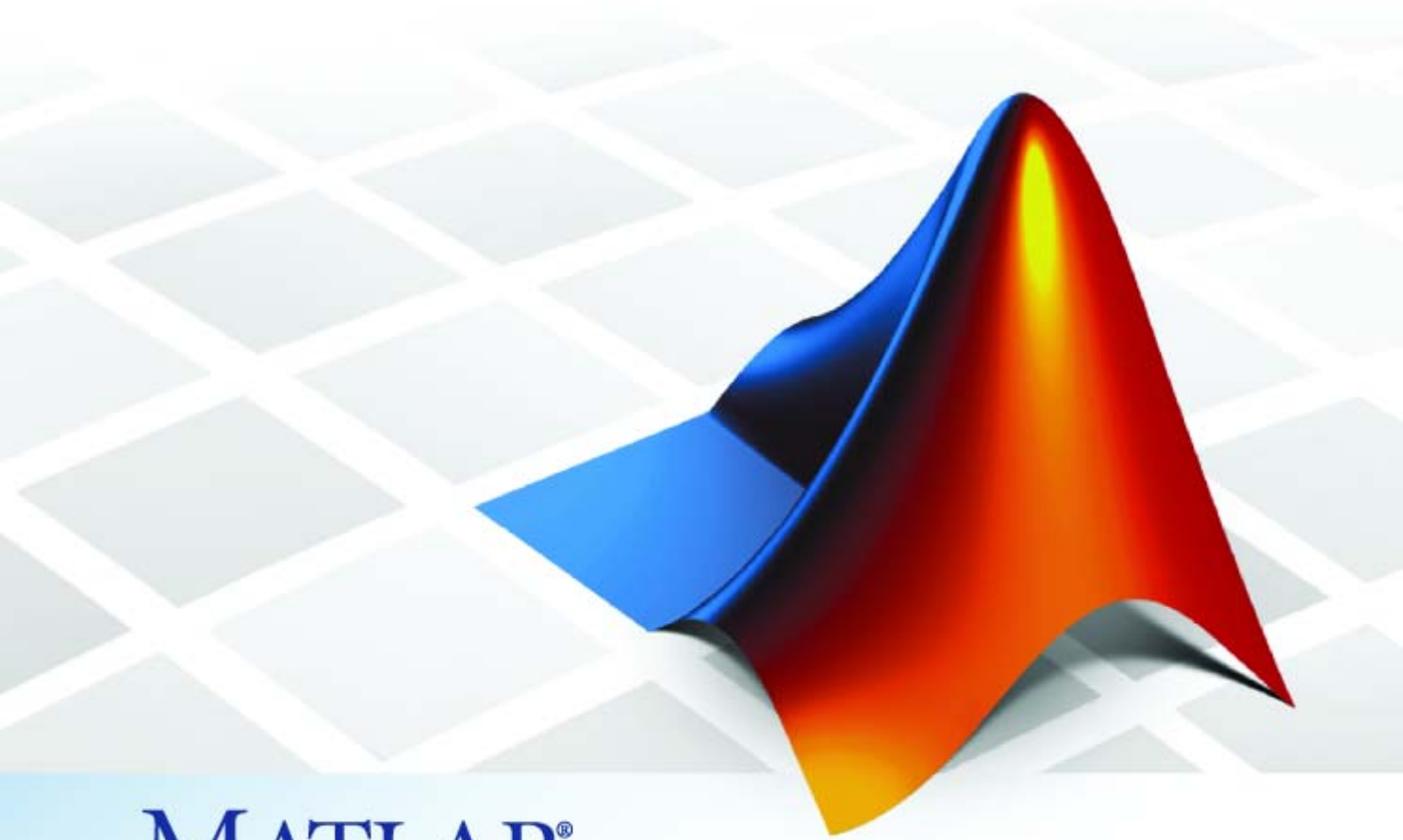
Fuzzy Logic Toolbox 2

User's Guide



MATLAB®

The MathWorks™

Accelerating the pace of engineering and science

How to Contact The MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support
suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Fuzzy Logic Toolbox User's Guide

© COPYRIGHT 1995–2007 The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, SimBiology, SimHydraulics, SimEvents, and xPC TargetBox are registered trademarks and The MathWorks, the L-shaped membrane logo, Embedded MATLAB, and PolySpace are trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

January 1995	First printing	
April 1997	Second printing	
January 1998	Third printing	
September 2000	Fourth printing	Revised for Version 2 (Release 12)
April 2003	Fifth printing	
June 2004	Online only	Updated for Version 2.1.3 (Release 14)
March 2005	Online only	Updated for Version 2.2.1 (Release 14SP2)
September 2005	Online only	Updated for Version 2.2.2 (Release 14SP3)
March 2006	Online only	Updated for Version 2.2.3 (Release 2006a)
September 2006	Online only	Updated for Version 2.2.4 (Release 2006b)
March 2007	Online only	Updated for Version 2.2.5 (Release 2007a)
September 2007	Online only	Revised for Version 2.2.6 (Release 2007b)

Contents

1 Getting Started

What Is Fuzzy Logic Toolbox? 1-2
Fuzzy Logic Toolbox Description 1-2
Installation 1-3
Using This Guide 1-3

What Is Fuzzy Logic? 1-5
Description of Fuzzy Logic 1-5
Why Use Fuzzy Logic? 1-8
When Not to Use Fuzzy Logic 1-9
What Can Fuzzy Logic Toolbox Do? 1-10

An Introductory Example: Fuzzy Versus Nonfuzzy Logic 1-12
The Basic Tipping Problem 1-12
The Nonfuzzy Approach 1-12
The Fuzzy Logic Approach 1-16
Problem Solution 1-17

2 Tutorial

Overview 2-2

Foundations of Fuzzy Logic 2-4
Fuzzy Sets 2-4
Membership Functions 2-8
Logical Operations 2-13
If-Then Rules 2-16

Fuzzy Inference Systems 2-20
What Are Fuzzy Interference Systems? 2-20

Tipping Problem Revisited 2-21
The Fuzzy Inference Diagram 2-27
Customization 2-29

Building Systems with Fuzzy Logic Toolbox 2-30
Another Tipping Problem 2-30
Getting Started 2-33
The FIS Editor 2-34
The Membership Function Editor 2-39
The Rule Editor 2-45
The Rule Viewer 2-49
The Surface Viewer 2-52
Importing and Exporting from the GUI Tools 2-53
Customizing Your Fuzzy System 2-54

Working from the Command Line 2-57
The Tipping Problem from the Command Line 2-57
System Display Functions 2-60
Building a System from Scratch 2-63
FIS Evaluation 2-66
The FIS Structure 2-66

Working with Simulink 2-71
An Example: Water Level Control 2-71
Building Your Own Fuzzy Simulink Models 2-78

Sugeno-Type Fuzzy Inference 2-83
What is Sugeno-Type Fuzzy Interference? 2-83
An Example: Two Lines 2-87
Comparison of Sugeno and Mamdani Methods 2-89

anfis and the ANFIS Editor GUI 2-90
Introduction 2-90
A Modeling Scenario 2-91
Model Learning and Inference Through ANFIS 2-91
Know Your Data 2-92
Constraints of anfis 2-94
Training Adaptive Neuro Fuzzy Inference Systems Using the ANFIS Editor GUI 2-94
ANFIS Editor GUI Example 1: Checking Data Helps Model Validation 2-98

ANFIS Editor GUI Example 2: Checking Data Does Not Validate Model 2-107
anfis from the Command Line 2-110
More on anfis and the ANFIS Editor GUI 2-117

Fuzzy Clustering 2-124
What is Data Clustering 2-124
Fuzzy C-Means Clustering 2-124
Subtractive Clustering 2-128
Data Clustering Using the Clustering GUI Tool 2-134

Stand-Alone C Code Fuzzy Inference Engine 2-138

3 Functions — By Category

GUI Tools and Plotting Functions 3-2

Membership Functions 3-3

FIS Data Structure Management 3-4

Advanced Techniques 3-5

Simulink 3-6

4 Functions — Alphabetical List

5 Blocks — By Category

Controllers 5-2

Logical Operators 5-3

Membership Functions 5-4

6 Blocks — Alphabetical List

A Examples

Introductory Examples A-2

Dinner for Two, from the Top A-2

Simulink Examples A-2

Cart and Pole Simulation A-2

Sugeno Fuzzy Inference Example A-2

ANFIS Editor GUI Examples A-2

Bibliography

B

Glossary

Index

1

Getting Started

What Is Fuzzy Logic Toolbox? (p. 1-2)	A collection of functions built on the MATLAB® numeric computing environment
What Is Fuzzy Logic? (p. 1-5)	Overview of fuzzy logic and its applications
An Introductory Example: Fuzzy Versus Nonfuzzy Logic (p. 1-12)	An example showing the difference between linear and fuzzy logic approach

What Is Fuzzy Logic Toolbox?

In this section...

"Fuzzy Logic Toolbox Description" on page 1-2

"Installation" on page 1-3

"Using This Guide" on page 1-3

Fuzzy Logic Toolbox Description

Fuzzy Logic Toolbox is a collection of functions built on the MATLAB® numeric computing environment. It provides tools for you to create and edit fuzzy inference systems within the framework of MATLAB, or if you prefer, you can integrate your fuzzy systems into simulations with Simulink®. You can even build stand-alone C programs that call on fuzzy systems you build with MATLAB. This toolbox relies heavily on graphical user interface (GUI) tools to help you accomplish your work, although you can work entirely from the command line if you prefer.

The toolbox provides three categories of tools:

- Command line functions
- Graphical interactive tools
- Simulink blocks and examples

The first category of tools is made up of functions that you can call from the command line or from your own applications. Many of these functions are MATLAB M-files, series of MATLAB statements that implement specialized fuzzy logic algorithms. You can view the MATLAB code for these functions using the statement

```
type function_name
```

You can change the way any toolbox function works by copying and renaming the M-file, then modifying your copy. You can also extend the toolbox by adding your own M-files.

Secondly, the toolbox provides a number of interactive tools that let you access many of the functions through a GUI. Together, the GUI-based tools provide an environment for fuzzy inference system design, analysis, and implementation.

The third category of tools is a set of blocks for use with the Simulink simulation software. These are specifically designed for high speed fuzzy logic inference in the Simulink environment.

What makes Fuzzy Logic Toolbox so powerful is the fact that most of human reasoning and concept formation is linked to the use of fuzzy rules. By providing a systematic framework for computing with fuzzy rules, Fuzzy Logic Toolbox greatly amplifies the power of human reasoning. Further amplification results from the use of MATLAB and graphical user interfaces, areas in which The MathWorks™ has unparalleled expertise.

Installation

To install this toolbox on a workstation, large machine, or a PC, see the installation documentation for that platform.

To determine if Fuzzy Logic Toolbox is already installed on your system, check for a subdirectory named `fuzzy` within the main toolbox directory or folder.

Using This Guide

If you are new to fuzzy logic, begin with "What Is Fuzzy Logic?" on page 1-5. This introduces the motivation behind fuzzy logic and leads you smoothly into the tutorial.

If you are an experienced fuzzy logic user, you may want to start at the beginning of Chapter 2, "Tutorial" to make sure you are comfortable with the fuzzy logic terminology in Fuzzy Logic Toolbox. If you just want an overview of each graphical tool and examples of specific fuzzy system tasks, turn directly to "Building Systems with Fuzzy Logic Toolbox" on page 2-30. This section does not include information on the adaptive data modeling application covered by the toolbox function `ANFIS`. The basic functionality of this tool can be found in "Training Adaptive Neuro Fuzzy Inference Systems Using the ANFIS Editor GUI" on page 2-94.

If you just want to start as soon as possible and experiment, you can open an example system right away by typing

```
fuzzy tipper
```

This displays the Fuzzy Inference System (FIS) editor for an example decision-making problem that has to do with how to tip in a restaurant.

All toolbox users should use Chapter 4, “Functions — Alphabetical List” for information on specific tools or functions. Reference descriptions include a synopsis of the function’s syntax, as well as a complete explanation of options and operation. Many reference descriptions also include helpful examples, a description of the function’s algorithm, and references to additional reading material. For GUI-based tools, the descriptions include options for invoking the tool.

What Is Fuzzy Logic?

In this section...

"Description of Fuzzy Logic" on page 1-5

"Why Use Fuzzy Logic?" on page 1-8

"When Not to Use Fuzzy Logic" on page 1-9

"What Can Fuzzy Logic Toolbox Do?" on page 1-10

Description of Fuzzy Logic

In recent years, the number and variety of applications of fuzzy logic have increased significantly. The applications range from consumer products such as cameras, camcorders, washing machines, and microwave ovens to industrial process control, medical instrumentation, decision-support systems, and portfolio selection.

To understand why use of fuzzy logic has grown, you must first understand what is meant by fuzzy logic.

Fuzzy logic has two different meanings. In a narrow sense, fuzzy logic is a logical system, which is an extension of multivalued logic. However, in a wider sense fuzzy logic (FL) is almost synonymous with the theory of fuzzy sets, a theory which relates to classes of objects with unsharp boundaries in which membership is a matter of degree. In this perspective, fuzzy logic in its narrow sense is a branch of FL. Even in its more narrow definition, fuzzy logic differs both in concept and substance from traditional multivalued logical systems.

In Fuzzy Logic Toolbox, fuzzy logic should be interpreted as FL, that is, fuzzy logic in its wide sense. The basic ideas underlying FL are explained very clearly and insightfully in the Introduction. What might be added is that the basic concept underlying FL is that of a linguistic variable, that is, a variable whose values are words rather than numbers. In effect, much of FL may be viewed as a methodology for computing with words rather than numbers. Although words are inherently less precise than numbers, their use is closer to human intuition. Furthermore, computing with words exploits the tolerance for imprecision and thereby lowers the cost of solution.

Another basic concept in FL, which plays a central role in most of its applications, is that of a fuzzy if-then rule or, simply, fuzzy rule. Although rule-based systems have a long history of use in AI, what is missing in such systems is a mechanism for dealing with fuzzy consequents and fuzzy antecedents. In fuzzy logic, this mechanisms provided by the calculus of fuzzy rules. The calculus of fuzzy rules serves as a basis for what might be called the Fuzzy Dependency and Command Language (FDCL). Although FDCL is not used explicitly in Fuzzy Logic Toolbox, it is effectively one of its principal constituents. In most of the applications of fuzzy logic, a fuzzy logic solution is, in reality, a translation of a human solution into FDCL.

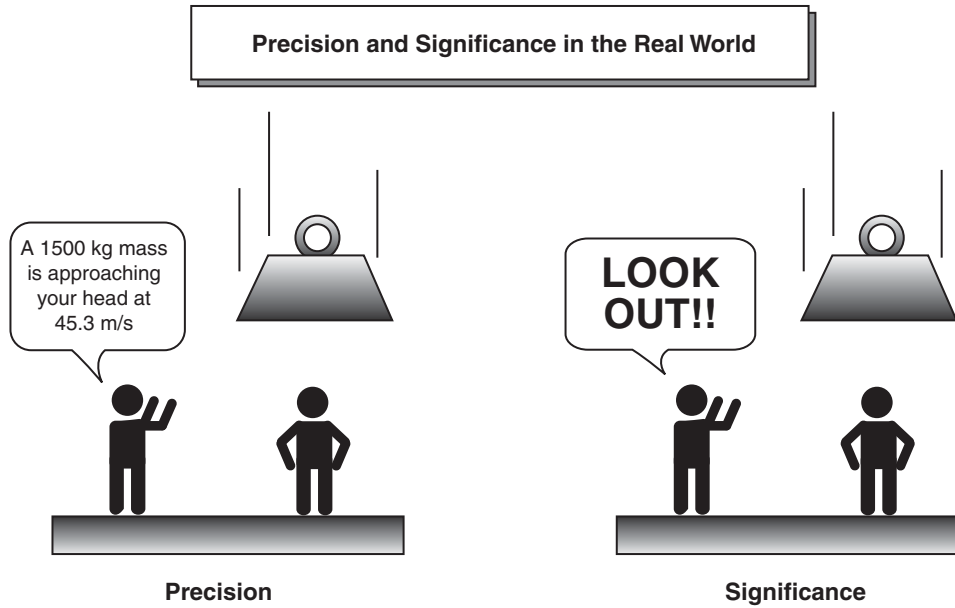
A trend that is growing in visibility relates to the use of fuzzy logic in combination with neurocomputing and genetic algorithms. More generally, fuzzy logic, neurocomputing, and genetic algorithms may be viewed as the principal constituents of what might be called soft computing. Unlike the traditional, hard computing, *soft computing* accommodates the imprecision of the real world. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and low solution cost. In the future, soft computing could play an increasingly important role in the conception and design of systems whose MIQ (Machine IQ) is much higher than that of systems designed by conventional methods.

Among various combinations of methodologies in soft computing, the one that has highest visibility at this juncture is that of fuzzy logic and neurocomputing, leading to neuro-fuzzy systems. Within fuzzy logic, such systems play a particularly important role in the induction of rules from observations. An effective method developed by Dr. Roger Jang for this purpose is called ANFIS (Adaptive Neuro-Fuzzy Inference System). This method is an important component of Fuzzy Logic Toolbox.

Fuzzy logic is all about the relative importance of precision: How important is it to be exactly right when a rough answer will do?

Fuzzy Logic Toolbox for use with MATLAB is a tool for solving problems with fuzzy logic. Fuzzy logic is a fascinating area of research because it does a good job of trading off between significance and precision—something that humans have been managing for a very long time.

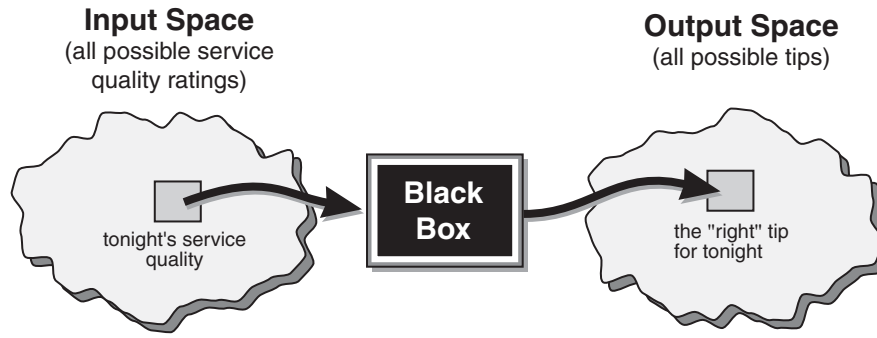
In this sense, fuzzy logic is both old and new because, although the modern and methodical science of fuzzy logic is still young, the concepts of fuzzy logic relies on age-old skills of human reasoning.



Fuzzy logic is a convenient way to map an input space to an output space. Mapping input to output is the starting point for everything. Consider the following examples:

- With information about how good your service was at a restaurant, a fuzzy logic system can tell you what the tip should be.
- With your specification of how hot you want the water, a fuzzy logic system can adjust the faucet valve to the right setting.
- With information about how far away the subject of your photograph is, a fuzzy logic system can focus the lens for you.
- With information about how fast the car is going and how hard the motor is working, a fuzzy logic system can shift gears for you.

A graphical example of an input-output map is shown in the following figure.



An input-output map for the tipping problem:
"Given the quality of service, how much should I tip?"

To determine the appropriate amount of tip requires mapping inputs to the appropriate outputs. Between the input and the output, the preceding figure shows a black box that can contain any number of things: fuzzy systems, linear systems, expert systems, neural networks, differential equations, interpolated multidimensional lookup tables, or even a spiritual advisor, just to name a few of the possible options. Clearly the list could go on and on.

Of the dozens of ways to make the black box work, it turns out that fuzzy is often the very best way. Why should that be? As Lotfi Zadeh, who is considered to be the father of fuzzy logic, once remarked: "In almost every case you can build the same product without fuzzy logic, but fuzzy is faster and cheaper."

Why Use Fuzzy Logic?

Here is a list of general observations about fuzzy logic:

- Fuzzy logic is conceptually easy to understand.

  The mathematical concepts behind fuzzy reasoning are very simple. Fuzzy logic is a more intuitive approach without the far-reaching complexity.

- Fuzzy logic is flexible.

  With any given system, it is easy to layer on more functionality without starting again from scratch.

- Fuzzy logic is tolerant of imprecise data.

Everything is imprecise if you look closely enough, but more than that, most things are imprecise even on careful inspection. Fuzzy reasoning builds this understanding into the process rather than tacking it onto the end.

- Fuzzy logic can model nonlinear functions of arbitrary complexity.

  You can create a fuzzy system to match any set of input-output data. This process is made particularly easy by adaptive techniques like Adaptive Neuro-Fuzzy Inference Systems (ANFIS), which are available in Fuzzy Logic Toolbox.

- Fuzzy logic can be built on top of the experience of experts.

  In direct contrast to neural networks, which take training data and generate opaque, impenetrable models, fuzzy logic lets you rely on the experience of people who already understand your system.

- Fuzzy logic can be blended with conventional control techniques.

  Fuzzy systems don't necessarily replace conventional control methods. In many cases fuzzy systems augment them and simplify their implementation.

- Fuzzy logic is based on natural language.

  The basis for fuzzy logic is the basis for human communication. This observation underpins many of the other statements about fuzzy logic. Because fuzzy logic is built on the structures of qualitative description used in everyday language, fuzzy logic is easy to use.

The last statement is perhaps the most important one and deserves more discussion. Natural language, which is used by ordinary people on a daily basis, has been shaped by thousands of years of human history to be convenient and efficient. Sentences written in ordinary language represent a triumph of efficient communication.

When Not to Use Fuzzy Logic

Fuzzy logic is not a cure-all. When should you not use fuzzy logic? The safest statement is the first one made in this introduction: fuzzy logic is a convenient way to map an input space to an output space. If you find it's not convenient, try something else. If a simpler solution already exists, use it. Fuzzy logic is the codification of common sense — use common sense when you implement it and you will probably make the right decision. Many controllers, for example,

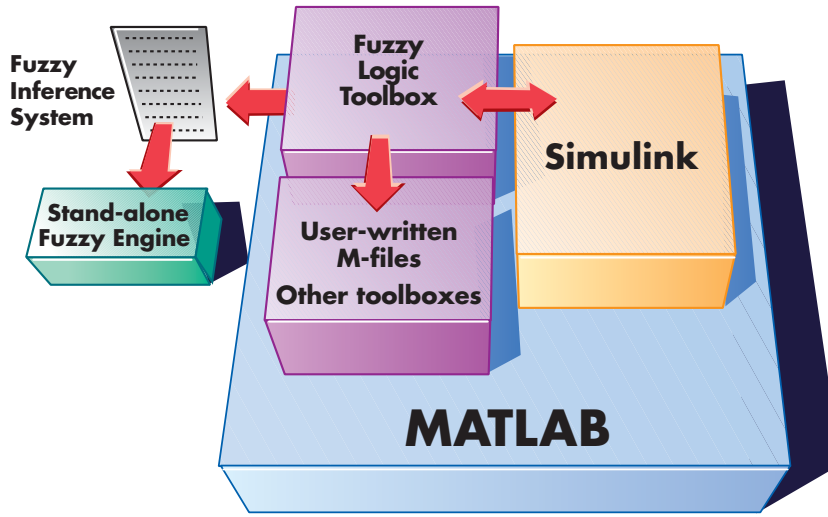
do a fine job without using fuzzy logic. However, if you take the time to become familiar with fuzzy logic, you'll see it can be a very powerful tool for dealing quickly and efficiently with imprecision and nonlinearity.

What Can Fuzzy Logic Toolbox Do?

Fuzzy Logic Toolbox allows you to do several things, but the most important thing it lets you do is create and edit fuzzy inference systems. You can create these systems using graphical tools or command-line functions, or you can generate them automatically using either clustering or adaptive neuro-fuzzy techniques.

If you have access to Simulink, you can easily test your fuzzy system in a block diagram simulation environment.

The toolbox also lets you run your own stand-alone C programs directly, without the need for Simulink. This is made possible by a stand-alone Fuzzy Inference Engine that reads the fuzzy systems saved from a MATLAB session. You can customize the stand-alone engine to build fuzzy inference into your own code. All provided code is ANSI compliant.



Because of the integrated nature of the MATLAB environment, you can create your own tools to customize Fuzzy Logic Toolbox or harness it with another

toolbox, such as the Control System Toolbox, Neural Network Toolbox, or Optimization Toolbox, to mention only a few of the possibilities.

An Introductory Example: Fuzzy Versus Nonfuzzy Logic

In this section...

"The Basic Tipping Problem" on page 1-12
"The Nonfuzzy Approach" on page 1-12
"The Fuzzy Logic Approach" on page 1-16
"Problem Solution" on page 1-17

The Basic Tipping Problem

To illustrate the value of fuzzy logic, examine both linear and fuzzy approaches to the following problem:

What is the right amount to tip your waitperson?

First, work through this problem the conventional (nonfuzzy) way, writing MATLAB commands that spell out linear and piecewise-linear relations. Then, look at the same system using fuzzy logic.

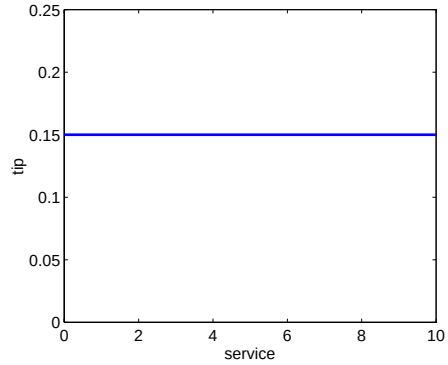
The Basic Tipping Problem. Given a number between 0 and 10 that represents the quality of service at a restaurant (where 10 is excellent), what should the tip be?

Note This problem is based on tipping as it is typically practiced in the United States. An average tip for a meal in the U.S. is 15%, though the actual amount may vary depending on the quality of the service provided.

The Nonfuzzy Approach

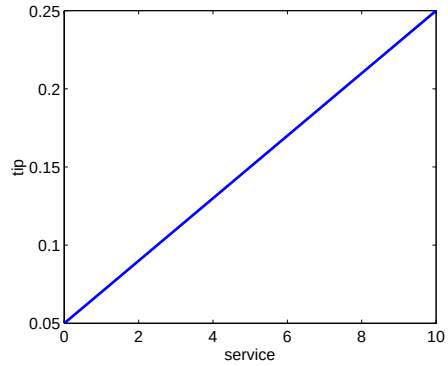
Begin with the simplest possible relationship. Suppose that the tip always equals 15% of the total bill.

```
tip = 0.15
```



This relationship does not take into account the quality of the service, so you need to add a new term to the equation. Because service is rated on a scale of 0 to 10, you might have the tip go linearly from 5% if the service is bad to 25% if the service is excellent. Now the relation looks like the following plot:

```
tip=0.20/10*service+0.05
```

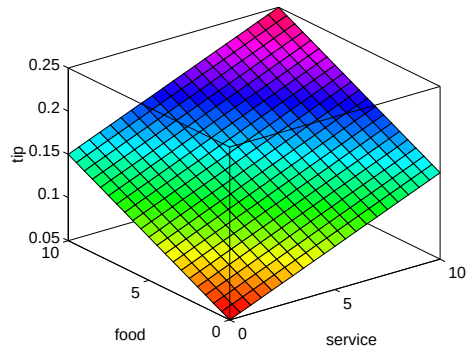


The formula does what you want it to do, and is straightforward. However, you may want the tip to reflect the quality of the food as well. This extension of the problem is defined as follows.

The Extended Tipping Problem. Given two sets of numbers between 0 and 10 (where 10 is excellent) that respectively represent the quality of the service and the quality of the food at a restaurant, what should the tip be?

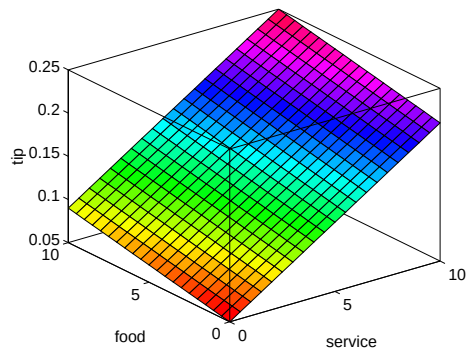
See how the formula is affected now that you have added another variable. Try the following equation:

```
tip = 0.20/20*(service+food)+0.05;
```



In this case, the results look satisfactory, but when you look at them closely, they do not seem quite right. Suppose you want the service to be a more important factor than the food quality. Specify that service accounts for 80% of the overall tipping grade and the food makes up the other 20%. Try this equation:

```
servRatio=0.8;
tip=servRatio*(0.20/10*service+0.05) + ...
    (1-servRatio)*(0.20/10*food+0.05);
```

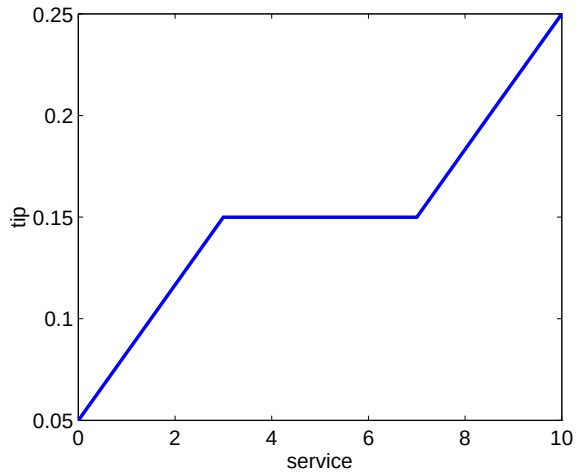


The response is still somehow too uniformly linear. Suppose you want more of a flat response in the middle, i.e., you want to give a 15% tip in general, but

want to also specify a variation if the service is exceptionally good or bad. This factor, in turn, means that the previous linear mappings no longer apply. You can still use the linear calculation with a piecewise linear construction. Now, return to the one-dimensional problem of just considering the service. You can string together a simple conditional statement using breakpoints like this.

```
if service<3,
    tip=(0.10/3)*service+0.05;
elseif service<7,
    tip=0.15;
elseif service<=10,
    tip=(0.10/3)*(service-7)+0.15;
end
```

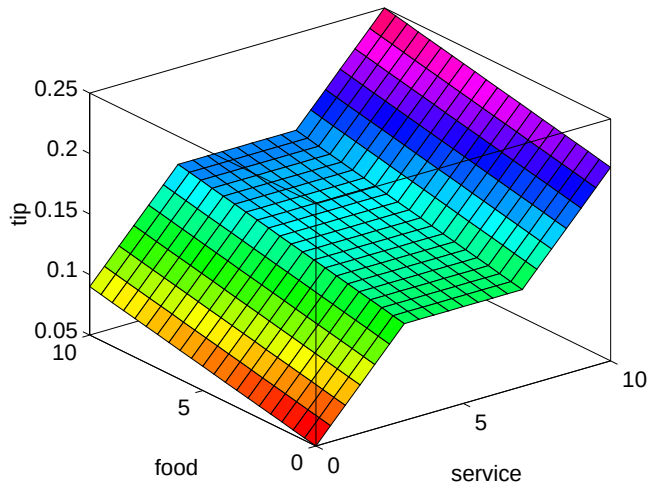
The plot now looks like the following figure:



If you extend this to two dimensions, where you take food into account again, something like the following output results.

```
servRatio=0.8;
if service<3,
    tip=((0.10/3)*service+0.05)*servRatio + ...
        (1-servRatio)*(0.20/10*food+0.05);
elseif service<7,
```

```
    tip=(0.15)*servRatio + ...
        (1-servRatio)*(0.20/10*food+0.05);
else,
    tip=((0.10/3)*(service-7)+0.15)*servRatio + ...
        (1-servRatio)*(0.20/10*food+0.05);
end
```



The plot looks good, but the function is surprisingly complicated. It was a little difficult to code this correctly, and it is definitely not easy to modify this code in the future. Moreover, it is even less apparent how the algorithm works to someone who did not see the original design process.

The Fuzzy Logic Approach

You need to capture the essentials of this problem, leaving aside all the factors that could be arbitrary. If you make a list of what really matters in this problem, you might end up with the following rule descriptions.

Tipping Problem Rules — Service Factor
If service is poor, then tip is cheap
If service is good, then tip is average
If service is excellent, then tip is generous

The order in which the rules are presented here is arbitrary. It does not matter which rules come first. If you want to include the food's effect on the tip, add the following two rules.

Tipping Problem Rules — Food Factor
If food is rancid, then tip is cheap
If food is delicious, then tip is generous

You can combine the two different lists of rules into one tight list of three rules like so.

Tipping Problem — Both Service and Food Factors
If service is poor or the food is rancid, then tip is cheap
If service is good, then tip is average
If service is excellent or food is delicious, then tip is generous

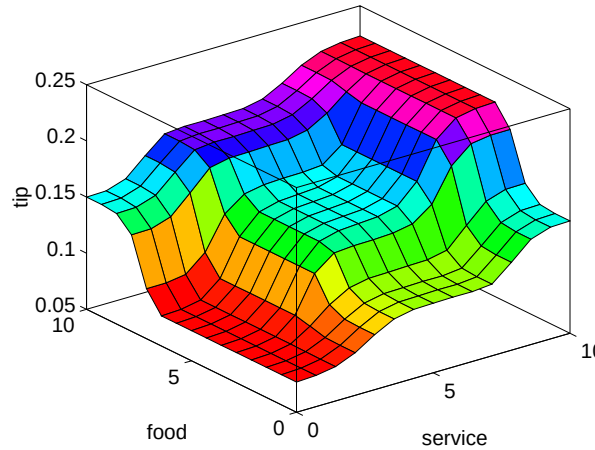
These three rules are the core of your solution. Coincidentally, you have just defined the rules for a fuzzy logic system. When you give mathematical meaning to the linguistic variables (what is an average tip, for example?) you have a complete fuzzy inference system. The methodology of fuzzy logic must also consider:

- How are the rules all combined?
- How do I define mathematically what an average tip is?

The next few chapters provide detailed answers to these questions. The details of the method don't really change much from problem to problem—the mechanics of fuzzy logic aren't terribly complex. What matters is that you understand that fuzzy logic is adaptable, simple, and easily applied.

Problem Solution

The following plot represents the fuzzy logic system that solves the tipping problem.



This plot was generated by the three rules that accounted for both service and food factors. The mechanics of how fuzzy inference works is explained in “Overview” on page 2-2, “Foundations of Fuzzy Logic” on page 2-4, and in “Fuzzy Inference Systems” on page 2-20. In “Building Systems with Fuzzy Logic Toolbox” on page 2-30, the entire tipping problem is worked through using the graphical tools in Fuzzy Logic Toolbox.

Observations

Consider some observations about the example so far. You found a piecewise linear relation that solved the problem. It worked, but it was problematic to derive, and when you wrote it down as code, it was not very easy to interpret. Conversely, the fuzzy logic system is based on some common sense statements. Also, you were able to add two more rules to the bottom of the list that influenced the shape of the overall output without needing to undo what had already been done, making the subsequent modification was relatively easy.

Moreover, by using fuzzy logic rules, the maintenance of the structure of the algorithm decouples along fairly clean lines. The notion of an average tip might change from day to day, city to city, country to country, but the underlying logic is the same: if the service is good, the tip should be average.

Recalibrating the Method

You can recalibrate the method quickly by simply shifting the fuzzy set that defines average without rewriting the fuzzy logic rules.

You can shift lists of piecewise linear functions, but there is a greater likelihood that recalibration will not be so quick and simple.

In the following example, the piecewise linear tipping problem slightly rewritten to make it more generic. It performs the same function as before, only now the constants can be easily changed.

```
% Establish constants
lowTip=0.05; averTip=0.15; highTip=0.25;
tipRange=highTip-lowTip;
badService=0; okayService=3;
goodService=7; greatService=10;
serviceRange=greatService-badService;
badFood=0; greatFood=10;
foodRange=greatFood-badFood;

% If service is poor or food is rancid, tip is cheap
if service<okayService,
    tip=(((averTip-lowTip)/(okayService-badService)) ...
        *service+lowTip)*servRatio + ...
        (1-servRatio)*(tipRange/foodRange*food+lowTip);
% If service is good, tip is average
elseif service<goodService,
    tip=averTip*servRatio + (1-servRatio)* ...
        (tipRange/foodRange*food+lowTip);
% If service is excellent or food is delicious, tip is generous
else,
    tip=(((highTip-averTip)/ ...
        (greatService-goodService))* ...
        (service-goodService)+averTip)*servRatio + ...
        (1-servRatio)*(tipRange/foodRange*food+lowTip);
end
```

As with all code, the more generality that is introduced, the less precise the algorithm becomes. While you can improve clarity by adding more comments,

or perhaps rewriting the algorithm in slightly more self-evident ways, but the piecewise linear methodology is not the optimal way to resolve this issue.

If you remove everything from the algorithm except for three comments, what remain are exactly the fuzzy logic rules you previously wrote down.

```
% If service is poor or food is rancid, tip is cheap
% If service is good, tip is average
% If service is excellent or food is delicious, tip is generous
```

If, as with a fuzzy system, the comment is identical with the code, think how much more likely your code is to have comments. Fuzzy logic lets the language that is clearest to you, high level comments, also have meaning to the machine, which is why it is a very successful technique for bridging the gap between people and machines.

By making the equations as simple as possible (linear) you make things simpler for the machine but more complicated for you. However, the limitation is really no longer the computer—it is your mental model of what the computer is doing. Computers have the ability to make things hopelessly complex; fuzzy logic reclaims the middle ground and lets the machine work with your preferences rather than the other way around.

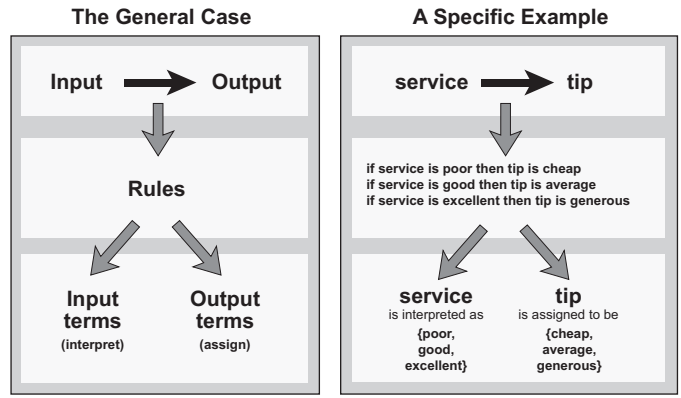
2

Tutorial

Overview (p. 2-2)	An overview of the fuzzy inference process
Foundations of Fuzzy Logic (p. 2-4)	Fuzzy sets, membership functions, if-then rules, and logical operations
Fuzzy Inference Systems (p. 2-20)	Mamdani-type and Sugeno-type inference systems
Building Systems with Fuzzy Logic Toolbox (p. 2-30)	Using the graphical user interface (GUI) tools
Working from the Command Line (p. 2-57)	Using toolbox functions from the MATLAB command line
Working with Simulink (p. 2-71)	Fuzzy Logic Toolbox working with simulation software
Sugeno-Type Fuzzy Inference (p. 2-83)	Output membership functions are either linear or constant
anfis and the ANFIS Editor GUI (p. 2-90)	Tools that apply fuzzy inference techniques to data modeling
Fuzzy Clustering (p. 2-124)	Identifies natural groupings of data that produces concise representation of a system's behavior
Stand-Alone C Code Fuzzy Inference Engine (p. 2-138)	Source codes for a stand-alone fuzzy inference engine

Overview

The point of fuzzy logic is to map an input space to an output space, and the primary mechanism for doing this is a list of if-then statements called rules. All rules are evaluated in parallel, and the order of the rules is unimportant. The rules themselves are useful because they refer to variables and the adjectives that describe those variables. Before you can build a system that interprets rules, you must define all the terms you plan on using and the adjectives that describe them. To say that the water is hot, you need to define the range that the water's temperature can be expected to vary as well as what we mean by the word *hot*. The following diagram provides a roadmap for the fuzzy inference process. It shows the general description of a fuzzy system on the left and a specific fuzzy system (the tipping example from Chapter 1, "Getting Started") on the right.



To summarize the concept of fuzzy inference depicted in this figure, *fuzzy inference is a method that interprets the values in the input vector and, based on some set of rules, assigns values to the output vector.*

This section is designed to guide you through the fuzzy logic process step by step by providing an introduction to the theory and practice of fuzzy logic. The first three sections of this section are the most important—they move from general to specific, first introducing underlying ideas and then discussing implementation details specific to the toolbox.

These three areas are as follows:

- "Foundations of Fuzzy Logic" on page 2-4, which is an introduction to the general concepts. If you are already familiar with fuzzy logic, you can to skip this section.
- "Fuzzy Inference Systems" on page 2-20, which explains the specific methods of fuzzy inference used in Fuzzy Logic Toolbox. Because the field of fuzzy logic uses many terms that do not yet have standard interpretations, read this section to become familiar with the fuzzy inference process as it is employed through Fuzzy Logic Toolbox.
- "Building Systems with Fuzzy Logic Toolbox" on page 2-30, which goes into detail about how you build and edit a fuzzy system using this toolbox. This topic provides a quick start orientation to the graphical user interface tools available in Fuzzy Logic Toolbox and guides you through the construction of a complete fuzzy inference system from start to finish.

After these three topics, there are additional topics, such as Simulink use, automatic rule generation, and demonstrations.

Foundations of Fuzzy Logic

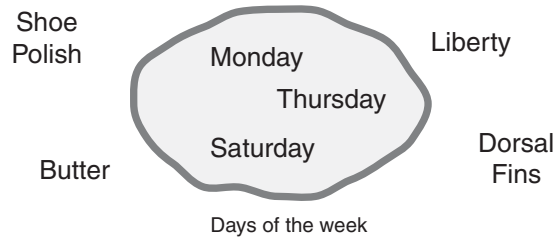
In this section...

"Fuzzy Sets" on page 2-4
"Membership Functions" on page 2-8
"Logical Operations" on page 2-13
"If-Then Rules" on page 2-16

Fuzzy Sets

Fuzzy logic starts with the concept of a fuzzy set. A *fuzzy set* is a set without a crisp, clearly defined boundary. It can contain elements with only a partial degree of membership.

To understand what a fuzzy set is, first consider the definition of a *classical set*. A classical set is a container that wholly includes or wholly excludes any given element. For example, the set of days of the week unquestionably includes Monday, Thursday, and Saturday. It just as unquestionably excludes butter, liberty, and dorsal fins, and so on.



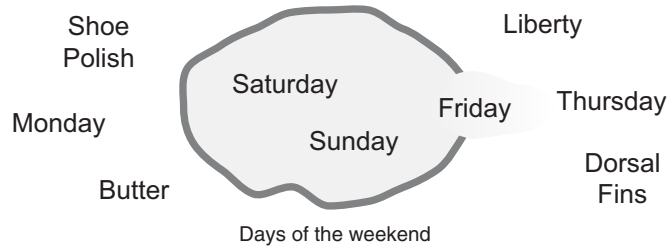
This type of set is called a classical set because it has been around for a long time. It was Aristotle who first formulated the Law of the Excluded Middle, which says X must either be in set A or in set not-A. Another version of this law is:

> *Of any subject, one thing must be either asserted or denied.*

To restate this law with annotations: "Of any subject (say Monday), one thing (a day of the week) must be either asserted or denied (I assert that Monday

is a day of the week)." This law demands that opposites, the two categories A and not-A, should between them contain the entire universe. Everything falls into either one group or the other. There is no thing that is both a day of the week and not a day of the week.

Now, consider the set of days comprising a weekend. The following diagram attempts to classify the weekend days.



Most would agree that Saturday and Sunday belong, but what about Friday? It feels like a part of the weekend, but somehow it seems like it should be technically excluded. Thus, in the preceding diagram, Friday tries its best to "straddle on the fence." Classical or normal sets would not tolerate this kind of classification. Either something is in or it is out. Human experience suggests something different, however, straddling the fence is part of life.

Of course individual perceptions and cultural background must be taken into account when you define what constitutes the weekend. Even the dictionary is imprecise, defining the weekend as the period from Friday night or Saturday to Monday morning. You are entering the realm where sharp-edged, yes-no logic stops being helpful. Fuzzy reasoning becomes valuable exactly when you work with how people really perceive the concept *weekend* as opposed to a simple-minded classification useful for accounting purposes only. More than anything else, the following statement lays the foundations for fuzzy logic.

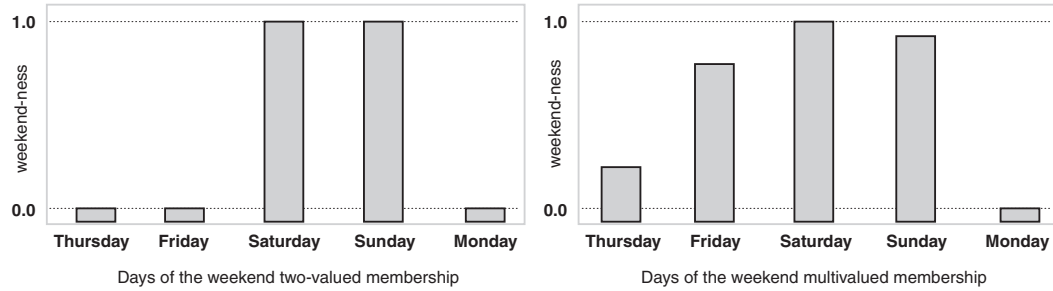
In fuzzy logic, the truth of any statement becomes a matter of degree.

Any statement can be fuzzy. The major advantage that fuzzy reasoning offers is the ability to reply to a yes-no question with a not-quite-yes-or-no answer. Humans do this kind of thing all the time (think how rarely you get a straight answer to a seemingly simple question), but it is a rather new trick for computers.

How does it work? Reasoning in fuzzy logic is just a matter of generalizing the familiar yes-no (Boolean) logic. If you give true the numerical value of 1 and false the numerical value of 0, this value indicates that fuzzy logic also permits in-between values like 0.2 and 0.7453. For instance:

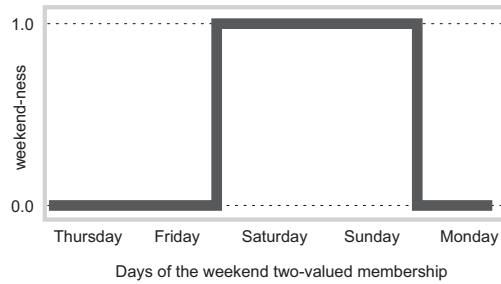
Q: Is Saturday a weekend day?
A: 1 (yes, or true)
Q: Is Tuesday a weekend day?
A: 0 (no, or false)
Q: Is Friday a weekend day?
A: 0.8 (for the most part yes, but not completely)
Q: Is Sunday a weekend day?
A: 0.95 (yes, but not quite as much as Saturday).

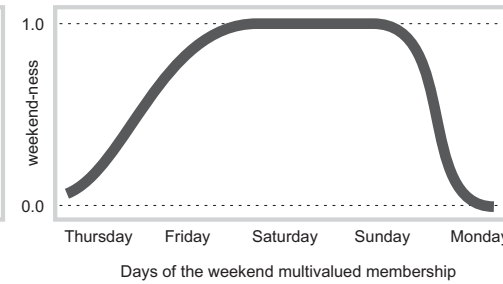
The following plot on the left shows the truth values for weekend-ness if you are forced to respond with an absolute yes or no response. On the right, is a plot that shows the truth value for weekend-ness if you are allowed to respond with fuzzy in-between values.



Technically, the representation on the right is from the domain of *multivalued logic* (or multivalent logic). If you ask the question "Is X a member of set A?" the answer might be yes, no, or any one of a thousand intermediate values in between. Thus, X might have partial membership in A. Multivalued logic stands in direct contrast to the more familiar concept of two-valued (or bivalent yes-no) logic.

To return to the example, now consider a continuous scale time plot of weekend-ness shown in the following plots.

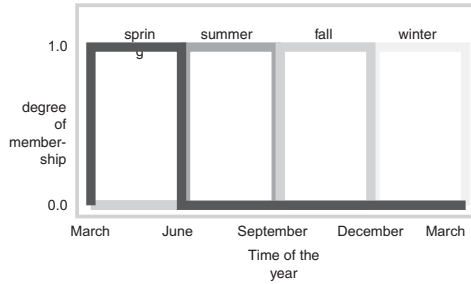


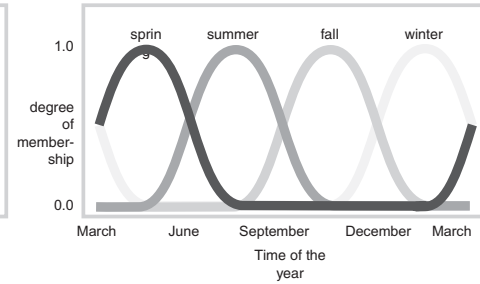


By making the plot continuous, you are defining the degree to which any given instant belongs in the weekend rather than an entire day. In the plot on the left, notice that at midnight on Friday, just as the second hand sweeps past 12, the weekend-ness truth value jumps discontinuously from 0 to 1. This is one way to define the weekend, and while it may be useful to an accountant, it may not really connect with your own real-world experience of weekend-ness.

The plot on the right shows a smoothly varying curve that accounts for the fact that all of Friday, and, to a small degree, parts of Thursday, partake of the quality of weekend-ness and thus deserve partial membership in the fuzzy set of weekend moments. The curve that defines the weekend-ness of any instant in time is a function that maps the input space (time of the week) to the output space (weekend-ness). Specifically it is known as a *membership function*. See "Membership Functions" on page 3-3 for a more detailed discussion.

As another example of fuzzy sets, consider the question of seasons. What season is it right now? In the northern hemisphere, summer officially begins at the exact moment in the earth's orbit when the North Pole is pointed most directly toward the sun. It occurs exactly once a year, in late June. Using the astronomical definitions for the season, you get sharp boundaries as shown on the left in the figure that follows. But what you experience as the seasons vary more or less continuously as shown on the right in the following figure (in temperate northern hemisphere climates).

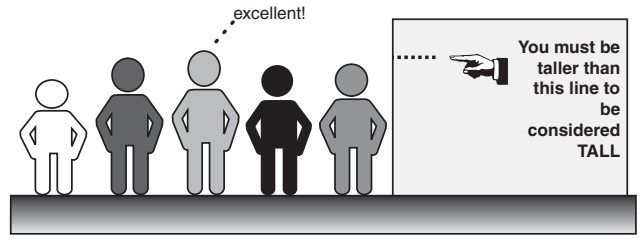




Membership Functions

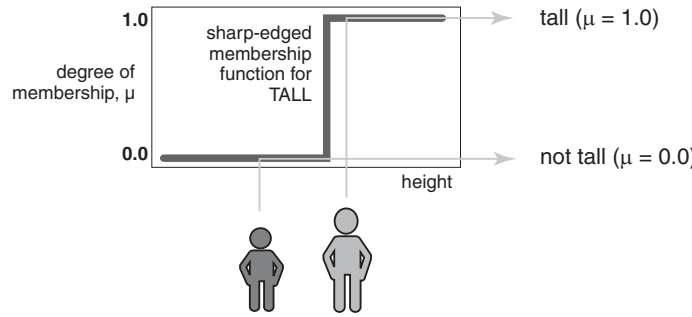
A *membership function* (MF) is a curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1. The input space is sometimes referred to as the *universe of discourse*, a fancy name for a simple concept.

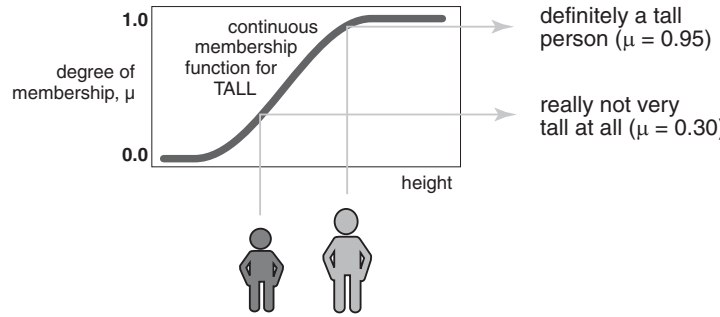
One of the most commonly used examples of a fuzzy set is the set of tall people. In this case, the universe of discourse is all potential heights, say from 3 feet to 9 feet, and the word tall would correspond to a curve that defines the degree to which any person is tall. If the set of tall people is given the well-defined (crisp) boundary of a classical set, you might say all people taller than 6 feet are officially considered tall. However, such a distinction is clearly absurd. It may make sense to consider the set of all real numbers greater than 6 because numbers belong on an abstract plane, but when we want to talk about real people, it is unreasonable to call one person short and another one tall when they differ in height by the width of a hair.



If the kind of distinction shown previously is unworkable, then what is the right way to define the set of tall people? Much as with the plot of weekend days, the figure following shows a smoothly varying curve that passes from

not-tall to tall. The output-axis is a number known as the membership value between 0 and 1. The curve is known as a *membership function* and is often given the designation of μ. This curve defines the transition from not tall to tall. Both people are tall to some degree, but one is significantly less tall than the other.





Subjective interpretations and appropriate units are built right into fuzzy sets. If you say "She's tall," the membership function tall should already take into account whether you are referring to a six-year-old or a grown woman. Similarly, the units are included in the curve. Certainly it makes no sense to say "Is she tall in inches or in meters?"

Membership Functions in Fuzzy Logic Toolbox

The only condition a membership function must really satisfy is that it must vary between 0 and 1. The function itself can be an arbitrary curve whose shape we can define as a function that suits us from the point of view of simplicity, convenience, speed, and efficiency.

A classical set might be expressed as

$$A = \{x \mid x > 6\}$$

A fuzzy set is an extension of a classical set. If X is the universe of discourse and its elements are denoted by x, then a fuzzy set A in X is defined as a set of ordered pairs.

$$A = \{x, \mu_A(x) \mid x \quad X\}$$

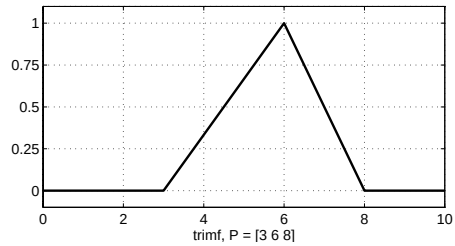
$\mu_A(x)$ is called the membership function (or MF) of x in A. The membership function maps each element of X to a membership value between 0 and 1.

Fuzzy Logic Toolbox includes 11 built-in membership function types. These 11 functions are, in turn, built from several basic functions:

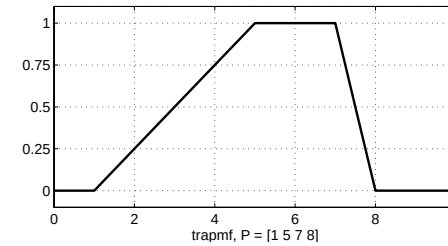
- piecewise linear functions
- the Gaussian distribution function
- the sigmoid curve
- quadratic and cubic polynomial curves

For detailed information on any of the membership functions mentioned next, turn to Chapter 4, “Functions — Alphabetical List”. By convention, all membership functions have the letters `mf` at the end of their names.

The simplest membership functions are formed using straight lines. Of these, the simplest is the *triangular* membership function, and it has the function name `trimf`. This function is nothing more than a collection of three points forming a triangle. The *trapezoidal* membership function, `trapmf`, has a flat top and really is just a truncated triangle curve. These straight line membership functions have the advantage of simplicity.



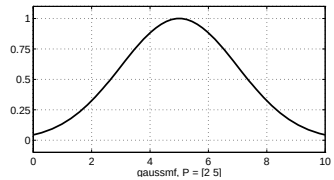
trimf



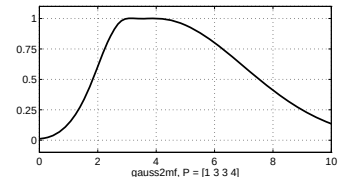
trapmf

Two membership functions are built on the *Gaussian* distribution curve: a simple Gaussian curve and a two-sided composite of two different Gaussian curves. The two functions are `gaussmf` and `gauss2mf`.

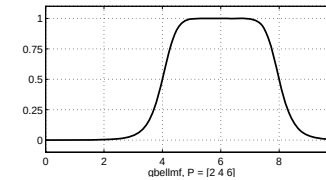
The *generalized bell* membership function is specified by three parameters and has the function name `gbellmf`. The bell membership function has one more parameter than the Gaussian membership function, so it can approach a non-fuzzy set if the free parameter is tuned. Because of their smoothness and concise notation, Gaussian and bell membership functions are popular methods for specifying fuzzy sets. Both of these curves have the advantage of being smooth and nonzero at all points.



gaussmf

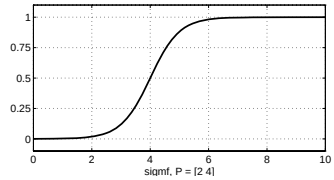


gauss2mf

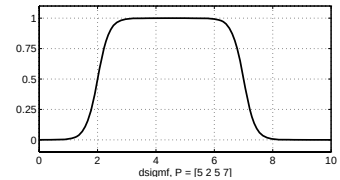


gbellmf

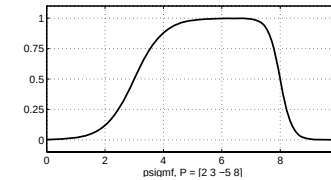
Although the Gaussian membership functions and bell membership functions achieve smoothness, they are unable to specify asymmetric membership functions, which are important in certain applications. Next, you define the *sigmoidal* membership function, which is either open left or right. Asymmetric and closed (i.e. not open to the left or right) membership functions can be synthesized using two sigmoidal functions, so in addition to the basic `sigmf`, you also have the difference between two sigmoidal functions, `dsigmf`, and the product of two sigmoidal functions `psigmf`.



sigmf

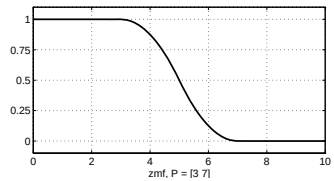


dsigmf

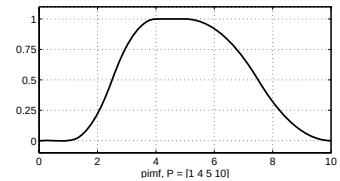


psigmf

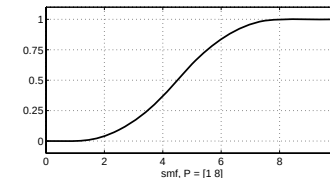
Polynomial based curves account for several of the membership functions in the toolbox. Three related membership functions are the *Z, S*, and *Pi curves*, all named because of their shape. The function `zmf` is the asymmetrical polynomial curve open to the left, `smf` is the mirror-image function that opens to the right, and `pimf` is zero on both extremes with a rise in the middle.



zmf



pimf



smf

There is a very wide selection to choose from when you're selecting your favorite membership function. Fuzzy Logic Toolbox also allows you to create your own membership functions if you find the list too restrictive. However, if a list based on expanded membership functions seems too complicated, just remember that you could probably get along very well with just one or two types of membership functions, for example the triangle and trapezoid functions. The selection is wide for those who want to explore the possibilities, but expansive membership functions are not necessary for good fuzzy inference systems. Finally, remember that more details are available on all these functions in the reference section.

Summary of Membership Functions

- Fuzzy sets describe vague concepts (e.g., fast runner, hot weather, weekend days).

- A fuzzy set admits the possibility of partial membership in it. (e.g., Friday is sort of a weekend day, the weather is rather hot).
- The degree an object belongs to a fuzzy set is denoted by a membership value between 0 and 1. (e.g., Friday is a weekend day to the degree 0.8).
- A membership function associated with a given fuzzy set maps an input value to its appropriate membership value.

Logical Operations

Now that you understand the fuzzy inference, you need to see how fuzzy inference connects with logical operations.

The most important thing to realize about fuzzy logical reasoning is the fact that it is a superset of standard Boolean logic. In other words, if you keep the fuzzy values at their extremes of 1 (completely true), and 0 (completely false), standard logical operations will hold. As an example, consider the following standard truth tables.

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

OR

A	not A
0	1
1	0

NOT

Now, because in fuzzy logic the truth of any statement is a matter of degree, can these truth tables be altered? The input values can be real numbers between 0 and 1. What function preserves the results of the AND truth table (for example) and also extend to all real numbers between 0 and 1?

One answer is the *min* operation. That is, resolve the statement A AND B, where A and B are limited to the range (0,1), by using the function $min(A,B)$. Using the same reasoning, you can replace the OR operation with the *max* function, so that A OR B becomes equivalent to $max(A,B)$. Finally, the operation NOT A becomes equivalent to the operation $1-A$. Notice how the previous truth table is completely unchanged by this substitution.

A	B	min(A,B)
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	max(A,B)
0	0	0
0	1	1
1	0	1
1	1	1

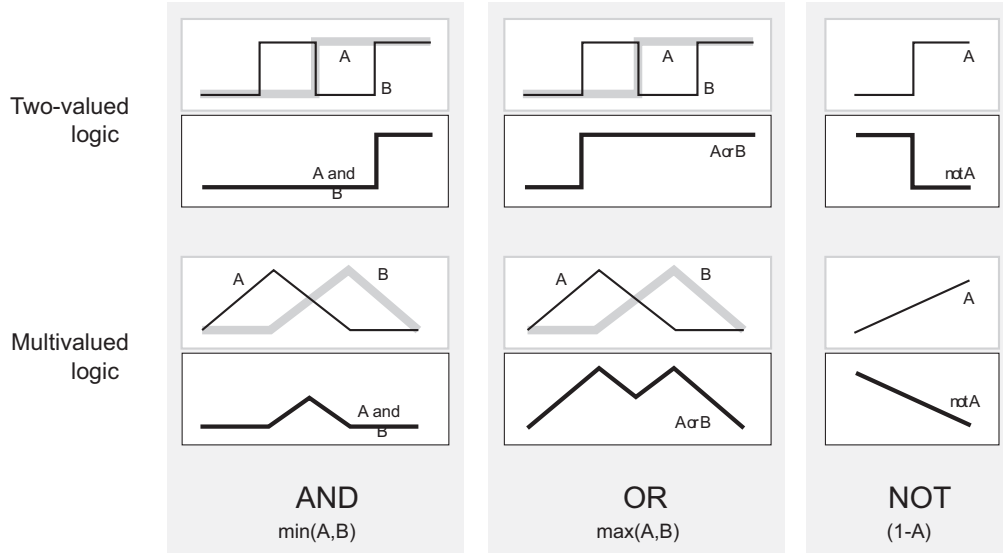
OR

A	1 - A
0	1
1	0

NOT

Moreover, because there is a function behind the truth table rather than just the truth table itself, you can now consider values other than 1 and 0.

The next figure uses a graph to show the same information. In this figure, the truth table is converted to a plot of two fuzzy sets applied together to create one fuzzy set. The upper part of the figure displays plots corresponding to the preceding two-valued truth tables, while the lower part of the figure displays how the operations work over a continuously varying range of truth values *A* and *B* according to the fuzzy operations you have defined.



Given these three functions, you can resolve any construction using fuzzy sets and the fuzzy logical operation AND, OR, and NOT.

Additional Fuzzy Operators

In this case, you defined only one particular correspondence between two-valued and multivalued logical operations for AND, OR, and NOT. This correspondence is by no means unique.

In more general terms, you are defining what are known as the fuzzy intersection or conjunction (AND), fuzzy union or disjunction (OR), and fuzzy complement (NOT). The classical operators for these functions are: AND = *min*, OR = *max*, and NOT = additive complement. Typically, most fuzzy logic applications make use of these operations and leave it at that. In general, however, these functions are arbitrary to a surprising degree. Fuzzy Logic Toolbox uses the classical operator for the fuzzy complement as shown in the previous figure, but also enables you to customize the AND and OR operators.

The intersection of two fuzzy sets *A* and *B* is specified in general by a binary mapping *T*, which aggregates two membership functions as follows:

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

For example, the binary operator *T* may represent the multiplication of

$\mu_A(x)$ and $\mu_B(x)$. These fuzzy intersection operators, which are usually referred to as *T*-norm (Triangular norm) operators, meet the following basic requirements:

A *T-norm* operator is a binary mapping $T(.,.)$ satisfying
boundary: $T(0, 0) = 0$, $T(a, 1) = T(1, a) = a$
monotonicity: $T(a, b) <= T(c, d)$ if $a <= c$ and $b <= d$
commutativity: $T(a, b) = T(b, a)$
associativity: $T(a, T(b, c)) = T(T(a, b), c)$

The first requirement imposes the correct generalization to crisp sets. The second requirement implies that a decrease in the membership values in *A* or *B* cannot produce an increase in the membership value in *A* intersection *B*. The third requirement indicates that the operator is indifferent to the order of the fuzzy sets to be combined. Finally, the fourth requirement allows us to take the intersection of any number of sets in any order of pairwise groupings.

Like fuzzy intersection, the fuzzy union operator is specified in general by a binary mapping S:

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

For example, the binary operator S can represent the addition of

$\mu_A(x)$ and $\mu_B(x)$. These fuzzy union operators, which are often referred to as *T-conorm (or S-norm)* operators, must satisfy the following basic requirements:

A *T-conorm* (or *S-norm*) operator is a binary mapping $S(.,.)$ satisfying
boundary: $S(1, 1) = 1$, $S(a, 0) = S(0, a) = a$
monotonicity: $S(a, b) <= S(c, d)$ if $a <= c$ and $b <= d$
commutativity: $S(a, b) = S(b, a)$
associativity: $S(a, S(b, c)) = S(S(a, b), c)$

Several parameterized *T*-norms and dual *T*-conorms have been proposed in the past, such as those of Yager[19], Dubois and Prade [3], Schweizer and Sklar [14], and Sugeno [15], found in the Appendix B, "Bibliography". Each of these provides a way to vary the gain on the function so that it can be very restrictive or very permissive.

If-Then Rules

Fuzzy sets and fuzzy operators are the subjects and verbs of fuzzy logic. These if-then rule statements are used to formulate the conditional statements that comprise fuzzy logic.

A single fuzzy if-then rule assumes the form

if x is A then y is B

where A and B are linguistic values defined by fuzzy sets on the ranges (universes of discourse) X and Y, respectively. The if-part of the rule "x is A" is called the *antecedent* or premise, while the then-part of the rule "y is B" is called the *consequent* or conclusion. An example of such a rule might be

If service is good then tip is average

The concept *good* is represented as a number between 0 and 1, and so the antecedent is an interpretation that returns a single number between 0 and

1. Conversely, *average* is represented as a fuzzy set, and so the consequent is an assignment that assigns the entire fuzzy set B to the output variable y. In the if-then rule, the word *is* gets used in two entirely different ways depending on whether it appears in the antecedent or the consequent. In MATLAB terms, this usage is the distinction between a relational test using "==" and a variable assignment using the "=" symbol. A less confusing way of writing the rule would be

If service == good then tip = average

In general, the input to an if-then rule is the current value for the input variable (in this case, *service*) and the output is an entire fuzzy set (in this case, *average*). This set will later be *defuzzified,* assigning one value to the output. The concept of defuzzification is described in the next section.

Interpreting an if-then rule involves distinct parts: first evaluating the antecedent (which involves *fuzzifying* the input and applying any necessary *fuzzy operators*) and second applying that result to the consequent (known as *implication*). In the case of two-valued or binary logic, if-then rules do not present much difficulty. If the premise is true, then the conclusion is true. If you relax the restrictions of two-valued logic and let the antecedent be a fuzzy statement, how does this reflect on the conclusion? The answer is a simple one. if the antecedent is true to some degree of membership, then the consequent is also true to that same degree.

Thus:

in binary logic: $p \rightarrow q$ (p and q are either both true or both false.)
in fuzzy logic: $0.5\,p \rightarrow 0.5\,q$ (Partial antecedents provide partial implication.)

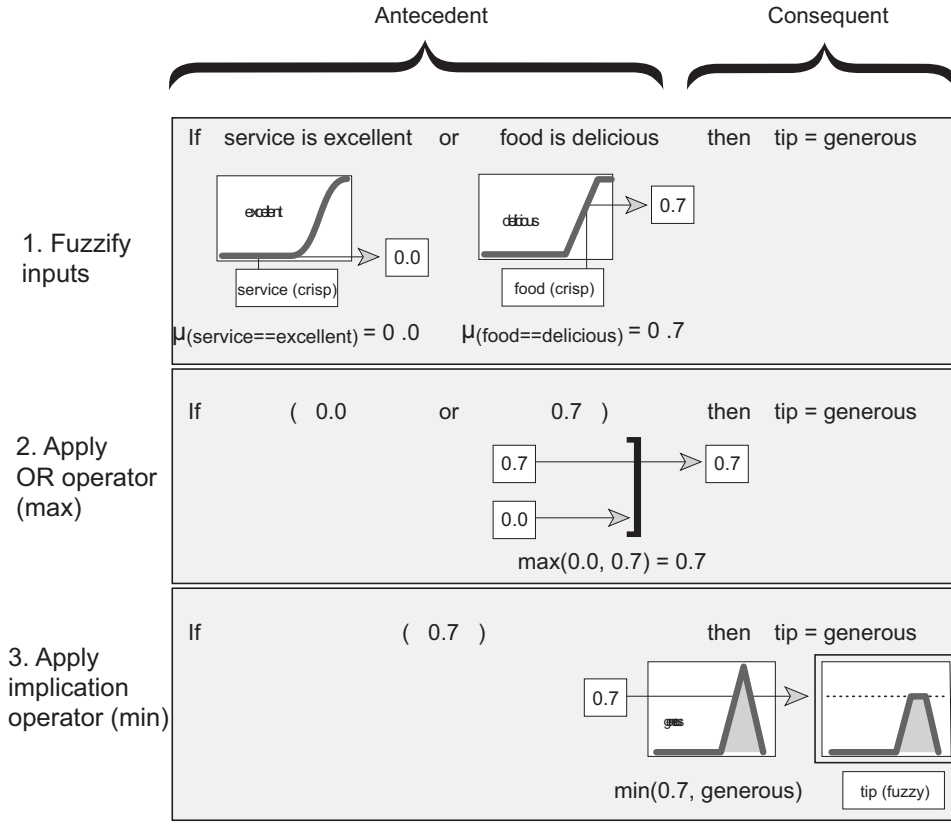
The antecedent of a rule can have multiple parts.

if sky is gray and wind is strong and barometer is falling, then ...

in which case all parts of the antecedent are calculated simultaneously and resolved to a single number using the logical operators described in the preceding section. The consequent of a rule can also have multiple parts.

if temperature is cold then hot water valve is open and cold water valve is shut

in which case all consequents are affected equally by the result of the antecedent. How is the consequent affected by the antecedent? The consequent specifies a fuzzy set be assigned to the output. The *implication function* then modifies that fuzzy set to the degree specified by the antecedent. The most common ways to modify the output fuzzy set are truncation using the `min` function (where the fuzzy set is chopped off as shown in the following figure) or scaling using the `prod` function (where the output fuzzy set is squashed). Both are supported by Fuzzy Logic Toolbox, but you use truncation for the examples in this section.



Summary of If-Then Rules

Interpreting if-then rules is a three-part process. This process is explained in detail in the next section:

1. **Fuzzify inputs**: Resolve all fuzzy statements in the antecedent to a degree of membership between 0 and 1. If there is only one part to the antecedent, then this is the degree of support for the rule.

2. **Apply fuzzy operator to multiple part antecedents**: If there are multiple parts to the antecedent, apply fuzzy logic operators and resolve the antecedent to a single number between 0 and 1. This is the degree of support for the rule.

3. **Apply implication method**: Use the degree of support for the entire rule to shape the output fuzzy set. The consequent of a fuzzy rule assigns an entire fuzzy set to the output. This fuzzy set is represented by a membership function that is chosen to indicate the qualities of the consequent. If the antecedent is only partially true, (i.e., is assigned a value less than 1), then the output fuzzy set is truncated according to the implication method.

In general, one rule alone is not effective. Two or more rules that can play off one another are needed. The output of each rule is a fuzzy set. The output fuzzy sets for each rule are then aggregated into a single output fuzzy set. Finally the resulting set is defuzzified, or resolved to a single number. "Fuzzy Inference Systems" on page 2-20 shows how the whole process works from beginning to end for a particular type of fuzzy inference system called a *Mamdani type*.

Fuzzy Inference Systems

In this section...

"What Are Fuzzy Interference Systems?" on page 2-20

"Tipping Problem Revisited" on page 2-21

"The Fuzzy Inference Diagram" on page 2-27

"Customization" on page 2-29

What Are Fuzzy Interference Systems?

Fuzzy inference is the process of formulating the mapping from a given input to an output using fuzzy logic. The mapping then provides a basis from which decisions can be made, or patterns discerned. The process of fuzzy inference involves all of the pieces that are described in the previous sections: "Membership Functions" on page 2-8, "Logical Operations" on page 2-13, and "If-Then Rules" on page 2-16. There are two types of fuzzy inference systems that can be implemented in Fuzzy Logic Toolbox: Mamdani-type and Sugeno-type. These two types of inference systems vary somewhat in the way outputs are determined. See the Bibliography for references to descriptions of these two types of fuzzy inference systems, [8], [11], [16].

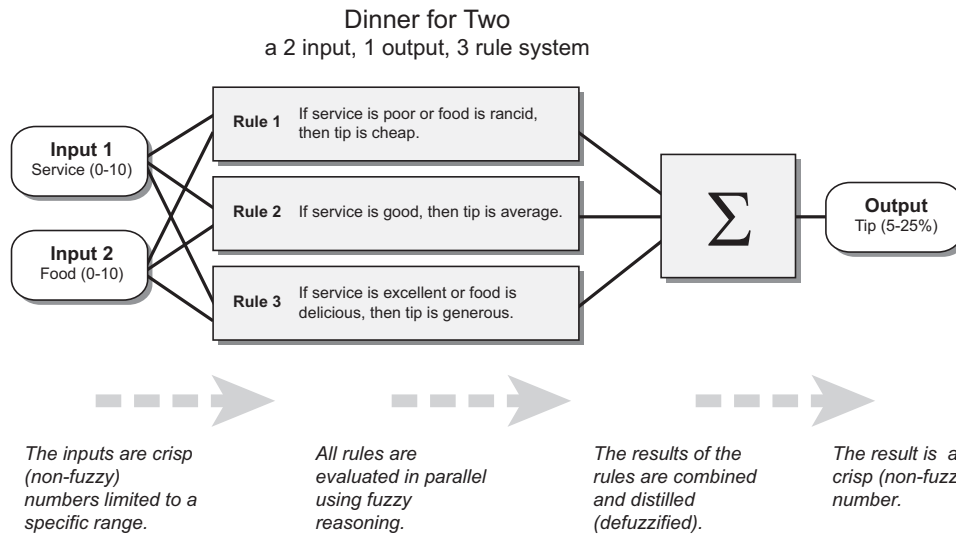
Fuzzy inference systems have been successfully applied in fields such as automatic control, data classification, decision analysis, expert systems, and computer vision. Because of its multidisciplinary nature, fuzzy inference systems are associated with a number of names, such as fuzzy-rule-based systems, fuzzy expert systems, fuzzy modeling, fuzzy associative memory, fuzzy logic controllers, and simply (and ambiguously) fuzzy systems.

Mamdani's fuzzy inference method is the most commonly seen fuzzy methodology. Mamdani's method was among the first control systems built using fuzzy set theory. It was proposed in 1975 by Ebrahim Mamdani [11] as an attempt to control a steam engine and boiler combination by synthesizing a set of linguistic control rules obtained from experienced human operators. Mamdani's effort was based on Lotfi Zadeh's 1973 paper on fuzzy algorithms for complex systems and decision processes [22]. Although the inference process described in the next few sections differs somewhat from the methods described in the original paper, the basic idea is much the same.

Mamdani-type inference, as defined for Fuzzy Logic Toolbox, expects the output membership functions to be fuzzy sets. After the aggregation process, there is a fuzzy set for each output variable that needs defuzzification. it is possible, and in many cases much more efficient, to use a single spike as the output membership function rather than a distributed fuzzy set. This type of output is sometimes known as a *singleton* output membership function, and it can be thought of as a pre-defuzzified fuzzy set. It enhances the efficiency of the defuzzification process because it greatly simplifies the computation required by the more general Mamdani method, which finds the centroid of a two-dimensional function. Rather than integrating across the two-dimensional function to find the centroid, you use the weighted average of a few data points. Sugeno-type systems support this type of model. In general, Sugeno-type systems can be used to model any inference system in which the output membership functions are either linear or constant.

Tipping Problem Revisited

This section reviews the two-input, one-output, three-rule tipping problem that you saw in the introduction, only in more detail. The basic structure of this example is shown in the following diagram.



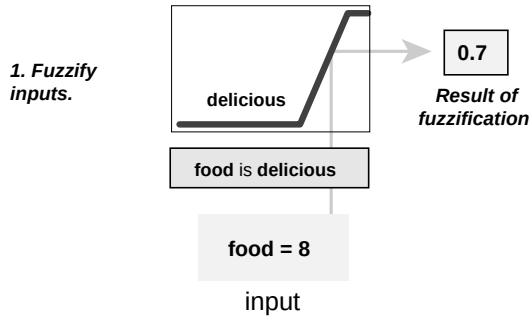
Information flows from left to right, from two inputs to a single output. The parallel nature of the rules is one of the more important aspects of fuzzy logic systems. Instead of sharp switching between modes based on breakpoints, logic flows smoothly from regions where the system's behavior is dominated by either one rule or another.

In Fuzzy Logic Toolbox, there are five parts of the fuzzy inference process: fuzzification of the input variables, application of the fuzzy operator (AND or OR) in the antecedent, implication from the antecedent to the consequent, aggregation of the consequents across the rules, and defuzzification. These sometimes cryptic and odd names have very specific meaning that are defined in the following steps.

Step 1. Fuzzify Inputs

The first step is to take the inputs and determine the degree to which they belong to each of the appropriate fuzzy sets via membership functions. In Fuzzy Logic Toolbox, the input is always a crisp numerical value limited to the universe of discourse of the input variable (in this case the interval between 0 and 10) and the output is a fuzzy degree of membership in the qualifying linguistic set (always the interval between 0 and 1). Fuzzification of the input amounts to either a table lookup or a function evaluation.

This example is built on three rules, and each of the rules depends on resolving the inputs into a number of different fuzzy linguistic sets: service is poor, service is good, food is rancid, food is delicious, and so on. Before the rules can be evaluated, the inputs must be fuzzified according to each of these linguistic sets. For example, to what extent is the food really delicious? The following figure shows how well the food at the hypothetical restaurant (rated on a scale of 0 to 10) qualifies, (via its membership function), as the linguistic variable delicious. In this case, we rated the food as an 8, which, given your graphical definition of delicious, corresponds to $\mu = 0.7$ for the delicious membership function.



In this manner, each input is fuzzified over all the qualifying membership functions required by the rules.

Step 2. Apply Fuzzy Operator

After the inputs are fuzzified, you know the degree to which each part of the antecedent is satisfied for each rule. If the antecedent of a given rule has more than one part, the fuzzy operator is applied to obtain one number that represents the result of the antecedent for that rule. This number is then applied to the output function. The input to the fuzzy operator is two or more membership values from fuzzified input variables. The output is a single truth value.

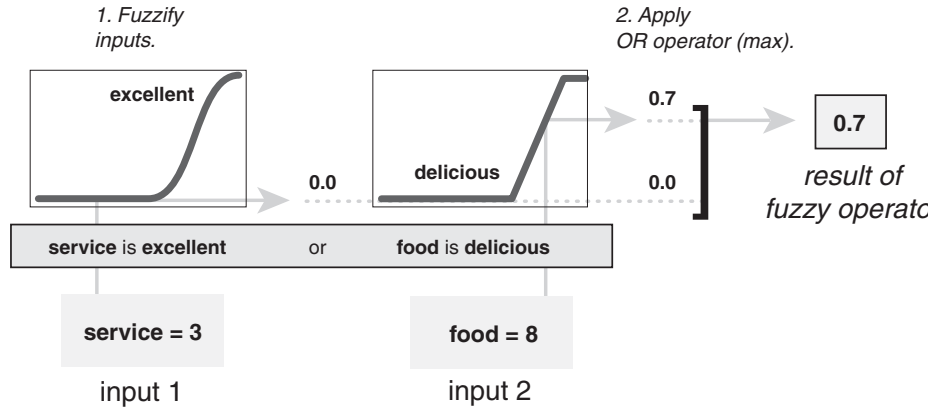
As is described in "Logical Operations" on page 2-13 section, any number of well-defined methods can fill in for the AND operation or the OR operation. In Fuzzy Logic Toolbox, two built-in AND methods are supported: *min* (minimum) and *prod* (product). Two built-in OR methods are also supported: *max* (maximum), and the probabilistic OR method *probor*. The probabilistic OR method (also known as the algebraic sum) is calculated according to the equation

$$probor(a,b) = a + b - ab$$

In addition to these built-in methods, you can create your own methods for AND and OR by writing any function and setting that to be your method of choice.

The following figure shows the OR operator *max* at work, evaluating the antecedent of the rule 3 for the tipping calculation. The two different pieces of the antecedent (service is excellent and food is delicious) yielded the fuzzy

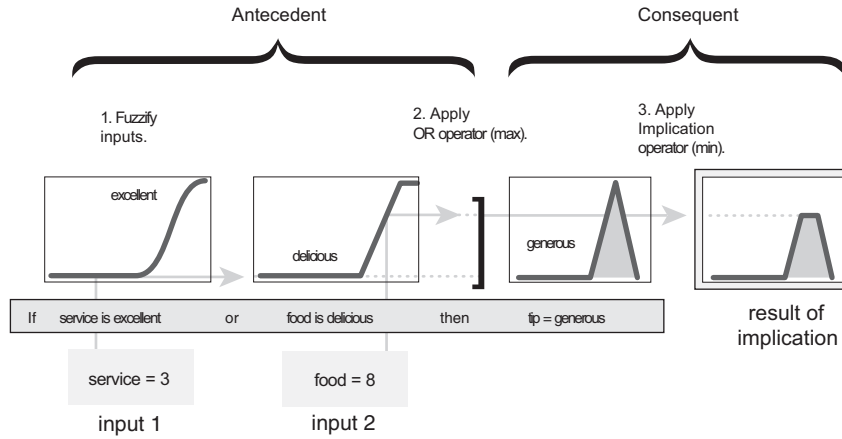
membership values 0.0 and 0.7 respectively. The fuzzy OR operator simply selects the maximum of the two values, 0.7, and the fuzzy operation for rule 3 is complete. The probabilistic OR method would still result in 0.7.



Step 3. Apply Implication Method

Before applying the implication method, you must determine the rule's weight. Every rule has a *weight* (a number between 0 and 1), which is applied to the number given by the antecedent. Generally, this weight is 1 (as it is for this example) and thus has no effect at all on the implication process. From time to time you may want to weight one rule relative to the others by changing its weight value to something other than 1.

After proper weighting has been assigned to each rule, the implication method is implemented. A consequent is a fuzzy set represented by a membership function, which weights appropriately the linguistic characteristics that are attributed to it. The consequent is reshaped using a function associated with the antecedent (a single number). The input for the implication process is a single number given by the antecedent, and the output is a fuzzy set. Implication is implemented for each rule. Two built-in methods are supported, and they are the same functions that are used by the AND method: *min* (minimum), which truncates the output fuzzy set, and *prod* (product), which scales the output fuzzy set.



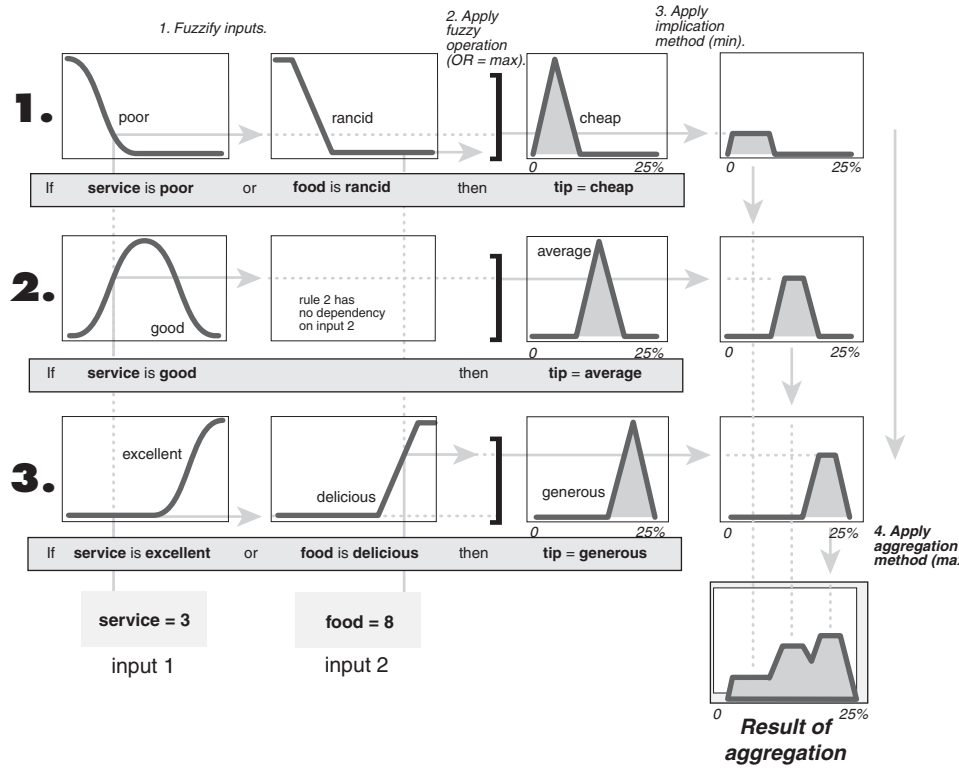
Step 4. Aggregate All Outputs

Because decisions are based on the testing of all of the rules in a FIS, the rules must be combined in some manner in order to make a decision. Aggregation is the process by which the fuzzy sets that represent the outputs of each rule are combined into a single fuzzy set. Aggregation only occurs once for each output variable, just prior to the fifth and final step, defuzzification. The input of the aggregation process is the list of truncated output functions returned by the implication process for each rule. The output of the aggregation process is one fuzzy set for each output variable.

As long as the aggregation method is commutative (which it always should be), then the order in which the rules are executed is unimportant. Three built-in methods are supported:

- `max` (maximum)
- `probor` (probabilistic OR)
- `sum` (simply the sum of each rule's output set)

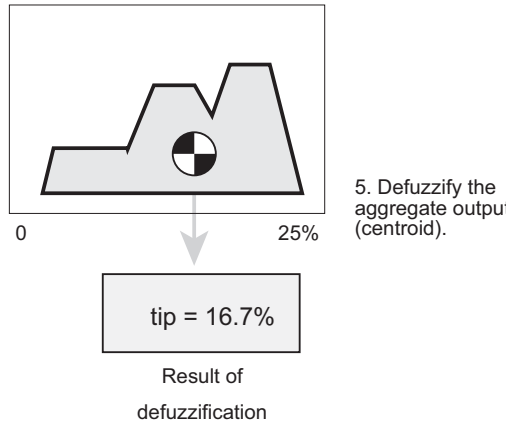
In the following diagram, all three rules have been placed together to show how the output of each rule is combined, or aggregated, into a single fuzzy set whose membership function assigns a weighting for every output (tip) value.



Step 5. Defuzzify

The input for the defuzzification process is a fuzzy set (the aggregate output fuzzy set) and the output is a single number. As much as fuzziness helps the rule evaluation during the intermediate steps, the final desired output for each variable is generally a single number. However, the aggregate of a fuzzy set encompasses a range of output values, and so must be defuzzified in order to resolve a single output value from the set.

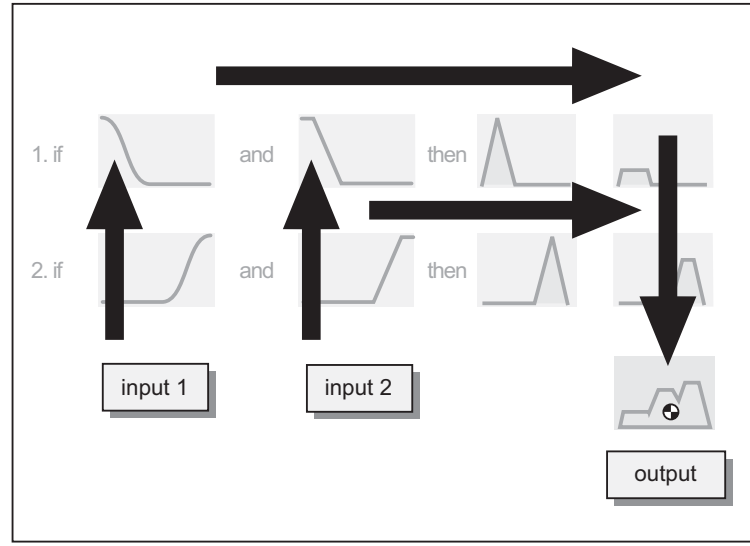
Perhaps the most popular defuzzification method is the centroid calculation, which returns the center of area under the curve. There are five built-in methods supported: centroid, bisector, middle of maximum (the average of the maximum value of the output set), largest of maximum, and smallest of maximum.



The Fuzzy Inference Diagram

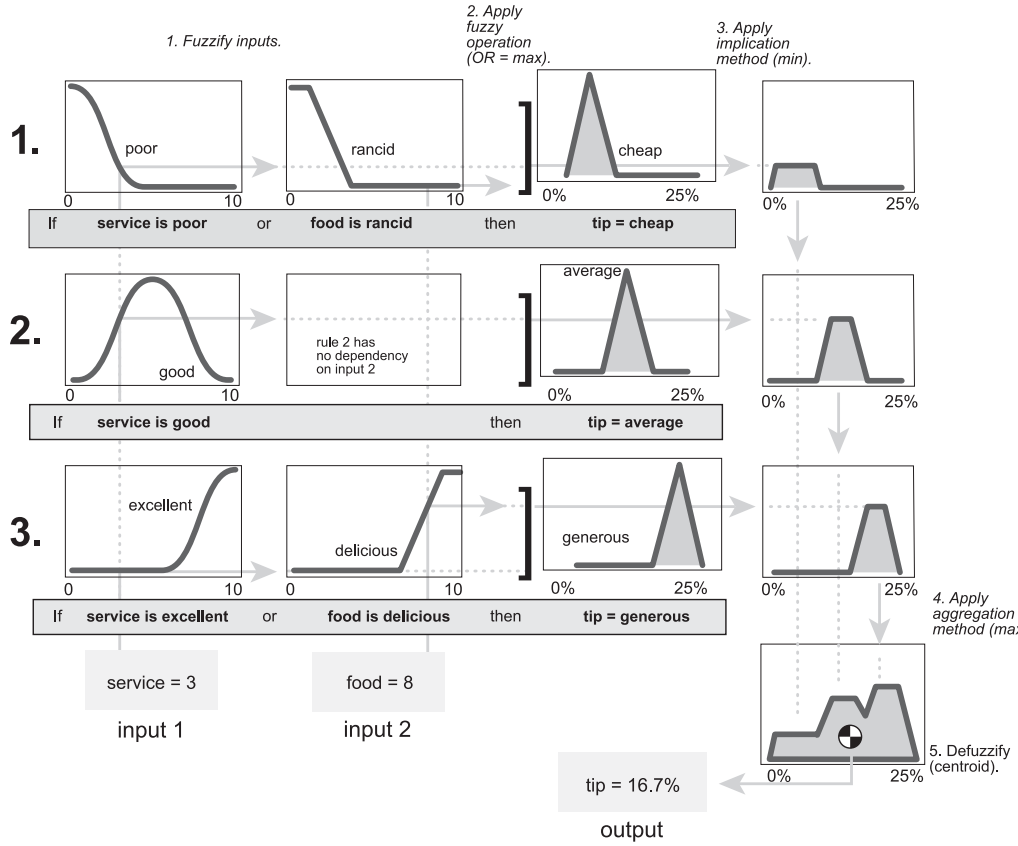
The fuzzy inference diagram is the composite of all the smaller diagrams presented so far in this section. It simultaneously displays all parts of the fuzzy inference process you have examined. Information flows through the fuzzy inference diagram as shown in the following figure.

Interpreting the fuzzy inference diagram



In this figure, the flow proceeds up from the inputs in the lower left, then across each row, or rule, and then down the rule outputs to finish in the lower right. This compact flow shows everything at once, from linguistic variable fuzzification all the way through defuzzification of the aggregate output.

The following figure shows the actual full-size fuzzy inference diagram. There is a lot to see in a fuzzy inference diagram, but after you become accustomed to it, you can learn a lot about a system very quickly. For instance, from this diagram with these particular inputs, you can easily see that the implication method is truncation with the *min* function. The *max* function is being used for the fuzzy OR operation. Rule 3 (the bottom-most row in the diagram shown previously) is having the strongest influence on the output. and so on. The Rule Viewer described in "The Rule Viewer" on page 2-49 is a MATLAB implementation of the fuzzy inference diagram.



Customization

One of the primary goals of Fuzzy Logic Toolbox is to have an open and easily modified fuzzy inference system structure. Thus, Fuzzy Logic Toolbox is designed to give you as much freedom as possible, within the basic constraints of the process described, to customize the fuzzy inference process for your application. For example, you can substitute your own MATLAB functions for any of the default functions used in the five steps detailed previously: you make your own membership functions, AND methods, OR methods, implication methods, aggregation methods, and defuzzification methods. "Building Systems with Fuzzy Logic Toolbox" on page 2-30 describes exactly how to build and implement a fuzzy inference system using the tools provided.

Building Systems with Fuzzy Logic Toolbox

In this section...

"Another Tipping Problem" on page 2-30
"Getting Started" on page 2-33
"The FIS Editor" on page 2-34
"The Membership Function Editor" on page 2-39
"The Rule Editor" on page 2-45
"The Rule Viewer" on page 2-49
"The Surface Viewer" on page 2-52
"Importing and Exporting from the GUI Tools" on page 2-53
"Customizing Your Fuzzy System" on page 2-54

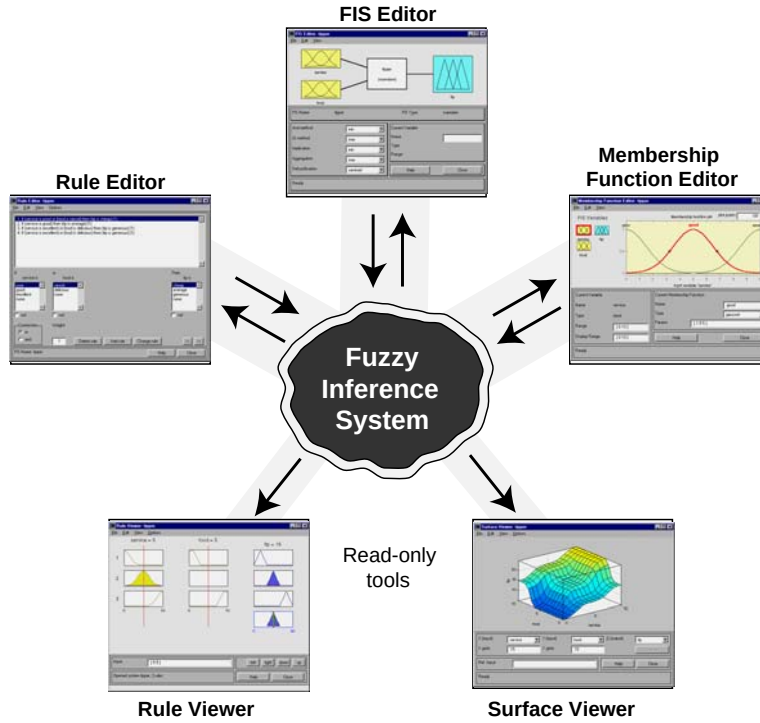
Another Tipping Problem

Now, you are going to work through a similar tipping example, only we'll be building it using the graphical user interface (GUI) tools provided by Fuzzy Logic Toolbox. Although it is possible to use Fuzzy Logic Toolbox by working strictly from the command line, in general it is much easier to build a system graphically. There are five primary GUI tools for building, editing, and observing fuzzy inference systems in Fuzzy Logic Toolbox:

- Fuzzy Inference System (FIS) Editor
- Membership Function Editor
- Rule Editor
- Rule Viewer
- Surface Viewer

These GUIs are dynamically linked, in that changes you make to the FIS using one of them, can affect what you see on any of the other open GUIs. You can have any or all of them open for any given system.

In addition to these five primary GUIs, the toolbox includes the graphical ANFIS Editor GUI, which is used for building and analyzing Sugeno-type adaptive neural fuzzy inference systems. The ANFIS Editor GUI is discussed in “Sugeno-Type Fuzzy Inference” on page 2-83.



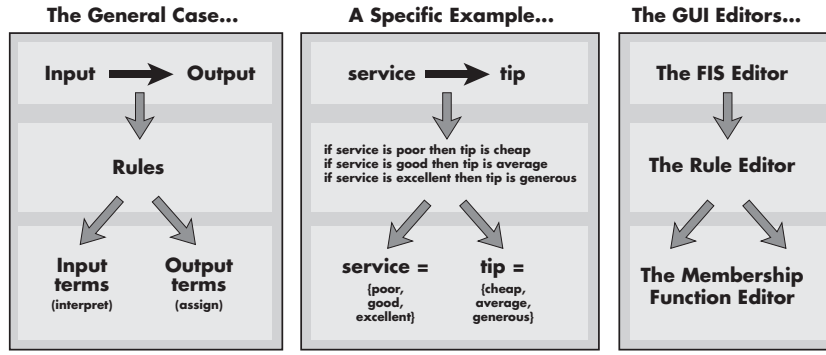
The FIS Editor handles the high-level issues for the system: How many input and output variables? What are their names? Fuzzy Logic Toolbox does not limit the number of inputs. However, the number of inputs may be limited by the available memory of your machine. If the number of inputs is too large, or the number of membership functions is too big, then it may also be difficult to analyze the FIS using the other GUI tools.

The Membership Function Editor is used to define the shapes of all the membership functions associated with each variable.

The Rule Editor is for editing the list of rules that defines the behavior of the system.

The Rule Viewer and the Surface Viewer are used for looking at, as opposed to editing, the FIS. They are strictly read-only tools. The Rule Viewer is a MATLAB based display of the fuzzy inference diagram shown at the end of the last section. Used as a diagnostic, it can show (for example) which rules are active, or how individual membership function shapes are influencing the results. The Surface Viewer is used to display the dependency of one of the outputs on any one or two of the inputs—that is, it generates and plots an output surface map for the system.

This section began with an illustration similar to the following one describing the main parts of a fuzzy inference system, only the next one shows how the three editors fit together. The two viewers examine the behavior of the entire system.



The five primary GUIs can all interact and exchange information. Any one of them can read and write both to the workspace and to a file (the read-only viewers can still exchange plots with the workspace and save them to a file). For any fuzzy inference system, any or all of these five GUIs may be open. If more than one of these editors is open for a single system, the various GUI windows are aware of the existence of the others, and, if necessary, updates related windows. Thus, if the names of the membership functions are changed using the Membership Function Editor, those changes are reflected in the rules shown in the Rule Editor. The editors for any number of different FIS systems may be open simultaneously. The FIS Editor, the Membership Function Editor, and the Rule Editor can all read and modify the FIS data,

but the Rule Viewer and the Surface Viewer do not modify the FIS data in any way.

Getting Started

We'll start with a basic description of a two-input, one-output tipping problem (based on tipping practices in the U.S.).

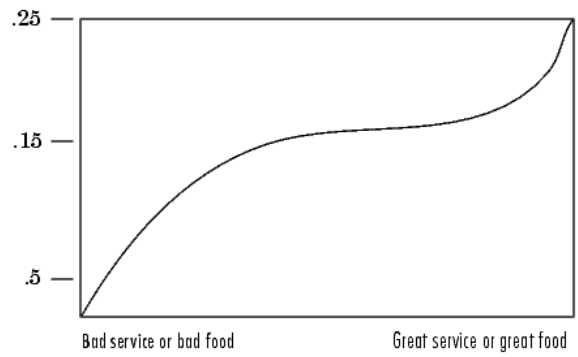
The Basic Tipping Problem

Given a number between 0 and 10 that represents the quality of service at a restaurant (where 10 is excellent), and another number between 0 and 10 that represents the quality of the food at that restaurant (again, 10 is excellent), what should the tip be?

The starting point is to write down the three golden rules of tipping, based on years of personal experience in restaurants.

1. *If the service is poor or the food is rancid, then tip is cheap.*
2. *If the service is good, then tip is average.*
3. *If the service is excellent or the food is delicious, then tip is generous.*

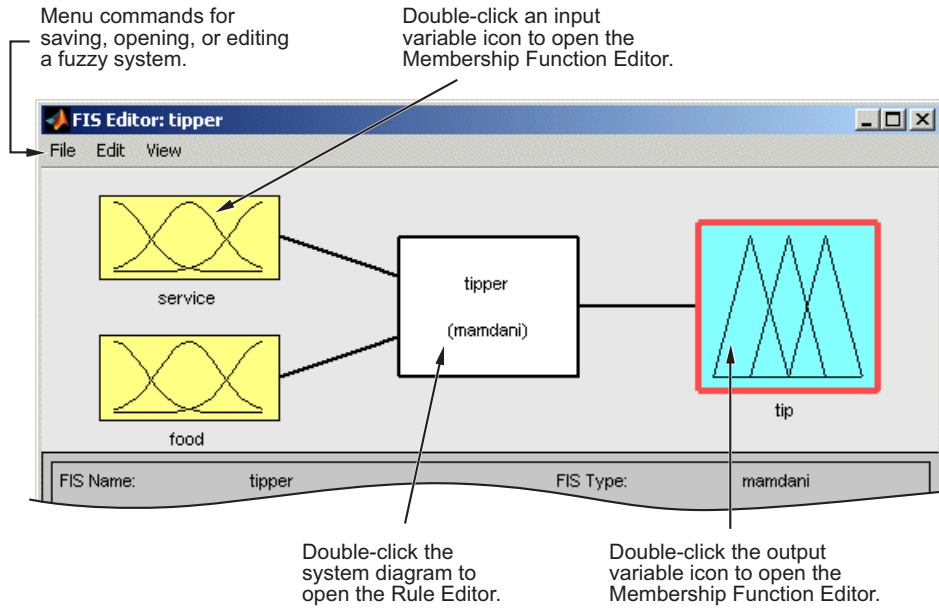
Assume that an average tip is 15%, a generous tip is 25%, and a cheap tip is 5%.

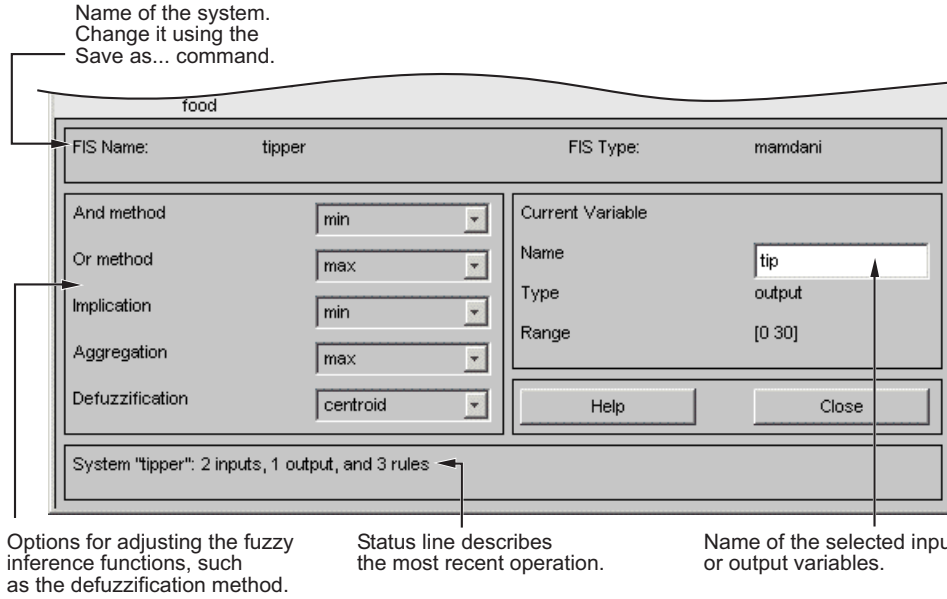


Obviously the numbers and the shape of the curve are subject to local traditions, cultural bias, and so on, but the three rules are generally universal.

Now that you know the rules and have an idea of what the output should look like, begin working with the GUI tools to construct a fuzzy inference system for this decision process.

The FIS Editor





The following discussion tells you how to build a new fuzzy inference system from scratch. If you want to save time and follow along quickly, you can load the prebuilt system by typing

```
fuzzy tipper
```

This command loads the FIS associated with the file `tipper.fis` (the `.fis` is implied) and launches the FIS Editor. However, if you load the prebuilt system, you will not build rules or construct membership functions.

The FIS Editor displays general information about a fuzzy inference system. There is a simple diagram at the top that shows the names of each input variable on the left, and those of each output variable on the right. The sample membership functions shown in the boxes are just icons and do not depict the actual shapes of the membership functions:

- Below the diagram is the name of the system and the type of inference used. The default, Mamdani-type inference, is what is described so far and what you continue to use for this example. Another slightly different type of inference, called Sugeno-type inference, is also available. This method is explained in “Sugeno-Type Fuzzy Inference” on page 2-83.

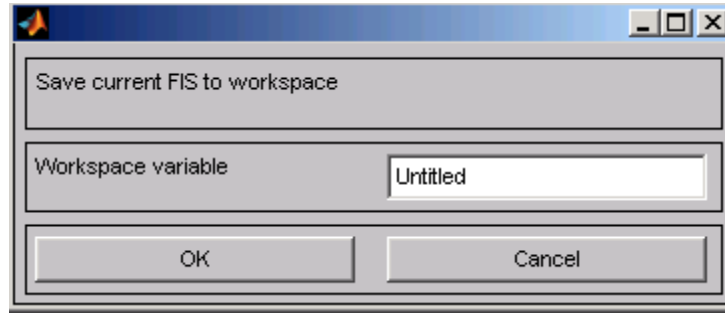
- Below the name of the fuzzy inference system, on the left side of the figure, are the pop-up menus that allow you to modify the various pieces of the inference process.
- On the right side at the bottom of the figure is the area that displays the name of either an input or output variable, its associated membership function type, and its range.
- The latter two fields are specified only after the membership functions have been.
- Below that region are the **Help** and **Close** buttons that call up online help and close the window, respectively. At the bottom is a status line that relays information about the system.

To start this system from scratch, type

```
fuzzy
```

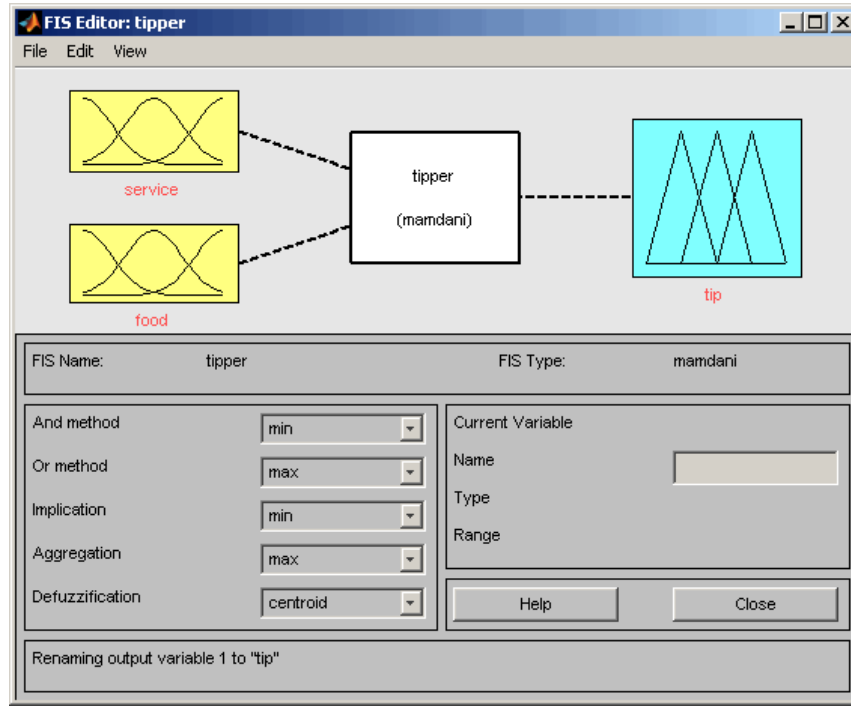
at the MATLAB prompt. The generic untitled FIS Editor opens, with one input, labeled **input1**, and one output, labeled **output1**. For this example, you construct a two-input, one output system. A second yellow box labeled **input2** appears. The two inputs in your example are **service** and **food**. The one output is **tip**. You want to change the variable names to reflect these designations:

1. Select **Edit > Add variable > Input**.
2. Click the yellow box **input1**. This box is highlighted with a red outline.
3. Edit the **Name** field from `input1` to `service`, and press **Enter**.
4. Click the yellow box **input2**. This box is highlighted with a red outline.
5. Edit the **Name** field from `input2` to `food`, and press **Enter**.
6. Click the blue box **output1**.
7. Edit the **Name** field from `output1` to `tip`, and press **Enter**.
8. Select **File > Export > To Workspace**.



9 Enter the **Workspace variable** name `tipper`, and click **OK**.

The diagram is updated to reflect the new names of the input and output variables. There is now a new variable in the workspace called `tipper` that contains all the information about this system. By saving to the workspace with a new name, you also rename the entire system. Your window looks something like the following diagram.

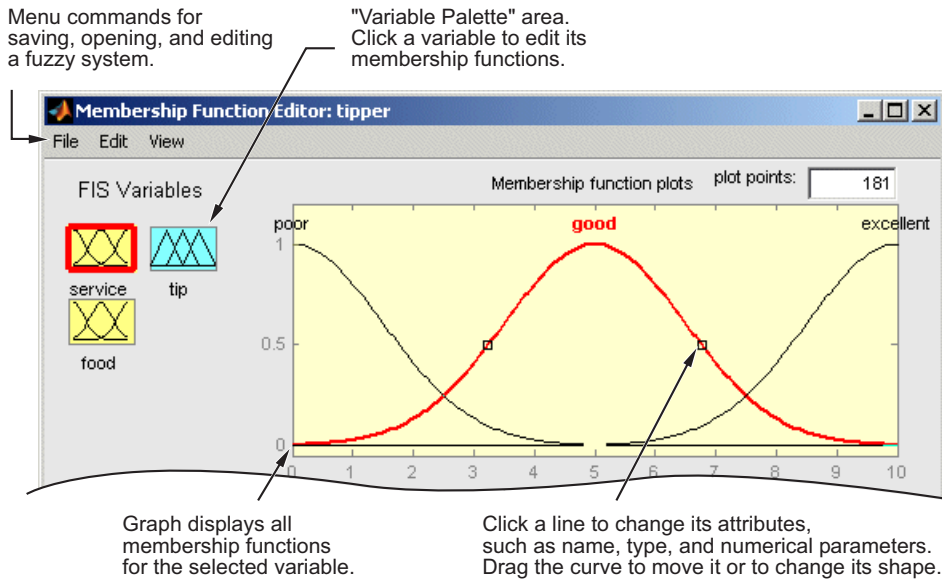


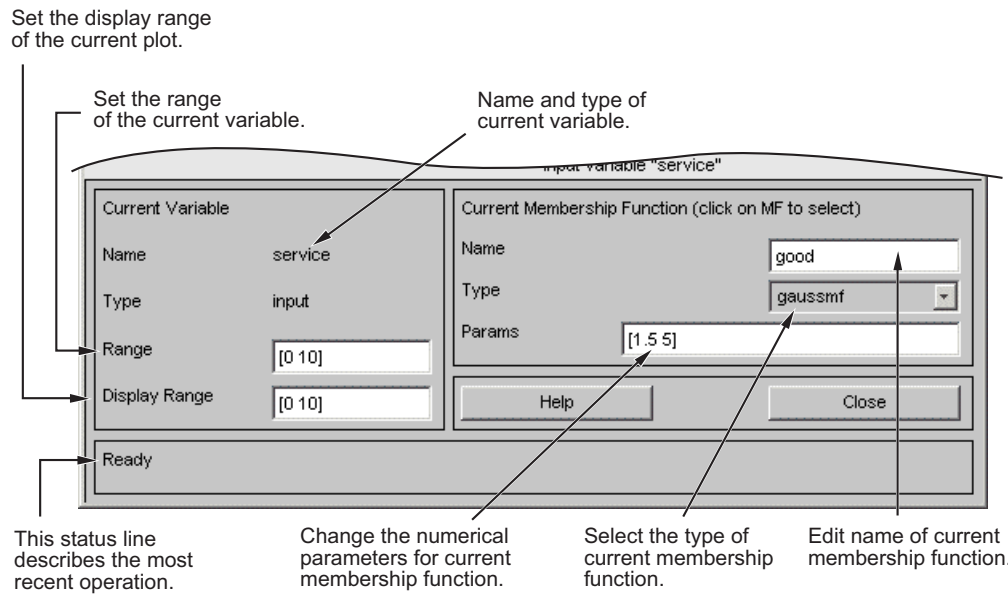
Leave the inference options in the lower left in their default positions for now. You have entered all the information you need for this particular GUI. Next, define the membership functions associated with each of the variables. To do this, open the Membership Function Editor.

You can open the Membership Function Editor in one of three ways:

- Within the FIS Editor window, select **Edit > Membership Functions.**.
- Within the FIS Editor window, double-click the blue icon called **tip**.
- At the command line, type `mfedit`.

The Membership Function Editor





The Membership Function Editor shares some features with the FIS Editor. In fact, all of the five basic GUI tools have similar menu options, status lines, and **Help** and **Close** buttons.

The Membership Function Editor is the tool that lets you display and edit all of the membership functions associated with all of the input and output variables for the entire fuzzy inference system.

When you open the Membership Function Editor to work on a fuzzy inference system that does not already exist in the workspace, there are no membership functions associated with the variables that you defined with the FIS Editor.

On the upper-left side of the graph area in the Membership Function Editor is a "Variable Palette" that lets you set the membership functions for a given variable.

To set up the membership functions associated with an input or an output variable for the FIS, select a FIS variable in this region by clicking it.

Next select the **Edit** pull-down menu, and choose **Add MFs ..** A new window appears, which allows you to select both the membership function type and the number of membership functions associated with the selected variable. In the lower-right corner of the window are the controls that let you change the name, type, and parameters (shape), of the membership function, after it is selected.

The membership functions from the current variable are displayed in the main graph. These membership functions can be manipulated in two ways. You can first use the mouse to select a particular membership function associated with a given variable quality, (such as poor, for the variable, service), and then drag the membership function from side to side. This action affects the mathematical description of the quality associated with that membership function for a given variable. The selected membership function can also be tagged for dilation or contraction by clicking on the small square drag points on the membership function, and then dragging the function with the mouse toward the *outside*, for dilation, or toward the *inside*, for contraction. This action changes the parameters associated with that membership function.

Below the Variable Palette is some information about the type and name of the current variable. There is a text field in this region that lets you change

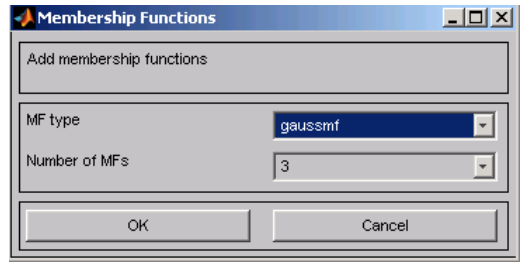
the limits of the current variable's range (universe of discourse) and another that lets you set the limits of the current plot (which has no real effect on the system).

The process of specifying the input membership functions for this two input tipper problem is as follows:

1. Select the **input** variable, `service`, by double-clicking on it. Set both the **Range** and the **Display Range** to the vector `[0 10]`.

2. Select **Remove All MFs** from the **Edit** menu. This removes all existing Membership Functions from the Membership Function Editor.

3. Select **Add MFs.** from the **Edit** menu. The following window opens:



4. Use the tab to choose `gaussmf` for **MF Type** and `3` for **Number of MFs**. This choice adds three Gaussian curves to the input variable service.

5. Click once on the curve with the left-most *hump*. Change the name of the curve to `poor`. To adjust the shape of the membership function, either use the mouse, as described previously, or type in a desired parameter change, and then click on the **membership function**. The desired parameter **Params** listing for this curve is `[1.5 0]`. The two inputs of **Params** represent the standard deviation and center for the Gaussian curve.

6. Name the curve with the middle hump, `good`. The desired parameter listing for this curve is `[1.5 5]`. Name the curve with the rightmost hump, `excellent`. The desired parameter for this curve is `[1.5 10]`.

7. Select the **input** variable, **food**, by clicking on it. Set both the **Range** and the **Display Range** to the vector `[0 10]`.

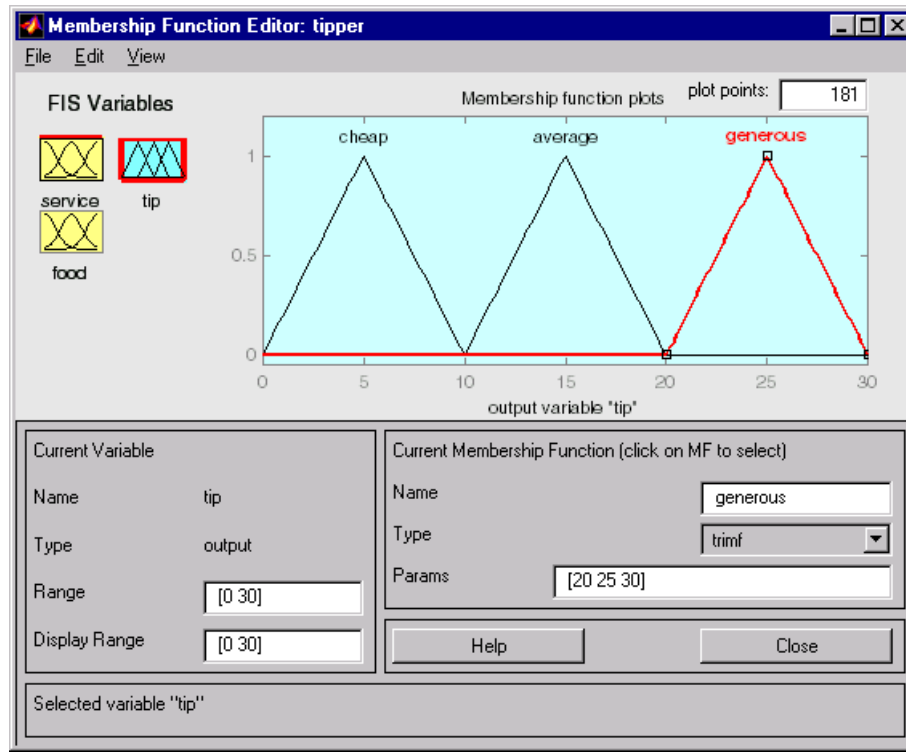
8. Select **Add MFs.** from the **Edit** menu and add two `trapmf` curves to the input variable `food`.

9. Click once directly on the curve with the leftmost trapezoid. Change the name of the curve to `rancid`. To adjust the shape of the membership function, either use the mouse, as described previously, or type in a desired parameter change, and then click on the **membership function**. The default parameter listing for this curve is `[0 0 1 3]`.

10. Name the curve with the rightmost trapezoid, `delicious`, and reset the associated parameters if desired.

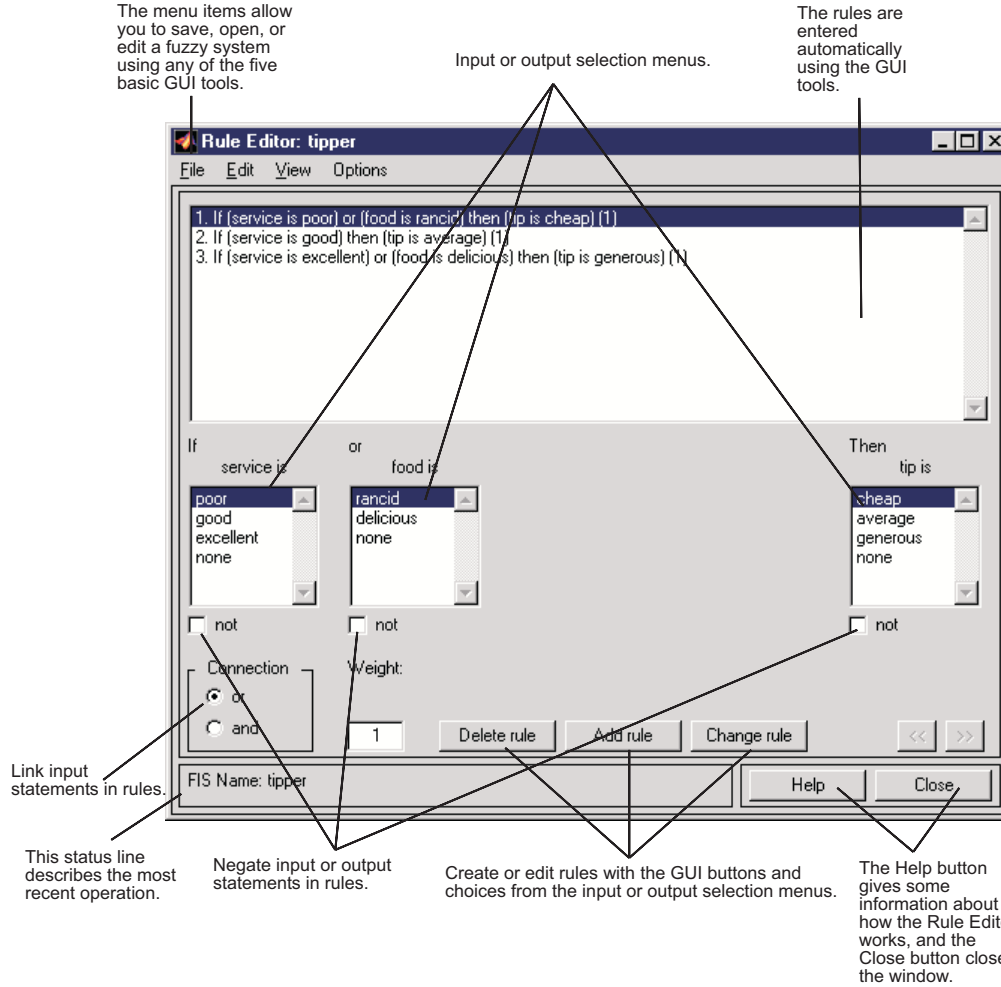
Next, create the membership functions for the output variable, **tip**. To create the output variable membership functions, use the Variable Palette on the left, selecting the output variable, **tip**. The inputs ranged from 0 to 10, but the output scale is going to be a tip between 5% and 25%.

Use triangular membership function types for the output. First, set the **Range** (and the **Display Range**) to `[0 30]`, to cover the output range. Initially, the *cheap* membership function has the parameters `[0 5 10]`, the *average* membership function are`[10 15 20]`, and the *generous* membership function are `[20 25 30]`. Your system should look similar to the following figure.



Now that the variables have been named and the membership functions have appropriate shapes and names, you can enter the rules. To call up the Rule Editor, go to the **Edit** menu and select **Rules**, or type `ruleedit` at the command line.

The Rule Editor



Constructing rules using the graphical Rule Editor interface is fairly self evident. Based on the descriptions of the input and output variables defined with the FIS Editor, the Rule Editor allows you to construct the rule statements automatically, From the GUI, you can:

- Create rules by selecting an item in each input and output variable box, and one **Connection** item and clicking **Add Rule**. You can choose none as

one of the variable qualities to exclude that variable from a given rule and choose `not` under any variable name to negate the associated quality.

- Delete a rule by selecting the rule and clicking **Delete Rule**.
- Edit a rule by changing the selection in the variable box and clicking **Change Rule**.
- Specify weight to a rule by typing in a desired number between `0` and `1` in **Weight**. If you do not specify the weight, it is assumed to be unity `(1)`.

Similar to those in the FIS Editor and the Membership Function Editor, the Rule Editor has the menu bar and the status line. The menu items allow you to open, close, save and edit a fuzzy system using the five basic GUI tools. From the menu, you can also:

- Set the format for the display by selecting **Options > Format**.
- Set the language by selecting **Options > Language**.

You can access information about the Rule Editor by clicking **Help** and close the GUI using **Close**.

To insert the first rule in the Rule Editor, select the following:

- `poor` under the variable **service**
- `rancid` under the variable **food**
- The **or** radio button, in the **Connection** block
- `cheap`, under the output variable, **tip**.

The resulting rule is

1. *If (service is poor) or (food is rancid) then (tip is cheap) (1)*

The numbers in the parentheses represent weights.

Follow a similar procedure to insert the second and third rules in the Rule Editor to get

1. *If (service is poor) or (food is rancid) then (tip is cheap) (1)*
2. *If (service is good) then (tip is average) (1)*
3. *If (service is excellent) or (food is delicious) then (tip is generous) (1)*

To change a rule, first click on the rule to be changed. Next make the desired changes to that rule, and then click **Change rule**. For example, to change the first rule to

1. *If (service not poor) or (food not rancid) then (tip is not cheap) (1)*

Select the **not** check box under each variable, and then click **Change rule**.

The **Format** pop-up menu from the **Options** menu indicates that you are looking at the verbose form of the rules. Try changing it to `symbolic`. You will see

1. *(service==poor) | (food==rancid) => (tip=cheap) (1)*
2. *(service==good) => (tip=average) (1)*
3. *(service==excellent) | (food==delicious) => (tip=generous) (1)*

There is not much difference in the display really, but it is slightly more language neutral, because it does not depend on terms like *if* and *then*. If you change the format to indexed, you see an extremely compressed version of the rules.

1 1, 1 (1) : 2
2 0, 2 (1) : 1
3 2, 3 (1) : 2

This is the version of the rules that the machine deals with.

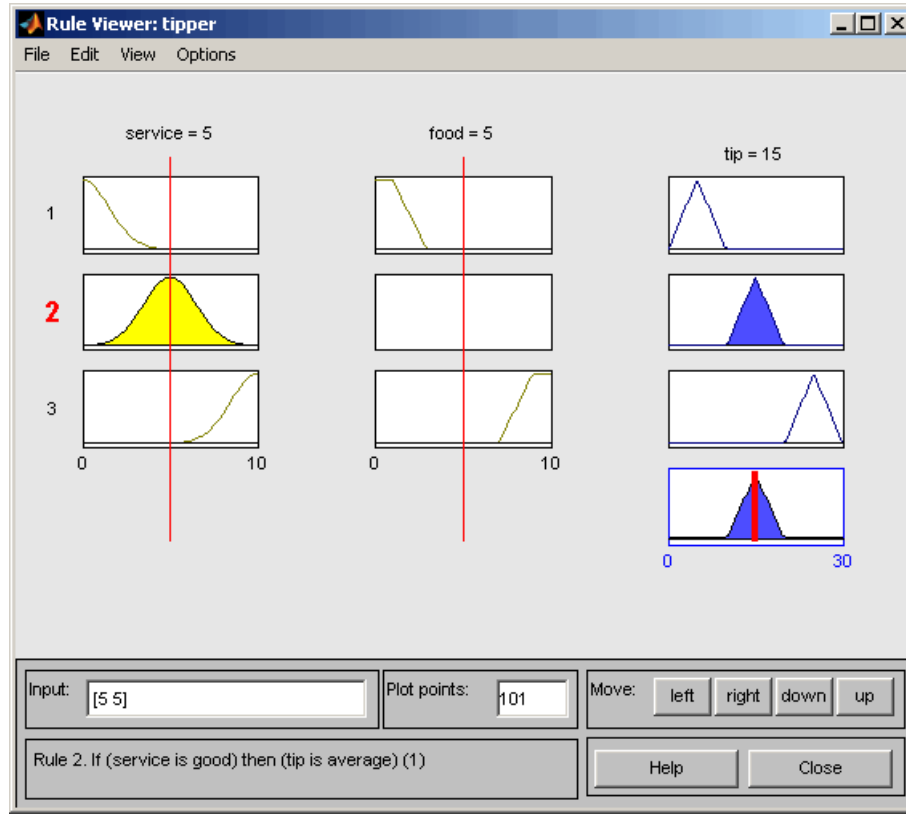
- The first column in this structure corresponds to the input variables.
- The second column corresponds to the output variable.
- The third column displays the weight applied to each rule.
- The fourth column is shorthand that indicates whether this is an OR (2) rule or an AND (1) rule.
- The numbers in the first two columns refer to the index number of the membership function.

A literal interpretation of rule 1 is "If input 1 is MF1 (the first membership function associated with input 1) or if input 2 is MF1, then output 1 should be MF1 (the first membership function associated with output 1) with the weight 1.

The symbolic format does not consider the terms, *if*, *then*, and so on. The indexed format doesn't even bother with the names of your variables. Obviously the functionality of your system doesn't depend on how well you have named your variables and membership functions. The whole point of naming variables descriptively is, as always, making the system easier for you to interpret. Thus, unless you have some special purpose in mind, it is probably be easier for you to continue with the **verbose** format.

At this point, the fuzzy inference system has been completely defined, in that the variables, membership functions, and the rules necessary to calculate tips are in place. Now, look at the fuzzy inference diagram presented at the end of the previous section and verify that everything is behaving the way you think it should. You can use the Rule Viewer, the next of the GUI tools we'll look at. From the **View** menu, select **Rules**.

The Rule Viewer



The Rule Viewer displays a roadmap of the whole fuzzy inference process. It is based on the fuzzy inference diagram described in the previous section. You see a single figure window with 10 plots nested in it. The three plots across the top of the figure represent the antecedent and consequent of the first rule. Each rule is a row of plots, and each column is a variable. The rule numbers are displayed on the left of each row. You can click on a rule number to view the rule in the status line.

- The first two columns of plots (the six yellow plots) show the membership functions referenced by the antecedent, or the if-part of each rule.
- The third column of plots (the three blue plots) shows the membership functions referenced by the consequent, or the then-part of each rule.

Notice that under **food**, there is a plot which is blank. This corresponds to the characterization of `none` for the variable **food** in the second rule.

- The fourth plot in the third column of plots represents the aggregate weighted decision for the given inference system.

  This decision will depend on the input values for the system. The defuzzified output is displayed as a bold vertical line on this plot.

The variables and their current values are displayed on top of the columns. In the lower left, there is a text field **Input** in which you can enter specific input values. For the two-input system, you will enter an input vector, `[9 8]`, for example, and then press**Enter**. You can also adjust these input values by clicking on any of the three plots for each input. This will move the red index line horizontally, to the point where you have clicked. Alternatively, you can also click and drag this line in order to change the input values. When you release the line, (or after manually specifying the input), a new calculation is performed, and you can see the whole fuzzy inference process take place:

- Where the index line representing service crosses the membership function line "service is poor" in the upper-left plot determines the degree to which rule one is activated.
- A yellow patch of color under the actual membership function curve is used to make the fuzzy membership value visually apparent.

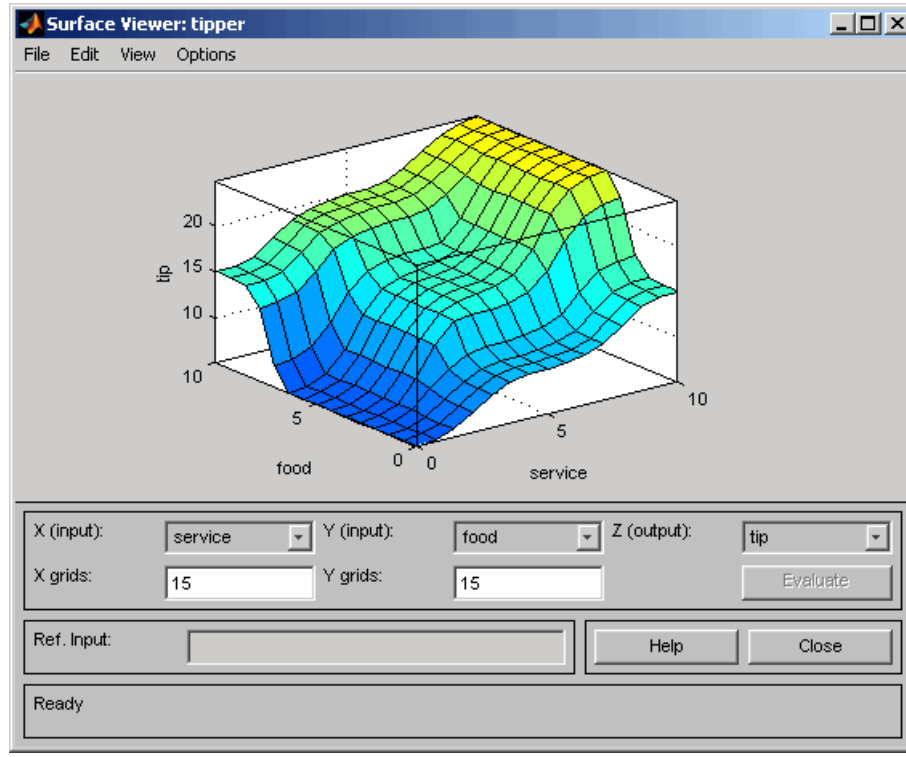
Each of the characterizations of each of the variables is specified with respect to the input index line in this manner. If you follow rule 1 across the top of the diagram, you can see the consequent "tip is cheap" has been truncated to exactly the same degree as the (composite) antecedent—this is the implication process in action. The aggregation occurs down the third column, and the resultant aggregate plot is shown in the single plot appearing in the lower right corner of the plot field. The defuzzified output value is shown by the thick line passing through the aggregate fuzzy set.

You can shift the plots using **left**, **right**, **down**, and **up**. The menu items allow you to save, open, or edit a fuzzy system using any of the five basic GUI tools.

The Rule Viewer allows you to interpret the entire fuzzy inference process at once. The Rule Viewer also shows how the shape of certain membership functions influences the overall result. Because it plots every part of every rule, it can become unwieldy for particularly large systems, but, for a relatively small number of inputs and outputs, it performs well (depending on how much screen space you devote to it) with up to 30 rules and as many as 6 or 7 variables.

The Rule Viewer shows one calculation at a time and in great detail. In this sense, it presents a sort of micro view of the fuzzy inference system. If you want to see the entire output surface of your system—the entire span of the output set based on the entire span of the input set— you need to open up the Surface Viewer. This viewer is the last of our five basic GUI tools in Fuzzy Logic Toolbox, and you open it by selecting **Surface** from the **View** menu.

The Surface Viewer



Upon opening the Surface Viewer, you see a three-dimensional curve that represents the mapping from food and service quality to tip amount. Because this curve represents a two-input one-output case, you can see the entire mapping in one plot. When we move beyond three dimensions overall, we start to encounter trouble displaying the results.

Accordingly, the Surface Viewer is equipped with drop-down menus **X (input):**, **Y (input):** and **Z (output):** that let you select any two inputs and any one output for plotting. Below these menus are two input fields **X grids:** and **Y grids:** that let you specify how many x-axis and y-axis grid lines you want to include. This capability allows you to keep the calculation time reasonable for complex problems.

Clicking **Evaluate** initiates the calculation, and the plot is generated after the calculation is complete. To change the *x*-axis or *y*-axis grid after the surface is in view, change the appropriate input field, and press **Enter**. The surface plot is updated to reflect the new grid settings.

The Surface Viewer has a special capability that is very helpful in cases with two (or more) inputs and one output: you can grab the axes, using the mouse and reposition them to get a different three-dimensional view on the data.

The **Ref. Input** field is used in situations when there are more inputs required by the system than the surface is mapping. You can edit this field to explicitly set inputs not specified in the surface plot.

Suppose you have a four-input one-output system and would like to see the output surface. The Surface Viewer can generate a three-dimensional output surface where any two of the inputs vary, but two of the inputs must be held constant because computer monitors cannot display a five-dimensional shape. In such a case, the input is a four-dimensional vector with `NaN`s holding the place of the varying inputs while numerical values indicates those values that remain fixed. A `NaN` is the IEEE symbol for not a number.

The menu items allow you to open, close, save and edit a fuzzy system using the five basic GUI tools. You can access information about the Surface Viewer by clicking **Help** and close the GUI using **Close**.

This concludes the quick walk-through of each of the main GUI tools. For the tipping problem, the output of the fuzzy system matches your original idea of the shape of the fuzzy mapping from service to tip fairly well. In hindsight, you might say, “Why bother? I could have just drawn a quick lookup table and been done an hour ago!” However, if you are interested in solving an entire class of similar decision-making problems, fuzzy logic may provide an appropriate tool for the solution, given its ease with which a system can be quickly modified.

Importing and Exporting from the GUI Tools

When you save a fuzzy system to a file, you are saving an ASCII text FIS file representation of that system with the file suffix `.fis`. This text file can be edited and modified and is simple to understand. When you save your fuzzy system to the MATLAB workspace, you are creating a variable (whose name

you choose) that acts as a MATLAB structure for the FIS system. FIS files and FIS structures represent the same system.

Note If you do not save your FIS to a file, but only save it to the MATLAB workspace, you are not able to recover it for use in a new MATLAB session.

Customizing Your Fuzzy System

If you want to include customized functions as part of your use of Fuzzy Logic Toolbox, you must follow a few guidelines. You may substitute customized functions for the AND, OR, aggregation, and defuzzification methods, provided your customized functions work in a similar way to `max`, `min`, or `prod` in MATLAB. That is, they must be able to operate down the columns of a matrix.

In MATLAB, for a matrix `x`, `min(x)` returns a row vector containing the minimum element from each column. For N-D arrays, `min(x)` operates along the first non-singleton dimension. The function `min(x,y)`, on the other hand, returns an array the same size as `x` and `y` populated with the smallest elements from `x` or `y`. Either one can be a scalar. Functions such as `max`, `prod`, and `mean` operate in a similar manner.

In Fuzzy Logic Toolbox, the implication method performs an element by element matrix operation, similar to the `min(x,y)` function in MATLAB, as in

```
a=[1 2; 3 4];
b=[2 2; 2 2];
min(a,b)
ans =
    1     2
    2     2
```

After you have defined your custom function using the procedure described in the next section, use the FIS Editor to substitute your custom function for a standard function using the following procedure:

1. Open the FIS Editor by typing `fuzzy` at the command line prompt.
2. In the lower-left panel, locate the method you want to replace.

3 In the pull-down menu, select `Custom`. A dialog box appears.

4 Enter the name of your custom function, and click **OK**.

Your custom function then replaces the standard function in all subsequent operations.

Custom Membership Functions

You can create your own membership functions using an M-file. The values these functions can take must be between 0 and 1. There is a limitation on customized membership functions in that they cannot use more than 16 parameters.

To define a custom membership function named `custmf`:

1 Create an M-file for a function, `custmf.m`, that takes values between 0 and 1, and depends on 16 parameters at most.

2 Choose the **Add Custom MF** item in the **Edit** menu on the Membership Function Editor GUI.

3 Enter your custom membership function M-file name, `custmf`, in the **M-file function name** text box.

4 Enter the vector of parameters you want to use to parameterize your customized membership function in the text box next to **Parameter list**.

5 Give the custom membership function a name different from any other membership function name you use in your FIS.

6 Click **OK**.

The following sample code for a custom membership function, `testmf1`, depends on eight parameters between 0 and 10.

```
function out = testmf1(x, params)
for i=1:length(x)
if x(i)<params(1)
 y(i)=params(1);
elseif x(i)<params(2)
```

```
 y(i)=params(2);
elseif x(i)<params(3)
 y(i)=params(3);
elseif x(i)<params(4)
 y(i)=params(4);
elseif x(i)<params(5)
 y(i)=params(5);
elseif x(i)<params(6)
 y(i)=params(6);
elseif x(i)<params(7)
 y(i)=params(7);
elseif x(i)<params(8)
 y(i)=params(8);
else
 y(i)=0;
end
end
out=.1*y';
```

You can try naming this file `testmf1.m` and loading it into the Membership Function Editor using the parameters of your choice.

Working from the Command Line

In this section...

"The Tipping Problem from the Command Line" on page 2-57

"System Display Functions" on page 2-60

"Building a System from Scratch" on page 2-63

"FIS Evaluation" on page 2-66

"The FIS Structure" on page 2-66

The Tipping Problem from the Command Line

The tipping system is one of many examples of fuzzy inference systems provided with Fuzzy Logic Toolbox. The FIS is always cast as a MATLAB structure. To load this system (rather than bothering with creating it from scratch), type

```
a = readfis('tipper.fis')
```

MATLAB responds with

```
a =
            name: 'tipper'
            type: 'mamdani'
       andMethod: 'min'
        orMethod: 'max'
    defuzzMethod: 'centroid'
       impMethod: 'min'
       aggMethod: 'max'
           input: [1x2 struct]
          output: [1x1 struct]
            rule: [1x3 struct]
```

The labels on the left of this listing represent the various components of the MATLAB structure associated with `tipper.fis`. You can access the various components of this structure by typing the component name after typing `a`. At the MATLAB command line, type

```
a.type
```

for example. MATLAB will respond with

```
ans =
mamdani
```

The function

```
getfis(a)
```

returns almost the same structure information that typing `a`, alone does.

`getfis(a)` returns

```
Name      = tipper
      Type      = mamdani
      NumInputs = 2
      InLabels  =
            service
            food
      NumOutputs = 1
      OutLabels =
            tip
      NumRules = 3
      AndMethod = min
      OrMethod = max
      ImpMethod = min
      AggMethod = max
      DefuzzMethod = centroid
```

Some of these fields are not part of the structure, `a`. Thus, you cannot get information by typing `a.Inlabels`, but you can get it by typing

```
getfis(a,'Inlabels')
```

Similarly, you can obtain structure information using `getfis` in this manner.

```
getfis(a,'input',1)
getfis(a,'output',1)
getfis(a,'input',1,'mf',1)
```

The `structure.field` syntax also generates this information. For more information on the syntax for MATLAB structures and cell arrays, see the MATLAB documentation.

For example, type

```
a.input
```

or

```
a.input(1).mf(1)
```

The function `getfis` is loosely modeled on the Handle Graphics® function `get`. The function `setfis` acts as the reciprocal to `getfis`. It allows you to change any property of a FIS. For example, if you wanted to change the name of this system, you could type

```
a = setfis(a,'name','gratuity');
```

However, because `a` is already a MATLAB structure, you can set this information more simply by typing

```
a.name = 'gratuity';
```

Now the FIS structure `a` has been changed to reflect the new name. If you want a little more insight into this FIS structure, try

```
showfis(a)
```

This syntax returns a printout listing all the information about `a`. This function is intended more for debugging than anything else, but it shows all the information recorded in the FIS structure

Because the variable, `a`, designates the fuzzy tipping system, you can display any of the GUIs for the tipping system directly from the command line. Any of the following functions will display the tipping system with the associated GUI:

- `fuzzy(a)` displays the FIS Editor.
- `mfedit(a)` displays the Membership Function Editor.

- `ruleedit(a)` displays the Rule Editor.
- `ruleview(a)` displays the Rule Viewer.
- `surfview(a)` displays the Surface Viewer.

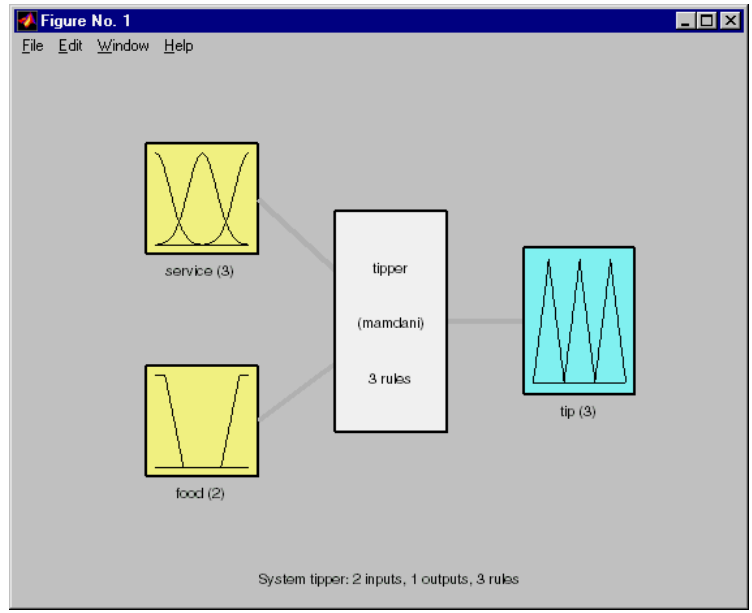
If, in addition, `a` is a Sugeno-type FIS, then `anfisedit(a)` displays the ANFIS Editor GUI.

When any of these GUIs is opened, you can access any of the other GUIs using the pull-down menu rather than the command line.

System Display Functions

There are three functions designed to give you a high-level view of your fuzzy inference system from the command line: `plotfis`, `plotmf`, and `gensurf`. The first of these displays the whole system as a block diagram much as it would appear on the FIS Editor.

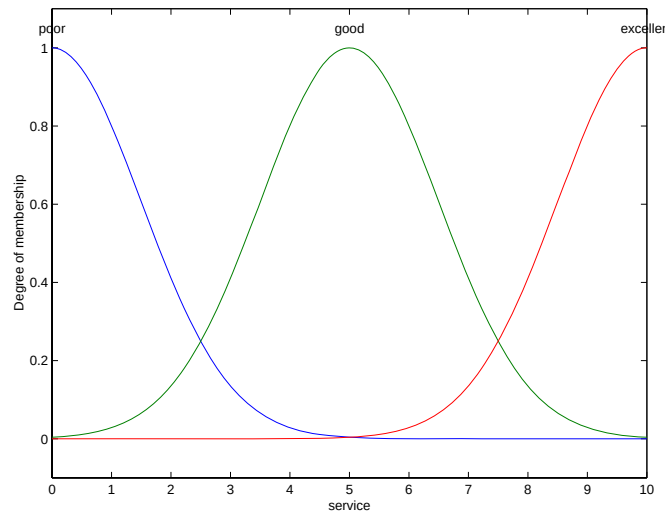
```
plotfis(a)
```



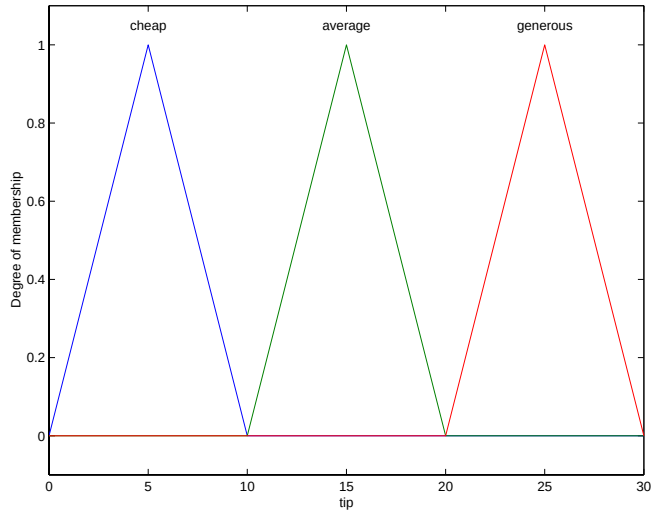
After closing any open MATLAB figures or GUI windows, the function `plotmf` plots all the membership functions associated with a given variable as follows.

```
plotmf(a,'input',1)
```

returns the following plots



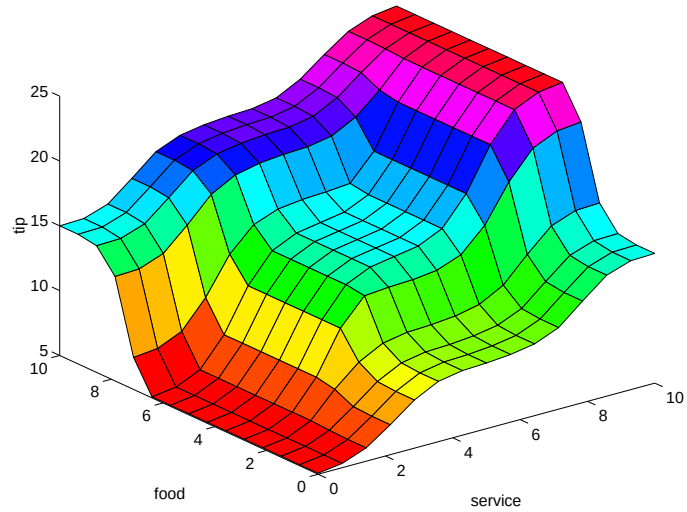

```
plotmf(a,'output',1)
```



These plots appear in the Membership Function Editor GUI, or in an open MATLAB figure, if `plotmf` is called while either of these is open.

Finally, the function `gensurf` plots any one or two inputs versus any one output of a given system. The result is either a two-dimensional curve, or a three-dimensional surface. When there are three or more inputs, `gensurf` must be generated with all but two inputs fixed, as is described in `gensurf`.

```
gensurf(a)
```



Building a System from Scratch

It is possible to use Fuzzy Logic Toolbox without bothering with the GUI tools at all. For instance, to build the tipping system entirely from the command line, you use the commands `newfis`, `addvar`, `addmf`, and `addrule`.

Probably the most difficult part of this process is learning the shorthand that the fuzzy inference systems use for building rules. Use the command line function, `addrule` to do this.

Each variable, input, or output, has an index number, and each membership function has an index number. The rules are built from statements such as the following:

If input1 is MF1 or input2 is MF3, then output1 is MF2 (weight = 0.5)

This rule is turned into a structure according to the following logic. If there are *m* inputs to a system and *n* outputs, then the first *m* vector entries of the rule structure correspond to inputs 1 through *m*.

- The entry in column 1 is the index number for the membership function associated with input 1.

- The entry in column 2 is the index number for the membership function associated with input 2, and so on.
- The next n columns work the same way for the outputs.
- Column $m + n + 1$ is the weight associated with that rule (typically 1) and column $m + n + 2$ specifies the connective used (where AND = 1 and OR = 2).

The structure associated with the preceding rule is

```
1 3 2 0.5 2
```

This sample code shows one way you can build the entire tipping system from the command line, using the MATLAB structure syntax.

```
a=newfis('tipper');
a.input(1).name='service';
a.input(1).range=[0 10];
a.input(1).mf(1).name='poor';
a.input(1).mf(1).type='gaussmf';
a.input(1).mf(1).params=[1.5 0];
a.input(1).mf(2).name='good';
a.input(1).mf(2).type='gaussmf';
a.input(1).mf(2).params=[1.5 5];
a.input(1).mf(3).name='excellent';
a.input(1).mf(3).type='gaussmf';
a.input(1).mf(3).params=[1.5 10];
a.input(2).name='food';
a.input(2).range=[0 10];
a.input(2).mf(1).name='rancid';
a.input(2).mf(1).type='trapmf';
a.input(2).mf(1).params=[-2 0 1 3];
a.input(2).mf(2).name='delicious';
a.input(2).mf(2).type='trapmf';
a.input(2).mf(2).params=[7 9 10 12];
a.output(1).name='tip';
a.output(1).range=[0 30];
a.output(1).mf(1).name='cheap'
a.output(1).mf(1).type='trimf';
a.output(1).mf(1).params=[0 5 10];
a.output(1).mf(2).name='average';
```

```
a.output(1).mf(2).type='trimf';
a.output(1).mf(2).params=[10 15 20];
a.output(1).mf(3).name='generous';
a.output(1).mf(3).type='trimf';
a.output(1).mf(3).params=[20 25 30];
a.rule(1).antecedent=[1 1];
a.rule(1).consequent=[1];
a.rule(1).weight=1;
a.rule(1).connection=2;
a.rule(2).antecedent=[2 0];
a.rule(2).consequent=[2];
a.rule(2).weight=1;
a.rule(2).connection=1;
a.rule(3).antecedent=[3 2];
a.rule(3).consequent=[3];
a.rule(3).weight=1;
a.rule(3).connection=2
```

Alternatively, you can build the entire tipping system from the command line using Fuzzy Logic Toolbox commands. These commands are in the `mktipper.m` demo file.

```
a=newfis('tipper');
a=addvar(a,'input','service',[0 10]);
a=addmf(a,'input',1,'poor','gaussmf',[1.5 0]);
a=addmf(a,'input',1,'good','gaussmf',[1.5 5]);
a=addmf(a,'input',1,'excellent','gaussmf',[1.5 10]);
a=addvar(a,'input','food',[0 10]);
a=addmf(a,'input',2,'rancid','trapmf',[-2 0 1 3]);
a=addmf(a,'input',2,'delicious','trapmf',[7 9 10 12]);
a=addvar(a,'output','tip',[0 30]);
a=addmf(a,'output',1,'cheap','trimf',[0 5 10]);
a=addmf(a,'output',1,'average','trimf',[10 15 20]);
a=addmf(a,'output',1,'generous','trimf',[20 25 30]);
ruleList=[ ...
1 1 1 1 2
2 0 2 1 1
3 2 3 1 2 ];
a=addrule(a,ruleList);
```

FIS Evaluation

To evaluate the output of a fuzzy system for a given input, use the function `evalfis`. For example, the following script evaluates `tipper` at the input, [1 2].

```
a = readfis('tipper');
evalfis([1 2], a)
ans =
    5.5586
```

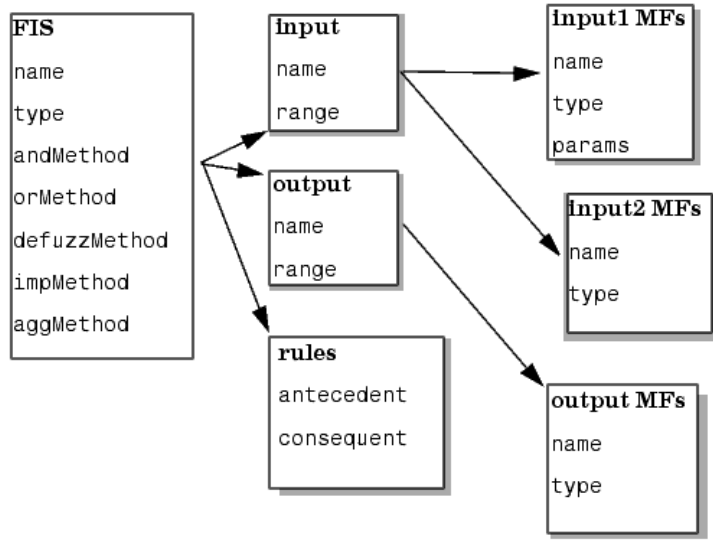
This function can also be used for multiple collections of inputs, because different input vectors are represented in different parts of the input structure.

```
evalfis([3 5; 2 7], a)
ans =
    12.2184
     7.7885
```

The FIS Structure

The FIS structure is the MATLAB object that contains all the fuzzy inference system information. This structure is stored inside each GUI tool. Access functions such as `getfis` and `setfis` make it easy to examine this structure.

All the information for a given fuzzy inference system is contained in the FIS structure, including variable names, membership function definitions, and so on. This structure can itself be thought of as a hierarchy of structures, as shown in the following diagram.



You can generate a listing of information on the FIS using the `showfis` command, as shown in the following code sample:

```
showfis(a)
1.  Name              tipper
2.  Type              mamdani
3.  Inputs/Outputs    [ 2 1 ]
4.  NumInputMFs       [ 3 2 ]
5.  NumOutputMFs      3
6.  NumRules          3
7.  AndMethod         min
8.  OrMethod          max
9.  ImpMethod         min
10. AggMethod         max
11. DefuzzMethod      centroid
12. InLabels          service
13.                   food
14. OutLabels         tip
15. InRange           [ 0 10 ]
16.                   [ 0 10 ]
17. OutRange          [ 0 30 ]
```

```
18. InMFLabels       poor
19.                  good
20.                  excellent
21.                  rancid
22.                  delicious
23. OutMFLabels      cheap
24.                  average
25.                  generous
26. InMFTypes        gaussmf
27.                  gaussmf
28.                  gaussmf
29.                  trapmf
30.                  trapmf
31. OutMFTypes       trimf
32.                  trimf
33.                  trimf
34. InMFParams       [ 1.5 0 0 0 ]
35.                  [ 1.5 5 0 0 ]
36.                  [ 1.5 10 0 0 ]
37.                  [ 0 0 1 3 ]
38.                  [ 7 9 10 10 ]
39. OutMFParams      [ 0 5 10 0 ]
40.                  [ 10 15 20 0 ]
41.                  [ 20 25 30 0 ]
42. Rule Antecedent  [ 1 1 ]
43.                  [ 2 0 ]
44.                  [ 3 2 ]
42. Rule Consequent  1
43.                  2
44.                  3
42. Rule Weight      1
43.                  1
44.                  1
42. Rule Connection  2
43.                  1
44.                  2
```

The list of command-line functions associated with FIS construction includes `getfis`, `setfis`, `showfis`, `addvar`, `addmf`, `addrule`, `rmvar`, and `rmmf`.

Saving FIS Files

A specialized text file format is used for saving fuzzy inference systems. The functions `readfis` and `writefis` are used for reading and writing these files.

If you prefer, you can modify the FIS by editing its `.fis` text file rather than using any of the GUIs. You should be aware, however, that changing one entry may oblige you to change another. For example, if you delete a membership function using this method, you also need to make certain that any rules requiring this membership function are also deleted.

The rules appear in indexed format in a `.fis` text file. The following sample shows the file `tipper.fis`.

```
[System]
Name='tipper'
Type='mamdani'
NumInputs=2
NumOutputs=1
NumRules=3
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='service'
Range=[0 10]
NumMFs=3
MF1='poor':'gaussmf',[1.5 0]
MF2='good':'gaussmf',[1.5 5]
MF3='excellent':'gaussmf',[1.5 10]

[Input2]
Name='food'
Range=[0 10]
NumMFs=2
MF1='rancid':'trapmf',[0 0 1 3]
MF2='delicious':'trapmf',[7 9 10 10]
```

```
[Output1]
Name='tip'
Range=[0 30]
NumMFs=3
MF1='cheap':'trimf',[0 5 10]
MF2='average':'trimf',[10 15 20]
MF3='generous':'trimf',[20 25 30]

[Rules]
1 1, 1 (1) : 2
2 0, 2 (1) : 1
3 2, 3 (1) : 2
```

Working with Simulink

In this section...

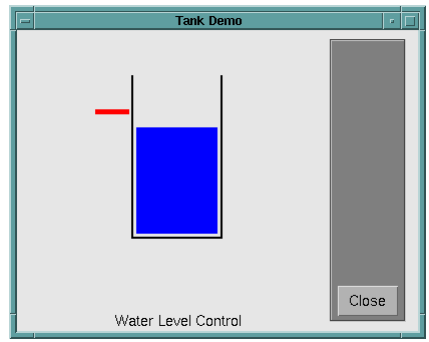
"An Example: Water Level Control" on page 2-71

"Building Your Own Fuzzy Simulink Models" on page 2-78

An Example: Water Level Control

Fuzzy Logic Toolbox is designed to work with Simulink. After you have created your fuzzy system using the GUI tools or some other method, you are ready to embed your system directly into a simulation.

Picture a tank with a pipe flowing in and a pipe flowing out. You can change the valve controlling the water that flows in, but the outflow rate depends on the diameter of the outflow pipe (which is constant) and the pressure in the tank (which varies with the water level). The system has some very nonlinear characteristics.



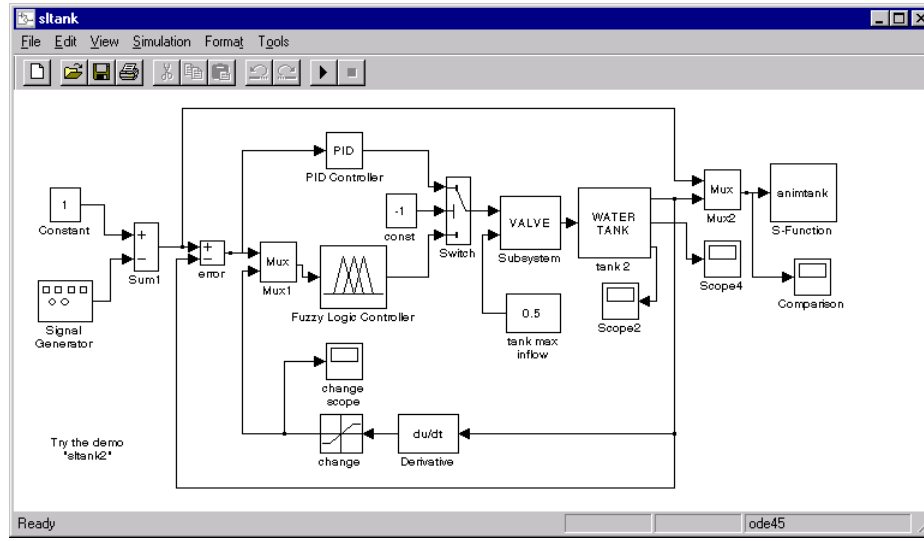
A controller for the water level in the tank needs to know the current water level and it needs to be able to set the valve. Your controller's input is the water level error (desired water level minus actual water level), and its output is the rate at which the valve is opening or closing. A first pass at writing a fuzzy controller for this system might be the following:

1. *If (level is okay) then (valve is no_change) (1)*
2. *If (level is low) then (valve is open_fast) (1)*
3. *If (level is high) then (valve is close_fast) (1)*

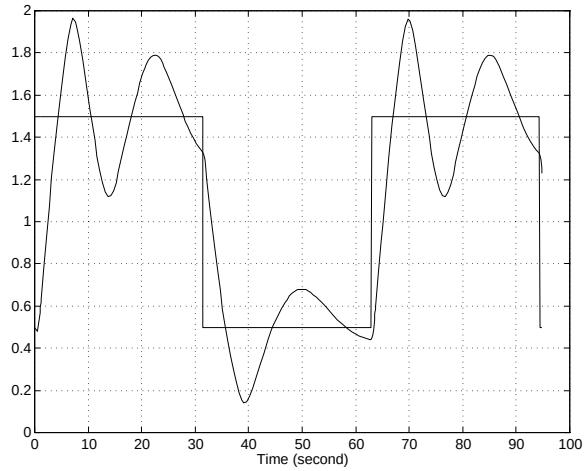
One of the great advantages of Fuzzy Logic Toolbox is the ability to take fuzzy systems directly into Simulink and test them out in a simulation environment. A Simulink block diagram for this system is shown in the following figure. It contains a Simulink block called the Fuzzy Logic Controller block. The Simulink block diagram for this system is `sltank`. Typing

```
sltank
```

at the command line, causes the system to appear. At the same time, the file `tank.fis` is loaded into the FIS structure `tank`.



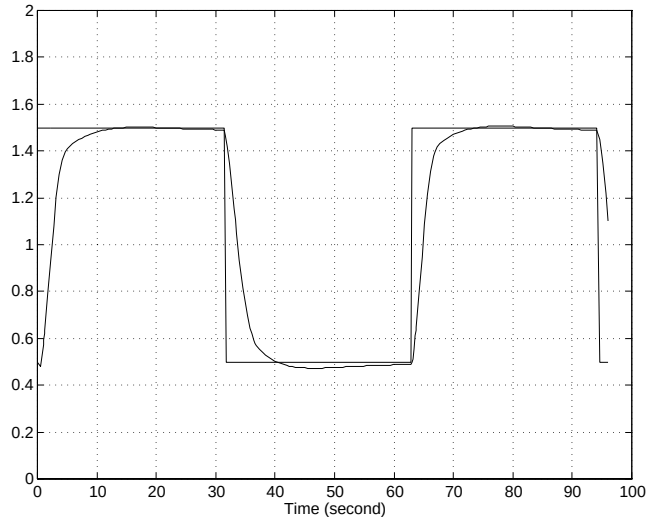
Some experimentation shows that three rules are not sufficient, because the water level tends to oscillate around the desired level. See the following plot:



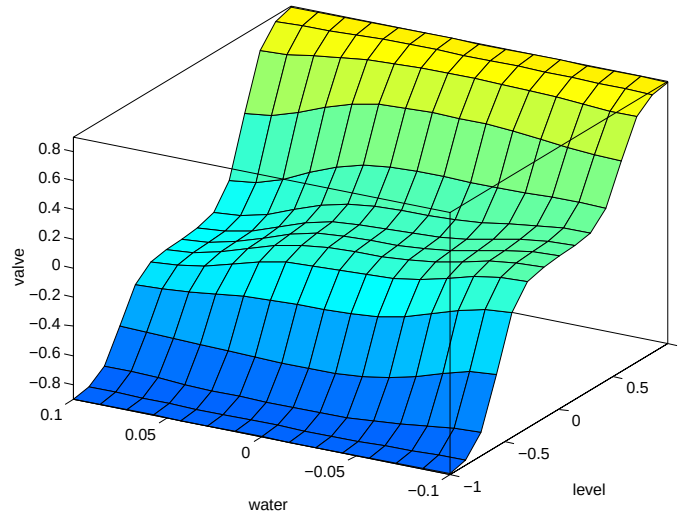
You need to add another input, the water level's rate of change, to slow down the valve movement when it gets close to the right level.

4. *If (level is good) and (rate is negative), then (valve is close_slow) (1)*
5. *If (level is good) and (rate is positive), then (valve is open_slow) (1)*

The demo, `sltank` is built with these five rules. With all five rules in operations, you can examine the step response by simulating this system. You do so by clicking **Start** from the pull-down menu under **Simulate**, and clicking the Comparison block. The result looks similar to the following plot.



One interesting feature of the water tank system is that the tank empties much more slowly than it fills up because of the specific value of the outflow diameter pipe. You can deal with this by setting the `close_slow` valve membership function to be slightly different from the `open_slow` setting. A PID controller does not have this capability. The valve command versus the water level change rate (depicted as *water*) and the relative water level change (depicted as *level*) surface looks like this. If you look closely, you can see a slight asymmetry to the plot.



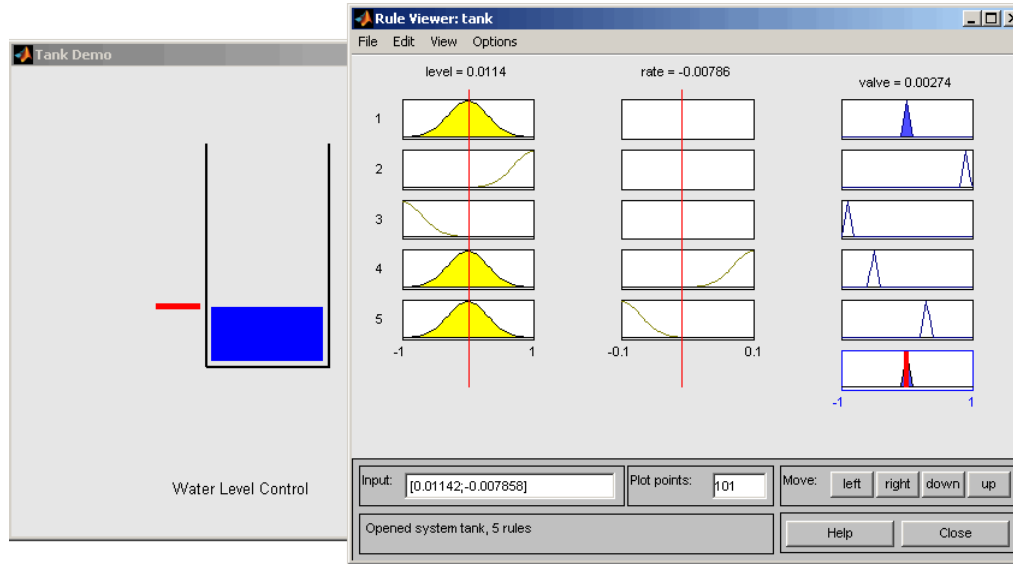
Because the MATLAB technical computing environment supports so many tools (like Control System Toolbox, Neural Network Toolbox, and so on), you can, for example, easily make a comparison of a fuzzy controller versus a linear controller or a neural network controller.

For a demonstration of how the Rule Viewer can be used to interact with a Fuzzy Logic Controller block in a Simulink model, type

```
sltankrule
```

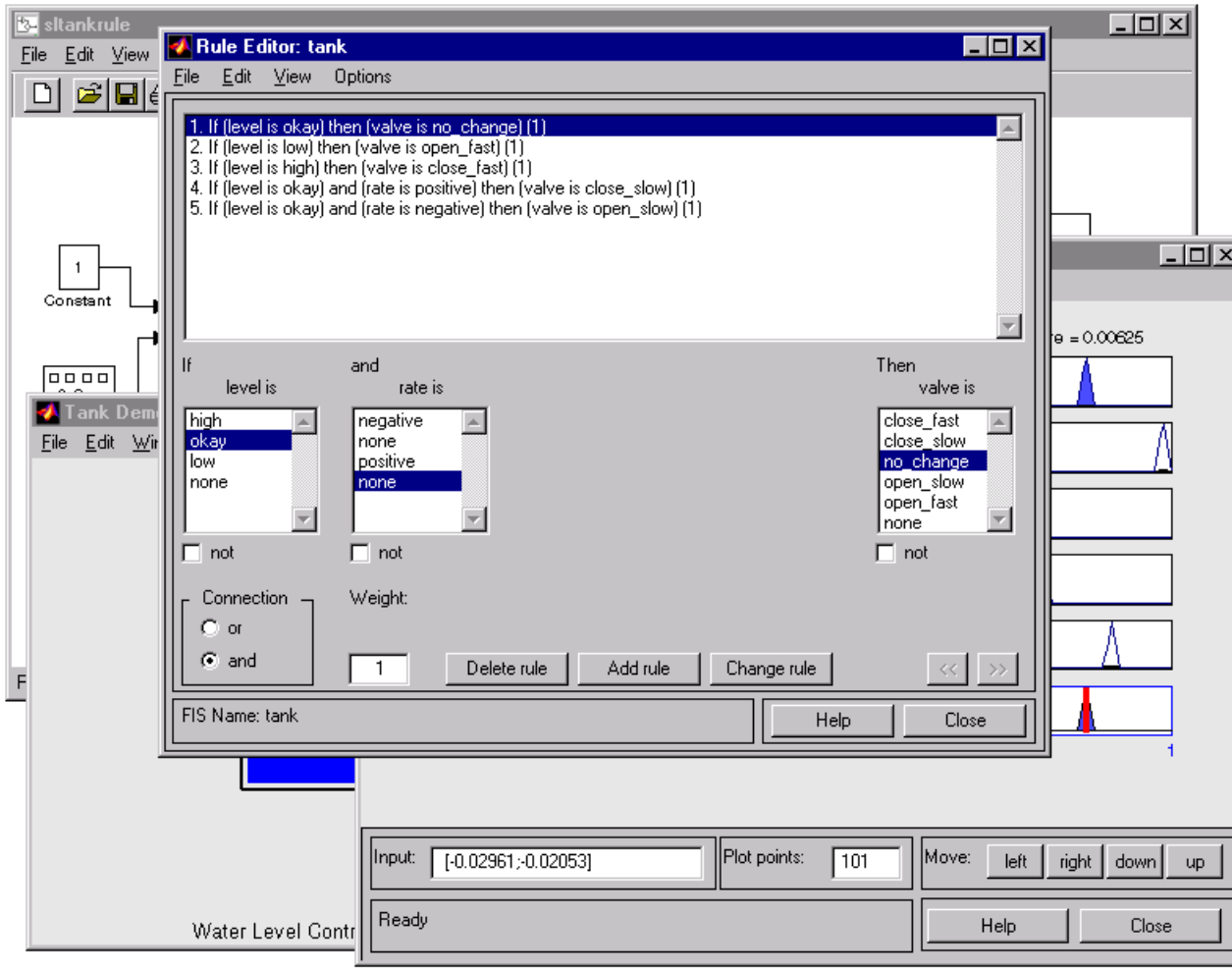
This demo contains a block called the Fuzzy Controller With Rule Viewer block.

In this demo, the Rule Viewer opens when you start the simulation. This Rule Viewer provides an animation of how the rules are fired during the water tank simulation. The windows that open when you simulate the `sltankrule` demo are depicted as follows.



The Rule Viewer that opens during the simulation can be used to access the Membership Function Editor, the Rule Editor, or any of the other GUIs, (see "The Membership Function Editor" on page 2-39, or "The Rule Editor" on page 2-45, for more information).

For example, you may want to open the Rule Editor to change one of your rules. To do so, select **Rules** under the **Edit** menu of the open Rule Viewer. Now, you can view or edit the rules for this Simulink model.



If you stop the simulation prior to selecting any of these editors, you should change your FIS. Remember to save any changes you make to your FIS to the workspace before you restart the simulation.

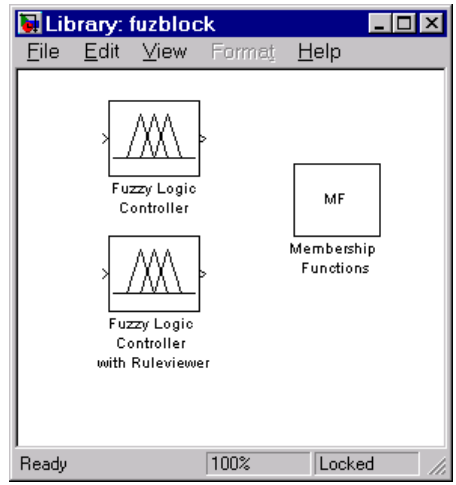
Building Your Own Fuzzy Simulink Models

To build your own Simulink systems that use fuzzy logic, simply copy the Fuzzy Logic Controller block out of `sltank` (or any of the other Simulink demo systems available with the toolbox) and place it in your own block diagram. You can also find the Fuzzy Logic Controller block in the Fuzzy Logic Toolbox library, which you can open either by selecting **Fuzzy Logic Toolbox** in the Simulink Library Browser, or by typing

```
fuzblock
```

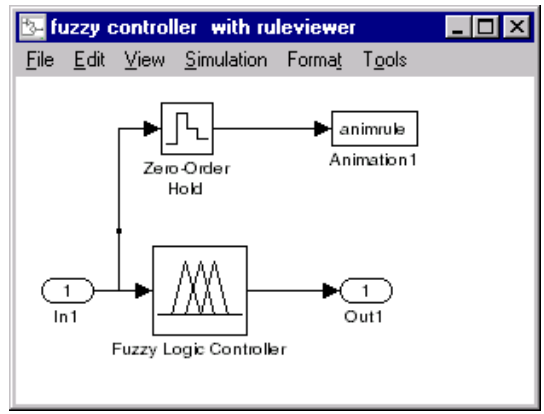
at the MATLAB prompt.

The following library appears.



The Fuzzy Logic Toolbox library contains the Fuzzy Logic Controller and Fuzzy Logic Controller with Rule Viewer blocks. It also includes a Membership Functions sublibrary that contains Simulink blocks for the built-in membership functions.

The Fuzzy Logic Controller with Rule Viewer block is an extension of the Fuzzy Logic Controller block. It allows you to visualize how rules are fired during simulation. Right-click on the Fuzzy Controller With Rule Viewer block, and select `Look Under Mask`, and the following window appears.



To initialize the Fuzzy Logic Controller blocks (with or without the Rule Viewer), double-click on the block and enter the name of the structure variable describing your FIS. This variable must be located in the MATLAB workspace.

About the Fuzzy Logic Controller Block

For most fuzzy inference systems, the Fuzzy Logic Controller block automatically generates a hierarchical block diagram representation of your FIS. This automatic model generation ability is called the *Fuzzy Wizard*. The block diagram representation only uses built-in Simulink blocks and, therefore, allows for efficient code generation. For more information about the Fuzzy Logic Controller block, see the `fuzblock` reference page.

The Fuzzy Wizard cannot handle FIS with custom membership functions or with AND, OR, IMP, and AGG functions outside of the following list:

- `orMethod: max`
- `andMethod: min,prod`
- `impMethod: min,prod`
- `aggMethod: max`

In these cases, the Fuzzy Logic Controller block uses the S-function `sffis` to simulate the FIS. For more information, see the `sffis` reference page.

Example: Cart and Pole Simulation

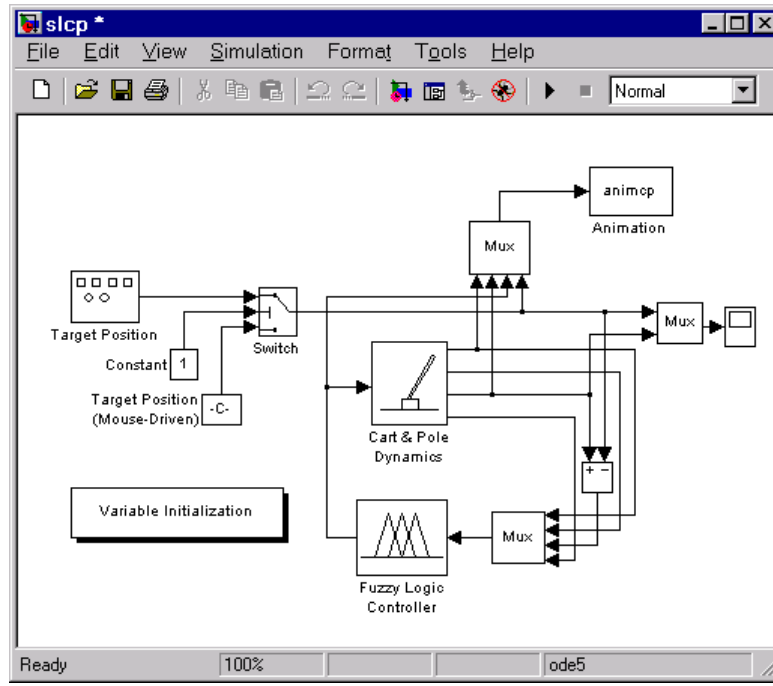
The cart and pole simulation is an example of a FIS model auto-generated by the Fuzzy Logic Controller block.

Type

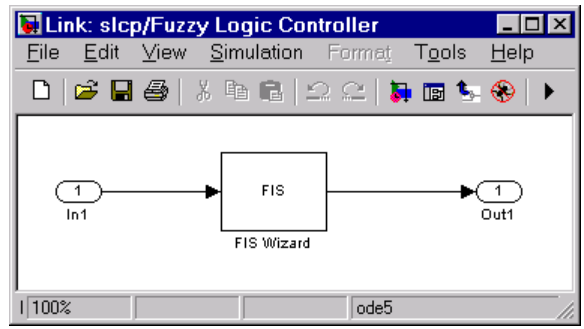
```
slcp
```

at the MATLAB prompt to open the simulation.

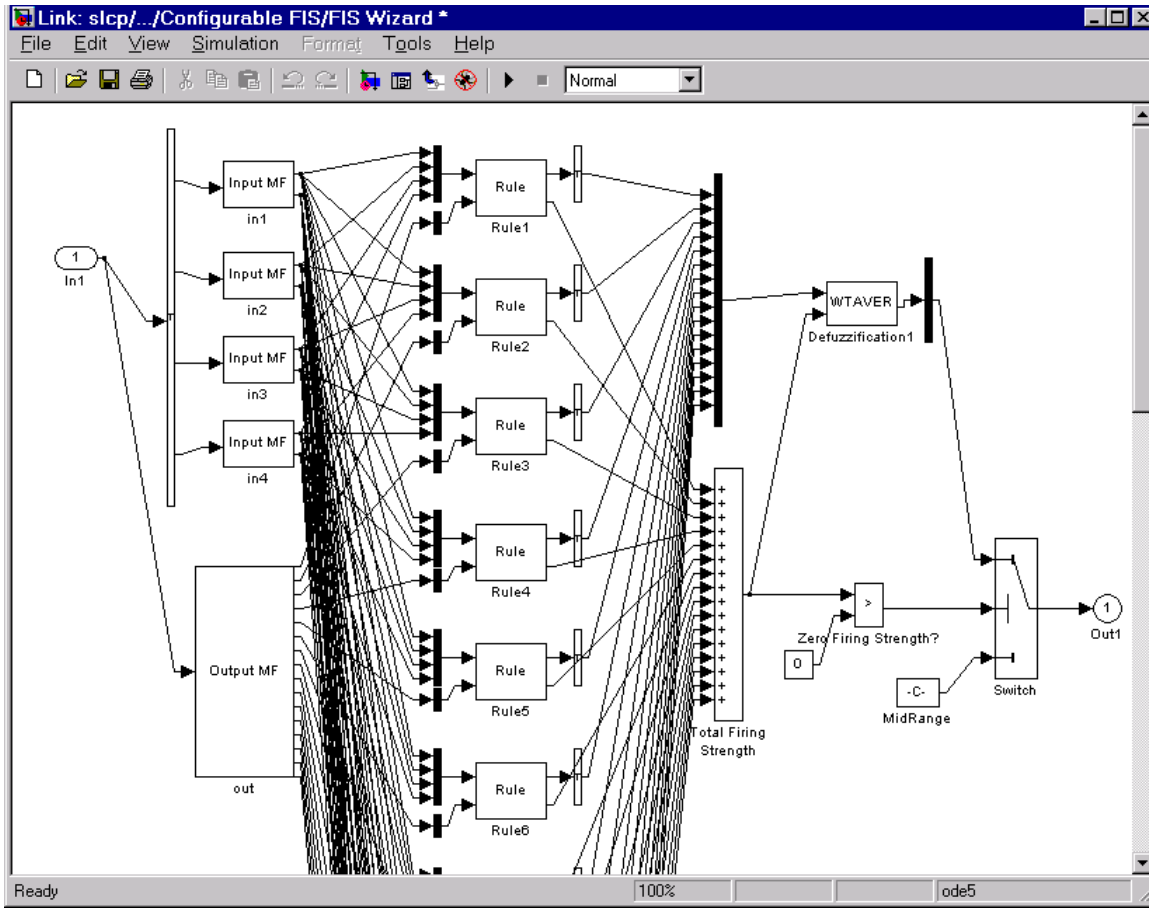
This model appears.



Right-click on the Fuzzy Logic Controller block, and select **Look under mask** from the right-click menu. The following subsystem opens.



Follow the same procedure to look under the mask of the FIS Wizard subsystem to see the implementation of your FIS. This following figure shows part of the implementation (the entire model is too large to show in this document).



As the figure shows, the Fuzzy Logic Controller block uses built-in Simulink blocks to implement your FIS. Although the models can grow complex, this representation is better suited than the S-function `sffis` for efficient code generation.

Sugeno-Type Fuzzy Inference

In this section...

"What is Sugeno-Type Fuzzy Interference?" on page 2-83

"An Example: Two Lines" on page 2-87

"Comparison of Sugeno and Mamdani Methods" on page 2-89

What is Sugeno-Type Fuzzy Interference?

The fuzzy inference process discussed so far is Mamdani's fuzzy inference method, the most common methodology. This section discusses the so-called Sugeno, or Takagi-Sugeno-Kang, method of fuzzy inference. Introduced in 1985 [16], it is similar to the Mamdani method in many respects. The first two parts of the fuzzy inference process, fuzzifying the inputs and applying the fuzzy operator, are exactly the same. The main difference between Mamdani and Sugeno is that the Sugeno output membership functions are either linear or constant.

A typical rule in a Sugeno fuzzy model has the form

If Input 1 = x and Input 2 = y, then Output is $z = ax + by + c$

For a zero-order Sugeno model, the output level z is a constant $(a=b=0)$.

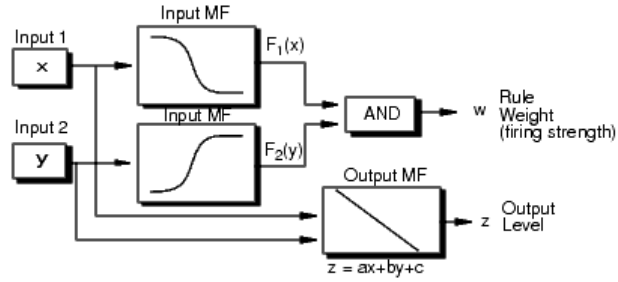
The output level z_i of each rule is weighted by the firing strength w_i of the rule. For example, for an AND rule with Input 1 = x and Input 2 = y, the firing strength is

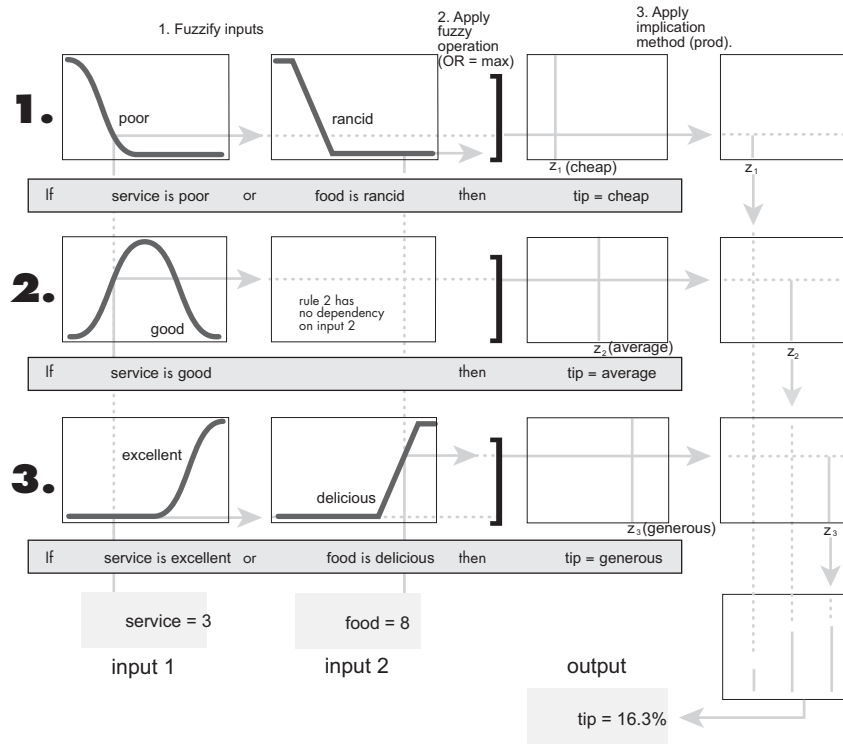
$$w_i = AndMethod\ (F_1(x),\ F_2(y))$$

where $F_{1,2}(.)$ are the membership functions for Inputs 1 and 2. The final output of the system is the weighted average of all rule outputs, computed as

$$\text{Final Output} = \frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} w_i}$$

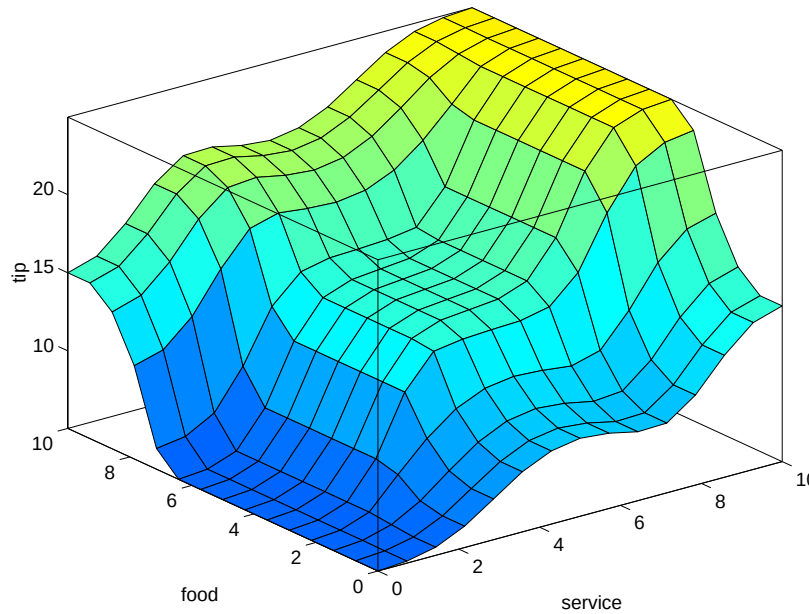
A Sugeno rule operates as shown in the following diagram.





The preceding figure shows the fuzzy tipping model developed in previous sections of this manual adapted for use as a Sugeno system. Fortunately, it is frequently the case that singleton output functions are completely sufficient for the needs of a given problem. As an example, the system `tippersg.fis` is the Sugeno-type representation of the now-familiar tipping model. If you load the system and plot its output surface, you will see that it is almost the same as the Mamdani system you have previously seen.

```
a = readfis('tippersg');
gensurf(a)
```



The easiest way to visualize first-order Sugeno systems is to think of each rule as defining the location of a moving singleton. That is, the singleton output spikes can move around in a linear fashion in the output space, depending on what the input is. This also tends to make the system notation very compact and efficient. Higher-order Sugeno fuzzy models are possible, but they introduce significant complexity with little obvious merit. Sugeno fuzzy models whose output membership functions are greater than first order are not supported by Fuzzy Logic Toolbox.

Because of the linear dependence of each rule on the input variables, the Sugeno method is ideal for acting as an interpolating supervisor of multiple linear controllers that are to be applied, respectively, to different operating conditions of a dynamic nonlinear system. For example, the performance of an aircraft may change dramatically with altitude and Mach number. Linear controllers, though easy to compute and well suited to any given flight condition, must be updated regularly and smoothly to keep up with the changing state of the flight vehicle. A Sugeno fuzzy inference system is extremely well suited to the task of smoothly interpolating the linear gains that would be applied across the input space; it is a natural and efficient

gain scheduler. Similarly, a Sugeno system is suited for modeling nonlinear systems by interpolating between multiple linear models.

An Example: Two Lines

To see a specific example of a system with linear output membership functions, consider the one input one output system stored in `sugeno1.fis`.

```
fismat = readfis('sugeno1');
getfis(fismat,'output',1)
   Name = output
   NumMFs = 2
   MFLabels =
    line1
    line2
   Range = [0 1]
```

The output variable has two membership functions.

```
getfis(fismat,'output',1,'mf',1)
   Name = line1
   Type = linear
   Params =
    -1    -1
getfis(fismat,'output',1,'mf',2)
 Name = line2
   Type = linear
   Params =
    1    -1
```

Further, these membership functions are linear functions of the input variable. The membership function `line1` is defined by the equation

$$output = (-1) \times input + (-1)$$

and the membership function `line2` is defined by the equation

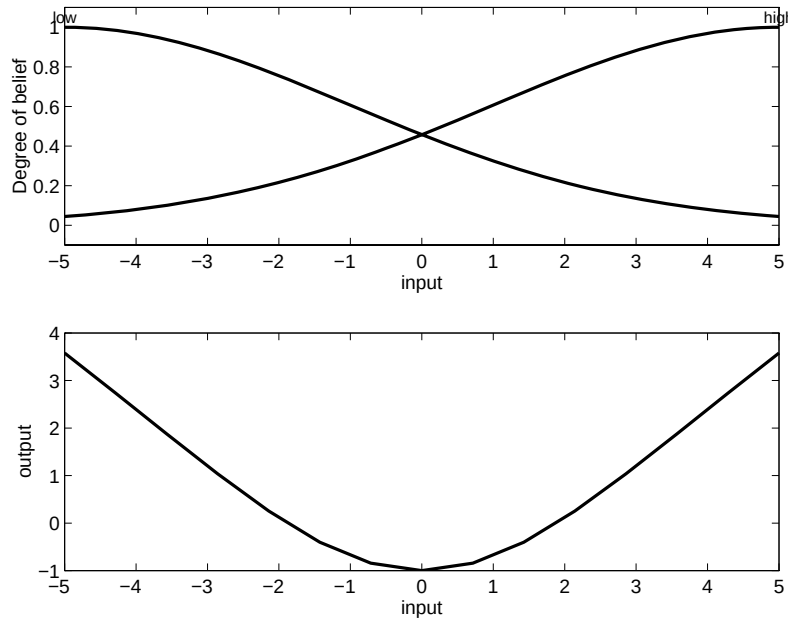
$$output = (1) \times input + (-1)$$

The input membership functions and rules define which of these output functions are expressed and when:

```
showrule(fismat)
ans =
1. If (input is low) then (output is line1) (1)
2. If (input is high) then (output is line2) (1)
```

The function `plotmf` shows us that the membership function `low` generally refers to input values less than zero, while `high` refers to values greater than zero. The function `gensurf` shows how the overall fuzzy system output switches smoothly from the line called `line1` to the line called `line2`.

```
subplot(2,1,1), plotmf(fismat,'input',1)
subplot(2,1,2),gensurf(fismat)
```



As this example shows, Sugeno-type system gives you the freedom to incorporate linear systems into your fuzzy systems. By extension, you could build a fuzzy system that switches between several optimal linear controllers as a highly nonlinear system moves around in its operating space.

Comparison of Sugeno and Mamdani Methods

Because it is a more compact and computationally efficient representation than a Mamdani system, the Sugeno system lends itself to the use of adaptive techniques for constructing fuzzy models. These adaptive techniques can be used to customize the membership functions so that the fuzzy system best models the data.

Note You can use the MATLAB command-line function `mam2sug` to convert a Mamdani system into a Sugeno system (not necessarily with a single output) with constant output membership functions. It uses the centroid associated with all of the output membership functions of the Mamdani system. See Chapter 4, "Functions — Alphabetical List" for details.

The following are some final considerations about the two different methods.

Advantages of the Sugeno Method

- It is computationally efficient.
- It works well with linear techniques (e.g., PID control).
- It works well with optimization and adaptive techniques.
- It has guaranteed continuity of the output surface.
- It is well suited to mathematical analysis.

Advantages of the Mamdani Method

- It is intuitive.
- It has widespread acceptance.
- It is well suited to human input.

anfis and the ANFIS Editor GUI

In this section...

"Introduction" on page 2-90

"A Modeling Scenario" on page 2-91

"Model Learning and Inference Through ANFIS" on page 2-91

"Know Your Data" on page 2-92

"Constraints of anfis" on page 2-94

"Training Adaptive Neuro Fuzzy Inference Systems Using the ANFIS Editor GUI" on page 2-94

"ANFIS Editor GUI Example 1: Checking Data Helps Model Validation" on page 2-98

"ANFIS Editor GUI Example 2: Checking Data Does Not Validate Model" on page 2-107

"anfis from the Command Line" on page 2-110

"More on anfis and the ANFIS Editor GUI" on page 2-117

Introduction

The basic structure of the type of fuzzy inference system seen thus far is a model that maps input characteristics to input membership functions, input membership function to rules, rules to a set of output characteristics, output characteristics to output membership functions, and the output membership function to a single-valued output or a decision associated with the output. You have considered only fixed membership functions that were chosen arbitrarily. You have applied fuzzy inference to only modeling systems whose rule structure is essentially predetermined by the user's interpretation of the characteristics of the variables in the model.

In this section we discuss the use of the function `anfis` and the ANFIS Editor GUI in Fuzzy Logic Toolbox. These tools apply fuzzy inference techniques to data modeling. As you have seen from the other fuzzy inference GUIs, the shape of the membership functions depends on parameters, and changing these parameters change the shape of the membership function. Instead of just looking at the data to choose the membership function parameters, you

choose membership function parameters automatically using these Fuzzy Logic Toolbox applications.

A Modeling Scenario

Suppose you want to apply fuzzy inference to a system for which you already have a collection of input/output data that you would like to use for modeling, model-following, or some similar scenario. You do not necessarily have a predetermined model structure based on characteristics of variables in your system.

In some modeling situations, you cannot discern what the membership functions should look like simply from looking at data. Rather than choosing the parameters associated with a given membership function arbitrarily, these parameters could be chosen so as to tailor the membership functions to the input/output data in order to account for these types of variations in the data values. In such cases, the *neuro-adaptive* learning techniques incorporated into `anfis` in Fuzzy Logic Toolbox can help.

Model Learning and Inference Through ANFIS

The neuro-adaptive learning method works similarly to that of neural networks. Neuro-adaptive learning techniques provide a method for the fuzzy modeling procedure to *learn* information about a data set. Then, Fuzzy Logic Toolbox computes the membership function parameters that best allow the associated fuzzy inference system to track the given input/output data. The Fuzzy Logic Toolbox function that accomplishes this membership function parameter adjustment is called `anfis`. The `anfis` function can be accessed either from the command line or through the ANFIS Editor GUI. Because the functionality of the command line function `anfis` and the ANFIS Editor GUI is similar, they are used somewhat interchangeably in this discussion, except when specifically describing the GUI.

What Is ANFIS?

The acronym ANFIS derives its name from *adaptive neuro-fuzzy inference system*. Using a given input/output data set, the toolbox function `anfis` constructs a fuzzy inference system (FIS) whose membership function parameters are tuned (adjusted) using either a backpropagation algorithm

alone or in combination with a least squares type of method. This adjustment allows your fuzzy systems to learn from the data they are modeling.

FIS Structure and Parameter Adjustment

A network-type structure similar to that of a neural network, which maps inputs through input membership functions and associated parameters, and then through output membership functions and associated parameters to outputs, can be used to interpret the input/output map.

The parameters associated with the membership functions changes through the learning process. The computation of these parameters (or their adjustment) is facilitated by a gradient vector. This gradient vector provides a measure of how well the fuzzy inference system is modeling the input/output data for a given set of parameters. When the gradient vector is obtained, any of several optimization routines can be applied in order to adjust the parameters to reduce some error measure. This error measure is usually defined by the sum of the squared difference between actual and desired outputs. `anfis` uses either back propagation or a combination of least squares estimation and backpropagation for membership function parameter estimation.

Know Your Data

The modeling approach used by `anfis` is similar to many system identification techniques. First, you hypothesize a parameterized model structure (relating inputs to membership functions to rules to outputs to membership functions, and so on). Next, you collect input/output data in a form that will be usable by `anfis` for training. You can then use `anfis` to *train* the FIS model to emulate the training data presented to it by modifying the membership function parameters according to a chosen error criterion.

In general, this type of modeling works well if the training data presented to `anfis` for training (estimating) membership function parameters is fully representative of the features of the data that the trained FIS is intended to model. In some cases however, data is collected using noisy measurements, and the training data cannot be representative of all the features of the data that will be presented to the model. In such situations, model validation is helpful.

Model Validation Using Testing and Checking Data Sets

Model validation is the process by which the input vectors from input/output data sets on which the FIS was not trained, are presented to the trained FIS model, to see how well the FIS model predicts the corresponding data set output values.

One problem with model validation for models constructed using adaptive techniques is selecting a data set that is both representative of the data the trained model is intended to emulate, yet sufficiently distinct from the training data set so as not to render the validation process trivial.

If you have collected a large amount of data, hopefully this data contains all the necessary representative features, so the process of selecting a data set for checking or testing purposes is made easier. However, if you expect to be presenting noisy measurements to your model, it is possible the training data set does not include all of the representative features you want to model.

The testing data set lets you check the generalization capability of the resulting fuzzy inference system. The idea behind using a checking data set for model validation is that after a certain point in the training, the model begins overfitting the training data set. In principle, the model error for the checking data set tends to decrease as the training takes place up to the point that overfitting begins, and then the model error for the checking data suddenly increases. Overfitting is accounted for by testing the FIS trained on the training data against the checking data, and choosing the membership function parameters to be those associated with the minimum checking error if these errors indicate model overfitting.

Usually, these training and checking data sets are collected based on observations of the target system and are then stored in separate files.

In the first example in the following section, two similar data sets are used for checking and training, but the checking data set is corrupted by a small amount of noise. This example illustrates of the use of the ANFIS Editor GUI with checking data to reduce the effect of model overfitting. In the second example, a training data set that is presented to `anfis` is sufficiently different than the applied checking data set. By examining the checking error sequence over the training period, it is clear that the checking data set is not good for model validation purposes. This example illustrates the use of the ANFIS Editor GUI to compare data sets.

Constraints of anfis

`anfis` is much more complex than the fuzzy inference systems discussed so far, and is not available for all of the fuzzy inference system options. Specifically, `anfis` only supports Sugeno-type systems, and these must have the following properties:

- Be first or zeroth order Sugeno-type systems.
- Have a single output, obtained using weighted average defuzzification. All output membership functions must be the same type and either be linear or constant.
- Have no rule sharing. Different rules cannot share the same output membership function, namely the number of output membership functions must be equal to the number of rules.
- Have unity weight for each rule.

An error occurs if your FIS structure does not comply with these constraints.

Moreover, `anfis` cannot accept all the customization options that basic fuzzy inference allows. That is, you cannot make your own membership functions and defuzzification functions; you must use the ones provided.

Training Adaptive Neuro Fuzzy Inference Systems Using the ANFIS Editor GUI

You can create, train, and test Sugeno-type fuzzy systems using the ANFIS Editor GUI. For more information about the supported Sugeno-type systems, see "Constraints of anfis" on page 2-94.

To start the GUI, type the following command at the MATLAB prompt:

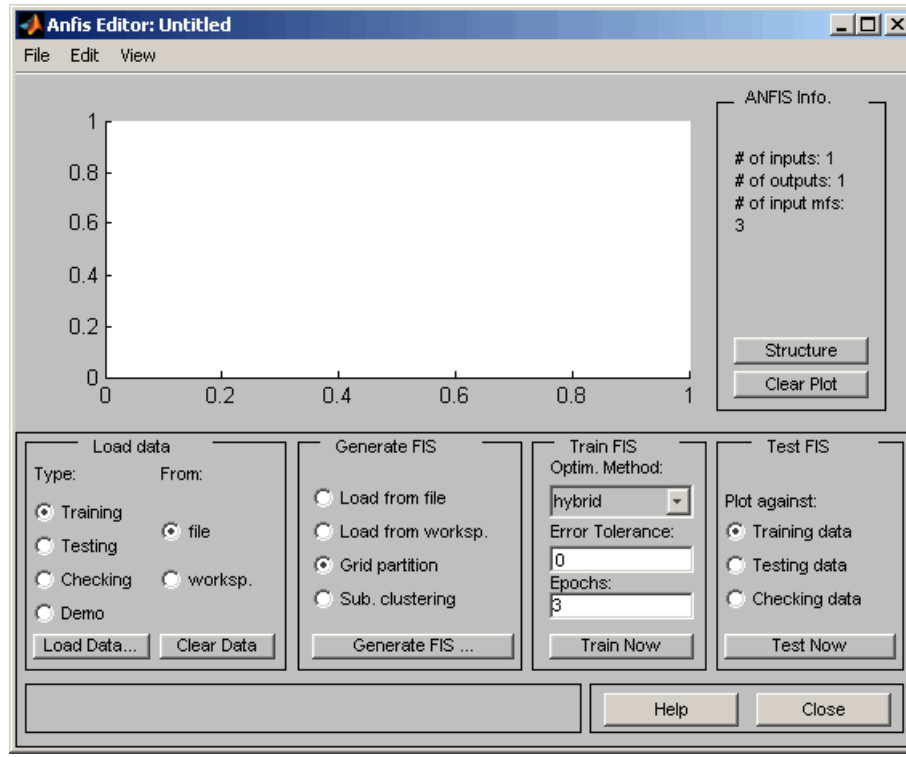
```
anfisedit
```

The ANFIS Editor GUI window shown in the following figure includes four distinct areas to support a typical workflow. The GUI lets you perform the following tasks:

1. Load, plot, and clear the data.

2. Generate or load the FIS model structure.
3. Train the FIS using optimization methods.
4. Validate the trained FIS model.

Access the online help topics by clicking **Help** in the ANFIS Editor.



Loading, Plotting, and Clearing the Data

To train a FIS, you must begin by loading a **Training** data set that contains the desired input/output data of the system to be modeled. Any data set you load must be an array with the data arranged as column vectors, and the output data in the last column.

You can also load **Testing** and **Checking** data in the GUI. For more information on testing and checking data sets, see "Model Validation Using Testing and Checking Data Sets" on page 2-93.

To load a data set using the **Load data** portion of the GUI:

1. Specify the data **Type**.
2. Select the data from a **file** or the MATLAB **worksp**.
3. Click **Load Data**.

After you load the data, it displays in the plot. The training, testing and checking data are annotated in blue as *circles*, *diamonds*, and *pluses* respectively.

To clear a specific data set from the GUI:

1. In the **Load data** area, select the data **Type**.
2. Click **Clear Data**.

This action also removes the corresponding data from the plot.

Generating or Loading the Initial FIS Structure

Before you start the FIS training, you must specify an initial FIS model structure. To specify the model structure, perform one of the following tasks:

- Load a previously saved Sugeno-type FIS structure from a file or the MATLAB workspace.
- Generate the initial FIS model by choosing one of the following partitioning techniques:
  - **Grid partition**— Generates a single-output Sugeno-type FIS by using grid partitioning on the data.
  - **Sub. clustering** — Generates an initial model for ANFIS training by first applying subtractive clustering on the data.

To view a graphical representation of the initial FIS model structure, click **Structure**.

Training the FIS

After loading the training data and generating the initial FIS structure, you can start training the FIS.

The following steps show you how to train the FIS.

1. In **Optim. Method**, choose **hybrid** or **backpropaga** as the optimization method.

   The optimization methods train the membership function parameters to emulate the training data.

   Note The **hybrid** optimization method is a combination of least-squares and backpropagation gradient descent method.

   ---

2. Enter the number of training **Epochs** and the training **Error Tolerance** to set the stopping criteria for training.

   The training process stops whenever the maximum epoch number is reached or the training error goal is achieved.

3. Click **Train Now** to train the FIS.

   This action adjusts the membership function parameters and displays the error plots.

Examine the error plots to determine overfitting during the training. If you notice the checking error increasing over iterations, it indicates model overfitting. For examples on model overfitting, see "ANFIS Editor GUI Example 1: Checking Data Helps Model Validation" on page 2-98 and "ANFIS Editor GUI Example 2: Checking Data Does Not Validate Model" on page 2-107.

Validating the Trained FIS

After the FIS is trained, validate the model using a **Testing** or **Checking** data that differs from the one you used to train the FIS. To validate the trained FIS:

1. Select the validation data set and click **Load Data**.
2. Click **Test Now**.

   This action plots the test data against the FIS output (shown in red) in the plot.

For more information on the use of testing data and checking data for model validation, see "Model Validation Using Testing and Checking Data Sets" on page 2-93.

ANFIS Editor GUI Example 1: Checking Data Helps Model Validation

In this section we look at an example that loads similar training and checking data sets. The checking data set is corrupted by noise.

Loading Data

To work both of the following examples, you load the training data sets (`fuzex1trnData` and `fuzex2trnData`) and the checking data sets (`fuzex1chkData` and `fuzex2chkData`), into the ANFIS Editor GUI from the workspace. You may also substitute your own data sets.

To load the data sets from the workspace into the ANFIS Editor GUI:

1. Type

   ```
   load fuzex1trnData.dat
   load fuzex2trnData.dat
   load fuzex1chkData.dat
   load fuzex2chkData.dat
   ```

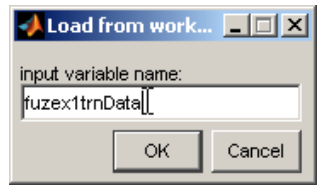
   in the MATLAB command line to load these data sets from the directory `fuzzydemos` into the MATLAB workspace.

Note You might also want to load your data set from the `fuzzydemos` or any other directory using the ANFIS Editor GUI.

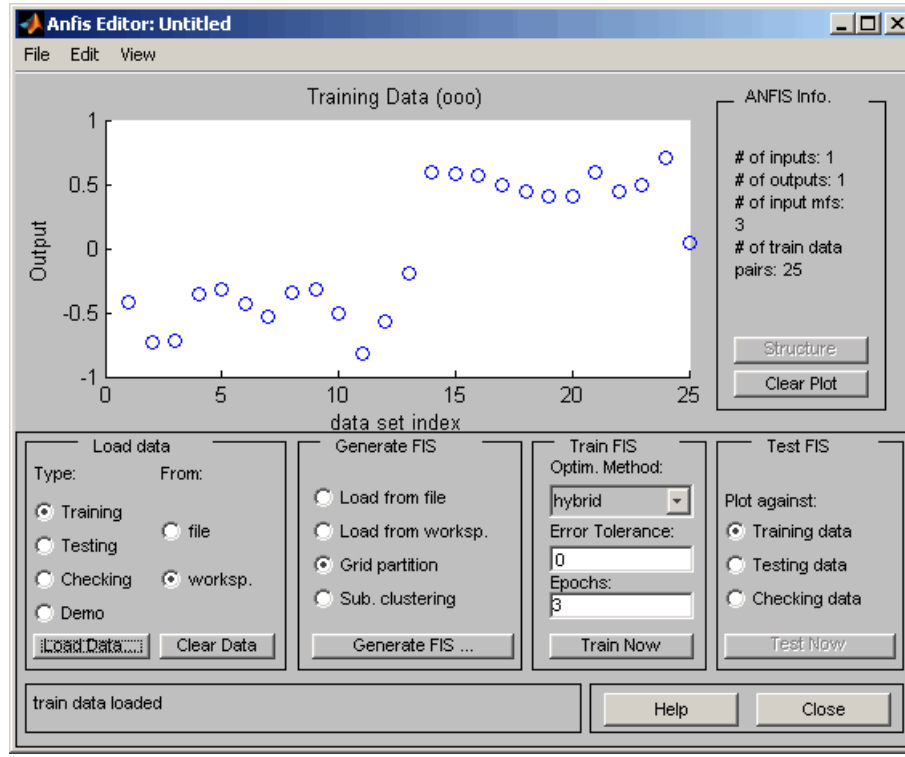
2. Open the ANFIS Editor GUI by typing `anfisedit` in the MATLAB command line.

3. To load the training data set from the workspace, select **Type: Training**, **From: worksp.** in the **Load data** portion of the ANFIS Editor GUI. Then, click the **Load Data** button.

   The small GUI window that opens allows you to type in a variable name from the workspace. Type in `fuzex1trnData` as shown in the following figure, and click **OK**.



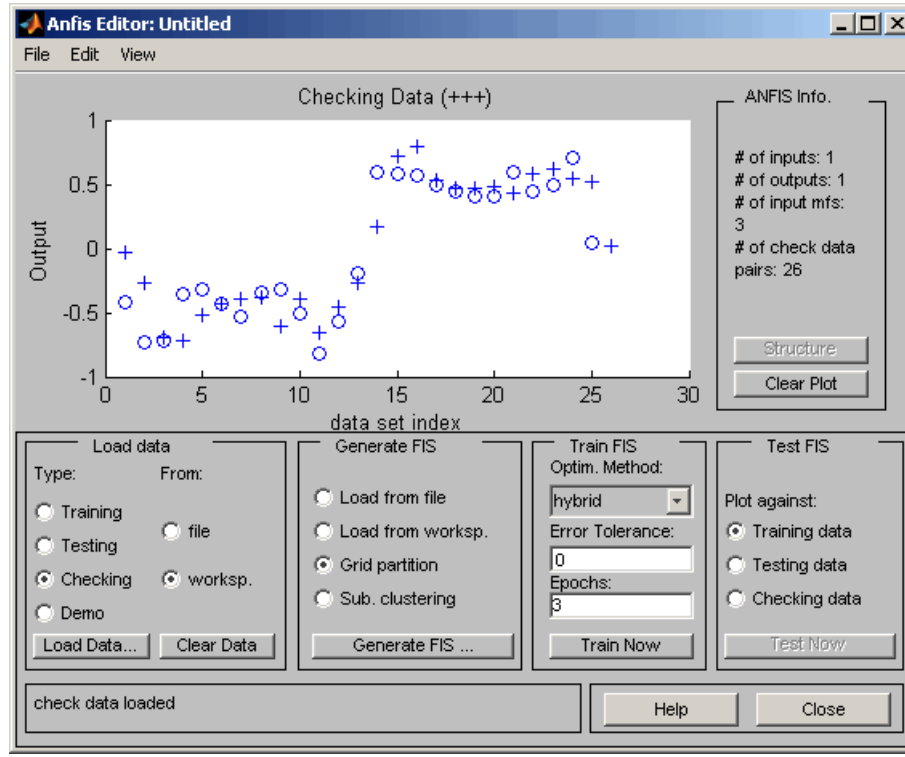
   The training data appears in the plot in the center of the GUI as a set of *circles*.



The horizontal axis is marked **data set index**. This index indicates the row from which that input data value was obtained (whether or not the input is a vector or a scalar).

4 Next, select **Checking** in the **Type** column of the **Load data** portion of the ANFIS Editor GUI to load `fuzex1chkData` from the workspace. Then, click the **Load Data** button. Type in `fuzex1trnData` as the variable name and click OK.

This data appears in the GUI plot as *pluses* superimposed on the training data.



This data set is used to train a fuzzy system by adjusting the membership function parameters that best model this data. The next step is to specify an initial fuzzy inference system for `anfis` to train.

Initializing and Generating Your FIS

You can either initialize the FIS parameters to your own preference, or if you do not have any preference for how you want the initial membership functions to be parameterized, you can let `anfis` initialize the parameters for you.

Automatic FIS Structure Generation with ANFIS

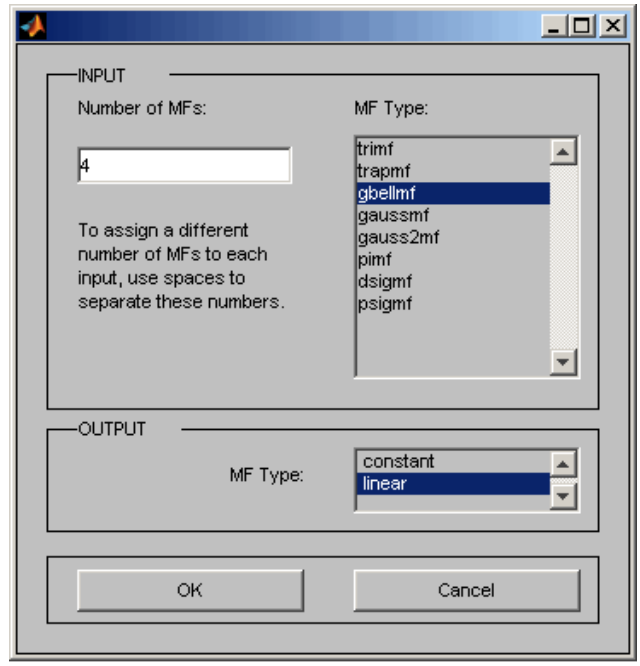
To initialize your FIS using `anfis`:

1 Choose **Grid partition,** the default partitioning method. The two partition methods, grid partitioning and subtractive clustering, are described later in

"Fuzzy C-Means Clustering" on page 2-124, and in "Subtractive Clustering" on page 2-128.

2 Click on the **Generate FIS** button. Clicking this button displays a menu from which you can choose the number of membership functions, **MFs**, and the type of input and output membership functions. There are only two choices for the output membership function: `constant` and `linear`. This limitation of output membership function choices is because `anfis` only operates on Sugeno-type systems.

3 Fill in the entries as shown in the following figure, and click **OK.**



You can also implement this FIS generation from the command line using the command `genfis1` (for grid partitioning) or `genfis2` (for subtractive clustering).

Specifying Your Own Membership Functions for ANFIS

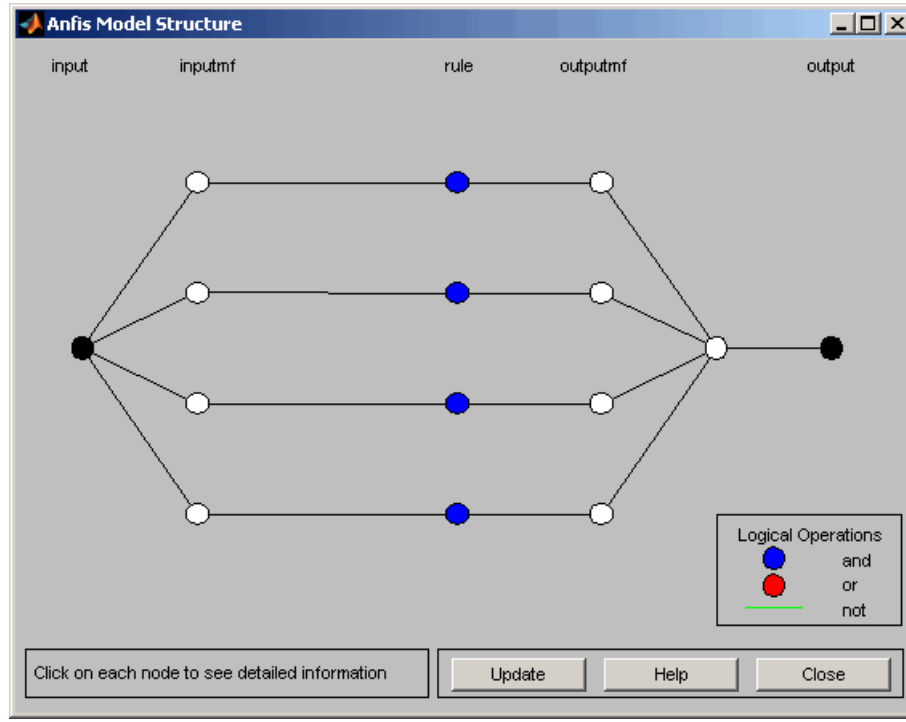
You can choose your own preferred membership functions with specific parameters to be used by `anfis` as an initial FIS for training.

To define your own FIS structure and parameters:

1. Open the **Membership functions** menu item from the **Edit** menu.
2. Add your desired membership functions (the custom membership option will be disabled for `anfis`). The output membership functions must either be all constant or all linear. For carrying out this and the following step, see "The FIS Editor" on page 2-34 and "The Membership Function Editor" on page 2-39.
3. Select the **Rules** menu item in the **Edit** menu, and use the Rule Editor to generate the rules (see "The Rule Editor" on page 2-45).
4. Select the **FIS Properties** menu item from the **Edit** menu. Name your FIS, and save it to either the workspace or to file.
5. Click the **Close** button to return to the ANFIS Editor GUI to train the FIS.
6. To load an existing FIS for ANFIS initialization, in the **Generate FIS** portion of the GUI, click **Load from worksp.** or **Load from file**. You load your FIS from a file if you have saved a FIS previously that you would like to use. Otherwise you load your FIS from the workspace.

Viewing Your FIS Structure

After you generate the FIS, you can view the model structure by clicking the **Structure** button in the middle of the right side of the GUI. A new GUI appears, as follows.



The branches in this graph are color coded. Color coding of branches characterize the rules and indicate whether or not *and*, *not*, or *or* are used in the rules. The input is represented by the left-most node and the output by the right-most node. The node represents a normalization factor for the rules. Clicking on the nodes indicates information about the structure.

You can view the membership functions or the rules by opening either the Membership Function Editor, or the Rule Editor from the **Edit** menu.

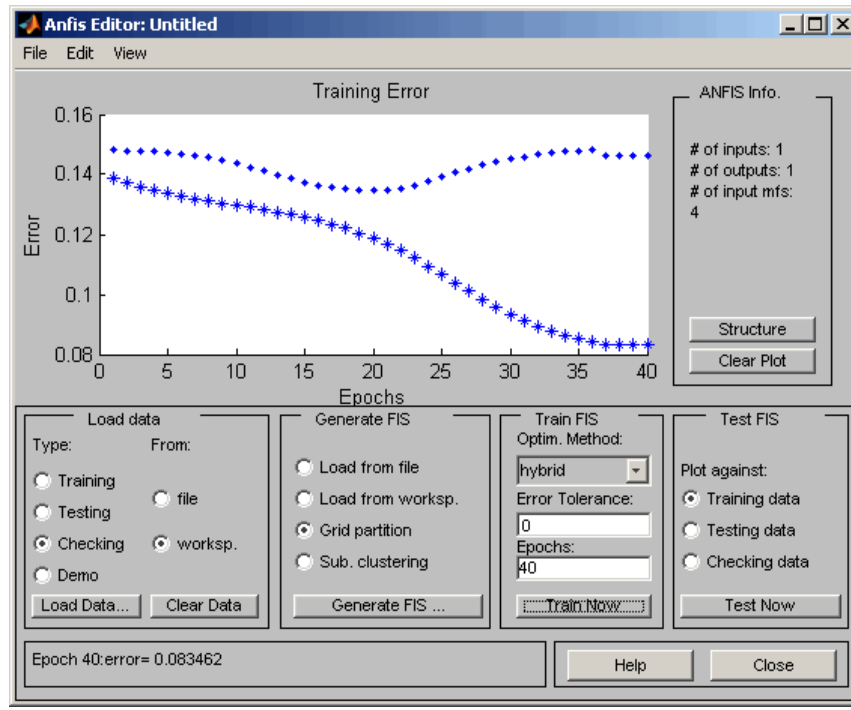
ANFIS Training

The two `anfis` parameter optimization method options available for FIS training are `hybrid` (the default, mixed least squares and backpropagation) and `backpropa` (backpropagation). **Error Tolerance** is used to create a training stopping criterion, which is related to the error size. The training will stop after the training data error remains within this tolerance. This is best left set to 0 if you are unsure how your training error may behave.

To start the training:

1. Leave the optimization method at `hybrid`.
2. Set the number of training epochs to 40, under the **Epochs** listing on the GUI (the default value is 3).
3. Select **Train Now**.

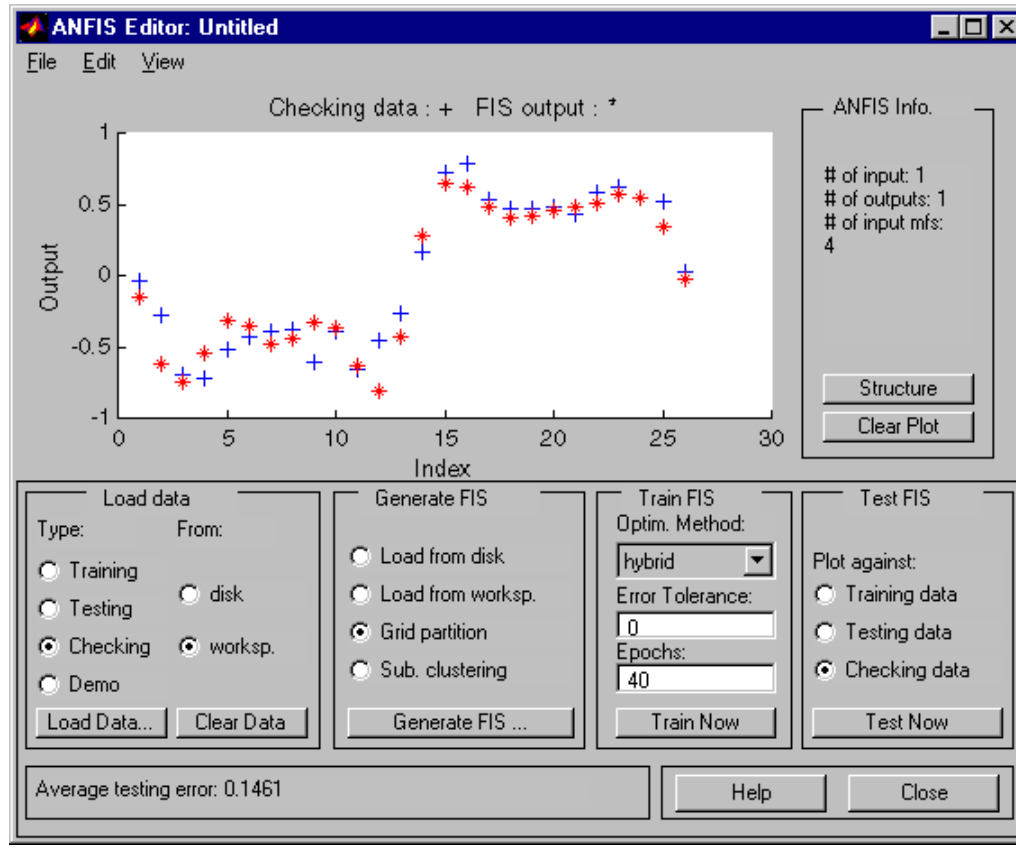
The following window appears on your screen.



The plot shows the checking error as ♦ ♦ on the top . The training error appears as * * on the bottom. The checking error decreases up to a certain point in the training, and then it increases. This increase represents the point of model overfitting. `anfis` chooses the model parameters associated with the minimum checking error (just prior to this jump point). This example shows why the checking data option of `anfis` is useful.

Testing Your Data Against the Trained FIS

To test your FIS against the checking data, select **Checking data** in the **Test FIS** portion of the ANFIS Editor GUI, and click **Test Now**. When you test the checking data against the FIS, it looks satisfactory.



Loading More Data with anfis

If you load data into `anfis` after clearing previously loaded data, you must make sure that the newly loaded data sets have the same number of inputs as the previously loaded ones did. Otherwise, you must start a new `anfisedit` session from the command line.

Checking Data Option and Clearing Data

If you do not want to use the checking data option of `anfis`, then do not load any checking data before you train the FIS. If you decide to retrain your FIS with no checking data, you can unload the checking data in one of two ways:

- Select the **Checking** option button in the **Load data** portion of the ANFIS Editor GUI, and then click **Clear Data** to unload the checking data.
- Close the ANFIS Editor GUI, and go to the MATLAB command line, and retype `anfisedit`. In this case you must reload the training data.

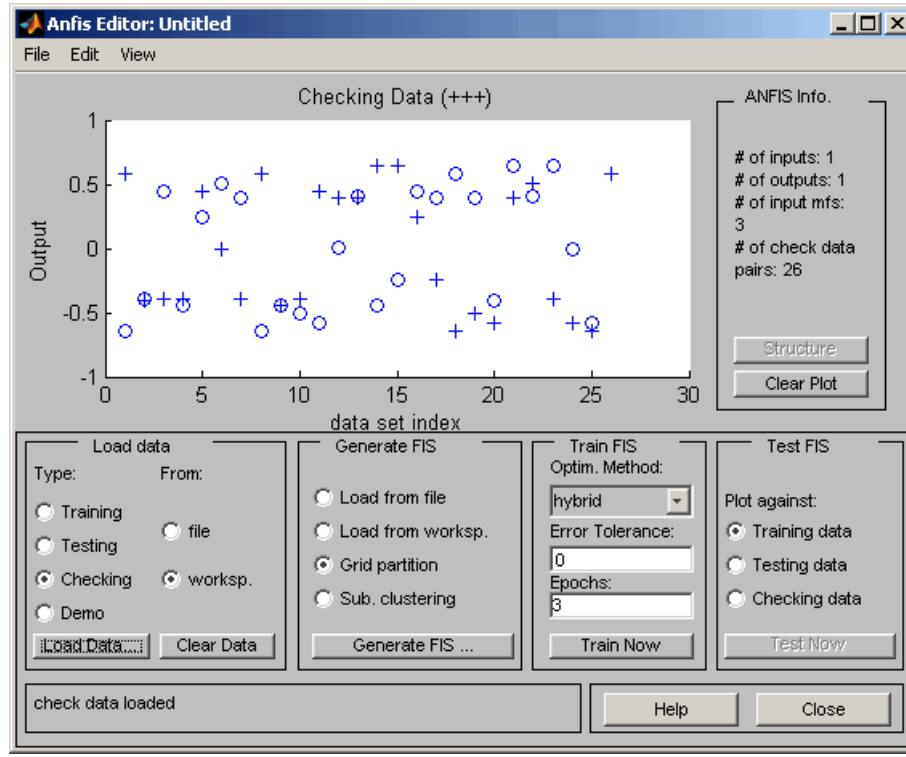
After clearing the data, you must regenerate your FIS. After the FIS is generated, you can use your first training experience to decide on the number of training epochs you want for the second round of training.

ANFIS Editor GUI Example 2: Checking Data Does Not Validate Model

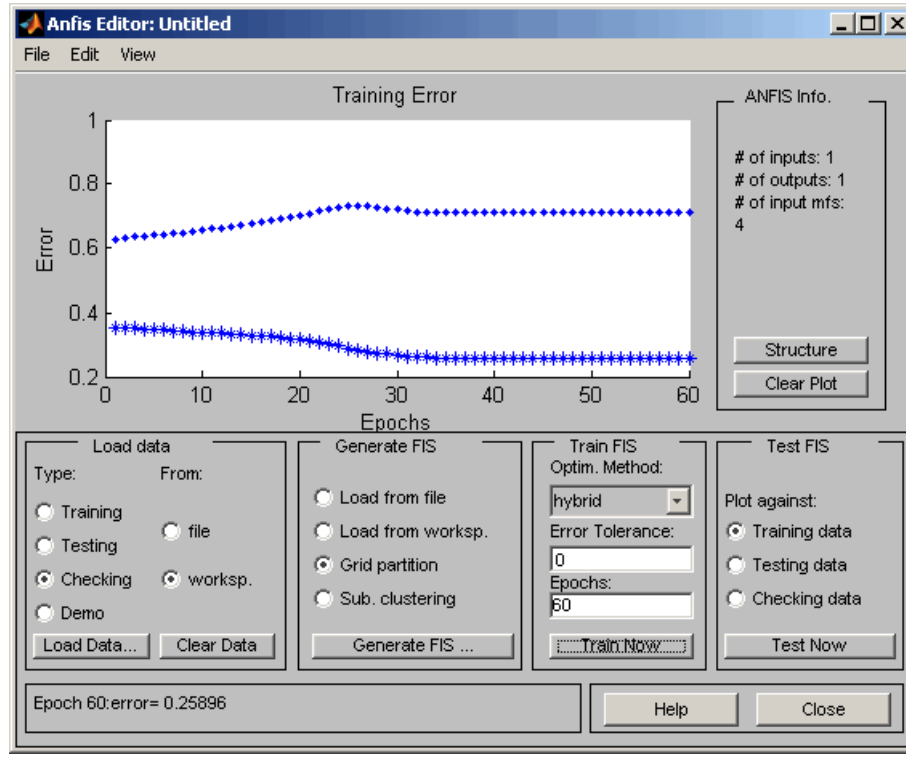
This example examines what happens when the training and checking data sets are sufficiently different. To see how the ANFIS Editor GUI can be used to learn something about data sets and how they differ:

1. Clear the ANFIS Editor GUI:
   - Clear both the training and checking data.
   - (optional) Click the **Clear Plot** button on the right.
2. Load `fuzex2trnData` and `fuzex2chkData` (respectively, the training data and checking data) from the MATLAB workspace just as you did in the previous example.

You should see a plot similar to the one in the following figure. The training data appears as *circles* superimposed with the checking data, appearing as *pluses*.



Train the FIS for this system exactly as you did in the previous example, except now choose 60 **Epochs** before training. You should get the following plot, showing the checking error as ♦ ♦ on top and the training error as * * on the bottom.

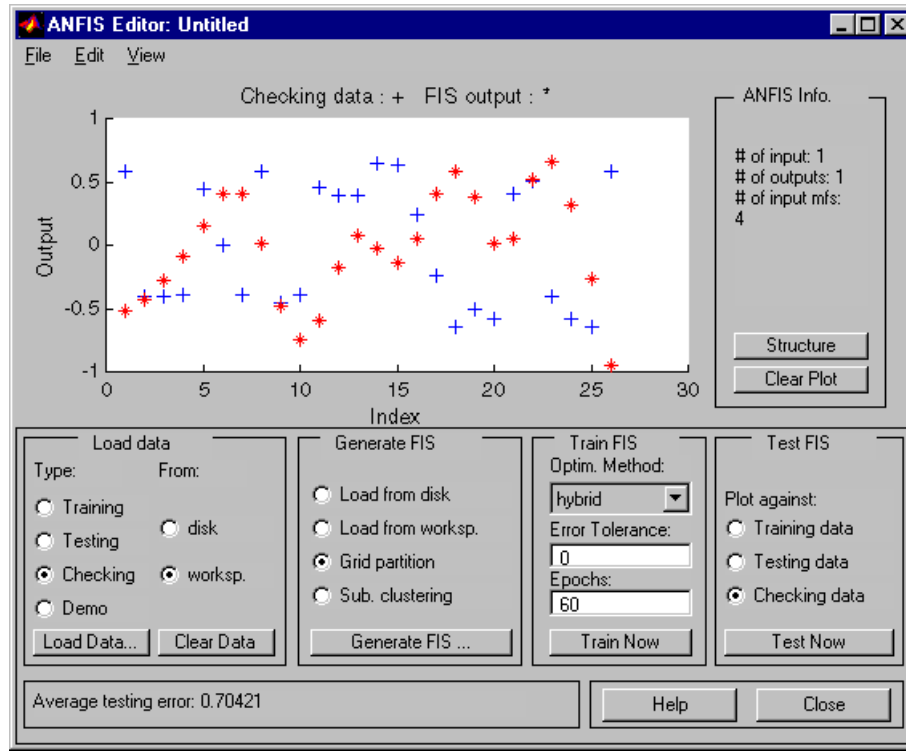


In this case, the checking error is quite large. It appears that the minimum checking error occurs within the first epoch. Using the checking data option with `anfis` automatically sets the FIS parameters to be those associated with the minimum checking error. Clearly this set of membership functions is not the best choice for modeling the training data.

This example illustrates the problem discussed earlier wherein the checking data set presented to `anfis` for training was sufficiently different from the training data set. As a result, the trained FIS did not capture the features of this data set very well. It is important to know the features of your data set well when you select your training and checking data. When you do not know the features of your data, you can analyze the checking error plots to see whether or not the checking data performed sufficiently well with the trained model.

In this example, the checking error is sufficiently large to indicate that either you need to select more data for training or modify your membership function choices (both the number of membership functions and the type). Otherwise, the system can be retrained without the checking data, if you think the training data sufficiently captures the features you are trying to represent.

To complete this example, test the trained FIS model against the checking data. To do so, select **Checking data** in the **Test FIS** portion of the GUI, and click **Test Now**. The following plot in the GUI indicates that there is quite a discrepancy between the checking data output and the FIS output.



anfis from the Command Line

Generating a FIS using the ANFIS Editor GUI is quite simple. However, as you saw in the last example, you need to be cautious about implementing the

checking data validation feature of `anfis`. You must check that the checking data error does what is supposed to. Otherwise, you need to retrain the FIS.

This section describes how to carry out the command line features of `anfis` on a chaotic times-series prediction example.

Using anfis for Chaotic Time-Series Prediction

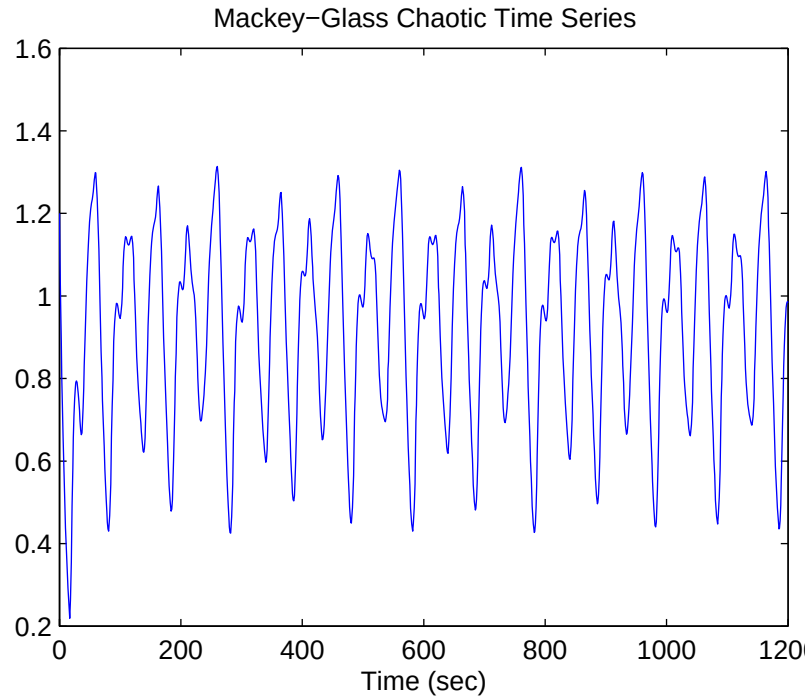
The demo `mgtsdemo` uses `anfis` to predict a time series that is generated by the following Mackey-Glass (MG) time-delay differential equation.

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t)$$

This time series is chaotic, and so there is no clearly defined period. The series does not converge or diverge, and the trajectory is highly sensitive to initial conditions. This benchmark problem is used in the neural network and fuzzy modeling research communities.

To obtain the time series value at integer points, we applied the fourth-order Runge-Kutta method to find the numerical solution to the previous MG equation; the result was saved in the file `mgdata.dat`. Assume $x(0) = 1.2$, $\tau = 17$, and $x(t) = 0$ for $t < 0$. To plot the MG time series, type

```
load mgdata.dat
t = mgdata(:, 1); x = mgdata(:, 2);
figure(1), plot(t, x);
```



In time-series prediction, you need to use known values of the time series up to the point in time, say, t, to predict the value at some point in the future, say, $t+P$. The standard method for this type of prediction is to create a mapping from D sample data points, sampled every Δ units in time, ($x(t-(D-1)\Delta)$,..., $x(t-\Delta)$, x(t)), to a predicted future value $x(t+P)$. Following the conventional settings for predicting the MG time series, set $D = 4$ and $\Delta = P = 6$. For each t, the input training data for `anfis` is a four-dimensional vector of the following form.

```
w(t) = [x(t-18) x(t-12) x(t-6) x(t)]
```

The output training data corresponds to the trajectory prediction.

```
s(t) = x(t+6)
```

For each t, ranging in values from 118 to 1117, the training input/output data is a structure whose first component is the four-dimensional input w, and whose second component is the output `s`. There is 1000 input/output data values. You use the first 500 data values for the `anfis` training (these become the training data set), while the others are used as checking data for validating the identified fuzzy model. This division of data values results in two 500-point data structures, `trnData` and `chkData`.

The following code generates this data:

```
for t=118:1117,
Data(t-117,:)=[x(t-18) x(t-12) x(t-6) x(t) x(t+6)];
end
trnData=Data(1:500, :);
chkData=Data(501:end, :);
```

To start the training, you need a FIS structure that specifies the structure and initial parameters of the FIS for learning. The `genfis1` function handles this specification.

```
fismat = genfis1(trnData);
```

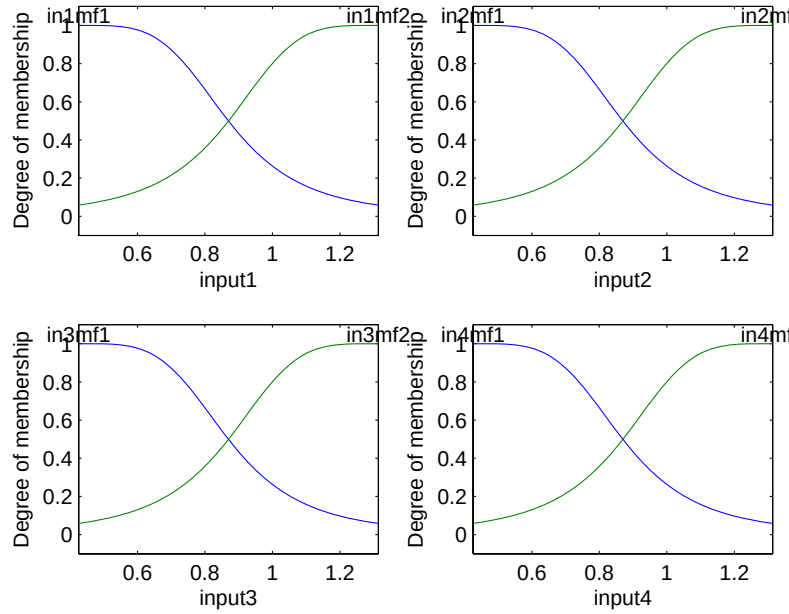
Because you did not specify numbers and types of membership functions used in the FIS, default values are assumed. These defaults provide two generalized bell membership functions on each of the four inputs, eight altogether. The generated FIS structure contains 16 fuzzy rules with 104 parameters. To achieve good generalization capability, it is important that the number of training data points be several times larger than the number parameters being estimated. In this case, the ratio between data and parameters is about five (500/104).

The function `genfis1` generates initial membership functions that are equally spaced and cover the whole input space. You can plot the input membership functions using the following commands.

```
figure(2)
subplot(2,2,1)
plotmf(fismat, 'input', 1)
subplot(2,2,2)
plotmf(fismat, 'input', 2)
subplot(2,2,3)
```

```
plotmf(fismat, 'input', 3)
subplot(2,2,4)
plotmf(fismat, 'input', 4)
```

These following plots show these initial membership functions.



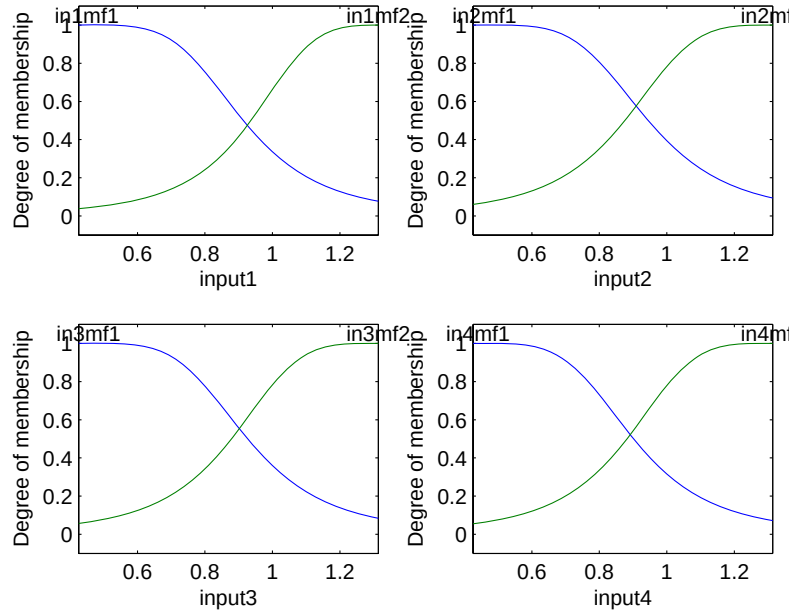
To start the training, type

```
[fismat1,error1,ss,fismat2,error2] = ...
   anfis(trnData,fismat,[],[],chkData);
```

Because the checking data option of `anfis` is invoked, the final FIS you choose is the one associated with the minimum checking error. This result is stored in `fismat2`. The following code plots these new membership functions.

```
figure(3)
subplot(2,2,1)
plotmf(fismat2, 'input', 1)
subplot(2,2,2)
plotmf(fismat2, 'input', 2)
subplot(2,2,3)
plotmf(fismat2, 'input', 3)
subplot(2,2,4)
plotmf(fismat2, 'input', 4)
```

The following plots show the result.



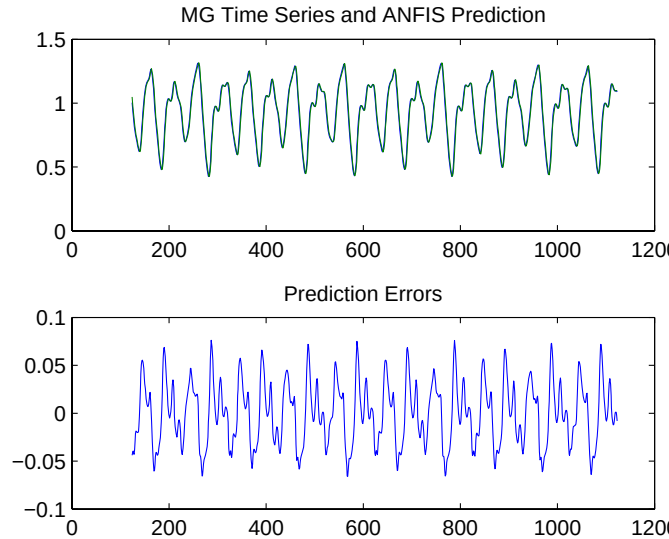
To plot the error signals, type

```
figure(4)
plot([error1; error2]);
```

In this case, `error1` and `error2` are the root mean squared error for the training and checking data, respectively.

In addition to these error plots, you may want to plot the FIS output versus the training or checking data. To compare the original MG time series and the fuzzy prediction side by side, try

```
figure(5)
anfis_output = evalfis([trnData(:,1:4); chkData(:,1:4)], ...
    fismat2);
index = 125:1124;
subplot(211), plot(t(index), [x(index) anfis_output]);
subplot(212), plot(t(index), x(index) - anfis_output);
```



The difference between the original MG time series and the `anfis` estimated values is very small; thus, you can only see one curve in the first plot. The prediction error is shown in the second plot with a much finer scale. You trained for only 10 epochs. If you apply more extensive training, you get better performance.

More on anfis and the ANFIS Editor GUI

The command `anfis` takes at least two and at most six input arguments. The general format is

```
[fismat1,trnError,ss,fismat2,chkError] = ...
    anfis(trnData,fismat,trnOpt,dispOpt,chkData,method);
```

where `trnOpt` (training options), `dispOpt` (display options), `chkData` (checking data), and `method` (training method), are optional. All output arguments are also optional. This section discusses the arguments and range components of the command line function `anfis`and the analogous functionality of the ANFIS Editor GUI.

When the ANFIS Editor GUI is invoked using `anfisedit`, only the training data set must exist prior to implementing `anfis`. In addition, the step-size is fixed when the adaptive neuro-fuzzy system is trained using this GUI tool.

Training Data

The training data, `trnData`, is a required argument to `anfis`, as well as to the ANFIS Editor GUI. Each row of `trnData` is a desired input/output pair of the target system you want to model Each row starts with an input vector and is followed by an output value. Therefore, the number of rows of `trnData` is equal to the number of training data pairs, and, because there is only one output, the number of columns of `trnData` is equal to the number of inputs plus one.

Input FIS Structure

You can obtain the input FIS structure, `fismat`, from any of the fuzzy editors:

- The FIS Editor
- The Membership Function Editor
- The Rule Editor from the ANFIS Editor GUI (which allows a FIS structure to be loaded from a file or the MATLAB workspace)
- The command line function, `genfis1` (for which you only need to give numbers and types of membership functions)

The FIS structure contains both the model structure, (which specifies such items as the number of rules in the FIS, the number of membership functions for each input, etc.), and the parameters, (which specify the shapes of membership functions).

There are two *methods* that `anfis` learning employs for updating membership function parameters:

- Backpropagation for all parameters (a steepest descent method)
- A hybrid method consisting of backpropagation for the parameters associated with the input membership functions, and least squares estimation for the parameters associated with the output membership functions

As a result, the training error decreases, at least locally, throughout the learning process. Therefore, the more the initial membership functions resemble the optimal ones, the easier it will be for the model parameter training to converge. Human expertise about the target system to be modeled may aid in setting up these initial membership function parameters in the FIS structure.

The `genfis1` function produces a FIS structure based on a fixed number of membership functions. This structure invokes the so-called *curse of dimensionality*, and causes excessive propagation of the number of rules when the number of inputs is moderately large, that is, more than four or five. Fuzzy Logic Toolbox offers a method that provides for some dimension reduction in the fuzzy inference system: you can generate a FIS structure using the clustering algorithm discussed in "Subtractive Clustering" on page 2-128. To use the clustering algorithm, you must select the **Sub. Clustering** option in the **Generate FIS** portion of the ANFIS Editor GUI before the FIS is generated. This subtractive clustering method partitions the data into groups called clusters, and generates a FIS with the minimum number rules required to distinguish the fuzzy qualities associated with each of the clusters.

Training Options

The ANFIS Editor GUI tool allows you to choose your desired error tolerance and number of training epochs.

Training option `trnOpt` for the command line `anfis` is a vector that specifies the stopping criteria and the step-size adaptation strategy:

- `trnOpt(1)`: number of training epochs, default = 10
- `trnOpt(2)`: error tolerance, default = 0
- `trnOpt(3)`: initial step-size, default = 0.01
- `trnOpt(4)`: step-size decrease rate, default = 0.9
- `trnOpt(5)`: step-size increase rate, default = 1.1

If any element of `trnOpt` is an `NaN` or missing, then the default value is taken. The training process stops if the designated epoch number is reached or the error goal is achieved, whichever comes first.

Usually, the step-size profile is a curve that increases initially, reaches some maximum, and then decreases for the remainder of the training. You achieve this ideal step-size profile by adjusting the initial step-size and the increase and decrease rates (`trnOpt(3) - trnOpt(5)`). The default values are set up to cover a wide range of learning tasks. For any specific application, you may want to modify these step-size options in order to optimize the training. However, there are no user-specified step-size options for training the adaptive neuro fuzzy inference system generated using the ANFIS Editor GUI.

Display Options

Display options apply only to the command-line function `anfis`.

For the command line `anfis`, the display options *argument*, `dispOpt`, is a vector of either 1s or 0s that specifies what information to display, (print in the MATLAB command window), before, during, and after the training process. A 1 is used to denote *print this option*, whereas a 0 denotes *do not print this option*:

- `dispOpt(1)`: display ANFIS information, default = 1
- `dispOpt(2)`: display error (each epoch), default = 1
- `dispOpt(3)`: display step-size (each epoch), default = 1
- `dispOpt(4)`: display final results, default = 1

The default mode displays all available information. If any element of `dispOpt` is `NaN` or missing, the default value is used.

Method

Both the ANFIS Editor GUI and the command line `anfis` apply either a backpropagation form of the steepest descent method for membership function parameter estimation, or a combination of backpropagation and the least-squares method to estimate membership function parameters. The choices for this argument are `hybrid` or `backpropagation`. These method choices are designated in the command line function, `anfis`, by 1 and 0, respectively.

Output FIS Structure for Training Data

`fismat1` is the output FIS structure corresponding to a minimal training error. This FIS structure is the one that you use to represent the fuzzy system when there is no checking data used for model cross-validation. This data also represents the FIS structure that is saved by the ANFIS Editor GUI when the checking data option is not used.

When you use the checking data option, the output saved is that associated with the minimum checking error.

Training Error

The training error is the difference between the training data output value, and the output of the fuzzy inference system corresponding to the same training data input value, (the one associated with that training data output value). The training error `trnError` records the root mean squared error (RMSE) of the training data set at each epoch. `fismat1` is the snapshot of the FIS structure when the training error measure is at its minimum. The ANFIS Editor GUI plots the training error versus epochs curve as the system is trained.

Step-Size

You cannot control the step-size options with the ANFIS Editor GUI. Using the command line `anfis`, the step-size array `ss` records the step-size during the training. Plotting `ss` gives the step-size profile, which serves as a reference for adjusting the initial step-size and the corresponding decrease and increase rates. The step-size (`ss`) for the command-line function `anfis` is updated according to the following guidelines:

- If the error undergoes four consecutive reductions, increase the step-size by multiplying it by a constant (`ssinc`) greater than one.
- If the error undergoes two consecutive combinations of one increase and one reduction, decrease the step-size by multiplying it by a constant (`ssdec`) less than one.

The default value for the initial step-size is 0.01; the default values for `ssinc` and `ssdec` are 1.1 and 0.9, respectively. All the default values can be changed via the training option for the command line `anfis`.

Checking Data

The checking data, `chkData`, is used for testing the generalization capability of the fuzzy inference system at each epoch. The checking data has the same format as that of the training data, and its elements are generally distinct from those of the training data.

The checking data is important for learning tasks for which the input number is large, and/or the data itself is noisy. A fuzzy inference system needs to track a given input/output data set well. Because the model structure used for `anfis` is fixed, there is a tendency for the model to overfit the data on which is it trained, especially for a large number of training epochs. If overfitting does occur, the fuzzy inference system may not respond well to other independent data sets, especially if they are corrupted by noise. A validation or checking data set can be useful for these situations. This data set is used to cross-validate the fuzzy inference model. This cross-validation requires applying the checking data to the model and then seeing how well the model responds to this data.

When the checking data option is used with `anfis`, either via the command line, or using the ANFIS Editor GUI, the checking data is applied to the model at each training epoch. When the command line `anfis` is invoked, the model parameters that correspond to the minimum checking error are returned via the output argument `fismat2`. The FIS membership function parameters computed using the ANFIS Editor GUI when both training and checking data are loaded are associated with the training epoch that has a minimum checking error.

The use of the minimum checking data error epoch to set the membership function parameters assumes

- The checking data is similar enough to the training data that the checking data error decreases as the training begins.
- The checking data increases at some point in the training after the data overfitting occurs.

Depending on the behavior of the checking data error, the resulting FIS may—or may not—be the one you need to use. Refer to “ANFIS Editor GUI Example 2: Checking Data Does Not Validate Model” on page 2-107.

Output FIS Structure for Checking Data

The output of the command line `anfis`, `fismat2`, is the output FIS structure with the minimum checking error. This FIS structure is the one that you should use for further calculation if checking data is used for cross validation.

Checking Error

The checking error is the difference between the checking data output value, and the output of the fuzzy inference system corresponding to the same checking data input value, which is the one associated with that checking data output value. The checking error `chkError` records the RMSE for the checking data at each epoch. `fismat2` is the snapshot of the FIS structure when the checking error is at its minimum. The ANFIS Editor GUI plots the checking error versus epochs curve as the system is trained.

Fuzzy Clustering

In this section...

"What is Data Clustering" on page 2-124
"Fuzzy C-Means Clustering" on page 2-124
"Subtractive Clustering" on page 2-128
"Data Clustering Using the Clustering GUI Tool" on page 2-134

What is Data Clustering

Clustering of numerical data forms the basis of many classification and system modeling algorithms. The purpose of clustering is to identify natural groupings of data from a large data set to produce a concise representation of a system's behavior.

Fuzzy Logic Toolbox is equipped with some tools that allow you to find clusters in input-output training data. You can use the cluster information to generate a Sugeno-type fuzzy inference system that best models the data behavior using a minimum number of rules. The rules partition themselves according to the fuzzy qualities associated with each of the data clusters. Use the command-line function, `genfis2` to automatically accomplish this type of FIS generation.

Fuzzy C-Means Clustering

Fuzzy c-means (FCM) is a data clustering technique wherein each data point belongs to a cluster to some degree that is specified by a membership grade. This technique was originally introduced by Jim Bezdek in 1981[1] as an improvement on earlier clustering methods. It provides a method that shows how to group data points that populate some multidimensional space into a specific number of different clusters.

The Fuzzy Logic Toolbox command line function `fcm` starts with an initial guess for the cluster centers, which are intended to mark the mean location of each cluster. The initial guess for these cluster centers is most likely incorrect. Additionally, `fcm` assigns every data point a membership grade for each cluster. By iteratively updating the cluster centers and the membership

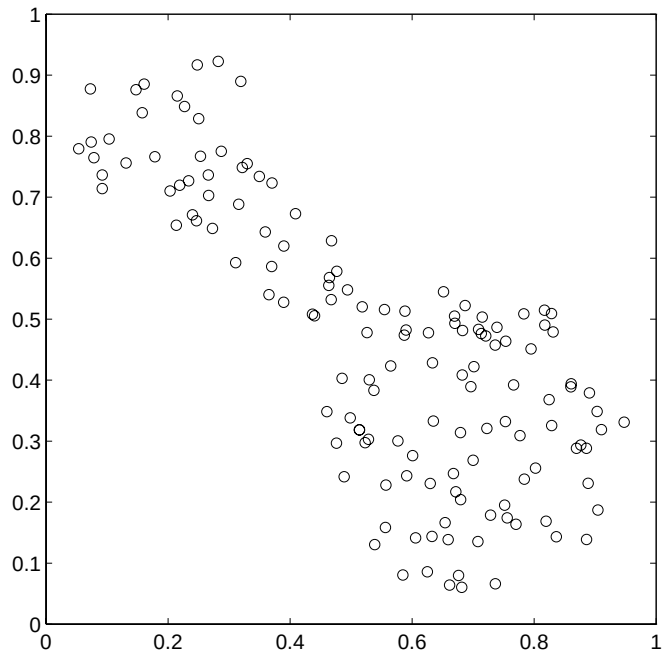
grades for each data point, `fcm` iteratively moves the cluster centers to the right location within a data set. This iteration is based on minimizing an objective function that represents the distance from any given data point to a cluster center weighted by that data point's membership grade.

The command line function `fcm` outputs a list of cluster centers and several membership grades for each data point. You can use the information returned by `fcm` to help you build a fuzzy inference system by creating membership functions to represent the fuzzy qualities of each cluster.

An Example: 2-D Clusters

You can use quasi-random two-dimensional data to illustrate how FCM clustering works. To load a data set and then review it, type

```
load fcmdata.dat
plot(fcmdata(:,1),fcmdata(:,2),'o')
```



Next, invoke the command-line function, `fcm`, and ask it to find two clusters in this data set

```
[center,U,objFcn] = fcm(fcmdata,2);
    Iteration count = 1, obj. fcn = 8.941176
    Iteration count = 2, obj. fcn = 7.277177
```

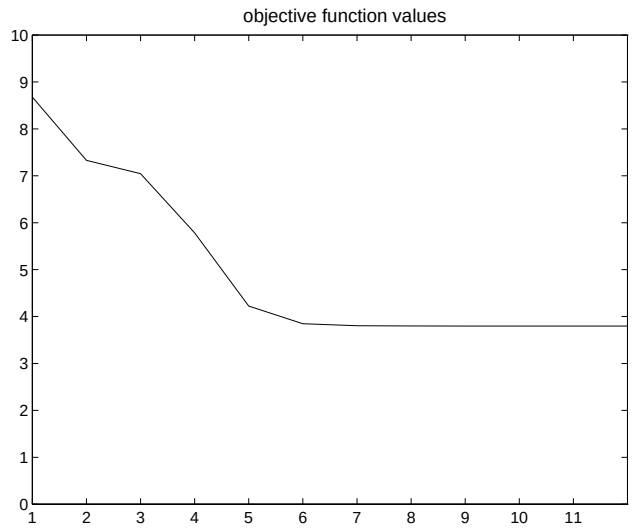
until the objective function is no longer decreasing much at all.

The variable `center` contains the coordinates of the two cluster centers, `U` contains the membership grades for each of the data points, and `objFcn` contains a history of the objective function across the iterations.

The `fcm` function is an iteration loop built on top of several other routines, namely `initfcm`, which initializes the problem, `distfcm`, which is used for distance calculations, and `stepfcm`, which steps through one iteration.

Plotting the objective function shows the progress of the clustering.

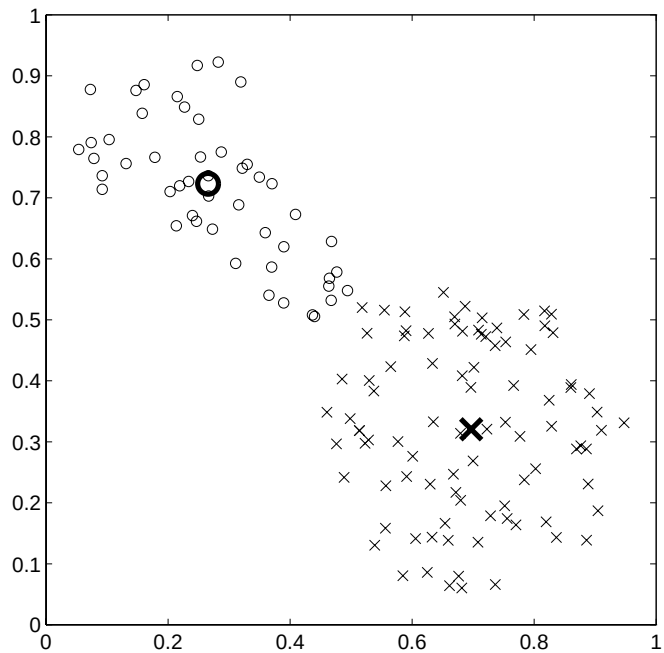
```
plot(objFcn)
```



Finally, this plot displays the two separate clusters classified by the `fcm` routine. Generate the next plot using

```
load fcmdata.dat
[center, U, obj_fcn] = fcm(fcmdata, 2);
maxU = max(U);
index1 = find(U(1, :) == maxU);
index2 = find(U(2, :) == maxU);
line(fcmdata(index1, 1), fcmdata(index1, 2), 'linestyle',...
'none','marker', 'o','color','g');
line(fcmdata(index2,1),fcmdata(index2,2),'linestyle',...
'none','marker', 'x','color','r');
hold on
plot(center(1,1),center(1,2),'ko','markersize',15,'LineWidth',2)
plot(center(2,1),center(2,2),'kx','markersize',15,'LineWidth',2)
```

In the following figure, large characters indicate cluster centers.



Subtractive Clustering

If you do not have a clear idea how many clusters there should be for a given set of data, *Subtractive clustering*, [2], is a fast, one-pass algorithm for estimating the number of clusters and the cluster centers in a set of data. The cluster estimates, which are obtained from the `subclust` function, can be used to initialize iterative optimization-based clustering methods (`fcm`) and model identification methods (like `anfis`). The `subclust` function finds the clusters by using the subtractive clustering method.

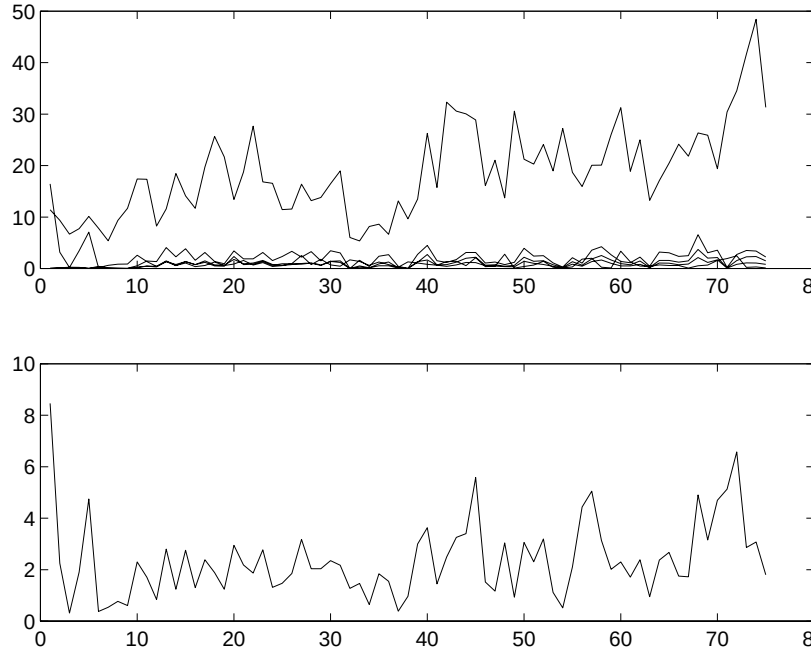
The `genfis2` function builds upon the `subclust` function to provide a fast, one-pass method to take input-output training data and generate a Sugeno-type fuzzy inference system that models the data behavior.

An Example: Suburban Commuting

In this example you apply the `genfis2` function to model the relationship between the number of automobile trips generated from an area and the area's demographics. Demographic and trip data are from 100 traffic analysis zones in New Castle County, Delaware. Five demographic factors are considered: population, number of dwelling units, vehicle ownership, median household income, and total employment. Hence, the model has five input variables and one output variable.

Load the data by typing

```
mytripdata
subplot(2,1,1), plot(datin)
subplot(2,1,2), plot(datout)
```



The function `tripdata` creates several variables in the workspace. Of the original 100 data points, use 75 data points as training data (`datin` and `datout`) and 25 data points as checking data, (as well as for test data to validate the model). The checking data input/output pairs are denoted by `chkdatin` and `chkdatout`. The `genfis2` function generates a model from data using clustering, and requires you to specify a cluster radius. The cluster radius indicates the range of influence of a cluster when you consider the data space as a unit hypercube. Specifying a small cluster radius usually yields many small clusters in the data, (resulting in many rules). Specifying a large cluster radius usually yields a few large clusters in the data, (resulting in fewer rules). The cluster radius is specified as the third argument of `genfis2`. The following syntax calls the `genfis2` function using a cluster radius of 0.5.

```
fismat=genfis2(datin,datout,0.5);
```

The genfis2 function is a fast, one-pass method that does not perform any iterative optimization. A FIS structure is returned; the model type for the FIS structure is a first order Sugeno model with three rules. Use evalfis to verify the model.

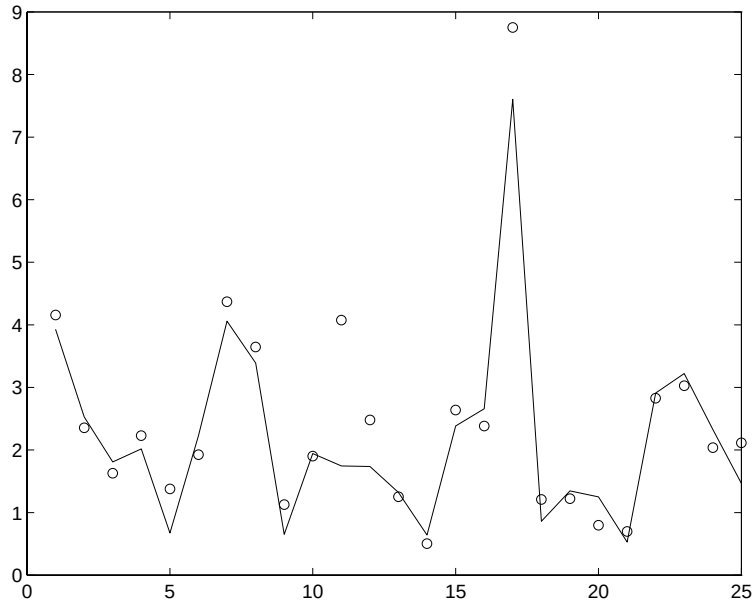
```
fuzout=evalfis(datin,fismat);
trnRMSE=norm(fuzout-datout)/sqrt(length(fuzout))
trnRMSE =
   0.5276
```

The variable trnRMSE is the root mean square error of the system generated by the training data. To validate the model, apply test data to the FIS. For this example, use the checking data for both checking and testing the FIS parameters.

```
chkfuzout=evalfis(chkdatin,fismat);
chkRMSE=norm(chkfuzout-chkdatout)/sqrt(length(chkfuzout))
chkRMSE =
   0.6170
```

This model does not perform as well on the testing data. A plot of the testing data reveals the difference.

```
plot(chkdatout)
hold on
plot(chkfuzout,'o')
hold off
```



At this point, you can use the optimization capability of `anfis` to improve the model. First, try using a relatively short `anfis` training (50 epochs) without implementing the checking data option, but test the resulting FIS model against the test data. The command-line syntax for this function is as follows:

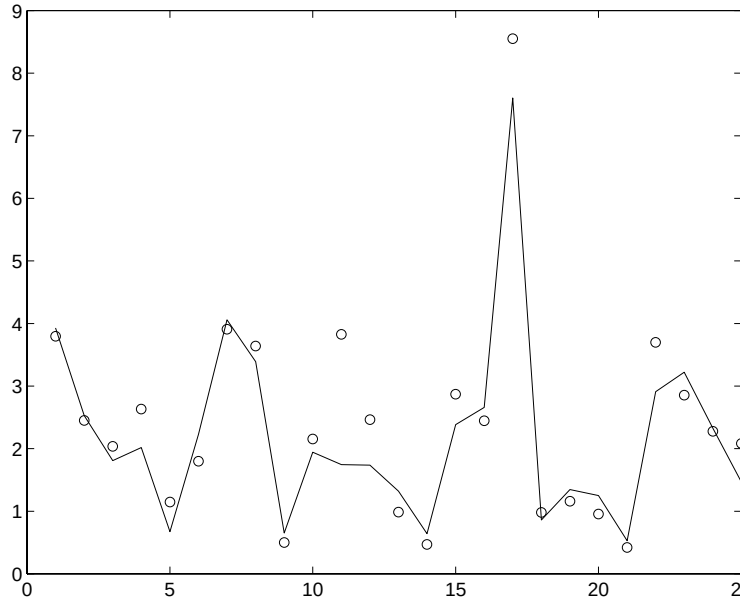
```
fismat2=anfis([datin datout],fismat,[50 0 0.1]);
```

After the training is done, type

```
fuzout2=evalfis(datin,fismat2);
trnRMSE2=norm(fuzout2-datout)/sqrt(length(fuzout2))
trnRMSE2 =
   0.3407
chkfuzout2=evalfis(chkdatin,fismat2);
chkRMSE2=norm(chkfuzout2-chkdatout)/sqrt(length(chkfuzout2))
chkRMSE2 =
   0.5827
```

The model has improved a lot with respect to the training data, but only a little with respect to the checking data. The following plot of the improved testing data.

```
plot(chkdatout)
hold on
plot(chkfuzout2,'o')
hold off
```



This plot shows that `genfis2` can be used as a stand-alone, fast method for generating a fuzzy model from data, or as a preprocessor to `anfis` for determining the initial rules. An important advantage of using a clustering method to find rules is that the resultant rules are more tailored to the input data than they are in a FIS generated without clustering. This reduces the problem of an excessive propagation of rules when the input data has a high dimension.

Overfitting

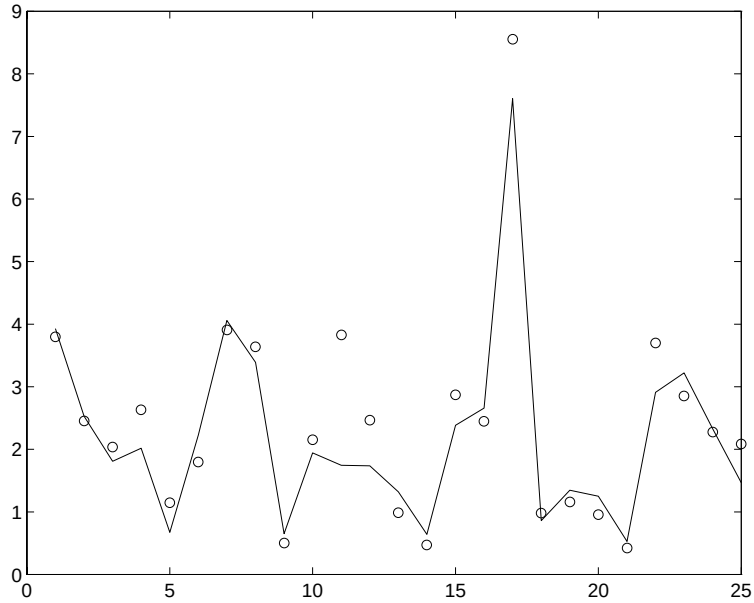
Next, consider what happens if you carry out a longer (200 epoch) training of this system using `anfis`, including its checking data option.

```
[fismat3,trnErr,stepSize,fismat4,chkErr]= ...
   anfis([datin datout],fismat2,[200 0 0.1],[], ...
   [chkdatin chkdatout]);
```

The long list of output arguments returns a history of the step sizes, the RMSE versus the training data, and the RMSE versus the checking data associated with each training epoch.

```
ANFIS training completed at epoch 200.
Minimal training RMSE = 0.326566
Minimal checking RMSE = 0.582545
```

This output looks satisfactory. The error with the training data is the lowest thus far and the error with the checking data is also slightly lower than before. This result suggests perhaps there is an overfit of the system to the training data. Overfitting occurs when you fit the fuzzy system to the training data so well that it no longer does a very good job of fitting the checking data. The result is a loss of generality. Review the following plot of error history against both the training data and the checking data.



This plot shows that the training error settles at about the 50th epoch point. The smallest value of the checking data error occurs at the 52nd epoch, after which it increases slightly, even as `anfis` continues to minimize the error against the training data all the way to the 200th epoch. Depending on the specified error tolerance, the plot also indicates the model's ability to generalize the test data.

You can clear the plotted data by clicking **Clear Plot**.

Data Clustering Using the Clustering GUI Tool

The Clustering GUI Tool implements the fuzzy data clustering functions `fcm` and `subclust` and lets you perform clustering on the data. For more information on the clustering functions, see "Fuzzy C-Means Clustering" on page 2-124 and "Subtractive Clustering" on page 2-128.

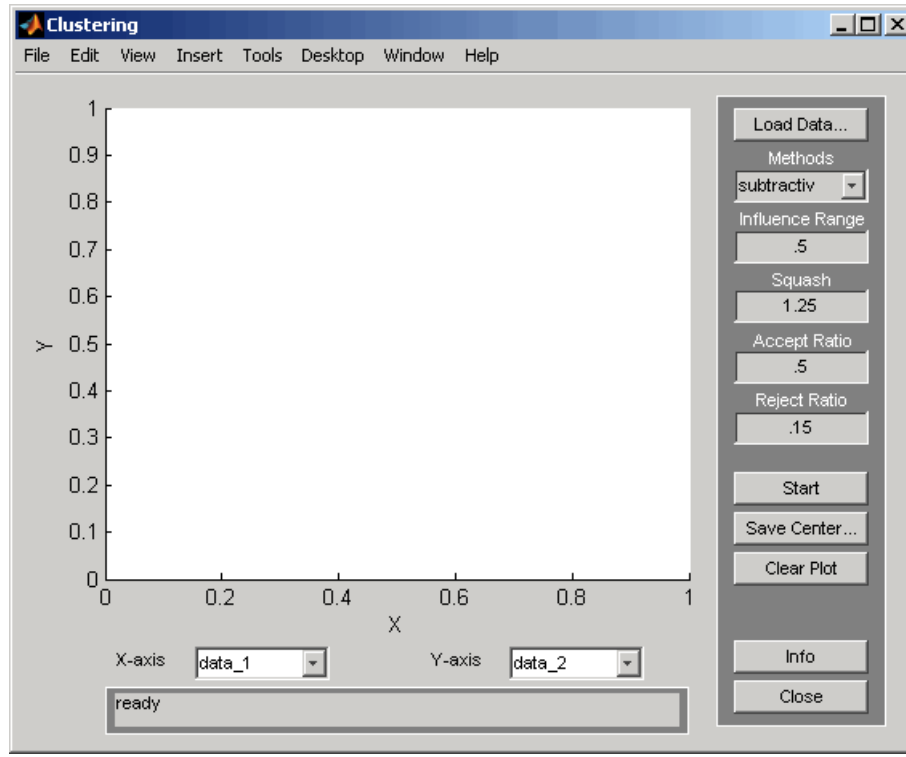
To start the GUI, type the following command at the MATLAB command prompt:

```
findcluster
```

The Clustering GUI Tool shown in the next figure lets you perform the following tasks:

1. Load and plot the data.
2. Start the clustering.
3. Save the cluster center.

Access the online help topics by clicking **Info** or using the **Help** menu in the Clustering GUI.



Loading and Plotting the Data

To load a data set in the GUI, perform either of the following actions:

- Click **Load Data**, and select the file containing the data.
- Open the GUI with a data set directly by invoking `findcluster` with the data set as the argument, in the MATLAB Command Window.

  The data set must have the extension `.dat`. For example, to load the data set, `clusterdemo.dat`, type `findcluster('clusterdemo.dat')`.

The Clustering GUI Tool works on multidimensional data sets, but displays only two of those dimensions on the plot. To select other dimensions in the data set for plotting, you can use the drop-down lists under **X-axis** and **Y-axis**.

Starting the Clustering

To start clustering the data:

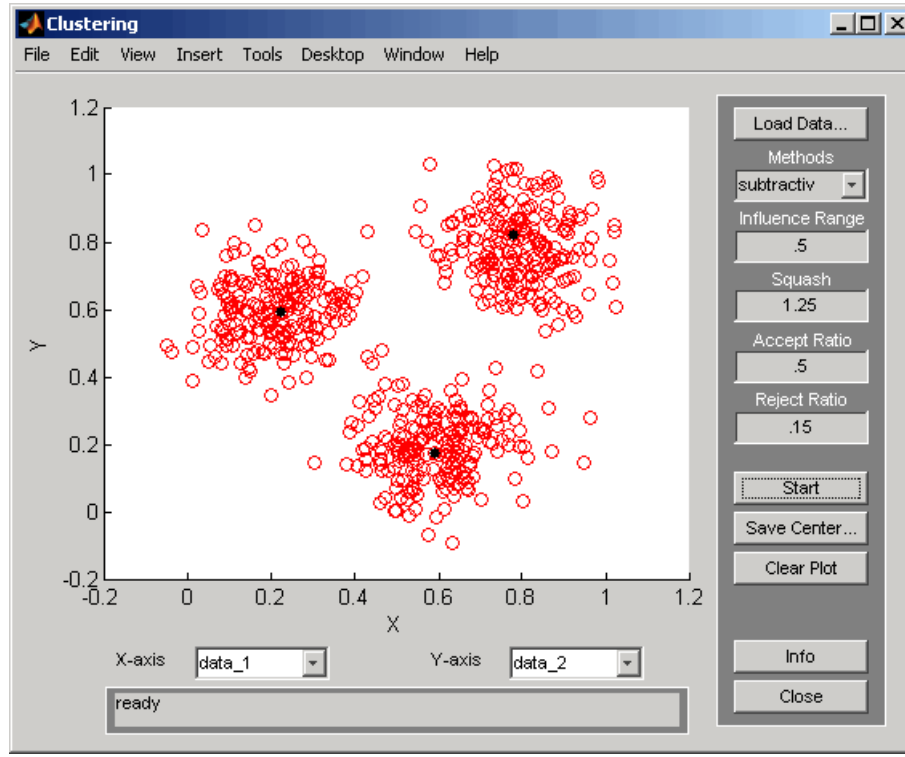
1. Choose the clustering function `fcm` (fuzzy C-Means clustering) or `subtractiv` (subtractive clustering) from the drop-down menu under **Methods**.

2. Set options for the selected method using the **Influence Range**, **Squash**, **Aspect Ratio**, and **Reject Ratio** fields.

   For more information on these methods and their options, refer to `fcm`, and `subclust` respectively.

3. Begin clustering by clicking **Start**.

   After clustering gets completed, the cluster centers appear in black as shown in the next figure.



Saving the Cluster Center

To save the cluster centers, click **Save Center**.

Stand-Alone C Code Fuzzy Inference Engine

In the `fuzzy/fuzzy` directory of the toolbox, you can find two C files, `fismain.c` and `fis.c`, which are provided as the source codes for a stand-alone fuzzy inference engine. The stand-alone C code fuzzy inference engine can read a FIS file and an input data file to perform fuzzy inference directly, or it can be embedded in other external applications.

To compile the stand-alone fuzzy inference engine on a UNIX system, type

```
%  cc -O -o fismain fismain.c -lm
```

Note `%` is only symbolic of a UNIX prompt, and that you do not have to type `fis.c` explicitly, since it is included in `fismain.c`

Upon successful compilation, type the executable command to see how it works.

```
% fismain
```

This command prompts the following message.

```
% Usage: fismain data_file fis_file
```

According to this message, `fismain` needs two files to do its work:

- A data file containing rows of input vectors
- A FIS file that specifies the fuzzy inference system under consideration

For example, consider a FIS structure file named, `mam21.fis`. You can prepare the input data file using MATLAB.

```
[x, y] = meshgrid(-5:5, -5:5);
input_data = [x(:) y(:)];
save fis_in input_data -ascii
```

This preparation saves all the input data as a 121-by-2 matrix in the ASCII file `fis_in`, where each row of the matrix represents an input vector.

Now, you can call the stand-alone code.

```
% fismain fis_in mam21.fis
```

This call generates 121 outputs on your screen. You can direct the outputs to another file.

```
% fismain fis_in mam21.fis > fis_out
```

Now, the file `fis_out` contains a 121-by-1 matrix. In general, each row of the output matrix represents an output vector. The syntax of `fismain` is similar to its MEX-file counterpart `evalfis.m`, except that all matrices are replaced with files.

To compare the results from the MATLAB MEX-file and the stand-alone executable, type the following within MATLAB.

```
fismat = readfis('mam21');
matlab_out = evalfis(input_data, fismat);
load fis_out
max(max(matlab_out - fis_out))
ans =
   4.9583e-13
```

This tiny difference comes from the limited length printout in the file `fis_out`. There are several things you should know about this stand-alone executable:

- It is ANSI C compatible.
- Customized functions are not allowed in the stand-alone executable, so you are limited to the 11 membership functions that come with the toolbox, as well as other factory settings for AND, OR, IMP, and AGG functions.
- `fismain.c` contains only the `main()` function and it is heavily documented for easy adaptation to other applications.
- To add a new membership function or new reasoning mechanism into the stand-alone code, you need to change the file `fis.c`, which contains all the necessary functions to perform the fuzzy inference process.

3

Functions — By Category

GUI Tools and Plotting Functions (p. 3-2)	Functions that invoke various graphical editors for fuzzy logic systems
Membership Functions (p. 3-3)	Membership functions for fuzzy inference systems
FIS Data Structure Management (p. 3-4)	Functions for creating and manipulating fuzzy inference systems
Advanced Techniques (p. 3-5)	Functions for generating FIS matrices, finding clusters, and training a FIS
Simulink (p. 3-6)	Fuzzy inference S-function and the Fuzzy Logic Simulink library

GUI Tools and Plotting Functions

`anfisedit`	Open ANFIS Editor GUI
`findcluster`	Interactive clustering GUI for fuzzy c-means and subclustering
`fuzzy`	Open basic Fuzzy Inference System editor
`mfedit`	Membership function editor
`plotfis`	Plot Fuzzy Inference System
`plotmf`	Plot all membership functions for given variable
`ruleedit`	Rule editor and parser
`ruleview`	Rule viewer and fuzzy inference diagram
`surfview`	Open Output Surface Viewer

Membership Functions

`dsigmf`	Built-in membership function composed of difference between two sigmoidal membership functions
`gauss2mf`	Gaussian combination membership function
`gaussmf`	Gaussian curve built-in membership function
`gbellmf`	Generalized bell-shaped built-in membership function
`pimf`	Π-shaped built-in membership function
`psigmf`	Built-in membership function composed of product of two sigmoidally shaped membership functions
`sigmf`	Sigmoidally shaped built-in membership function
`smf`	S-shaped built-in membership function
`trapmf`	Trapezoidal-shaped built-in membership function
`trimf`	Triangular-shaped built-in membership function
`zmf`	Z-shaped built-in membership function

FIS Data Structure Management

`addmf`	Add membership function to Fuzzy Inference System
`addrule`	Add rule to Fuzzy Inference System
`addvar`	Add variable to Fuzzy Inference System
`defuzz`	Defuzzify membership function
`evalfis`	Perform fuzzy inference calculations
`evalmf`	Generic membership function evaluation
`gensurf`	Generate Fuzzy Inference System output surface
`getfis`	Fuzzy system properties
`mf2mf`	Translate parameters between membership functions
`newfis`	Create new Fuzzy Inference System
`parsrule`	Parse fuzzy rules
`readfis`	Load Fuzzy Inference System from file
`rmmf`	Remove membership function from Fuzzy Inference System
`rmvar`	Remove variables from Fuzzy Inference System
`setfis`	Set fuzzy system properties
`showfis`	Display annotated Fuzzy Inference System
`showrule`	Display Fuzzy Inference System rules
`writefis`	Save Fuzzy Inference System to file

Advanced Techniques

`anfis`	Training routine for Sugeno-type Fuzzy Inference System (MEX only)
`fcm`	Fuzzy c-means clustering
`genfis1`	Generate Fuzzy Inference System structure from data without data clustering
`genfis2`	Generate Fuzzy Inference System structure from data using subtractive clustering
`genfis3`	Generate Fuzzy Inference System structure from data using FCM clustering
`subclust`	Find cluster centers with subtractive clustering

Simulink

`fuzblock`	Simulink fuzzy logic library
`sffis`	Fuzzy inference S-function for Simulink

4

Functions — Alphabetical List

addmf

Purpose

Add membership function to Fuzzy Inference System

Syntax

```
a = addmf(a,'varType',varIndex,'mfName','mfType',mfParams)
```

Description

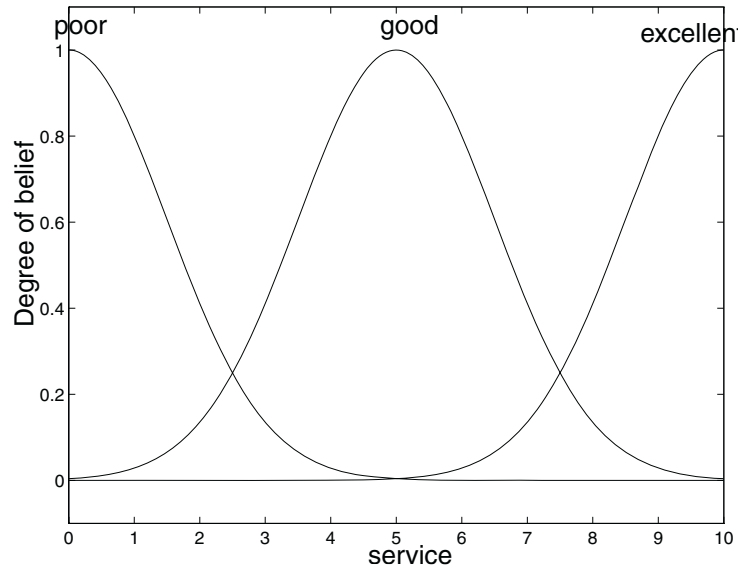
A membership function can be added only to a variable in an existing MATLAB workspace FIS. Indices are assigned to membership functions in the order in which they are added, so the first membership function added to a variable is always known as membership function number one for that variable. You cannot add a membership function to input variable number two of a system if only one input has been defined.

The function requires six input arguments in this order:

1. A MATLAB variable name of a FIS structure in the workspace
2. A string representing the type of variable you want to add the membership function to (`'input'` or `'output'`)
3. The index of the variable you want to add the membership function to
4. A string representing the name of the new membership function
5. A string representing the type of the new membership function
6. The vector of parameters that specify the membership function

Examples

```
a = newfis('tipper');
a = addvar(a,'input','service',[0 10]);
a = addmf(a,'input',1,'poor','gaussmf',[1.5 0]);
a = addmf(a,'input',1,'good','gaussmf',[1.5 5]);
a = addmf(a,'input',1,'excellent','gaussmf',[1.5 10]);
plotmf(a,'input',1)
```



See Also `addrule`, `addvar`, `plotmf`, `rmmf`, `rmvar`

Purpose

Add rule to Fuzzy Inference System

Syntax

```
a = addrule(a,ruleList)
```

Description

`addrule` has two arguments. The first argument is the MATLAB workspace variable FIS name. The second argument is a matrix of one or more rows, each of which represents a given rule. The format that the rule list matrix must take is very specific. If there are m inputs to a system and n outputs, there must be exactly $m + n + 2$ columns to the rule list.

The first m columns refer to the inputs of the system. Each column contains a number that refers to the index of the membership function for that variable.

The next n columns refer to the outputs of the system. Each column contains a number that refers to the index of the membership function for that variable.

The $m + n + 1$ column contains the weight that is to be applied to the rule. The weight must be a number between zero and one and is generally left as one.

The $m + n + 2$ column contains a 1 if the fuzzy operator for the rule's antecedent is AND. It contains a 2 if the fuzzy operator is OR.

Examples

```
ruleList=[
 1 1 1 1 1
 1 2 2 1 1];
a = addrule(a,ruleList);
```

If the system a has two inputs and one output, the first rule can be interpreted as:

"If Input 1 is MF 1 and Input 2 is MF 1, then Output 1 is MF 1."

See Also

`addmf`, `addvar`, `parsrule`, `rmmf`, `rmvar`, `showrule`

Purpose

Add variable to Fuzzy Inference System

Syntax

```
a = addvar(a,'varType','varName',varBounds)
```

Description

`addvar` has four arguments in this order:

- The name of a FIS structure in the MATLAB workspace
- A string representing the type of the variable you want to add (`'input'` or `'output'`)
- A string representing the name of the variable you want to add
- The vector describing the limiting range values for the variable you want to add

Indices are applied to variables in the order in which they are added, so the first input variable added to a system is always known as input variable number one for that system. Input and output variables are numbered independently.

Examples

```
a = newfis('tipper');
a = addvar(a,'input','service',[0 10]);
getfis(a,'input',1)
```

MATLAB replies

```
Name = service
NumMFs = 0
MFLabels =
Range = [0 10]
```

See Also

`addmf`, `addrule`, `rmmf`, `rmvar`

Purpose

Training routine for Sugeno-type Fuzzy Inference System (MEX only)

Syntax

```
[fis,error,stepsize] = anfis(trnData)
[fis,error,stepsize] = anfis(trnData,initFis)
[fis,error,stepsize] = anfis(trnData,numMFs)
[fis,error,stepsize,chkFis,chkErr] = ...
 anfis(trnData,initFis,trnOpt,dispOpt,chkData,optMethod)
[fis,error,stepsize,chkFis,chkErr] = ...
 anfis(trnData,numMFs,trnOpt,dispOpt,chkData,optMethod)
```

Description

This syntax is the major training routine for Sugeno-type fuzzy inference systems. `anfis` uses a hybrid learning algorithm to identify parameters of Sugeno-type fuzzy inference systems. It applies a combination of the least-squares method and the backpropagation gradient descent method for training FIS membership function parameters to emulate a given training data set. `anfis` can also be invoked using an optional argument for model validation. The type of model validation that takes place with this option is a checking for model overfitting, and the argument is a data set called the checking data set.

The arguments in the description for `anfis` are as follows. Note that you can specify the arguments `trnOpt`, `dispOpt`, `chkData`, and `optMethod` as empty, `[]`, when necessary:

- `trnData`: the name of a training data set. This matrix contains data input in all but the last column. The last column contains a single vector of output data.
- `initFis`: the name of a fuzzy inference system (FIS) used to provide `anfis` with an initial set of membership functions for training. Without this option, `anfis` uses `genfis1` to implement a default initial FIS for training. This default FIS has two membership functions of the Gaussian type, when it is invoked with only one argument. If `initFis` is provided as a single number (or a vector), it is taken as the number of membership functions (or the vector) whose entries are the respective numbers of membership functions associated with each respective input when these numbers differ for each input). In this case, both arguments of `anfis` are passed

to genfis1 to generate a valid FIS structure before starting the training process.

- numMFs: the number of membership functions. Use numMFs, an integer scalar value, as the second argument to anfis when you do not already have a FIS to train and you want anfis to build a default initial FIS using your data. Each input and output to this FIS is characterized by one or more membership functions. Specify the number of membership functions in numMFs.
- trnOpt: a vector of training options. When a training option is entered as NaN, the default options is in force. These options are as follows:
  - trnOpt(1): training epoch number (default: 10)
  - trnOpt(2): training error goal (default: 0)
  - trnOpt(3): initial step size (default: 0.01)
  - trnOpt(4): step size decrease rate (default: 0.9)
  - trnOpt(5): step size increase rate (default: 1.1)
- dispOpt: a vector of display options that specify what message to display in the MATLAB Command Window during training. The default value for a display option is 1, which means that the corresponding information is displayed. A 0 means the corresponding information is not displayed. When a display option is entered as NaN, the default options will be in force. These options are as follows:
  - dispOpt(1): ANFIS information, such as numbers of input and output membership functions, and so on (default: 1)
  - dispOpt(2): error (default: 1)
  - dispOpt(3): step size at each parameter update (default: 1)
  - dispOpt(4): final results (default: 1)
- chkData: the name of an optional checking data set for overfitting model validation. This data set is a matrix in the same format as the

training data set. When you supply `chkData` as an input argument, you must also supply `chkFis` and `chkErr` as output arguments.

- `optMethod`: an optional optimization method used in membership function parameter training: either `1` for the hybrid method or `0` for the backpropagation method. The default method is the hybrid method, which is a combination of least-squares estimation with backpropagation. The default method is invoked whenever the entry for this argument is anything but `0`.

The training process stops whenever the designated epoch number is reached or the training error goal is achieved.

Note When `anfis` is invoked with two or more arguments, optional arguments take on their default values if they are entered as `NaN`s or empty matrices. Default values can be changed directly by modifying the file `anfis.m`. Either `NaN`s or empty matrices must be used as placeholders for variables if you do not want to specify them, but do want to specify succeeding arguments, for example, when you implement the checking data option of `anfis`.

The range variables in the previous description for `anfis` are as follows:

- `fis` is the FIS structure whose parameters are set according to a minimum training error criterion.
- `error` or `chkErr` is an array of root mean squared errors representing the training data error signal and the checking data error signal, respectively. The function only returns `chkErr` when you supply `chkData` as an input argument.
- `stepsize` is an array of step sizes. The step size is decreased (by multiplying it with the component of the training option corresponding to the step size decrease rate) if the error measure undergoes two consecutive combinations of an increase followed by a decrease. The step size is increased (by multiplying it with

the increase rate) if the error measure undergoes four consecutive decreases.

- `chkFis` is the FIS structure whose parameters are set according to a minimum checking error criterion. The function only returns `chkFis` when you supply `chkData` as an input argument.

`anfis` has certain restrictions (see "Constraints of anfis" on page 2-94 for more information).

Examples

```
x = (0:0.1:10)';
y = sin(2*x)./exp(x/5);
trnData = [x y];
numMFs = 5;
mfType = 'gbellmf';
epoch_n = 20;
in_fis = genfis1(trnData,numMFs,mfType);
out_fis = anfis(trnData,in_fis,20);
plot(x,y,x,evalfis(x,out_fis));
legend('Training Data','ANFIS Output');
```

References

Jang, J.-S. R., "Fuzzy Modeling Using Generalized Neural Networks and Kalman Filter Algorithm," *Proc. of the Ninth National Conf. on Artificial Intelligence (AAAI-91)*, pp. 762-767, July 1991.

Jang, J.-S. R., "ANFIS: Adaptive-Network-based Fuzzy Inference Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 3, pp. 665-685, May 1993.

See Also

`anfisedit`, `genfis1`

Purpose

Open ANFIS Editor GUI

Syntax

```
anfisedit('a')
anfisedit(a)
anfisedit
```

Description

Using `anfisedit`, you bring up the ANFIS Editor GUI from which you can load a data set and train `anfis`. The ANFIS Editor GUI invoked using `anfisedit('a')`, opens the ANFIS Editor GUI from which you can implement `anfis` using a FIS structure stored as a file `a.fis`.

`anfisedit(a)` operates the same way for a FIS structure a, stored as a variable in the MATLAB workspace.

Refer to "anfis and the ANFIS Editor GUI" on page 2-90 for more information about how to use `anfisedit`.

Menu Items

On the ANFIS Editor GUI, there is a menu bar that allows you to open related GUI tools, open and save systems, and so on. The **File** menu is the same as the one found on the FIS Editor. Refer to `fuzzy` for more information:

- Use the following **Edit** menu item:

  Undo to undo the most recent change.

  FIS properties to invoke the FIS Editor.

  Membership functions to invoke the Membership Function Editor.

  Rules to invoke the Rule Editor.
- Use the following **View** menu items:

  Rules to invoke the Rule Viewer.

  Surface to invoke the Surface Viewer.

See Also

`fuzzy`, `mfedit`, `ruleedit`, `ruleview`, `surfview`

Purpose

Convert Fuzzy Logic Toolbox Version 1.0 Fuzzy Inference System matrix to current-version Fuzzy Inference System structure

Syntax

```
fis_new=convertfis(fis_old)
```

Description

`convertfis` takes a Version 1.0 FIS matrix and converts it to a FIS structure compatible with the current version.

Purpose

Defuzzify membership function

Syntax

```
out = defuzz(x,mf,type)
```

Description

`defuzz(x,mf,type)` returns a defuzzified value `out`, of a membership function `mf` positioned at associated variable value `x`, using one of several defuzzification strategies, according to the argument, `type`. The variable `type` can be one of the following:

- `centroid`: centroid of area
- `bisector`: bisector of area
- `mom`: mean value of maximum
- `som`: smallest (absolute) value of maximum
- `lom`: largest (absolute) value of maximum

If `type` is not one of those listed, Fuzzy Logic Toolbox assumes it to be a user-defined function. `x` and `mf` are passed to this function to generate the defuzzified output.

Examples

```
x = -10:0.1:10;
mf = trapmf(x,[-10 -8 -4 7]);
xx = defuzz(x,mf,'centroid');
```

Purpose

Built-in membership function composed of difference between two sigmoidal membership functions

Syntax

```
y = dsigmf(x,[a1 c1 a2 c2])
```

Description

The sigmoidal membership function used depends on the two parameters a and c and is given by

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

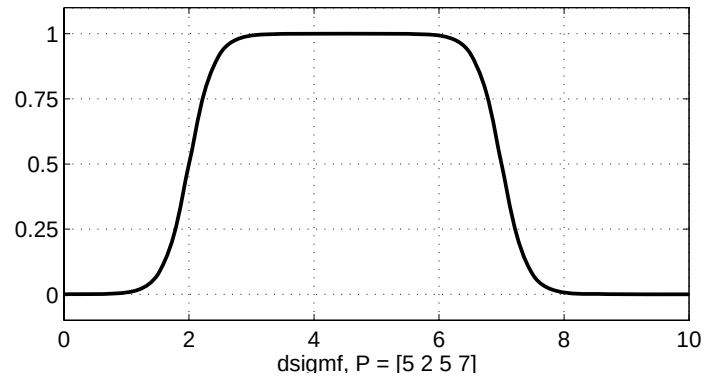
The membership function `dsigmf` depends on four parameters, `a1`, `c1`, `a2`, and `c2`, and is the difference between two of these sigmoidal functions.

$f_1(x;\ a_1,\ c_1)$ - $f_2(x;\ a_2,\ c_2)$

The parameters are listed in the order: $[a_1\ c_1\ a_2\ c_2]$.

Examples

```
x=0:0.1:10;
y=dsigmf(x,[5 2 5 7]);
plot(x,y)
xlabel('dsigmf, P=[5 2 5 7]')
```



See Also

gaussmf, gauss2mf, gbellmf, evalmf, mf2mf, pimf, psigmf, sigmf, smf, trapmf, trimf, zmf

Purpose

Perform fuzzy inference calculations

Syntax

```
output= evalfis(input,fismat)
output= evalfis(input,fismat, numPts)
[output, IRR, ORR, ARR]= evalfis(input,fismat)
[output, IRR, ORR, ARR]= evalfis(input,fismat,
numPts)
```

Description

`evalfis` has the following arguments:

- `input`: a number or a matrix specifying input values. If `input` is an M-by-N matrix, where N is number of input variables, then `evalfis` takes each row of `input` as an input vector and returns the M-by-L matrix to the variable, `output`, where each row is an output vector and L is the number of output variables.
- `fismat`: a FIS structure to be evaluated.
- `numPts`: an optional argument that represents the number of sample points on which to evaluate the membership functions over the input or output range. If this argument is not used, the default value of 101 points is used.

The range labels for `evalfis` are as follows:

- `output`: the output matrix of size M-by-L, where M represents the number of input values specified previously, and L is the number of output variables for the FIS.

The optional range variables for `evalfis` are only calculated when the `input` argument is a row vector, (only one set of inputs is applied). These optional range variables are

- `IRR`: the result of evaluating the input values through the membership functions. This matrix is of the size *numRules*-by-*N*, where *numRules* is the number of rules, and *N* is the number of input variables.

- `ORR`: the result of evaluating the output values through the membership functions. This matrix is of the size `numPts`-by-*numRules***L*, where *numRules* is the number of rules, and *L* is the number of outputs. The first *numRules* columns of this matrix correspond to the first output, the next *numRules* columns of this matrix correspond to the second output, and so forth.
- `ARR`: the `numPts`-by-*L* matrix of the aggregate values sampled at `numPts` along the output range for each output.

When it is invoked with only one range variable, this function computes the output vector, `output`, of the fuzzy inference system specified by the structure, `fismat`, for the input value specified by the number or matrix, `input`.

Examples

```
fismat = readfis('tipper');
out = evalfis([2 1; 4 9],fismat)
```

This syntax generates the response

```
out =
 7.0169
 19.6810
```

See Also

`ruleview`, `gensurf`

Purpose

Generic membership function evaluation

Syntax

```
y = evalmf(x,mfParams,mfType)
```

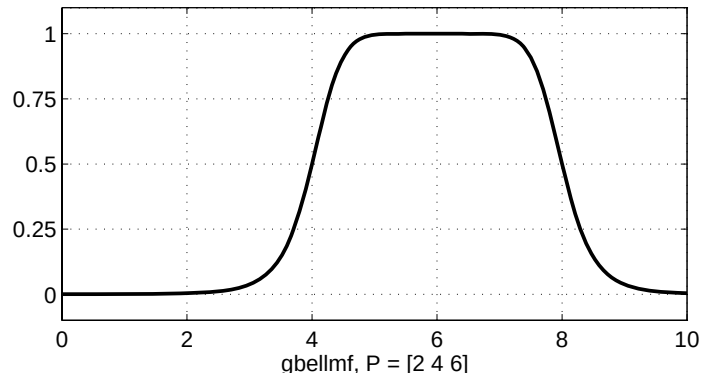
Description

`evalmf` evaluates any membership function, where x is the variable range for the membership function evaluation, `mfType` is a membership function from the toolbox, and `mfParams` are appropriate parameters for that function.

If you want to create your own custom membership function, `evalmf` still works, because it evaluates any membership function whose name it does not recognize.

Examples

```
x=0:0.1:10;
mfparams = [2 4 6];
mftype = 'gbellmf';
y=evalmf(x,mfparams,mftype);
plot(x,y)
xlabel('gbellmf, P=[2 4 6]')
```



See Also

`dsigmf`, `gaussmf`, `gauss2mf`, `gbellmf`, `evalmf`, `mf2mf`, `pimf`, `psigmf`, `sigmf`, `smf`, `trapmf`, `trimf`, `zmf`

Purpose

Fuzzy c-means clustering

Syntax

```
[center,U,obj_fcn] = fcm(data,cluster_n)
```

Description

`[center, U, obj_fcn] = fcm(data, cluster_n)` applies the fuzzy c-means clustering method to a given data set.

The input arguments of this function are

- `data`: data set to be clustered; each row is a sample data point
- `cluster_n`: number of clusters (greater than one)

The output arguments of this function are

- `center`: matrix of final cluster centers where each row provides the center coordinates
- `U`: final fuzzy partition matrix (or membership function matrix)
- `obj_fcn`: values of the objective function during iterations

`fcm(data,cluster_n,options)` uses an additional argument variable, `options`, to control clustering parameters, introduce a stopping criteria, set the iteration information display, or both.

- `options(1)`: exponent for the partition matrix `U` (default: 2.0)
- `options(2)`: maximum number of iterations (default: 100)
- `options(3)`: minimum amount of improvement (default: 1e-5)
- `options(4)`: info display during iteration (default: 1)

If any entry of `options` is `NaN`, the default value for that option is used instead. The clustering process stops when the maximum number of iterations is reached or when the objective function improvement between two consecutive iterations is less than the minimum amount of improvement specified.

Examples

```
data = rand(100, 2);
[center,U,obj_fcn] = fcm(data, 2);
plot(data(:,1), data(:,2),'o');
maxU = max(U);
index1 = find(U(1,:) == maxU);
index2 = find(U(2, :) == maxU);
line(data(index1,1),data(index1, 2),'linestyle','none',...
     'marker','*','color','g');
line(data(index2,1),data(index2, 2),'linestyle','none',...
     'marker', '*','color','r');
```

Purpose

Interactive clustering GUI for fuzzy c-means and subclustering

Syntax

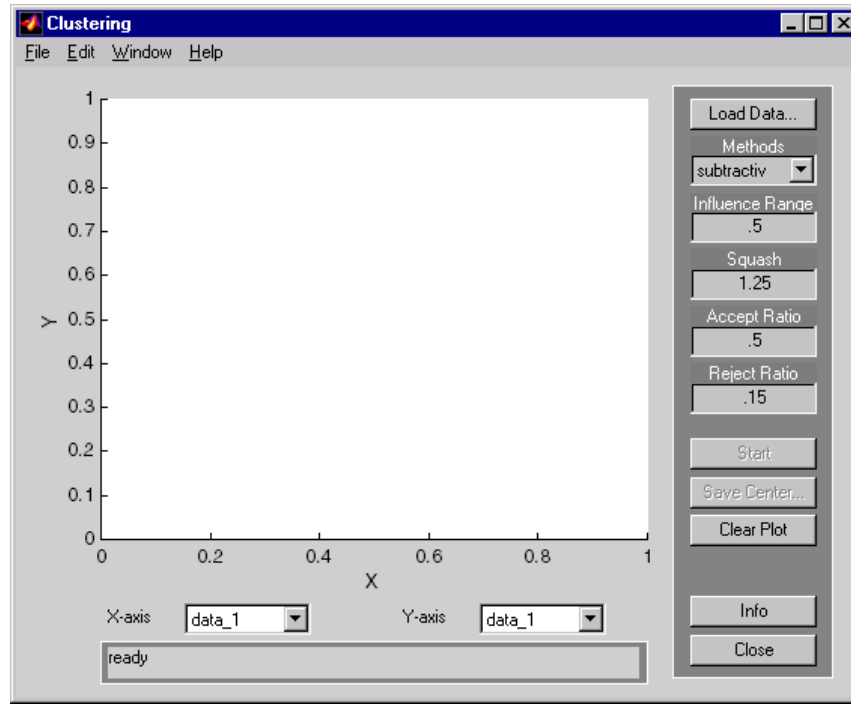
```
findcluster findcluster('file.dat')
```

Description

`findcluster` opens a GUI to implement either the fuzzy c-means (**fcm**), the fuzzy subtractive clustering (**subtractiv**) using the pull-down tab under **Method** on the GUI, or both. Data is entered using the **Load Data** button. The options for each of these methods are set to default values. These default values can be changed. See `fcm` reference page for a description of the options for fuzzy c-means. The `subclust` reference page provides a description of the options for fuzzy subclustering.

This tool works on multidimensional data sets, but only displays two of those dimensions. Use the pull-down tabs under **X-axis** and **Y-axis** to select which data dimension you want to view. For example, if you have data that is five-dimensional, this tool labels the data as data_1, data_2, data_3, data_4, data_5, in the order in which the data appears in the data set. **Start** to perform the clustering, and **Save Center** to save the cluster center.

When operating on a data set called `file.dat`, `findcluster` (`file.dat`) loads the data set automatically, plotting up to the first two dimensions of the data only. You can still choose which two dimensions of the data you want to cluster after the GUI appears.



Examples

```
findcluster('clusterdemo.dat')
```

See Also

`fcm`, `subclust`

Purpose

Perform fuzzy arithmetic

Syntax

```
C = fuzarith(X, A, B, operator)
```

Description

Using interval arithmetic, `C = fuzarith(X, A, B, operator)` returns a fuzzy set `C` as the result of applying the function represented by the string, `operator`, which performs a binary operation on the sampled convex fuzzy sets `A` and `B`. The elements of `A` and `B` are derived from convex functions of the sampled universe, `X`:

- `A`, `B`, and `X` are vectors of the same dimension.
- `operator` is one of the following strings: `'sum'`, `'sub'`, `'prod'`, and `'div'`.
- The returned fuzzy set `C` is a column vector with the same length as `X`.

Note Fuzzy addition might generate the message `"divide by zero"` but this does not affect the accuracy of this function.

Examples

```
point_n = 101;   % this determines MF's resolution
  min_x = -20; max_x = 20; % universe is [min_x, max_x]
  x = linspace(min_x, max_x, point_n)';
  A = trapmf(x, [-10 -2 1 3]); % trapezoidal fuzzy set A
  B = gaussmf(x, [2 5]);  % Gaussian fuzzy set B
  C1 = fuzarith(x, A, B, 'sum');
  subplot(2,1,1);
  plot(x, A, 'b--', x, B, 'm:', x, C1, 'c');
  title('fuzzy addition A+B');
  C2 = fuzarith(x, A, B, 'sub');
  subplot(2,1,2);
  plot(x, A, 'b--', x, B, 'm:', x, C2, 'c');
  title('fuzzy subtraction A-B');
  C3 = fuzarith(x, A, B, 'prod');
```

Purpose

Simulink fuzzy logic library

Syntax

```
fuzblock
```

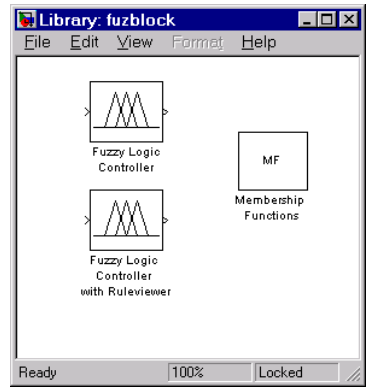
Description

This command opens a Simulink library that contains two Simulink blocks you can use:

- The Fuzzy Logic Controller
- The Fuzzy Logic Controller With Rule Viewer (See also `ruleview`.) This block forces the Rule Viewer to pop open during a Simulink simulation.

To open the dialog box for the Fuzzy Logic Controller (with or without the Rule Viewer), double-click the block. Enter the name of the structure variable describing your FIS. This variable must be located in the MATLAB workspace. For more discussion about these blocks, see "Building Your Own Fuzzy Simulink Models" on page 2-78.

If the fuzzy inference system has multiple inputs, multiplex these inputs together before feeding them into either the Fuzzy Logic Controller or the Fuzzy Logic Controller With Rule Viewer block. Similarly, if the system has multiple outputs, these signals are passed out of the block on one multiplexed line.



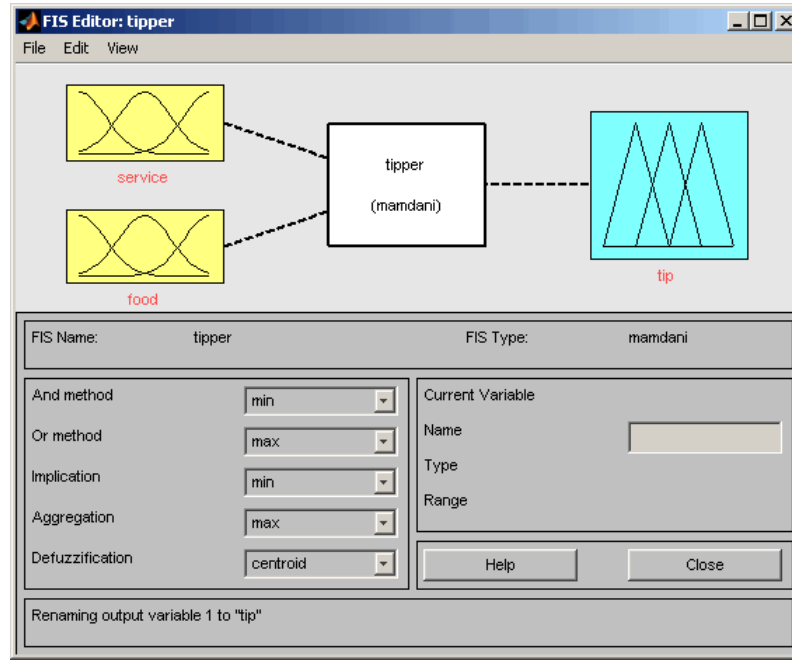
See Also `sffis`, `ruleview`

Purpose

Open basic Fuzzy Inference System editor

Syntax

```
fuzzy
fuzzy(fismat)
```



This GUI tool allows you to edit the highest level features of the fuzzy inference system, such as the number of input and output variables, the defuzzification method used, and so on. Refer to "The FIS Editor" on page 2-34 for more information about how to use the GUIs associated with `fuzzy`.

The FIS Editor is the high-level display for any fuzzy logic inference system. It allows you to call the various other editors to operate on the FIS. This interface allows convenient access to all other editors with an emphasis on maximum flexibility for interaction with the fuzzy system.

The Diagram

The diagram displayed at the top of the window shows the inputs, outputs, and a central fuzzy rule processor. Click one of the variable boxes to make the selected box the current variable. You should see the box highlighted in red. Double-click one of the variables to bring up the Membership Function Editor. Double-click the fuzzy rule processor to bring up the Rule Editor. If a variable exists but is not mentioned in the rule base, it is connected to the rule processor block with a dashed rather than a solid line.

Menu Items

The FIS Editor displays a menu bar that allows you to open related GUI tools, open and save systems, and so on.

- Under **File** select

  New FIS > Mamdani to open a new Mamdani-style system with no variables and no rules called `Untitled`.

  New FIS > Sugeno to open a new Sugeno-style system with no variables and no rules called `Untitled`.

  Import > From workspace to load a system from a specified FIS structure variable in the workspace.

  Import > From file to load a system from a specified `.fis` file.

  Export > To workspace. to save the system to a FIS structure variable in the workspace.

  Export > To file to save the current system to a `.fis` file.

  Print to print what is displayed in the GUI.

  Close to close the GUI.

- Under **Edit** select

  Undo to undo the most recent change.

  Add variable > Input to add another input to the current system.

  Add variable > Output to add another output to the current system.

  Remove Selected Variable to delete a selected variable.

Membership functions to invoke the Membership Function Editor.

Rules to invoke the Rule Editor.

- Under **View** select

  Rules to invoke the Rule Viewer.

  Surface to invoke the Surface Viewer.

Inference Method Pop-up Menus

Five pop-up menus are provided to change the functionality of the five basic steps in the fuzzy implication process:

- **And method**: Choose `min`, `prod`, or `Custom`, for a custom operation.
- **Or method**: Choose `max`, `probor` (probabilistic or), or `Custom`, for a custom operation.
- **Implication**: Choose `min`, `prod`, or `Custom`, for a custom operation. This selection is not available for Sugeno-style fuzzy inference.
- **Aggregation**: Choose `max`, `sum`, `probor`, or `Custom`, for a custom operation. This selection is not available for Sugeno-style fuzzy inference.
- **Defuzzification**: For Mamdani-style inference, choose `centroid`, `bisector`, `mom` (middle of maximum), `som` (smallest of maximum), `lom` (largest of maximum), or `Custom`, for a custom operation. For Sugeno-style inference, choose between `wtaver` (weighted average) or `wtsum` (weighted sum).

See Also

`mfedit`, `ruleedit`, `ruleview`, `surfview`, `anfisedit`

Purpose

Gaussian combination membership function

Syntax

```
y = gauss2mf(x,[sig1 c1 sig2 c2])
```

Description

The Gaussian function depends on two parameters *sig* and *c* as given by

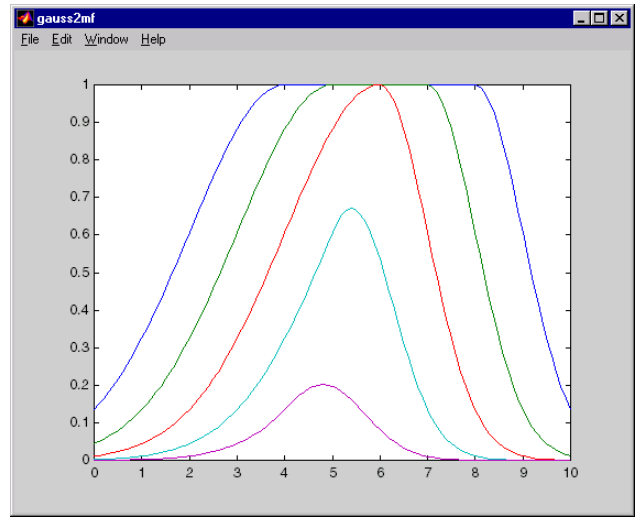
$$f(x;\sigma,c) = e^{\frac{-(x-c)^2}{2\sigma^2}}$$

The function `gauss2mf` is a combination of two of these two parameters. The first function, specified by *sig1* and *c1*, determines the shape of the leftmost curve. The second function specified by *sig2* and *c2* determines the shape of the right-most curve. Whenever *c1* < *c2*, the `gauss2mf` function reaches a maximum value of 1. Otherwise, the maximum value is less than one. The parameters are listed in the order:

[*sig1*, *c1*, *sig2*, *c2*] .

Examples

```
x = (0:0.1:10)';
        y1 = gauss2mf(x, [2 4 1 8]);
        y2 = gauss2mf(x, [2 5 1 7]);
        y3 = gauss2mf(x, [2 6 1 6]);
        y4 = gauss2mf(x, [2 7 1 5]);
        y5 = gauss2mf(x, [2 8 1 4]);
        plot(x, [y1 y2 y3 y4 y5]);
        set(gcf, 'name', 'gauss2mf', 'numbertitle', 'off');
```



See Also

`dsigmf`, `gauss2mf`, `gbellmf`, `evalmf`, `mf2mf`, `pimf`, `psigmf`, `sigmf`, `smf`, `trapmf`, `trimf`, `zmf`

Purpose Gaussian curve built-in membership function

Syntax

```
y = gaussmf(x,[sig c])
```

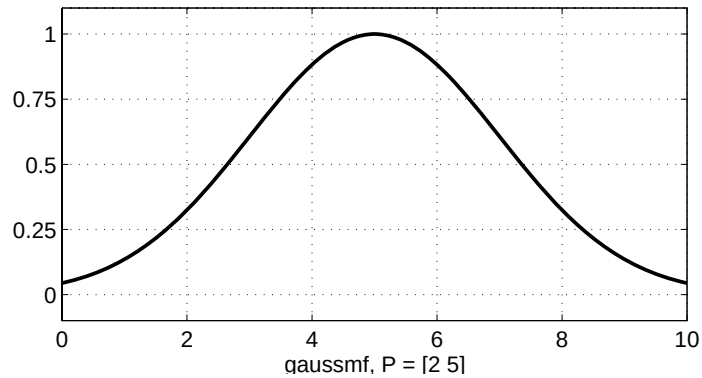
Description The symmetric Gaussian function depends on two parameters σ and c as given by

$$f(x;\sigma,c) = e^{\frac{-(x-c)^2}{2\sigma^2}}$$

The parameters for gaussmf represent the parameters σ and c listed in order in the vector [`sig c`].

Examples

```
x=0:0.1:10;
y=gaussmf(x,[2 5]);
plot(x,y)
xlabel('gaussmf, P=[2 5]')
```



See Also `dsigmf`, `gaussmf`, `gbellmf`, `evalmf`, `mf2mf`, `pimf`, `psigmf`, `sigmf`, `smf`, `trapmf`, `trimf`, `zmf`

Purpose

Generalized bell-shaped built-in membership function

Syntax

```
y = gbellmf(x,params)
```

Description

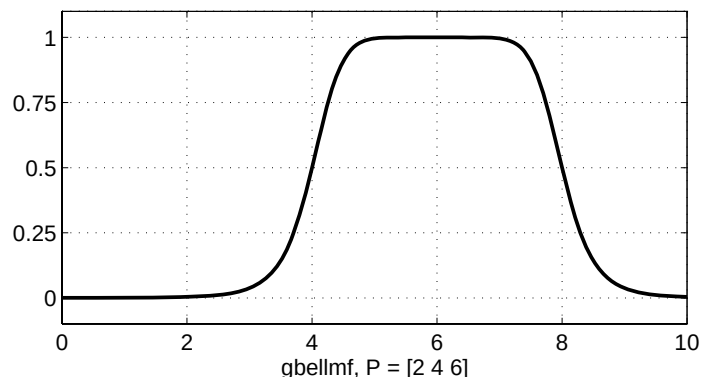
The generalized bell function depends on three parameters a, b, and c as given by

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

where the parameter b is usually positive. The parameter c locates the center of the curve. Enter the parameter vector `params`, the second argument for `gbellmf`, as the vector whose entries are a, b, and c, respectively.

Examples

```
x=0:0.1:10;
y=gbellmf(x,[2 4 6]);
plot(x,y)
xlabel('gbellmf, P=[2 4 6]')
```



See Also

`dsigmf`, `gaussmf`, `gauss2mf`, `evalmf`, `mf2mf`, `pimf`, `psigmf`, `sigmf`, `smf`, `trapmf`, `trimf`, `zmf`

Purpose

Generate Fuzzy Inference System structure from data without data clustering

Syntax

```
fismat = genfis1(data)
fismat = genfis1(data,numMFs,inmftype, outmftype)
```

Description

`genfis1` generates a Sugeno-type FIS structure used as initial conditions (initialization of the membership function parameters) for `anfis` training. `genfis1`(`data`, `numMFs`, `inmftype`, `outmftype`) generates a FIS structure from a training data set, `data`, using a grid partition on the data (no clustering).

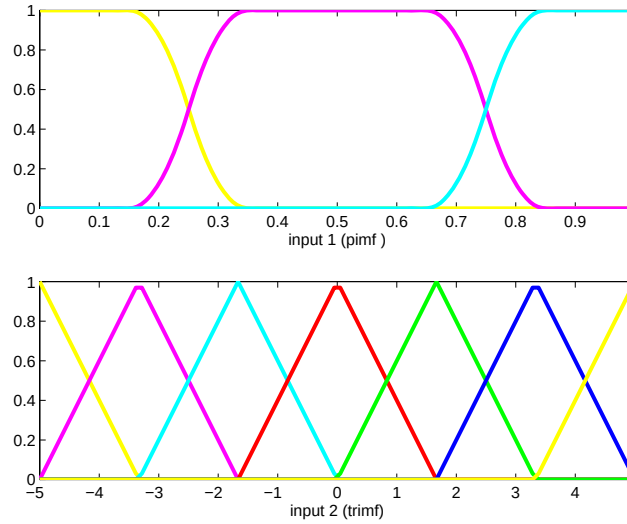
The arguments for `genfis1` are as follows:

- `data` is the training data matrix, which must be entered with all but the last columns representing input data, and the last column representing the single output.
- `numMFs` is a vector whose coordinates specify the number of membership functions associated with each input. If you want the same number of membership functions to be associated with each input, then make `numMFs` a single number.
- `inmftype` is a string array in which each row specifies the membership function type associated with each input. Again, this can be a one-dimensional single string if the type of membership functions associated with each input is the same.
- `outmftype` is a string that specifies the membership function type associated with the output. There can only be one output, because this is a Sugeno-type system. The output membership function type must be either `linear` or `constant`.

The number of membership functions associated with the output is the same as the number of rules generated by `genfis1`. The default number of membership functions, `numMFs`, is 2; the default input membership function type is `'gbellmf'`; and the default output membership function type is `'linear'`. These are used whenever `genfis1` is invoked without the last three arguments.

Examples

```
data = [rand(10,1) 10*rand(10,1)-5 rand(10,1)];
numMFs = [3 7];
mfType = str2mat('pimf','trimf');
fismat = genfis1(data,numMFs,mfType);
[x,mf] = plotmf(fismat,'input',1);
subplot(2,1,1), plot(x,mf);
xlabel('input 1 (pimf)');
[x,mf] = plotmf(fismat,'input',2);
subplot(2,1,2), plot(x,mf);
xlabel('input 2 (trimf)');
```



See Also

anfis

Purpose

Generate Fuzzy Inference System structure from data using subtractive clustering

Syntax

```
fismat = genfis2(Xin,Xout,radii)
fismat = genfis2(Xin,Xout,radii,xBounds)
fismat = genfis2(Xin,Xout,radii,xBounds,options)
```

Description

Given separate sets of input and output data, `genfis2` generates a FIS using fuzzy subtractive clustering. When there is only one output, `genfis2` may be used to generate an initial FIS for `anfis` training by first implementing subtractive clustering on the data. `genfis2` accomplishes this by extracting a set of rules that models the data behavior.

The rule extraction method first uses the `subclust` function to determine the number of rules and antecedent membership functions and then uses linear least squares estimation to determine each rule's consequent equations. This function returns a FIS structure that contains a set of fuzzy rules to cover the feature space.

The arguments for `genfis2` are as follows:

- `Xin` is a matrix in which each row contains the input values of a data point.
- `Xout` is a matrix in which each row contains the output values of a data point.
- `radii` is a vector that specifies a cluster center's range of influence in each of the data dimensions, assuming the data falls within a unit hyperbox.

  For example, if the data dimension is 3 (e.g., `Xin` has two columns and `Xout` has one column), `radii` = [0.5 0.4 0.3] specifies that the ranges of influence in the first, second, and third data dimensions (i.e., the first column of `Xin`, the second column of `Xin`, and the column of `Xout`) are 0.5, 0.4, and 0.3 times the width of the data space, respectively. If `radii` is a scalar value, then this scalar

value is applied to all data dimensions, i.e., each cluster center has a spherical neighborhood of influence with the given radius.

- `xBounds` is a 2-by-*N* optional matrix that specifies how to map the data in `Xin` and `Xout` into a unit hyperbox, where *N* is the data (row) dimension. The first row of `xBounds` contains the minimum axis range values and the second row contains the maximum axis range values for scaling the data in each dimension.

  For example, `xBounds` = [-10 0 -1; 10 50 1] specifies that data values in the first data dimension are to be scaled from the range [-10 +10] into values in the range [0 1]; data values in the second data dimension are to be scaled from the range [0 50]; and data values in the third data dimension are to be scaled from the range [-1 +1]. If `xBounds` is an empty matrix or not provided, then `xBounds` defaults to the minimum and maximum data values found in each data dimension.

- `options` is an optional vector for specifying algorithm parameters to override the default values. These parameters are explained in the help text for `subclust`. Default values are in place when this argument is not specified.

Examples

```
fismat = genfis2(Xin,Xout,0.5)
```

This example shows the minimum number of arguments needed to use this function. In this case, a range of influence of 0.5 is specified for all data dimensions.

```
fismat = genfis2(Xin,Xout,[0.5 0.25 0.3])
```

This equation assumes the combined data dimension is 3. Suppose `Xin` has two columns and `Xout` has one column, then 0.5 and 0.25 are the ranges of influence for each of the `Xin` data dimensions, and 0.3 is the range of influence for the `Xout` data dimension.

```
fismat = genfis2(Xin,Xout,0.5,[-10 -5 0; 10 5 20])
```

This equations specifies how to normalize the data in `Xin` and `Xout` into values in the range [0 1] for processing. Suppose `Xin` has two columns and `Xout` has one column, then the data in the first column of `Xin` are scaled from [-10 +10], the data in the second column of `Xin` are scaled from [-5 +5], and the data in `Xout` are scaled from [0 20].

See Also

`subclust`

Purpose

Generate Fuzzy Inference System structure from data using FCM clustering

Syntax

```
fismat = genfis3(Xin,Xout)
fismat = genfis3(Xin,Xout,type)
fismat = genfis3(Xin,Xout,type,cluster_n)
fismat = genfis3(Xin,Xout,type,cluster_n,fcmoptions)
```

Description

Given separate sets of input and output data, `genfis3` generates a FIS using fuzzy c-means (FCM) clustering by extracting a set of rules that models the data behavior. The rule extraction method first uses the `fcm` function to determine the number of rules and membership functions for the antecedents and consequents.

`fismat = genfis3(Xin,Xout)` generates a Sugeno-type FIS structure (`fismat`) given input data `Xin` and output data `Xout`. The matrices `Xin` and `Xout` have one column per FIS input and output, respectively.

`fismat = genfis3(Xin,Xout,type)` generates a FIS structure of the specified `type`, where `type` is either `'mamdani'` or `'sugeno'`.

`fismat = genfis3(Xin,Xout,type,cluster_n)` generates a FIS structure of the specified `type` and allows you to specify the number of clusters (`cluster_n`) to be generated by FCM. The number of clusters determines the number of rules and membership functions in the generated FIS. `cluster_n` must be an integer or `'auto'`, which uses the `subclust` algorithm with a radii of 0.5 to find the number of clusters.

`fismat = genfis3(Xin,Xout,type,cluster_n,fcmoptions)` generates a FIS structure of the specified `type` and number of clusters and uses the specified `fcmoptions` for the FCM algorithm. If you omit `fcmoptions`, the default fcm values are used. See `fcm` for information about these parameters.

Examples

Generate a FIS using default values.

```
Xin1 =  7*rand(50,1);
Xin2 = 20*rand(50,1)-10;
Xin  = [Xin1 Xin2];
```

```
Xout =  5*rand(50,1);
fismat = genfis3(Xin,Xout);
```

Generate a Mamdani FIS with 3 clusters.

```
fismat = genfis3(Xin,Xout,'mamdani',3)
```

See Also

anfis, fcm, genfis1, genfis2, subclust

Purpose

Generate Fuzzy Inference System output surface

Syntax

```
gensurf(fis)
gensurf(fis,inputs,output)
gensurf(fis,inputs,output,grids)
gensurf(fis,inputs,output,grids,refinput)
[x,y,z]=gensurf(...)
```

Description

`gensurf(fis)` generates a plot of the output surface of a given fuzzy inference system (`fis`) using the first two inputs and the first output.

`gensurf(fis,inputs,output)` generates a plot using the inputs (one or two) and output (only one is allowed) given, respectively, by the vector, `inputs`, and the scalar, `output`.

`gensurf(fis,inputs,output,grids)` allows you to specify the number of grids in the X (first, horizontal) and Y (second, vertical) directions. If `grids` is a two element vector, the grids in the X and Y directions can be set independently.

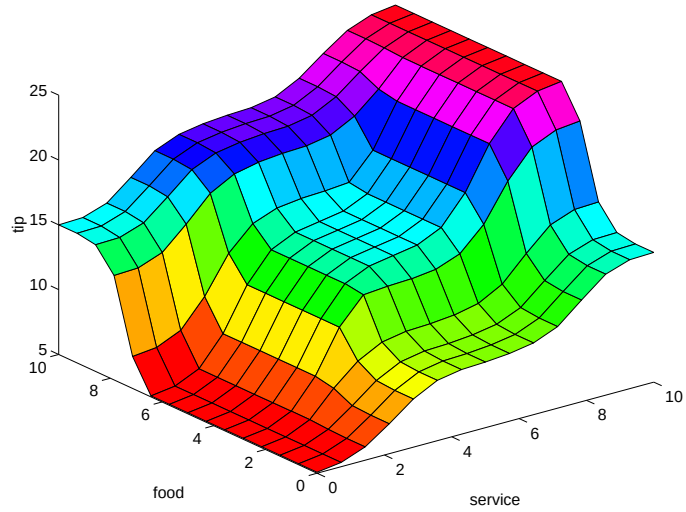
`gensurf(fis,inputs,output,grids,refinput)` can be used if there are more than two outputs. The length of the vector `refinput` is the same as the number of inputs:

- Enter `NaN`s for the entries of `refinput` corresponding to the inputs whose surface is being displayed.
- Enter real double scalars to fix the values of other inputs.

`[x,y,z]=gensurf(...)` returns the variables that define the output surface and suppresses automatic plotting.

Example 1

```
a = readfis('tipper');
gensurf(a)
```



Example 2

```
a = gensurf(Temp,[1 2],1,[20 20],[nan nan 0.2]);
```

This equation generates the surface of a three-input FIS named `Temp` from its first two inputs to its first output, while fixing a reference value for the third input at .2.

See Also

`evalfis`, `surfview`

Purpose

Fuzzy system properties

Syntax

```
getfis(a)
getfis(a,'fisprop')
getfis(a, 'vartype', varindex)
getfis(a,'vartype',varindex,'varprop')
getfis(a,'vartype',varindex,'mf',mfindex)
getfis(a,'vartype',varindex,'mf',mfindex,'mfprop')
```

Description

This function provides the fundamental access for the FIS structure. With this one function you can learn about every part of the fuzzy inference system.

The arguments for `getfis` are as follows:

- `a`: the name of a workspace variable FIS structure.
- `'fisprop'`: a string indicating the field you want to access. Possible fields include:

  `'name'`

  `'type'`

  `'numinputs'`

  `'numoutputs'`

  `'numinputmfs'`

  `'numoutputmfs'`

  `'numrules'`

  `'andmethod'`

  `'ormethod'`

  `'impmethod'`

  `'aggmethod'`

  `'defuzzmethod'`

```
'inlabels'
'outlabels'
'inrange'
'outrange'
'inmfs'
'outmfs'
'inmflabels'
'outmflabels'
'inmftypes'
'outmftypes'
'inmfparams'
'outmfparams'
'rulelist'
```

`'Name'` or `'NumInputs'`.

- `'vartype'`: a string indicating the type of variable you want (either `input` or `output`).
- `varindex`: an integer indicating the index of the variable you want (1, for input 1, for example).
- `'varprop'`: a string indicating the variable property you want. Possible strings are `name`, `range`, `nummfs`, and `mflabels`. See the examples for sample string names.
- `'mf'`: a required string that indicates you are searching for membership function information.
- `mfindex`: the index of the membership function for which you are seeking information.
- `'mfprop'`: a string indicating values for membership function properties. Possible strings are`name`, `type`, and `params`.

You can also access fuzzy system properties directly using MATLAB syntax for structures (including dot notation). See the examples that follow.

Examples

One input argument (`output` is the empty set)

```
a = readfis('tipper');
getfis(a)
         Name = tipper
         Type = mamdani
         NumInputs = 2
         InLabels =
                 service
                 food
         NumOutputs = 1
         OutLabels =
                 tip
         NumRules = 3
         AndMethod = min
         OrMethod = max
         ImpMethod = min
         AggMethod = max
         DefuzzMethod = centroid
```

Two input arguments

```
getfis(a,'type')
ans =
mamdani
```

or

```
a.type
ans =
mamdani
```

Three input arguments (`output` is the empty set)

```
getfis(a,'input',1)
        Name = service
        NumMFs = 3
        MFLabels =
                poor
                good
                excellent
        Range = [0 10]
```

or

```
a.input(1)
ans =
     name: 'service'
    range: [0 10]
       mf: [1x3 struct]
```

Four input arguments

```
getfis(a,'input',1,'name')
ans =
service
```

or

```
a.input(1).name
ans =
service
```

Five input arguments

```
getfis(a,'input',1,'mf',2)
        Name = good
        Type = gaussmf
        Params =
    1.5000    5.0000
```

or

```
a.input(1).mf(2)
ans =
      name: 'good'
      type: 'gaussmf'
    params: [1.5000 5]
```

Six input arguments

```
getfis(a,'input',1,'mf',2,'name')
ans =
good
```

or

```
a.input(1).mf(2).name
ans =
good
```

See Also

`setfis`, `showfis`

Purpose

Transform Mamdani Fuzzy Inference System into Sugeno Fuzzy Inference System

Syntax

```
sug_fis=mam2sug(mam_fis)
```

Description

`mam2sug (mam_fis)` transforms a (not necessarily a single output) Mamdani FIS structure `mam_fis` into a Sugeno FIS structure `sug_fis`. The returned Sugeno system has constant output membership functions. These constants are determined by the centroids of the consequent membership functions of the original Mamdani system. The antecedent remains unchanged.

Examples

```
mam_fismat = readfis('mam22.fis');
sug_fismat = mam2sug(mam_fismat);
subplot(2,2,1); gensurf(mam_fismat, [1 2], 1);
title('Mamdani system (Output 1)');
subplot(2,2,2); gensurf(sug_fismat, [1 2], 1);
title('Sugeno system (Output 1)');
subplot(2,2,3); gensurf(mam_fismat, [1 2],2);
title('Mamdani system (Output 2)');
subplot(2,2,4); gensurf(sug_fismat, [1 2],2);
title('Sugeno system (Output 2)');
```

Purpose

Translate parameters between membership functions

Syntax

```
outParams = mf2mf(inParams,inType,outType)
```

Description

This function translates any built-in membership function type into another, in terms of its parameter set. In principle, `mf2mf` mimics the symmetry points for both the new and old membership functions.

Caution

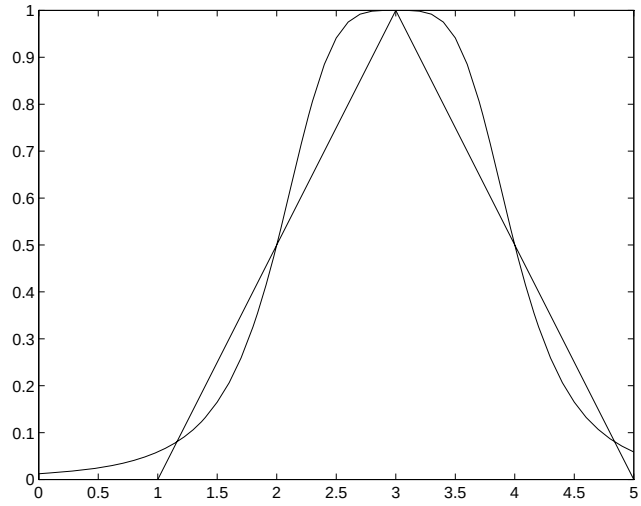
Occasionally this translation results in lost information, so that if the output parameters are translated back into the original membership function type, the transformed membership function does not look the same as it did originally.

The input arguments for `mf2mf` are as follows:

- `inParams`: the parameters of the membership function you are transforming
- `inType`: a string name for the type of membership function you are transforming
- `outType`: a string name for the new membership function you are transforming to

Examples

```
x=0:0.1:5;
mfp1 = [1 2 3];
mfp2 = mf2mf(mfp1,'gbellmf','trimf');
plot(x,gbellmf(x,mfp1),x,trimf(x,mfp2))
```



See Also

`dsigmf`, `gaussmf`, `gauss2mf`, `gbellmf`, `evalmf`, `pimf`, `psigmf`, `sigmf`, `smf`, `trapmf`, `trimf`, `zmf`

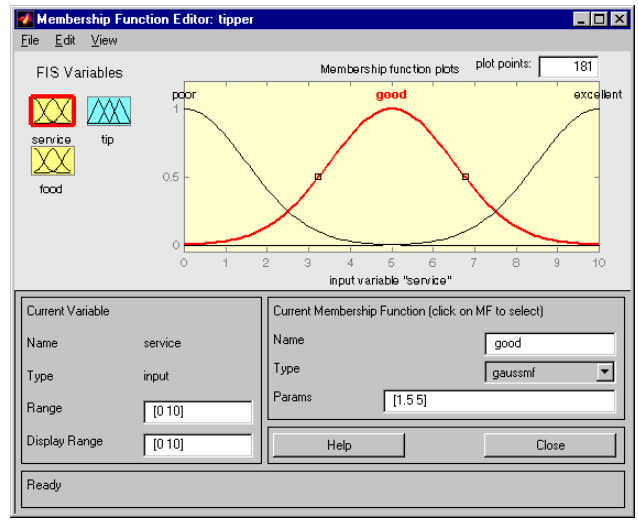
Purpose

Membership function editor

Syntax

```
mfedit('a')
mfedit(a)
mfedit
```

Description



`mfedit('a')` generates a membership function editor that allows you to inspect and modify all the membership functions for your FIS stored in the file, `a.fis`.

`mfedit(a)` operates on a MATLAB workspace variable, for a FIS structure, `a`.

`mfedit` alone opens the membership function editor with no FIS loaded.

For each membership function you can change the name, the type, and the parameters. Eleven built-in membership functions are provided for you to choose from, although of course you can always create your own specialized versions. Refer to "The Membership Function Editor" on page 2-39 for more information about how to use `mfedit`.

Select the icon for the variable on the upper left side of the diagram (under **FIS Variables**) to display its associated membership functions in the plot region. Select membership functions by clicking once on them or their labels.

Menu Items

On the Membership Function Editor, there is a menu bar that allows you to open related GUI tools, open and save systems, and so on. The **File** menu for the Membership Function Editor is the same as the one found on the FIS Editor. Refer to "The Membership Function Editor" on page 2-39 for more information.

- Under **Edit**, select:

  Undo to undo the most recent change.

  Add MFs to add membership functions to the current variable.

  Add Custom MF to add a customized membership function to the current variable.

  Remove Selected MF to delete the current membership function.

  Remove All MFs to delete all membership functions of the current variable.

  FIS properties to invoke the FIS Editor.

  Rules to invoke the Rule Editor.

- Under **View**, select:

  Rules to invoke the Rule Viewer.

  Surface to invoke the Surface Viewer.

Membership Function Pop-up Menu

There are 11 built-in membership functions to choose from, and you also have the option of installing a customized membership function.

See Also

`fuzzy`, `ruleedit`, `ruleview`, `surfview`

Purpose

Create new Fuzzy Inference System

Syntax

```
a=newfis(fisName,fisType,andMethod,orMethod,impMethod,
...
 aggMethod,defuzzMethod)
```

Description

This function creates new FIS structures. `newfis` has up to seven input arguments, and the output argument is a FIS structure. The seven input arguments are as follows:

- `fisName` is the string name of the FIS structure, `fisName.fis` you create.
- `fisType` is the type of FIS.
- `andMethod`, `orMethod`, `impMethod`, `aggMethod`, and `defuzzMethod`, respectively, provide the methods for AND, OR, implication, aggregation, and defuzzification.

Examples

The following example shows what the defaults are for each of the methods.

```
a=newfis('newsys');
getfis(a)
```

returns

```
Name = newsys
Type = mamdani
NumInputs = 0
InLabels =
NumOutputs = 0
OutLabels =
NumRules   0
AndMethod   min
OrMethod   max
ImpMethod   min
AggMethod   max
```

```
            DefuzzMethod   centroid
ans =
     [newsys]
```

See Also

readfis, writefis

Purpose

Parse fuzzy rules

Syntax

```
fis2 = parsrule(fis,txtRuleList)
fis2 = parsrule(fis,txtRuleList,ruleFormat)
fis2 = parsrule(fis,txtRuleList,ruleFormat,lang)
```

Description

This function parses the text that defines the rules (`txtRuleList`) for a MATLAB workspace FIS variable, `fis`, and returns a FIS structure with the appropriate rule list in place. If the original input FIS structure, `fis`, has any rules initially, they are replaced in the new structure, `fis2`.

Three different rule formats (indicated by `ruleFormat`) are supported: `'verbose'`, `'symbolic'`, and `'indexed'`. The default format is `'verbose'`. When the optional language argument, `lang`, is used, the rules are parsed in verbose mode, assuming the key words are in the language, `lang`. This language must be either ’`'english'`, `'francais'`, or `'deutsch'`. The key language words in English are if, then, is, AND, OR, and NOT.

Examples

```
a = readfis('tipper');
ruleTxt = 'if service is poor then tip is generous';
a2 = parsrule(a,ruleTxt,'verbose');
showrule(a2)
ans =
 1. If (service is poor) then (tip is generous) (1)
```

See Also

`addrule`, `ruleedit`, `showrule`

Purpose

Π-shaped built-in membership function

Syntax

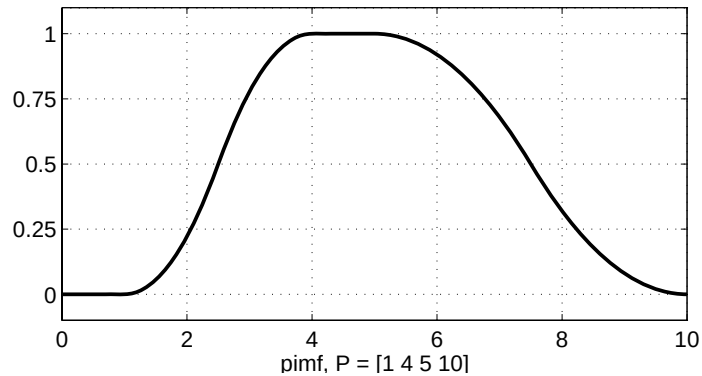
```
y = pimf(x,[a b c d])
```

Description

This spline-based curve is so named because of its Π shape. The membership function is evaluated at the points determined by the vector `x`. The parameters *a* and *d* locate the "feet" of the curve, while *b* and *c* locate its "shoulders."

Examples

```
x=0:0.1:10;
y=pimf(x,[1 4 5 10]);
plot(x,y)
xlabel('pimf, P=[1 4 5 10]')
```



See Also

`dsigmf`, `evalmf`, `gauss2mf`, `gaussmf`, `gbellmf`, `mf2mf`, `psigmf`, `sigmf`, `smf`, `trapmf`, `trimf`, `zmf`

Purpose

Plot Fuzzy Inference System

Syntax

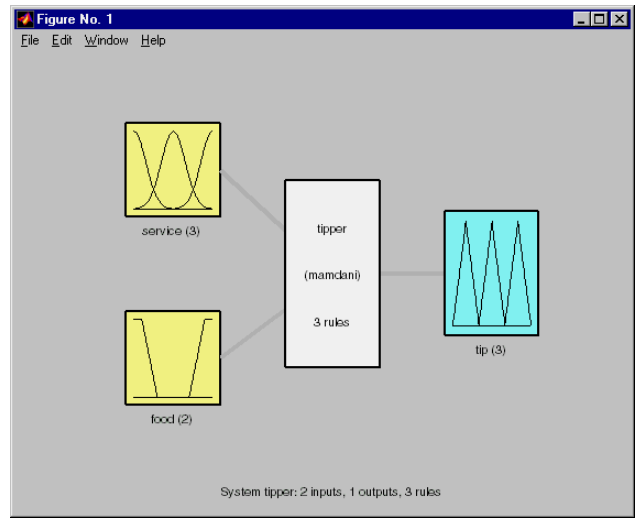
```
plotfis(fismat)
```

Description

This function displays a high level diagram of a FIS, `fismat`. Inputs and their membership functions appear to the left of the FIS structural characteristics, while outputs and their membership functions appear on the right.

Examples

```
a = readfis('tipper');
plotfis(a)
```



See Also

`evalmf`, `plotmf`

Purpose Plot all membership functions for given variable

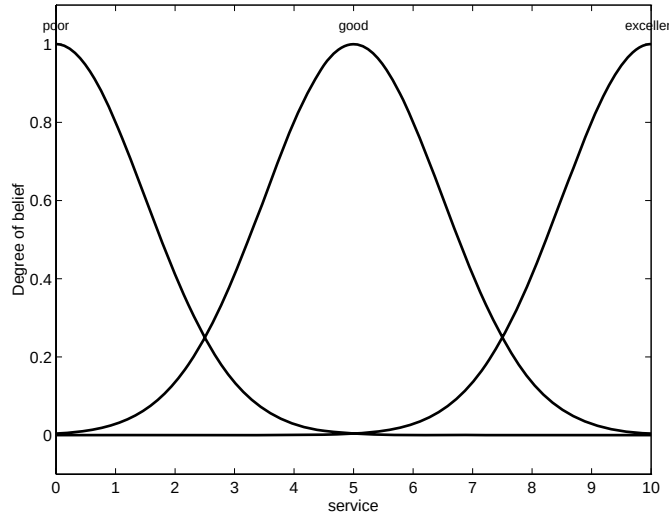
Syntax

```
plotmf(fismat,varType,varIndex)
```

Description This function plots all of the membership functions in the FIS called `fismat` associated with a given variable whose type and index are respectively given by *`varType`* (must be `'input'` or `'output'`), and `varIndex`. This function can also be used with the MATLAB function, `subplot`.

Examples

```
a = readfis('tipper');
plotmf(a,'input',1)
```



See Also `evalmf`, `plotfis`

Purpose

Probabilistic OR

Syntax

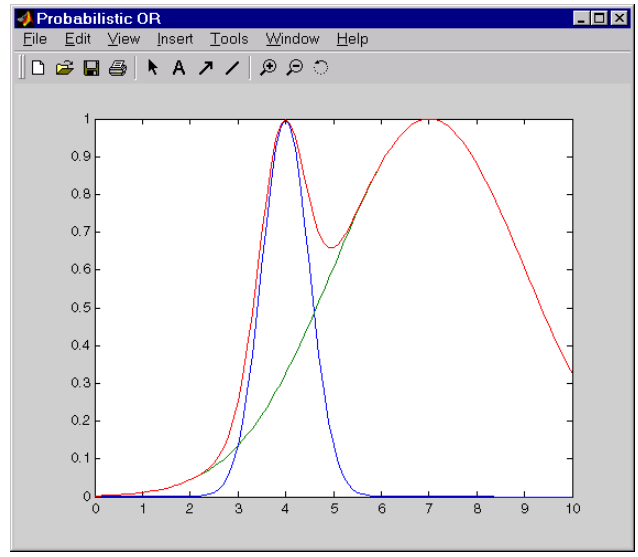
```
y = probor(x)
```

Description

`y = probor(x)` returns the probabilistic OR (also known as the *algebraic sum*) of the columns of `x`. if `x` has two rows such that `x = [a; b]`, then `y = a + b - ab`. If `x` has only one row, then `y = x`.

Examples

```
x = (0:0.1:10);
figure('Name','Probabilistic OR','NumberTitle','off');
y1 = gaussmf(x, [0.5 4]);
y2 = gaussmf(x, [2 7]);
yy = probor([y1; y2]);
plot(x,[y1; y2; yy])
```



Purpose

Built-in membership function composed of product of two sigmoidally shaped membership functions

Syntax

```
y = psigmf(x,[a1 c1 a2 c2])
```

Description

The sigmoid curve plotted for the vector `x` depends on two parameters a and c as given by

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

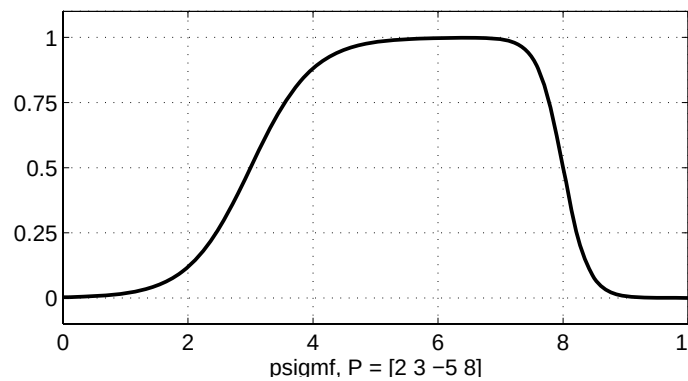
`psigmf` is simply the product of two such curves plotted for the values of the vector `x`

$$f_1(x;\, a_1,\, c_1) \times f_2(x;\, a_2,\, c_2)$$

The parameters are listed in the order $[a_1\ c_1\ a_2\ c_2]$.

Examples

```
x=0:0.1:10;
y=psigmf(x,[2 3 -5 8]);
plot(x,y)
xlabel('psigmf, P=[2 3 -5 8]')
```



See Also

`dsigmf`, `gaussmf`, `gauss2mf`, `gbellmf`, `evalmf`, `mf2mf`, `pimf`, `sigmf`, `smf`, `trapmf`, `trimf`, `zmf`

Purpose

Load Fuzzy Inference System from file

Syntax

```
fismat = readfis('filename')
```

Description

Read a fuzzy inference system from a `.fis` file (named `filename`) and import the resulting file into the workspace.

`fismat = readfis` (no input arguments) opens a `uigetfile` dialog box to assist with the name and directory location of the file.

Examples

```
fismat = readfis('tipper');
getfis(fismat)
```

returns

```
 Name = tipper
 Type = mamdani
 NumInputs = 2
 InLabels =
  service
  food
 NumOutputs = 1
 OutLabels =
  tip
 NumRules = 3
 AndMethod = min
 OrMethod = max
 ImpMethod = min
 AggMethod = max
 DefuzzMethod = centroid
ans =
tipper
```

See Also

`writefis`

Purpose

Remove membership function from Fuzzy Inference System

Syntax

```
fis = rmmf(fis,'varType',varIndex,'mf',mfIndex)
```

Description

`fis = rmmf(fis,varType,varIndex,'mf',mfIndex)` removes the membership function, `mfIndex`, of variable type `varType`, of index `varIndex`, from the fuzzy inference system associated with the workspace FIS structure, `fis`:

- The string *`varType`* must be `'input'` or `'output'`.
- `varIndex` is an integer for the index of the variable. This index represents the order in which the variables are listed.
- The argument `'mf'` is a string representing the membership function.
- `mfIndex` is an integer for the index of the membership function. This index represents the order in which the membership functions are listed.

Examples

```
a = newfis('mysys');
a = addvar(a,'input','temperature',[0 100]);
a = addmf(a,'input',1,'cold','trimf',[0 30 60]);
getfis(a,'input',1)
```

returns

```
          Name = temperature
          NumMFs = 1
          MFLabels =
                  cold
          Range = [0 100]
b = rmmf(a,'input',1,'mf',1);
getfis(b,'input',1)
```

returns

```
          Name = temperature
```

```
NumMFs = 0
MFLabels =
Range = [0 100]
```

See Also

addmf, addrule, addvar, plotmf, rmvar

Purpose

Remove variables from Fuzzy Inference System

Syntax

```
fis2 = rmvar(fis,'varType',varIndex)
[fis2,errorStr] = rmvar(fis,'varType',varIndex)
```

Description

`fis2 = rmvar(fis,'varType',varIndex)` removes the variable `'varType'`, of index `varIndex`, from the fuzzy inference system associated with the workspace FIS structure, `fis`:

- The string *varType* must be `'input'` or `'output'`.
- `varIndex` is an integer for the index of the variable. This index represents the order in which the variables are listed.

`[fis2,errorStr] = rmvar(fis,'varType',varIndex)` returns any error messages to the string, `errorStr`.

This command automatically alters the rule list to keep its size consistent with the current number of variables. You must delete from the FIS any rule that contains a variable you want to remove, before removing it. You cannot remove a fuzzy variable currently in use in the rule list.

Examples

```
a = newfis('mysys');
a = addvar(a,'input','temperature',[0 100]);
getfis(a)
```

returns

```
Name = mysys
     Type      = mamdani
     NumInputs = 1
     InLabels  =
           temperature
     NumOutputs = 0
     OutLabels =
     NumRules = 0
     AndMethod = min
```

```
        OrMethod = max
        ImpMethod = min
        AggMethod = max
        DefuzzMethod = centroid
ans =
mysys
b = rmvar(a,'input',1);
getfis(b)
```

returns

```
          Name = mysys
          Type = mamdani
          NumInputs = 0
          InLabels =
          NumOutputs = 0
          OutLabels =
          NumRules = 0
        AndMethod = min
        OrMethod = max
        ImpMethod = min
        AggMethod = max
        DefuzzMethod = centroid
ans =
mysys
```

See Also

addmf, addrule, addvar, rmmf

Purpose

Rule editor and parser

Syntax

```
ruleedit('a')
ruleedit(a)
```

Description

The Rule Editor, when invoked using `ruleedit('a')`, is used to modify the rules of a FIS structure stored in a file, `a.fis`. It can also be used to inspect the rules being used by a fuzzy inference system.

To use this editor to create rules, you must first define all of the input and output variables you want to use with the FIS editor. You can create the rules using the listbox and check box choices for input and output variables, connections, and weights. Refer to "The Rule Editor" on page 2-45 for more information about how to use `ruleedit`.

The syntax `ruleedit(a)` is used when you want to operate on a workspace variable for a FIS structure called `a`.

Menu Items

On the Rule Editor, there is a menu bar that allows you to open related GUI tools, open and save systems, and so on. The **File** menu for the Rule Editor is the same as the one found on the FIS Editor. Refer to `fuzzy` for more information:

- Use the following **Edit** menu item:

  Undo to undo the most recent change.

  FIS properties to invoke the FIS Editor.

  Membership functions to invoke the Membership Function Editor.
- Use the following **View** menu items:

  Rules to invoke the Rule Viewer.

  Surface to invoke the Surface Viewer.
- Use the **Options** menu items:

  Language to select the language: **English**, **Deutsch**, and **Francais**

  Format to select the format:

Verbose uses the words "if," "then," "AND," "OR," and so on to create actual sentences.

Symbolic substitutes some symbols for the words used in the verbose mode. For example, “if *A* AND *B* then *C*” becomes “*A* & *B* => *C*.”

Indexed mirrors how the rule is stored in the FIS structure.

See Also

`addrule`, `fuzzy`, `mfedit`, `parsrule`, `ruleview`, `showrule`, `surfview`

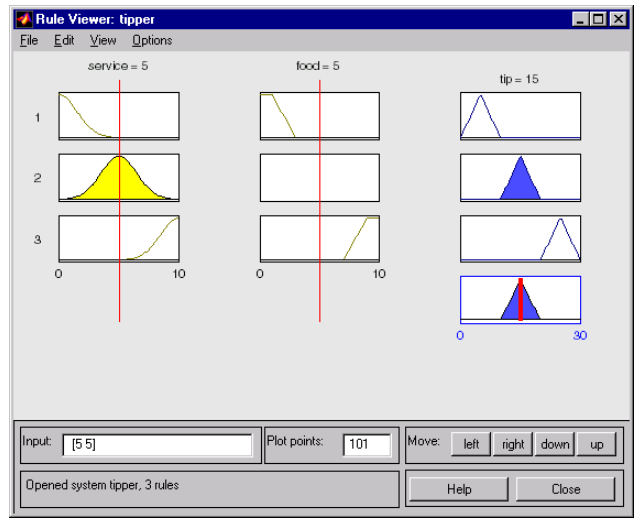
Purpose

Rule viewer and fuzzy inference diagram

Syntax

```
ruleview('a')
```

Description



The Rule Viewer invoked using `ruleview('a')` depicts the fuzzy inference diagram for a FIS stored in a file, `a.fis`. It is used to view the entire implication process from beginning to end. You can move around the line indices that correspond to the inputs and then watch the system readjust and compute the new output. Refer to "The Rule Viewer" on page 2-49 for more information about how to use `ruleview`.

Menu Items

On the Rule Viewer, there is a menu bar that allows you to open related GUI tools, open and save systems, and so on. The **File** menu for the Rule Viewer is the same as the one found on the FIS Editor. Refer to `fuzzy` for more information.

- Use the **Edit** menu items:

  Undo to undo the most recent action

  FIS properties to invoke the FIS Editor

Membership functions to invoke the Membership Function Editor

Rules to invoke the Rule Editor

- Use the **View** menu item:

  Surface to invoke the Surface Viewer

- Use the **Options** menu item:

  Format to set the format in which the rule appears: **Verbose, Symbolic,** or **Indexed**.

If you click on the rule numbers on the left side of the fuzzy inference diagram, the rule associated with that number appears in the Status Bar at the bottom of the Rule Viewer.

See Also

`fuzzy`, `mfedit`, `ruleedit`, `surfview`

Purpose

Set fuzzy system properties

Syntax

```
a = setfis(a,'fispropname','newfisprop')
a = setfis(a,'vartype',varindex,'varpropname','newvarprop')
a = setfis(a,'vartype',varindex,'mf',mfindex,
...
 'mfpropname','newmfprop');
```

Description

The command `setfis` can be called with three, five, or seven input arguments, depending on whether you want to set a property of the entire FIS structure, for a particular variable belonging to that FIS structure, or for a particular membership function belonging to one of those variables. The arguments are:

- `a` — a variable name of a FIS from the workspace
- `'vartype'` — a string indicating the variable type: `input` or `output`
- `varindex` — the index of the input or output variable
- `'mf'` — a required string for the fourth argument of a seven-argument call for `setfis`, indicating this variable is a membership function
- `mfindex` — the index of the membership function belonging to the chosen variable
- `'fispropname'` — a string indicating the property of the FIS field you want to set: `name`, `type`, `andmethod`, `ormethod`, `impmethod`, `aggmethod`, `defuzzmethod`
- `'newfisprop'` — a string describing the name of the FIS property or method you want to set
- `'varpropname'` — a string indicating the name of the variable field you want to set: `name` or `range`
- `'newvarprop'` — a string describing the name of the variable you want to set (for *name*), or an array describing the range of that variable (for `range`)

- 'mfpropname' — a string indicating the name of the membership function field you want to set: name, type, or params.
- 'newmfprop' — a string describing the name or type of the membership function field want to set (for name or type), or an array describing the range of the parameters (for params)

Examples

setfsi is called with three arguments:

```
a = readfis('tipper');
a2 = setfis(a, 'name', 'eating');
getfis(a2, 'name');
```

which results in

```
out =
eating
```

If it is used with five arguments, setfis updates two variable properties.

```
a2 = setfis(a,'input',1,'name','help');
getfis(a2,'input',1,'name')
ans =
 help
```

If used with seven arguments, setfis updates any of several membership function properties.

```
a2 = setfis(a,'input',1,'mf',2,'name','wretched');
getfis(a2,'input',1,'mf',2,'name')
ans =
 wretched
```

See Also

getfis

Purpose

Fuzzy inference S-function for Simulink

Syntax

```
output = sffis(t,x,u,flag,fismat)
```

Description

In most cases, the Fuzzy Logic Controller block generates a custom block diagram for your FIS. The Fuzzy Wizard, however, cannot handle FIS with custom membership functions or with AND, OR, IMP, and AGG functions outside of the following list:

- `orMethod:max`
- `andMethod: min,prod`
- `impMethod: min,prod`
- `aggMethod: max`

In these cases, the Fuzzy Logic Controller block uses the S-function `sffis` to simulate the FIS. For more discussion of this topic, see "Building Your Own Fuzzy Simulink Models" on page 2-78.

The MEX-file `sffis` is used by Simulink to undertake the calculation normally performed by `evalfis`. It has been optimized to work in the Simulink environment. `sffis` builds a data structure in memory during the initialization phase of a Simulink simulation, which it then continues to use until the simulation is complete.

The arguments `t`, `x`, and `flag` are standard Simulink S-function arguments (see Writing S-Functions in the Simulink documentation). The argument `u` is the input to the MATLAB workspace FIS structure, `fismat`. If, for example, there are two inputs to `fismat`, then `u` is a two-element vector.

See Also

`evalfis`, `fuzblock`

Purpose

Display annotated Fuzzy Inference System

Syntax

```
showfis(fismat)
```

Description

`showfis(fismat)` prints a version of the MATLAB workspace variable FIS, `fismat`, allowing you to see the significance and contents of each field of the structure.

Examples

```
a = readfis('tipper');
showfis(a)
```

Returns:

```
1.  Name             tipper
2.  Type             mamdani
3.  Inputs/Outputs   [2 1]
4.  NumInputMFs      [3 2]
5.  NumOutputMFs     3
6.  NumRules         3
7.  AndMethod        min
8.  OrMethod         max
9.  ImpMethod        min
10. AggMethod        max
11. DefuzzMethod     centroid
12. InLabels         service
13.                  food
14. OutLabels        tip
15. InRange          [0 10]
16.                  [0 10]
17. OutRange         [0 30]
18. InMFLabels       poor
19.                  good
20.                  excellent
21.                  rancid
22.                  delicious
23. OutMFLabels      cheap
```

```
  24.                   average
  25.                   generous
  26. InMFTypes         gaussmf
  27.                   gaussmf
  28.                   gaussmf
  29.                   trapmf
  30.                   trapmf
  31. OutMFTypes        trimf
  32.                   trimf
  33.                   trimf
  34. InMFParams        [1.5 0 0 0]
  35.                   [1.5 5 0 0]
  36.                   [1.5 10 0 0]
  37.                   [0 0 1 3]
  38.                   [7 9 10 10]
  39. OutMFParams       [0 5 10 0]
  40.                   [10 15 20 0]
  41.                   [20 25 30 0]
  42. Rule Antecedent   [1 1]
  43.                   [2 0]
  44.                   [3 2]
  42. Rule Consequent   1
  43.                   2
  44.                   3
  42. Rule Weight       1
  43.                   1
  44.                   1
  42. Rule Connection   2
  43.                   1
  44.                   2
```

See Also

getfis

Purpose

Display Fuzzy Inference System rules

Syntax

```
showrule(fis)
showrule(fis,indexList)
showrule(fis,indexList,format)
showrule(fis,indexList,format,Lang)
```

Description

This command is used to display the rules associated with a given system. It can be invoked with one to four arguments. The first argument, `fis`, is required. This argument is the MATLAB workspace variable name for a FIS structure. The second (optional) argument `indexList` is the vector of rules you want to display. The third argument (optional) is the string representing the format in which the rules are returned. `showrule` can return the rule in any of three different formats: `'verbose'` (the default mode, for which English is the default language), `'symbolic'`, and `'indexed'`, for membership function index referencing.

When used with four arguments, the forth argument must be `verbose`, and `showrule(fis,indexList,format,Lang)` displays the rules in the language given by `lang`, which must be either `'english'`, `'francais'`, or `'deutsch'`.

Examples

```
a = readfis('tipper');
showrule(a,1)
ans =
1. If (service is poor) or (food is rancid)
   then (tip is cheap) (1)

showrule(a,2)
ans =
2. If (service is good) then (tip is average) (1)

showrule(a,[3 1],'symbolic')
ans =
3. (service==excellent) | (food==delicious) =>
   (tip=generous) (1)
```

```
1. (service==poor) | (food==rancid) => (tip=cheap) (1)

showrule(a,1:3,'indexed')
ans =
1 1, 1 (1) : 2
2 0, 2 (1) : 1
3 2, 3 (1) : 2
```

See Also

addrule, parsrule, ruleedit

Purpose

Sigmoidally shaped built-in membership function

Syntax

```
y = sigmf(x,[a c])
```

Description

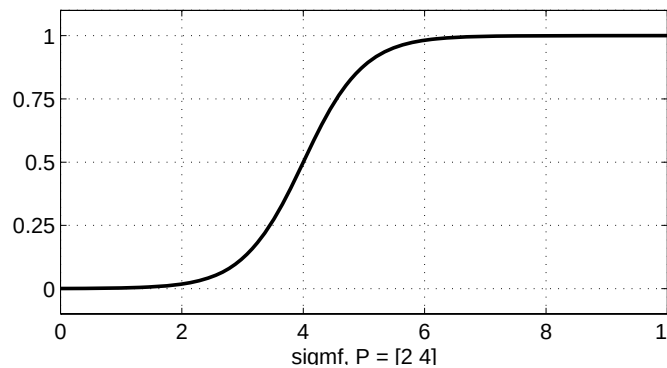
The sigmoidal function, `sigmf(x,[a c])`, as given in the following equation by $f(x,a,c)$ is a mapping on a vector `x`, and depends on two parameters a and c.

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

Depending on the sign of the parameter a, the sigmoidal membership function is inherently open to the right or to the left, and thus is appropriate for representing concepts such as "very large" or "very negative." More conventional-looking membership functions can be built by taking either the product or difference of two different sigmoidal membership functions. For more information see`sigmf` and `psigmf`.

Examples

```
x=0:0.1:10;
y=sigmf(x,[2 4]);
plot(x,y)
xlabel('sigmf, P=[2 4]')
```



See Also

dsigmf, gaussmf, gauss2mf, gbellmf, evalmf, mf2mf, pimf, psigmf, sigmf, smf, trapmf, trimf, zmf

Purpose

S-shaped built-in membership function

Syntax

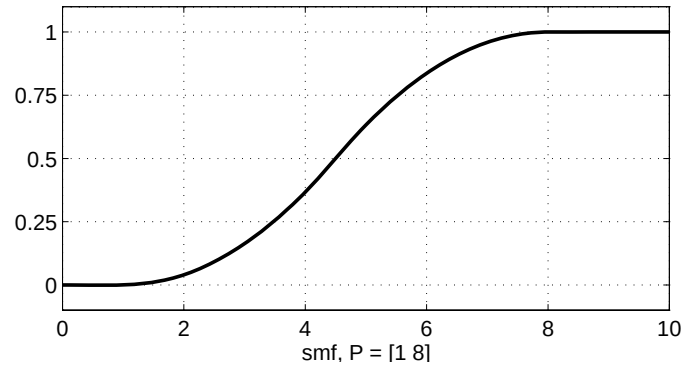
```
y = smf(x,[a b])
```

Description

This spline-based curve is a mapping on the vector `x` and is named because of its S-shape. The parameters a and b locate the extremes of the sloped portion of the curve.

Examples

```
x=0:0.1:10;
y=smf(x,[1 8]);
plot(x,y)
xlabel('smf, P=[1 8]')
```



See Also

`dsigmf`, `gaussmf`, `gauss2mf`, `gbellmf`, `evalmf`, `mf2mf`, `pimf`, `psigmf`, `sigmf`, `smf`, `trapmf`, `trimf`, `zmf`

subclust

Purpose

Find cluster centers with subtractive clustering

Syntax

```
[C,S] = subclust(X,radii,xBounds,options)
```

Description

This function estimates the cluster centers in a set of data by using the subtractive clustering method. The subtractive clustering method assumes each data point is a potential cluster center and calculates a measure of the likelihood that each data point would define the cluster center, based on the density of surrounding data points. The algorithm:

- Selects the data point with the highest potential to be the first cluster center
- Removes all data points in the vicinity of the first cluster center (as determined by `radii`), in order to determine the next data cluster and its center location
- Iterates on this process until all of the data is within `radii` of a cluster center

The subtractive clustering method is an extension of the mountain clustering method proposed by R. Yager.

The matrix `X` contains the data to be clustered; each row of `X` is a data point. The variable `radii` is a vector of entries between 0 and 1 that specifies a cluster center's range of influence in each of the data dimensions, assuming the data falls within a unit hyperbox. Small `radii` values generally result in finding a few large clusters. The best values for `radii` are usually between 0.2 and 0.5.

For example, if the data dimension is two (`X` has two columns), `radii=[0.5 0.25]` specifies that the range of influence in the first data dimension is half the width of the data space and the range of influence in the second data dimension is one quarter the width of the data space. If `radii` is a scalar, then the scalar value is applied to all data dimensions, i.e., each cluster center has a spherical neighborhood of influence with the given radius.

`xBounds` is a 2-by-N matrix that specifies how to map the data in `X` into a unit hyperbox, where N is the data dimension. This argument is optional if `X` is already normalized. The first row contains the minimum axis range values and the second row contains the maximum axis range values for scaling the data in each dimension.

For example, `xBounds = [-10 -5; 10 5]` specifies that data values in the first data dimension are to be scaled from the range [-10 +10] into values in the range [0 1]; data values in the second data dimension are to be scaled from the range [-5 +5] into values in the range [0 1]. If `xBounds` is an empty matrix or not provided, then `xBounds` defaults to the minimum and maximum data values found in each data dimension.

The `options` vector can be used for specifying clustering algorithm parameters to override the default values. These components of the vector `options` are specified as follows:

- `options(1) = quashFactor`: This factor is used to multiply the radii values that determine the neighborhood of a cluster center, so as to quash the potential for outlying points to be considered as part of that cluster. (default: 1.25)
- `options(2) = acceptRatio`: This factor sets the potential, as a fraction of the potential of the first cluster center, above which another data point is accepted as a cluster center. (default: 0.5)
- `options(3) = rejectRatio`: This factor sets the potential, as a fraction of the potential of the first cluster center, below which a data point is rejected as a cluster center. (default: 0.15)
- `options(4) = verbose`: If this term is not zero, then progress information is printed as the clustering process proceeds. (default: 0)

The function returns the cluster centers in the matrix `C`; each row of `C` contains the position of a cluster center. The returned `S` vector contains the sigma values that specify the range of influence of a cluster center in each of the data dimensions. All cluster centers share the same set of sigma values.

Examples

```
[C,S] = subclust(X,0.5)
```

This command sets the minimum number of arguments needed to use this function. A range of influence of 0.5 has been specified for all data dimensions.

```
[C,S] = subclust(X,[0.5 0.25 0.3],[],[2.0 0.8 0.7])
```

This command assumes the data dimension is 3 (`X` has 3 columns) and uses a range of influence of 0.5, 0.25, and 0.3 for the first, second, and third data dimension, respectively. The scaling factors for mapping the data into a unit hyperbox are obtained from the minimum and maximum data values. The `squashFactor` is set to 2.0, indicating that you only want to find clusters that are far from each other. The `acceptRatio` is set to 0.8, indicating that only data points that have a very strong potential for being cluster centers are accepted. The `rejectRatio` is set to 0.7, indicating that you want to reject all data points without a strong potential.

References

Chiu, S., "Fuzzy Model Identification Based on Cluster Estimation," *Journal of Intelligent & Fuzzy Systems*, Vol. 2, No. 3, Sept. 1994.

Yager, R. and D. Filev, "Generation of Fuzzy Rules by Mountain Clustering," *Journal of Intelligent & Fuzzy Systems*, Vol. 2, No. 3, pp. 209-219, 1994.

See Also

`genfis2`

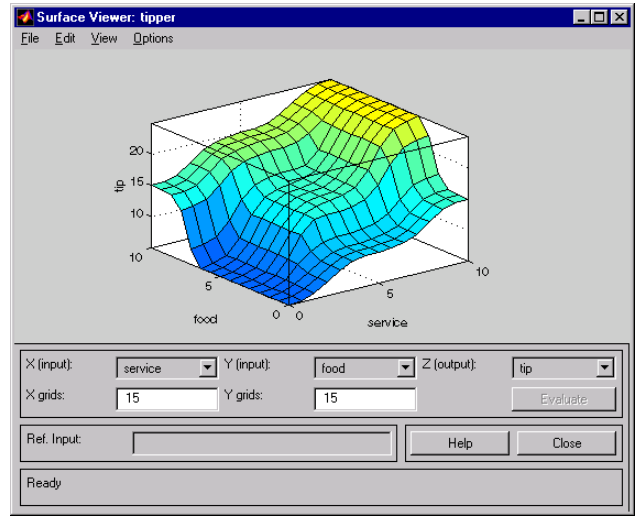
Purpose

Open Output Surface Viewer

Syntax

```
surfview('a')
```

Description



The Surface Viewer invoked using `surfview('a')` is a GUI tool that lets you examine the output surface of a FIS stored in a file, `a.fis`, for any one or two inputs. Because it does not alter the fuzzy system or its associated FIS structure in any way, Surface Viewer is a read-only editor. Using the pop-up menus, you select the two input variables you want assigned to the two input axes (X and Y), as well the output variable you want assigned to the output (or Z) axis. Click **Evaluate** to perform the calculation and plot the output surface.

By clicking on the plot axes and dragging the mouse, you can manipulate the surface so that you can view it from different angles.

If there are more than two inputs to your system, you must supply the constant values associated with any unspecified inputs in the reference input section.

Refer to "The Surface Viewer" on page 2-52 for more information about how to use `surfview`.

Menu Items

On the Surface Viewer, there is a menu bar that allows you to open related GUI tools, open and save systems, and so on. The **File** menu for the Surface Viewer is the same as the one found on the FIS Editor. Refer to `fuzzy` for more information:

- Use the **Edit** menu items:

  Undo to undo the most recent action

  FIS properties to invoke the FIS Editor

  Membership functions to invoke the Membership Function Editor

  Rules... to invoke the Rule Editor

- Use the **View** menu item:

  Rules to invoke the Rule Viewer

- Use the **Options** menu items:

  Plot to choose among eight different kinds of plot styles.

  Color Map to choose among several different color schemes.

  Always evaluate to automatically evaluate and plot a new surface every time you make a change that affects the plot (like changing the number of grid points). This option is the default. To clear this option, select it once more.

See Also

`anfisedit`, `fuzzy`, `gensurf`, `mfedit`, `ruleedit`, `ruleview`

Purpose

Trapezoidal-shaped built-in membership function

Syntax

```
y = trapmf(x,[a b c d])
```

Description

The trapezoidal curve is a function of a vector, x, and depends on four scalar parameters a, b, c, and d, as given by

$$f(x;a,b,c,d) = \begin{cases} 0, & x \le a \\ \dfrac{x-a}{b-a}, & a \le x \le b \\ 1, & b \le x \le c \\ \dfrac{d-x}{d-c}, & c \le x \le d \\ 0, & d \le x \end{cases}$$

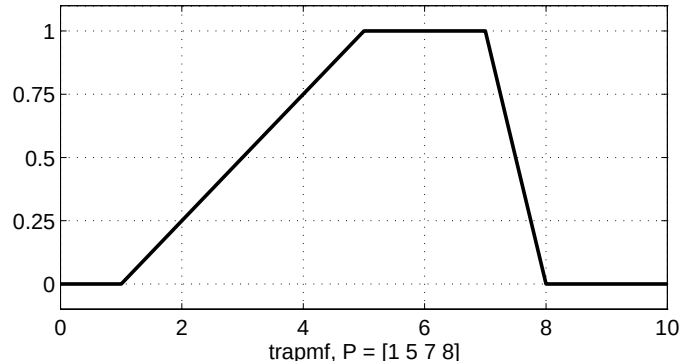
or, more compactly, by

$$f(x;a,b,c,d) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), o\right)$$

The parameters a and d locate the “feet” of the trapezoid and the parameters b and c locate the “shoulders.”

Examples

```
x=0:0.1:10;
y=trapmf(x,[1 5 7 8]);
plot(x,y)
xlabel('trapmf, P=[1 5 7 8]')
```



See Also

`dsigmf`, `gaussmf`, `gauss2mf`, `gbellmf`, `evalmf`, `mf2mf`, `pimf`, `psigmf`, `sigmf`, `smf`, `trimf`, `zmf`

Purpose

Triangular-shaped built-in membership function

Syntax

```
y = trimf(x,params)
y = trimf(x,[a b c])
```

Description

The triangular curve is a function of a vector, x, and depends on three scalar parameters a, b, and c, as given by

$$f(x;a,b,c) = \begin{cases} 0, & x \le a \\ \dfrac{x-a}{b-a}, & a \le x \le b \\ \dfrac{c-x}{c-b}, & b \le x \le c \\ 0, & c \le x \end{cases}$$

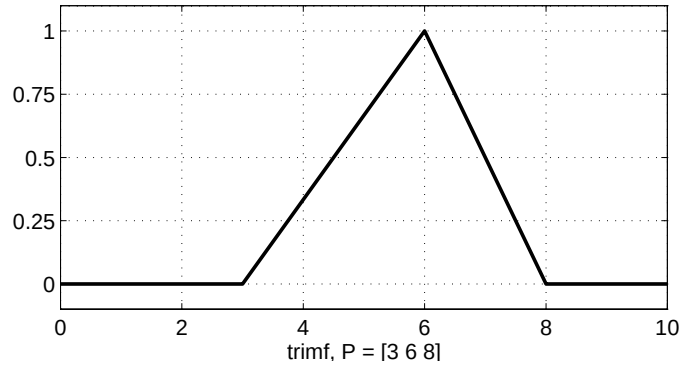
or, more compactly, by

$$f(x;a,b,c) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), o\right)$$

The parameters a and c locate the "feet" of the triangle and the parameter c locates the peak.

Examples

```
x=0:0.1:10;
y=trimf(x,[3 6 8]);
plot(x,y)
xlabel('trimf, P=[3 6 8]')
```



See Also

`dsigmf`, `gaussmf`, `gauss2mf`, `gbellmf`, `evalmf`, `mf2mf`, `pimf`, `psigmf`, `sigmf`, `smf`, `trapmf`

Purpose

Save Fuzzy Inference System to file

Syntax

```
writefis(fismat)
writefis(fismat,'filename')
writefis(fismat,'filename','dialog')
```

Description

`writefis` saves a MATLAB workspace FIS structure, `fismat`, as a `.fis` file.

`writefis(fismat)` opens a dialog box to assist with the naming and directory location of the file.

`writefis(fismat,'filename')` writes a `.fis` file corresponding to the FIS structure, `fismat`, to a file called `filename.fis`. No dialog box appears, and the file is saved to the current directory.

`writefis(fismat,'filename', 'dialog')` opens a dialog box with the default name `filename.fis` supplied.

The extension `.fis` is only added to `filename` if it is not already included in the name.

Examples

```
a = newfis('tipper');
a = addvar(a,'input','service',[0 10]);
a = addmf(a,'input',1,'poor','gaussmf',[1.5 0]);
a = addmf(a,'input',1,'good','gaussmf',[1.5 5]);
a = addmf(a,'input',1,'excellent','gaussmf',[1.5 10]);
writefis(a,'my_file')
```

See Also

`readfis`

Purpose Z-shaped built-in membership function

Syntax

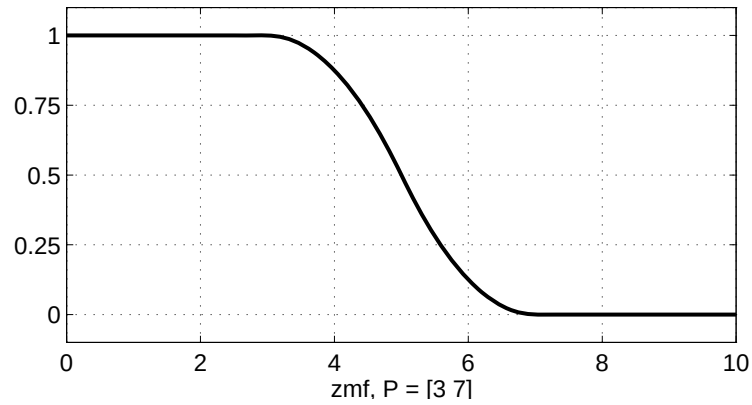
```
y = zmf(x,[a b])
```

Description This spline-based function of x is so named because of its Z-shape. The parameters a and b locate the extremes of the sloped portion of the curve as given by.

$$\begin{Bmatrix} 1, & x \le a \\ 1-2\left(\dfrac{x-a}{b-a}\right)^2, & a \le x \le \dfrac{a+b}{2} \\ 2\left(b-\dfrac{x}{b-a}\right)^2, & \dfrac{a+b}{2} \le x \le b \\ 0, & x \ge b \end{Bmatrix}$$

Examples

```
x=0:0.1:10;
y=zmf(x,[3 7]);
plot(x,y)
xlabel('zmf, P=[3 7]')
```



See Also

dsigmf, gaussmf, gauss2mf, gbellmf, evalmf, mf2mf, pimf, psigmf, sigmf, smf, trapmf, trimf

5

Blocks — By Category

Controllers (p. 5-2)	Controller blocks that implement fuzzy inference systems
Logical Operators (p. 5-3)	Probabilistic OR blocks for use with membership function blocks
Membership Functions (p. 5-4)	Blocks that implement various membership functions

Controllers

Fuzzy Logic Controller

Fuzzy Logic Controller with Ruleviewer

Fuzzy inference system in Simulink

Fuzzy inference system with Ruleviewer in Simulink

Logical Operators

Probabilistic OR	Probabilistic OR function in Simulink
Probabilistic Rule Agg	Probabilistic OR function, rule aggregation method

Membership Functions

Diff. Sigmoidal MF	Difference of two sigmoids membership function in Simulink
Gaussian MF	Gaussian membership function in Simulink
Gaussian2 MF	Combination of two Gaussians membership function in Simulink
Generalized Bell MF	Generalized bell membership function in Simulink
Pi-shaped MF	Pi-shaped membership function in Simulink
Prod. Sigmoidal MF	Product of two sigmoids membership function in Simulink
S-shaped MF	S-shaped membership function in Simulink
Sigmoidal MF	Sigmoidal membership function in Simulink
Trapezoidal MF	Trapezoidal membership function in Simulink
Triangular MF	Triangular membership function in Simulink
Z-shaped MF	Z-shaped membership function in Simulink

6

Blocks — Alphabetical List

Diff. Sigmoidal MF

Purpose

Difference of two sigmoids membership function in Simulink

Description

The Diff. Sigmoidal MF block implements a membership function in Simulink based on the difference between two sigmoids. The two sigmoid curves are given by

$$f_k(x) = \frac{1}{1 + \exp(-a_k(x - c_k))}$$

where k=1,2. The parameters $a_1$and a_2 control the slopes of the left and right curves. The parameters c_1 and c_2 control the points of inflection for the left and right curves. The parameters a_1 and a_2 should be positive.

See Also

`dsigmf`

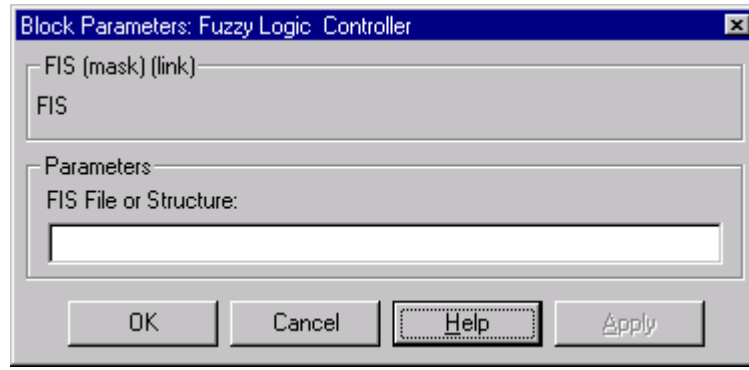
Purpose

Fuzzy inference system in Simulink

Description

The Fuzzy Logic Controller block implements a fuzzy inference system (FIS) in Simulink. See "Working with Simulink" on page 2-71 for a discussion of how to use this block.

Dialog Box



FIS File or Structure

Specify your fuzzy inference system.

See Also

Fuzzy Logic Controller with Ruleviewer

Fuzzy Logic Controller with Ruleviewer

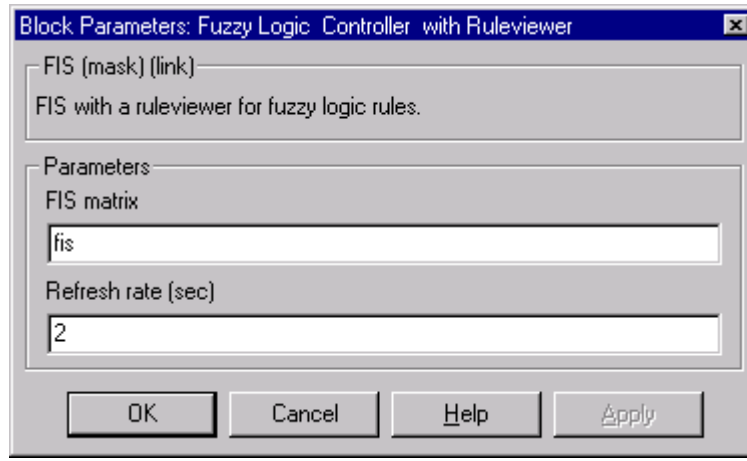
Purpose

Fuzzy inference system with Ruleviewer in Simulink

Description

The Fuzzy Logic Controller with Ruleviewer block implements a fuzzy inference system (FIS) with the ruleviewer in Simulink. See "Working with Simulink" on page 2-71 for a discussion of how to use this block.

Dialog Box



FIS matrix

Specify your fuzzy inference system as a file or structure.

Refresh rate (sec)

Specify the refresh rate in seconds.

See Also

Fuzzy Logic Controller

Purpose

Gaussian membership function in Simulink

Description

The Gaussian MF block implements a membership function in Simulink based on a symmetric Gaussian. The Gaussian curve is given by

$$f(x) = \exp\left(\frac{-0.5(x-c)^2}{\sigma^2}\right)$$

where c is the mean and σ is the variance.

See Also

`gaussmf`

Gaussian2 MF

Purpose

Combination of two Gaussians membership function in Simulink

Description

The Gaussian2 MF block implements a membership function based on a combination of two Gaussians. The two Gaussian functions are given by

$$f_k(x) = \exp\left(\frac{-0.5(x - c_k)^2}{\sigma_k^{\,2}}\right)$$

where k=1,2. The parameters c_1 and σ_1 are the mean and variance defining the left-most curve. The parameters c_2 and σ_2 are the mean and variance defining the right-most curve.

See Also

`gauss2mf`

Purpose

Generalized bell membership function in Simulink

Description

The Generalized Bell MF block implements a membership function in Simulink based on a generalized bell-shaped curve. The generalized bell-shaped curve is given by

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

where the parameters a and b vary the width of the curve and the parameter c locates the center of the curve. The parameter b should be positive.

See Also

`gbellmf`

Pi-shaped MF

Purpose

Pi-shaped membership function in Simulink

Description

The Pi-shaped MF block implements a membership function in Simulink based on a spline-based curve, so named because of its Π shape. The parameters a and d locate the left and right base points or "feet" of the curve. The parameters b and c set the left and right top point or "shoulders" of the curve.

See Also

`pimf`

Purpose

Probabilistic OR function in Simulink

Description

The Probabilistic OR block outputs the probabilistic OR value for the vector signal input, based on

$$y = 1 - \mathrm{prod}(1 - x)$$

See Also

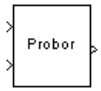
`probor`, Probabilistic Rule Agg

Probabilistic Rule Agg

Purpose

Probabilistic OR function, rule aggregation method

Description



The Probabilistic Rule Agg block outputs the element-wise(.*) probabilistic OR value of the two inputs based on

$$y = 1 - \mathrm{prod}\left(1 - [a;b]\right)$$

The two inputs, a and b, are row vectors.

See Also

`probor`, Probabilistic OR

Purpose

Product of two sigmoids membership function in Simulink

Description

The Prod. Sigmoidal MF block implements a membership function based on the product of two sigmoidal curves. The two sigmoidal curves are given by

$$f_k(x) = \frac{1}{1 + \exp\left(-a_k\left(x - c_k\right)\right)}$$

where k=1,2 The parameters a_1 and a_2 control the slopes of the left and right curves. The parameters c_1 and c_2 control the points of inflection for the left and right curves. Parameters a_1 and a_2 should be positive and negative respectively.

See Also

`psigmf`

S-shaped MF

Purpose

S-shaped membership function in Simulink

Description

The S-shaped MF block implements an S-shaped membership function in Simulink. Going from left to right the function increases from 0 to 1. The parameters `a` and `b` locate the left and right extremes of the sloped portion of the curve.

See Also

`smf`

Purpose

Sigmoidal membership function in Simulink

Description

The Sigmoidal MF block implements a sigmoidal membership function given by

$$f(x) = \frac{1}{1 + \exp(-a(x - c))}$$

When the sign of a is positive the curve increases from left to right. Conversely, when the sign of a is negative the curve decreases from left to right. The parameter c sets the point of inflection of the curve.

See Also

`sigmf`

Trapezoidal MF

Purpose

Trapezoidal membership function in Simulink

Description

The Trapezoidal MF block implements a trapezoidal-shaped membership function. The parameters `a` and `d` set the left and right "feet," or base points, of the trapezoid. The parameters `b` and `c` set the "shoulders," or top of the trapezoid.

See Also

`trapmf`

Purpose

Triangular membership function in Simulink

Description

The Triangular MF block implements a triangular-shaped membership function. The parameters `a` and `c` set the left and right "feet," or base points, of the triangle. The parameter `b` sets the location of the triangle peak.

See Also

`trimf`

Z-shaped MF

Purpose

Z-shaped membership function in Simulink

Description

The Z-shaped MF block implements a Z-shaped membership function. Going from left to right the function decreases from 1 to 0. The parameters `a` and `b` locate the left and right extremes of the sloped portion of the curve.

See Also

`trimf`, `zmf`

A

Examples

Use this list to find examples in the documentation.

Introductory Examples

"The Nonfuzzy Approach" on page 1-12
"The Fuzzy Logic Approach" on page 1-16

Dinner for Two, from the Top

"Another Tipping Problem" on page 2-30

Simulink Examples

"An Example: Water Level Control" on page 2-71

Cart and Pole Simulation

"Example: Cart and Pole Simulation" on page 2-80

Sugeno Fuzzy Inference Example

"An Example: Two Lines" on page 2-87

ANFIS Editor GUI Examples

"ANFIS Editor GUI Example 1: Checking Data Helps Model Validation" on page 2-98
"ANFIS Editor GUI Example 2: Checking Data Does Not Validate Model" on page 2-107

B

Bibliography

[1] Bezdec, J.C., *Pattern Recognition with Fuzzy Objective Function Algorithms,* Plenum Press, New York, 1981.

[2] Chiu, S., "Fuzzy Model Identification Based on Cluster Estimation," *Journal of Intelligent & Fuzzy Systems*, Vol. 2, No. 3, Spet. 1994.

[3] Dubois, D. and H. Prade, *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, New York, 1980.

[4] Jang, J.-S. R., "Fuzzy Modeling Using Generalized Neural Networks and Kalman Filter Algorithm," *Proc. of the Ninth National Conf. on Artificial Intelligence (AAAI-91)*, pp. 762-767, July 1991.

[5] Jang, J.-S. R., "ANFIS: Adaptive-Network-based Fuzzy Inference Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 3, pp. 665-685, May 1993.

[6] Jang, J.-S. R. and N. Gulley, "Gain scheduling based fuzzy controller design," *Proc. of the International Joint Conference of the North American Fuzzy Information Processing Society Biannual Conference, the Industrial Fuzzy Control and Intelligent Systems Conference, and the NASA Joint Technology Workshop on Neural Networks and Fuzzy Logic*, San Antonio, Texas, Dec. 1994.

[7] Jang, J.-S. R. and C.-T. Sun, "Neuro-fuzzy modeling and control, *Proceedings of the IEEE*, March 1995.

[8] Jang, J.-S. R. and C.-T. Sun, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice Hall, 1997.

[9] Kaufmann, A. and M.M. Gupta, *Introduction to Fuzzy Arithmetic*, V.N. Reinhold, 1985.

[10] Lee, C.-C., "Fuzzy logic in control systems: fuzzy logic controller-parts 1 and 2," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 20, No. 2, pp 404-435, 1990.

[11] Mamdani, E.H. and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, Vol. 7, No. 1, pp. 1-13, 1975.

[12] Mamdani, E.H., “Advances in the linguistic synthesis of fuzzy controllers,” *International Journal of Man-Machine Studies*, Vol. 8, pp. 669-678, 1976.

[13] Mamdani, E.H., “Applications of fuzzy logic to approximate reasoning using linguistic synthesis,” *IEEE Transactions on Computers*, Vol. 26, No. 12, pp. 1182-1191, 1977.

[14] Schweizer, B. and A. Sklar, “Associative functions and abstract semi-groups,” *Publ. Math Debrecen*, 10:69-81, 1963.

[15] Sugeno, M., “Fuzzy measures and fuzzy integrals: a survey,” (M.M. Gupta, G. N. Saridis, and B.R. Gaines, editors) *Fuzzy Automata and Decision Processes*, pp. 89-102, North-Holland, NY, 1977.

[16] Sugeno, M., *Industrial applications of fuzzy control*, Elsevier Science Pub. Co., 1985.

[17] Wang, L.-X., *Adaptive fuzzy systems and control: design and stability analysis*, Prentice Hall, 1994.

[18] Widrow, B. and D. Stearns, *Adaptive Signal Processing*, Prentice Hall, 1985.

[19] Yager, R., “On a general class of fuzzy connectives,” *Fuzzy Sets and Systems*, 4:235-242, 1980.

[20] Yager, R. and D. Filev, “Generation of Fuzzy Rules by Mountain Clustering,” *Journal of Intelligent & Fuzzy Systems*, Vol. 2, No. 3, pp. 209-219, 1994.

[21] Zadeh, L.A., “Fuzzy sets,” *Information and Control*, Vol. 8, pp. 338-353, 1965.

[22] Zadeh, L.A., "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 3, No. 1, pp. 28-44, Jan. 1973.

[23] Zadeh, L.A., "The concept of a linguistic variable and its application to approximate reasoning, Parts 1, 2, and 3," *Information Sciences*, 1975, 8:199-249, 8:301-357, 9:43-80.

[24] Zadeh, L.A., "Fuzzy Logic," *Computer*, Vol. 1, No. 4, pp. 83-93, 1988.

[25] Zadeh, L.A., "Knowledge representation in fuzzy logic," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, pp. 89-100, 1989.

Glossary

Adaptive Neuro-Fuzzy Inference System
(ANFIS) A technique for automatically tuning Sugeno-type inference systems based on training data.

aggregation
The combination of the consequents of each rule in a Mamdani fuzzy inference system in preparation for defuzzification.

antecedent
The initial (or "if") part of a fuzzy rule.

consequent
The final (or "then") part of a fuzzy rule.

defuzzification
The process of transforming a fuzzy output of a fuzzy inference system into a crisp output.

degree of fulfillment
See **firing strength**

degree of membership
The output of a membership function, this value is always limited to between 0 and 1. Also known as a membership value or membership grade.

firing strength
The degree to which the antecedent part of a fuzzy rule is satisfied. The firing strength may be the result of an AND or an OR operation, and it shapes the output function for the rule. Also known as *degree of fulfillment*.

fuzzification
The process of generating membership values for a fuzzy variable using membership functions.

fuzzy c-means clustering
A data clustering technique wherein each data point belongs to a cluster to a degree specified by a membership grade.

fuzzy inference system (FIS)
The overall name for a system that uses fuzzy reasoning to map an input space to an output space.

fuzzy operators
AND, OR, and NOT operators. These are also known as *logical connectives*.

fuzzy set
A set that can contain elements with only a partial degree of membership.

fuzzy singleton
A fuzzy set with a membership function that is unity at a one particular point and zero everywhere else.

implication
The process of shaping the fuzzy set in the consequent based on the results of the antecedent in a Mamdani-type FIS.

Mamdani-type inference
A type of fuzzy inference in which the fuzzy sets from the consequent of each rule are combined through the aggregation operator and the resulting fuzzy set is defuzzified to yield the output of the system.

membership function (MF)
A function that specifies the degree to which a given input belongs to a set or is related to a concept.

singleton output function
An output function that is given by a spike at a single number rather than a continuous curve. In Fuzzy Logic Toolbox, it is only supported as part of a zero-order Sugeno model.

subtractive clustering

A technique for automatically generating fuzzy inference systems by detecting clusters in input-output training data.

Sugeno-type inference

A type of fuzzy inference in which the consequent of each rule is a linear combination of the inputs. The output is a weighted linear combination of the consequents.

T-conorm

A two-input function that describes a superset of fuzzy union (OR) operators, including maximum, algebraic sum, and any of several parameterized T-conorms Also known as *S-norm*.

T-norm

A two-input function that describes a superset of fuzzy intersection (AND) operators, including minimum, algebraic product, and any of several parameterized T-norms.

Index

A

addmf 2-65
addrule 2-65
addvar 2-68
aggregation 2-21
AND 2-15
anfis 2-90
    options 2-119
ANFIS 2-91
anfisedit 2-60
antecedent 2-19

B

backpropagation 2-104

C

chaotic time series 2-124
checking data 2-93 2-98
checking error 2-123
clustering 2-124
clustering algorithms Glossary-3
clustering GUI 2-135
consequent 2-16
convertfis 4-11

D

defuzz 4-12
defuzzification 2-17
defuzzify 2-26
degree of membership 2-4
distfcm 2-126
dsigmf 2-11

E

error tolerance 2-104
evalfis 2-66
evalmf 4-17

F

fcm (fuzzy c-means) 2-124
findcluster 2-134
FIS 2-29
    C-code 2-138
    Editor 2-30
    files 2-69
    generating 2-101
    Mamdani-type 2-35
    matrix 2-66
    saving a FIS 2-54
    structure 2-92
fuzblock 2-78
fuzzification 2-17
fuzzy 4-25
fuzzy c-means clustering 3-5 4-18
fuzzy clustering 2-119
fuzzy inference system (FIS) 2-3
fuzzy inference system in Simulink 5-2 6-3
fuzzy inference system with Ruleviewer in
    Simulink 5-2 6-4
fuzzy operators 2-13
fuzzy set 2-4

G

gauss2mf 2-11
gaussian 2-11
gaussmf 2-11
gbellmf 2-11
genfis 2-113
genfis1 2-102
genfis2 2-102
gensurf 2-62
getfis 2-58
grid partition 2-101

H

hybrid method 2-104

I

if-then rules 2-17
    antecedent 2-16
    implication 2-17
implication 2-17
    See also if-then rules
`initfcm` 2-126

L

logical operations 2-13

M

`mam2sug` 4-46
Mamdani-style inference Glossary-2
Mamdani-type inference 2-19
max 2-25
membership function 2-7
    mf editor 2-104
Membership Function Editor 2-30
membership functions
    bell 2-11
    custom 2-55
    Gaussian 2-11
    Pi 2-12
    S 2-12
    sigmoidal 2-11
    Z 2-12
MF. *See* membership function
`mf2mf` 4-47
`mfedit` 4-49
min 2-28
model validation 2-92

N

neuro-fuzzy inference 2-91
`newfis` 2-65
NOT 2-15

O

OR 2-15

P

`pimf` 2-12
`plotfis` 2-60
`plotmf` 2-61
probabilistic OR 2-23
probor 2-25
`psigmf` 2-11

R

`readfis` 2-57
`rmmf` 2-68
`rmvar` 2-68
Rule Editor 2-30
rule formats 4-10
Rule Viewer 2-30
`ruleedit` 4-64
`ruleview` 4-66

S

`setfis` 2-59
`sffis` 4-70
`showfis` 2-59
`showrule` 4-73
`sigmf` 2-11
Simulink blocks
    fuzzy controller with ruleviewer 2-75
    Fuzzy Logic Controller 2-72
Simulink, working with 2-71
singleton 2-21
`sltank` 2-72
`smf` 2-12
stand-alone C code 2-138
stand-alone fuzzy inference engine 2-138
step size 2-121

`structure.field` syntax 2-59
`subclust` 4-78
subtractive clustering 2-101 2-128
Sugeno 2-128
Sugeno-type FIS 2-102
See also Sugeno-type inference
Sugeno-type inference 2-21
sum 2-25
Surface Viewer 2-30
`surfview` 4-81

T

T-conorm 2-16
T-norm 2-15
testing data 2-98
training data 2-93
training error 2-121
trapezoidal 2-10
`trapmf` 2-10
`trimf` 4-85

W

`writefis` 4-87

Z

`zmf` 2-12