)
let min = tnorms::minimum(a, b);              // 0.3
let product = tnorms::algebraic_product(a, b); // 0.21
let bounded = tnorms::bounded_difference(a, b); // 0.0

// S-norms (OR operations)
let max = snorms::maximum(a, b);             // 0.7
let sum = snorms::algebraic_sum(a, b);       // 0.79
let bounded = snorms::bounded_sum(a, b);     // 1.0
```

Linguistic Variables

```rust
use fuzzy_logic_study::LinguisticVariable;

// Create linguistic variable
let mut temperature = LinguisticVariable::new("Temperature", (0.0, 50.0));

// Add fuzzy sets
temperature.add_set(cold);
temperature.add_set(warm);
temperature.add_set(hot);

// Fuzzify a crisp value
let memberships = temperature.fuzzify(22.0);
// Returns: [("Cold", 0.0), ("Warm", 0.6), ("Hot", 0.0)]

// Classify (get max membership)
let classification = temperature.classify(22.0);
// Returns: Some("Warm")
```

Defuzzification

```rust
use fuzzy_logic_study::defuzzification::{
    defuzzify, DefuzzMethod, create_universe
};

// Create universe
let universe = create_universe(0.0, 10.0, 101);

// Defuzzify
let crisp = defuzzify(&fuzzy_set, &universe, DefuzzMethod::Centroid);
let crisp = defuzzify(&fuzzy_set, &universe, DefuzzMethod::MeanOfMaximum);
let crisp = defuzzify(&fuzzy_set, &universe, DefuzzMethod::Bisector);
```

Fuzzy Inference System

```rust
use fuzzy_logic_study::inference::{FuzzyInferenceSystem, FuzzyRule, and, or};
use std::collections::HashMap;

// Create FIS
let mut fis = FuzzyInferenceSystem::new("Temperature Control");

// Add rule: IF temp is cold THEN power is high
let rule = FuzzyRule::new(
    "Rule1",
    Box::new(|inputs| {
        inputs.get("temp_cold").copied().unwrap_or(0.0)
    }),
    "power_high",
);

fis.add_rule(rule);

// Infer
let mut inputs = HashMap::new();
inputs.insert("temp_cold".to_string(), 0.8);
let outputs = fis.infer(&inputs);
```

📊 Common Membership Function Formulas

Triangular

```
        { 0,                  x ≤ a
μ(x) =  { (x-a)/(b-a),        a < x ≤ b
        { (c-x)/(c-b),        b < x ≤ c
        { 0,                  x > c
```

Trapezoidal

```
        { 0,                  x ≤ a
        { (x-a)/(b-a),        a < x ≤ b
μ(x) =  { 1,                  b < x ≤ c
        { (d-x)/(d-c),        c < x ≤ d
        { 0,                  x > d
```

Gaussian

```
μ(x) = exp(-(x-c)²/(2σ²))
```

Sigmoid

```
μ(x) = 1 / (1 + exp(-a(x-c)))
```

Fuzzy Rules Syntax

Python

```python
# Simple rule
rule1 = ctrl.Rule(temperature['cold'], power['high'])

# AND
rule2 = ctrl.Rule(temperature['cold'] & humidity['high'], power['high'])

# OR
rule3 = ctrl.Rule(temperature['hot'] | humidity['high'], power['low'])

# NOT
rule4 = ctrl.Rule(~temperature['cold'], power['low'])

# Complex
rule5 = ctrl.Rule(
    (temperature['cold'] & humidity['high']) | temperature['very_cold'],
    power['very_high']
)
```

Rust

```rust
// In Rust, you build rules with closures
let rule = FuzzyRule::new(
    "Complex Rule",
    Box::new(|inputs| {
        let cold = inputs.get("temp_cold").copied().unwrap_or(0.0);
        let humid = inputs.get("humid_high").copied().unwrap_or(0.0);
        and(vec![cold, humid])  // AND operation
    }),
    "power_high",
);

// OR operation
Box::new(|inputs| {
    or(vec![cold, humid])
})
```

```
// NOT operation
Box::new(|inputs| {
    not(cold)
})
```

Common Patterns

Temperature Controller Pattern

```
IF temperature is COLD THEN heating is HIGH
IF temperature is COMFORTABLE THEN heating is OFF
IF temperature is HOT THEN cooling is HIGH
```

Tipping System Pattern

```
IF service is POOR OR food is POOR THEN tip is LOW
IF service is GOOD AND food is GOOD THEN tip is HIGH
```

Decision Making Pattern

```
IF risk is HIGH THEN investment is LOW
IF risk is LOW AND return is HIGH THEN investment is HIGH
```

Performance Tips

Python

- Use NumPy arrays for vectorized operations
- Avoid loops when possible
- Use `interp_membership` for single value lookups
- Cache membership function results if evaluating multiple times

Rust

- Use `--release` flag for production: `cargo run --release`
- Membership functions are already optimized
- Consider parallel processing for large datasets
- Use appropriate precision (f32 vs f64)

Common Issues & Solutions

Python

Issue: `ValueError: antecedent must be an Antecedent`

- **Solution**: Make sure variables are defined as `ctrl.Antecedent` or `ctrl.Consequent`

Issue: Membership functions don't overlap properly

- **Solution**: Check your `[a, b, c]` or `[a, b, c, d]` values for proper ordering

Rust

Issue: Borrow checker errors with closures

- **Solution**: Use `move` keyword and clone data before closure

Issue: Type mismatch with `Box<dyn Fn>`

- **Solution**: Add `+ Send + Sync` traits: `Box<dyn Fn(f64) -> f64 + Send + Sync>`

🎓 Key Formulas

Centroid Defuzzification:

```
x* = Σ(μ(xi) × xi) / Σ(μ(xi))
```

Fuzzy Complement:

```
μ_NOT_A(x) = 1 - μ_A(x)
```

Fuzzy Union (Max):

```
μ_A∪B(x) = max(μ_A(x), μ_B(x))
```

Fuzzy Intersection (Min):

```
μ_A∩B(x) = min(μ_A(x), μ_B(x))
```

💡 Quick Tips

1. **Start Simple**: Begin with triangular membership functions
2. **Test Boundaries**: Always test edge cases (min, max values)
3. **Visualize**: Plot membership functions to verify correctness

4. **Rule Coverage**: Ensure all input combinations have rules
5. **Smooth Transitions**: Overlap membership functions by 20-50%
6. **Defuzzification**: Centroid is most common, but experiment!

📚 Further Reading

- Zadeh, L.A. (1965). "Fuzzy Sets"
- Mamdani & Assilian (1975). "Experiment in Linguistic Synthesis"
- Ross, T.J. "Fuzzy Logic with Engineering Applications"

Happy Fuzzy Logic Learning! 🎉