How to Contact MathWorks

Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us
Phone: 508-647-7000

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

Fuzzy Logic Toolbox™ User's Guide

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 and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents
MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.
<table>
<thead>
<tr>
<th>Date</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 1995</td>
<td>First printing</td>
<td>Revised for Version 2 (Release 12)</td>
</tr>
<tr>
<td>April 1997</td>
<td>Second printing</td>
<td></td>
</tr>
<tr>
<td>January 1998</td>
<td>Third printing</td>
<td></td>
</tr>
<tr>
<td>September 2000</td>
<td>Fourth printing</td>
<td></td>
</tr>
<tr>
<td>April 2003</td>
<td>Fifth printing</td>
<td></td>
</tr>
<tr>
<td>June 2004</td>
<td>Online only</td>
<td>Updated for Version 2.1.3 (Release 14)</td>
</tr>
<tr>
<td>March 2005</td>
<td>Online only</td>
<td>Updated for Version 2.2.1 (Release 14SP2)</td>
</tr>
<tr>
<td>September 2005</td>
<td>Online only</td>
<td>Updated for Version 2.2.2 (Release 14SP3)</td>
</tr>
<tr>
<td>March 2006</td>
<td>Online only</td>
<td>Updated for Version 2.2.3 (Release 2006a)</td>
</tr>
<tr>
<td>September 2006</td>
<td>Online only</td>
<td>Updated for Version 2.2.4 (Release 2006b)</td>
</tr>
<tr>
<td>March 2007</td>
<td>Online only</td>
<td>Updated for Version 2.2.5 (Release 2007a)</td>
</tr>
<tr>
<td>September 2007</td>
<td>Online only</td>
<td>Revised for Version 2.2.6 (Release 2007b)</td>
</tr>
<tr>
<td>March 2008</td>
<td>Online only</td>
<td>Revised for Version 2.2.7 (Release 2008a)</td>
</tr>
<tr>
<td>October 2008</td>
<td>Online only</td>
<td>Revised for Version 2.2.8 (Release 2008b)</td>
</tr>
<tr>
<td>March 2009</td>
<td>Online only</td>
<td>Revised for Version 2.2.9 (Release 2009a)</td>
</tr>
<tr>
<td>September 2009</td>
<td>Online only</td>
<td>Revised for Version 2.2.10 (Release 2009b)</td>
</tr>
<tr>
<td>March 2010</td>
<td>Online only</td>
<td>Revised for Version 2.2.11 (Release 2010a)</td>
</tr>
<tr>
<td>September 2010</td>
<td>Online only</td>
<td>Revised for Version 2.2.12 (Release 2010b)</td>
</tr>
<tr>
<td>April 2011</td>
<td>Online only</td>
<td>Revised for Version 2.2.13 (Release 2011a)</td>
</tr>
<tr>
<td>September 2011</td>
<td>Online only</td>
<td>Revised for Version 2.2.14 (Release 2011b)</td>
</tr>
<tr>
<td>March 2012</td>
<td>Online only</td>
<td>Revised for Version 2.2.15 (Release 2012a)</td>
</tr>
<tr>
<td>September 2012</td>
<td>Online only</td>
<td>Revised for Version 2.2.16 (Release 2012b)</td>
</tr>
<tr>
<td>March 2013</td>
<td>Online only</td>
<td>Revised for Version 2.2.17 (Release 2013a)</td>
</tr>
<tr>
<td>September 2013</td>
<td>Online only</td>
<td>Revised for Version 2.2.18 (Release 2013b)</td>
</tr>
<tr>
<td>March 2014</td>
<td>Online only</td>
<td>Revised for Version 2.2.19 (Release 2014a)</td>
</tr>
<tr>
<td>October 2014</td>
<td>Online only</td>
<td>Revised for Version 2.2.20 (Release 2014b)</td>
</tr>
<tr>
<td>March 2015</td>
<td>Online only</td>
<td>Revised for Version 2.2.21 (Release 2015a)</td>
</tr>
<tr>
<td>September 2015</td>
<td>Online only</td>
<td>Revised for Version 2.2.22 (Release 2015b)</td>
</tr>
<tr>
<td>March 2016</td>
<td>Online only</td>
<td>Revised for Version 2.2.23 (Release 2016a)</td>
</tr>
<tr>
<td>September 2016</td>
<td>Online only</td>
<td>Revised for Version 2.2.24 (Release 2016b)</td>
</tr>
<tr>
<td>March 2017</td>
<td>Online only</td>
<td>Revised for Version 2.2.25 (Release 2017a)</td>
</tr>
<tr>
<td>September 2017</td>
<td>Online only</td>
<td>Revised for Version 2.3 (Release 2017b)</td>
</tr>
<tr>
<td>March 2018</td>
<td>Online only</td>
<td>Revised for Version 2.3.1 (Release 2018a)</td>
</tr>
<tr>
<td>September 2018</td>
<td>Online only</td>
<td>Revised for Version 2.4 (Release 2018b)</td>
</tr>
<tr>
<td>March 2019</td>
<td>Online only</td>
<td>Revised for Version 2.5 (Release 2019a)</td>
</tr>
<tr>
<td>September 2019</td>
<td>Online only</td>
<td>Revised for Version 2.6 (Release 2019b)</td>
</tr>
</tbody>
</table>
# Getting Started

## Fuzzy Logic Toolbox Product Description

- Key Features

## What Is Fuzzy Logic?

- Description of Fuzzy Logic
- Why Use Fuzzy Logic?
- When Not to Use Fuzzy Logic
- What Can Fuzzy Logic Toolbox Software Do?

## Foundations of Fuzzy Logic

- Overview
- Fuzzy Sets
- Membership Functions
- Logical Operations
- If-Then Rules
- References

## Fuzzy Inference Process

- Fuzzify Inputs
- Apply Fuzzy Operator
- Apply Implication Method
- Aggregate All Outputs
- Defuzzify
- Fuzzy Inference Diagram

## Membership Function Gallery

## Defuzzification Methods

## Fuzzy vs. Nonfuzzy Logic
## Fuzzy Inference System Modeling

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mamdani and Sugeno Fuzzy Inference Systems</strong></td>
<td>2-2</td>
</tr>
<tr>
<td>Mamdani Fuzzy Inference Systems</td>
<td>2-2</td>
</tr>
<tr>
<td>Sugeno Fuzzy Inference Systems</td>
<td>2-3</td>
</tr>
<tr>
<td><strong>Type-2 Fuzzy Inference Systems</strong></td>
<td>2-7</td>
</tr>
<tr>
<td>Interval Type-2 Membership Functions</td>
<td>2-7</td>
</tr>
<tr>
<td>Type-2 Fuzzy Inference Systems</td>
<td>2-8</td>
</tr>
<tr>
<td>Fuzzy Inference Process for Type-2 Fuzzy Systems</td>
<td>2-8</td>
</tr>
<tr>
<td>Type-Reduction Methods</td>
<td>2-15</td>
</tr>
<tr>
<td><strong>Build Fuzzy Systems Using Fuzzy Logic Designer</strong></td>
<td>2-17</td>
</tr>
<tr>
<td>Fuzzy Logic Toolbox Graphical User Interface Tools</td>
<td>2-17</td>
</tr>
<tr>
<td>The Basic Tipping Problem</td>
<td>2-19</td>
</tr>
<tr>
<td>The Fuzzy Logic Designer</td>
<td>2-20</td>
</tr>
<tr>
<td>The Membership Function Editor</td>
<td>2-25</td>
</tr>
<tr>
<td>The Rule Editor</td>
<td>2-33</td>
</tr>
<tr>
<td>The Rule Viewer</td>
<td>2-36</td>
</tr>
<tr>
<td>The Surface Viewer</td>
<td>2-38</td>
</tr>
<tr>
<td>Importing and Exporting Fuzzy Inference Systems</td>
<td>2-40</td>
</tr>
<tr>
<td><strong>Build Fuzzy Systems at the Command Line</strong></td>
<td>2-41</td>
</tr>
<tr>
<td><strong>Build Fuzzy Systems Using Custom Functions</strong></td>
<td>2-52</td>
</tr>
<tr>
<td>Build Fuzzy Inference Systems Using Custom Functions in Fuzzy Logic Designer</td>
<td>2-52</td>
</tr>
<tr>
<td>Specify Custom Membership Functions</td>
<td>2-53</td>
</tr>
<tr>
<td>Specify Custom Inference Functions</td>
<td>2-59</td>
</tr>
<tr>
<td>Specify Custom Type-Reduction Functions</td>
<td>2-65</td>
</tr>
<tr>
<td>Use Custom Functions in Code Generation</td>
<td>2-66</td>
</tr>
<tr>
<td><strong>Fuzzy Trees</strong></td>
<td>2-68</td>
</tr>
<tr>
<td>Types of Hierarchical Structures</td>
<td>2-68</td>
</tr>
<tr>
<td>Add or Remove FIS Tree Outputs</td>
<td>2-75</td>
</tr>
<tr>
<td>Use the Same Value for Multiple inputs of FIS Tree</td>
<td>2-75</td>
</tr>
<tr>
<td>Update Fuzzy Inference Systems in FIS Tree</td>
<td>2-76</td>
</tr>
<tr>
<td>Tune a Fuzzy Tree</td>
<td>2-76</td>
</tr>
<tr>
<td><strong>Fuzzy PID Control with Type-2 FIS</strong></td>
<td>2-78</td>
</tr>
</tbody>
</table>
Fuzzy Logic Image Processing ................................................................. 2-97

Fuzzy Inference System Tuning

Tuning Fuzzy Inference Systems ....................................................... 3-2
Tune Mamdani Fuzzy Inference System ........................................... 3-33
Tune FIS Tree for Gas Mileage Prediction ....................................... 3-45
Predict Chaotic Time Series Using Type-2 FIS ............................... 3-62
Tune Fuzzy Systems using Custom Cost Function ............................. 3-78

Neuro-Adaptive Learning and ANFIS ................................................. 3-92
FIS Structure ................................................................. 3-92
Training Data ............................................................... 3-93
Training Options .............................................................. 3-94
Display Options ............................................................... 3-95
Training Validation .............................................................. 3-96
Training Results ................................................................. 3-97
Training Algorithm Differences ..................................................... 3-98

Train Adaptive Neuro-Fuzzy Inference Systems ............................. 3-100
Load Training Data ............................................................... 3-100
Generate or Load FIS Structure .................................................... 3-104
Train FIS ................................................................. 3-109
Validate Trained FIS ............................................................ 3-111
Importance of Checking Data ......................................................... 3-112

Save Training Error Data to MATLAB Workspace ......................... 3-117
Predict Chaotic Time-Series using ANFIS ....................................... 3-125
Modeling Inverse Kinematics in a Robotic Arm ............................. 3-135
Adaptive Noise Cancellation Using ANFIS ..................................... 3-146
Nonlinear System Identification .................................................... 3-156
Data Clustering

Fuzzy Clustering ................................................. 4-2
What Is Data Clustering? ................................. 4-2
Fuzzy C-Means Clustering .................................. 4-2
Subtractive Clustering ....................................... 4-3
References .................................................. 4-3

Cluster Quasi-Random Data Using Fuzzy C-Means Clustering ................................................. 4-4

Adjust Fuzzy Overlap in Fuzzy C-Means Clustering .................................................. 4-8

Fuzzy C-Means Clustering ........................................ 4-12

Fuzzy C-Means Clustering for Iris Data ......................................... 4-17

Model Suburban Commuting Using Subtractive Clustering ................................................. 4-22

Modeling Traffic Patterns using Subtractive Clustering ................................................. 4-34

Data Clustering Using Clustering Tool ................................................. 4-48
Load and Plot Data .................................................. 4-49
Cluster Data .................................................. 4-49
Save Cluster Centers ............................................. 4-51

Fuzzy Logic in Simulink

Simulate Fuzzy Inference Systems in Simulink ................................................. 5-2
Simulate Fuzzy Inference System ................................................. 5-2
Access Intermediate Fuzzy Inference Results ................................................. 5-11
Simulation Modes ................................................. 5-12
Map Command-Line Functionality to Fuzzy Logic Controller Block .......................................... 5-12

Water Level Control in a Tank ......................................................... 5-15

Temperature Control in a Shower .................................................. 5-23

Implement Fuzzy PID Controller in Simulink Using Lookup Table ........................................... 5-30

6

Deployment

Deploy Fuzzy Inference Systems .................................................. 6-2
  Generate Code in Simulink ...................................................... 6-2
  Generate Code in MATLAB ..................................................... 6-2

Generate Code for Fuzzy System Using Simulink Coder .......... 6-4

Generate Structured Text for Fuzzy System Using Simulink PLC Coder ......................................... 6-9

Generate Code for Fuzzy System Using MATLAB Coder .......... 6-11
Getting Started

• “Fuzzy Logic Toolbox Product Description” on page 1-2
• “What Is Fuzzy Logic?” on page 1-3
• “Foundations of Fuzzy Logic” on page 1-10
• “Fuzzy Inference Process” on page 1-28
• “Membership Function Gallery” on page 1-37
• “Defuzzification Methods” on page 1-40
• “Fuzzy vs. Nonfuzzy Logic” on page 1-47
Fuzzy Logic Toolbox Product Description
Design and simulate fuzzy logic systems

Fuzzy Logic Toolbox provides MATLAB® functions, apps, and a Simulink® block for analyzing, designing, and simulating systems based on fuzzy logic. The product guides you through the steps of designing fuzzy inference systems. Functions are provided for many common methods, including fuzzy clustering and adaptive neurofuzzy learning.

The toolbox lets you model complex system behaviors using simple logic rules, and then implement these rules in a fuzzy inference system. You can use it as a stand-alone fuzzy inference engine. Alternatively, you can use fuzzy inference blocks in Simulink and simulate the fuzzy systems within a comprehensive model of the entire dynamic system.

Key Features

• Fuzzy Logic Design app for building fuzzy inference systems and viewing and analyzing results
• Membership functions for creating fuzzy inference systems
• Support for AND, OR, and NOT logic in user-defined rules
• Standard Mamdani and Sugeno-type fuzzy inference systems
• Automated membership function shaping through neuroadaptive and fuzzy clustering learning techniques
• Ability to embed a fuzzy inference system in a Simulink model
• Ability to generate embeddable C code or stand-alone executable fuzzy inference engines
What Is Fuzzy Logic?

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 software, fuzzy logic should be interpreted as FL, that is, fuzzy logic in its wide sense. The basic ideas underlying FL are explained in “Foundations of Fuzzy Logic” on page 1-10. 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 Artificial Intelligence (AI), what is missing in such systems is a mechanism for dealing with fuzzy consequents and fuzzy antecedents. In fuzzy logic, this mechanism is 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 the 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 the 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?

You can use Fuzzy Logic Toolbox software with MATLAB technical computing software as 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.
Determining 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 software.

- **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 Software Do?

You can create and edit fuzzy inference systems with Fuzzy Logic Toolbox software. 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 software, 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. 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 the toolbox or harness it with another toolbox, such as the Control System Toolbox™, Deep Learning Toolbox™, or Optimization Toolbox™ software.
See Also

More About

• “Foundations of Fuzzy Logic” on page 1-10
• “Fuzzy vs. Nonfuzzy Logic” on page 1-47
Foundations of Fuzzy Logic

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 on the right.

The General Case

![Diagram of the general case of fuzzy inference.]

A Specific Example

![Diagram of a specific example of fuzzy inference.]

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 topic guides you through the fuzzy logic process step by step by providing an introduction to the theory and practice of fuzzy logic.
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.

![Diagram of fuzzy set]

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 1-14 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 three feet to nine 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 six 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 six 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 \( \mu \). 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 Software**

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.
\[ A \{ x, \mu_A(x) | x \in X \} \]

\[ A = \{ x, \mu_A(x) | x \in X \} \]

\( \mu_A(x) \) 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.

The toolbox includes 11 built-in membership function types. These 11 functions are, in turn, built from several basic functions:

- Piece-wise linear functions
- Gaussian distribution function
- Sigmoid curve
- Quadratic and cubic polynomial curves

For detailed information on any of the membership functions mentioned next, see the corresponding reference page.

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

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 \texttt{gaussmf} and \texttt{gauss2mf}.

The generalized bell membership function is specified by three parameters and has the function name \texttt{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.

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 \texttt{sigmf}, you also have the difference between two sigmoidal functions, \texttt{dsigmf}, and the product of two sigmoidal functions \texttt{psigmf}.

Polynomial based curves account for several of the membership functions in the toolbox. Three related membership functions are the \texttt{Z}, \texttt{S}, and \texttt{Pi} curves, all named because of their shape. The function \texttt{zmf} is the asymmetrical polynomial curve open to the left, \texttt{smf} is the mirror-image function that opens to the right, and \texttt{pimf} is zero on both extremes with a rise in the middle.
There is a very wide selection to choose from when you’re selecting a membership function. You can also create your own membership functions with the toolbox. 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.
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 \( \text{min} \) operation. That is, resolve the statement \( A \text{ AND } B \), where \( A \) and \( B \) are limited to the range \((0,1)\), by using the function \( \text{min}(A,B) \). Using the same reasoning, you can replace the OR operation with the \( \text{max} \) function, so that \( A \text{ OR } B \) becomes equivalent to \( \text{max}(A,B) \). Finally, the operation \( \text{NOT } A \) becomes equivalent to the operation \( 1 - A \). Notice how the previous truth table is completely unchanged by this substitution.

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 software 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 $T(.,.)$ with the following properties:

- **Boundary** — $T(0,0) = 0, T(a,1) = T(1,a) = a$
- **Monotonicity** — $T(a,b) \leq T(c,d)$ if $a \leq c$ and $b \leq d$
- **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 pair-wise 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 $S(.,.)$ with the following properties:

- **Boundary** — $S(1,1) = 1, S(a,0) = S(0,a) = a$
- **Monotonicity** — $S(a,b) \leq S(c,d)$ if $a \leq c$ and $b \leq d$
- **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 [10], Dubois and Prade [1], Schweizer and Sklar [7], and Sugeno [8]. 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 two steps:

• Evaluation of the antecedent — Fuzzifying the inputs and applying any necessary fuzzy operators.
• Application of the result to the consequent.

The second step is known as implication. For an if-then rule, the antecedent, p, implies the consequent, q. In binary logic, if p is true, then q is also true (p → q). In fuzzy logic, if p is true to some degree of membership, then q is also true to the same degree (0.5p → 0.5q). In both cases, if p is false, then the value of q is undetermined.
The antecedent of a rule can have multiple parts.

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

In this 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 this 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 truncated as shown in the following figure) or scaling using the prod function (where the output fuzzy set is squashed). Both are supported by the toolbox, but you use truncation for the examples in this section.
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.

**Summary of If-Then Rules**

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>Consequent</th>
</tr>
</thead>
<tbody>
<tr>
<td>If service is excellent or food is delicious then tip = generous</td>
<td></td>
</tr>
<tr>
<td>[ \mu(\text{service}==\text{excellent}) = 0.0 ] [ \mu(\text{food}==\text{delicious}) = 0.7 ]</td>
<td></td>
</tr>
<tr>
<td>If (0.0 or 0.7) then tip = generous</td>
<td></td>
</tr>
<tr>
<td>[ \max(0.0, 0.7) = 0.7 ]</td>
<td></td>
</tr>
<tr>
<td>If (0.7) then tip = generous</td>
<td></td>
</tr>
<tr>
<td>[ \min(0.7, \text{generous}) ]</td>
<td></td>
</tr>
</tbody>
</table>

**Foundations of Fuzzy Logic**

1-25
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. “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17 shows how the whole process works from beginning to end for a particular type of fuzzy inference system called a *Mamdani type*.

**References**


**See Also**

**More About**

- "What Is Fuzzy Logic?" on page 1-3
- "Fuzzy Inference Process" on page 1-28
- "Fuzzy vs. Nonfuzzy Logic" on page 1-47
Fuzzy Inference Process

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 the pieces that are described in “Membership Functions” on page 1-14, “Logical Operations” on page 1-19, and “If-Then Rules” on page 1-23.

This section describes the fuzzy inference process and uses the example of the two-input, one-output, three-rule tipping problem from “The Basic Tipping Problem” on page 2-19. 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 an important aspect of fuzzy logic systems. Instead of sharp switching between modes based on breakpoints, logic flows smoothly from regions where one rule or another dominates.

Fuzzy inference process comprises of five parts:

- Fuzzification of the input variables on page 1-29
- Application of the fuzzy operator (AND or OR) in the antecedent on page 1-30
- Implication from the antecedent to the consequent on page 1-31
• Aggregation of the consequents across the rules on page 1-31
• Defuzzification on page 1-33

A fuzzy inference diagram on page 1-34 displays all parts of the fuzzy inference process — from fuzzification through defuzzification.

**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 (fuzzification). In Fuzzy Logic Toolbox software, the input is always a crisp numerical value limited to the universe of discourse of the input variable (in this case, the interval from 0 through 10). The output is a fuzzy degree of membership in the qualifying linguistic set (always the interval from 0 through 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 several 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 delicious? The following figure shows how well the food at the hypothetical restaurant (rated on a scale from 0 through 10) qualifies as the linguistic variable delicious using a membership function. In this case, we rate the food as an 8, which, given the 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.
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 rule has more than one part, the fuzzy operator is applied to obtain one number that represents the result of the rule antecedent. 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 1-19 section, any number of well-defined methods can fill in for the AND operation or the OR operation. In the 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:

\[ \text{probor}(a, b) = a + b - ab \]

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.
Apply Implication Method

Before applying the implication method, you must determine the rule weight. Every rule has a weight (a number from 0 through 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 on the implication process. However, you can decrease the effect of 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.

Note  Sugeno systems always use the product implication method.

Aggregate All Outputs

Since decisions are based on testing all the rules in a FIS, the rule outputs must be combined in some manner. 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, which is before the final defuzzification step. 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, then the order in which the rules are executed is unimportant. Three built-in methods are supported:

- \texttt{max} (maximum)
- \texttt{probor} (probabilistic OR)
- \texttt{sum} (sum of the rule output sets)

In the following diagram, all three rules are displayed to show how the rule outputs are aggregated into a single fuzzy set whose membership function assigns a weighting for every output (tip) value.
Note Sugeno systems always use the sum aggregation method.

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 to obtain a single output value from the set.

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

While the aggregate output fuzzy set covers a range from 0% though 30%, the defuzzified value is between 5% and 25%. These limits correspond to the centroids of the cheap and generous membership functions, respectively.

**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.
In this figure, the flow proceeds up from the inputs in the lower left, across each row, and then down the rule outputs 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. Using a fuzzy inference diagram, you can learn a lot about how the system operates. For instance, for the particular inputs in this diagram, you can see that the implication method is truncation with the \textit{min} function. The \textit{max} function is used for the fuzzy OR operation. Rule 3 (the bottom-most row in the diagram shown previously) has the strongest influence on the output. The Rule Viewer described in “The Rule Viewer” on page 2-36 is an implementation of the fuzzy inference diagram.
See Also

More About
- “Foundations of Fuzzy Logic” on page 1-10
Membership Function Gallery

This example shows how to display 11 membership functions supported in the Fuzzy Logic Toolbox.

Define membership functions.

```matlab
mf = [...
    fismf('trapmf',[-19 -17 -12 -7]) ...
    fismf('gbellmf',[3 4 -8]) ...
    fismf('trimf',[-9 -1 2]) ...
    fismf('gaussmf',[11 17]) ...
    fismf('gauss2mf',[3 10 5 13]) ...
    fismf('smf',[11 17]) ...
    fismf('zmf',[-18 -10]) ...
    fismf('psigmf',[2 -11 -5 -4]) ...
    fismf('dsigmf',[5 -3 1 5]) ...
    fismf('pimf',[0 7 11 15]) ...
    fismf('sigmf',[2 15]) ...
];
```

For more information on the different membership functions and their parameters, see their respective function reference pages.

Evaluate the membership functions.

```matlab
x = linspace(-20,20,201);
y = evalmf(mf,x);
```

Plot the evaluated membership functions with labels.

```matlab
subplot(2,1,1);
plot(x,y(1:6,:));
axis([min(x) max(x) 0 1.2]);
text((mf(1).Parameters(2)+mf(1).Parameters(3))/2,1.1,mf(1).Type,...
    'horizon','center');
text(mf(2).Parameters(3),1.1,mf(2).Type,...
    'horizon','center');
text(mf(3).Parameters(2),1.1,mf(3).Type,...
    'horizon','center');
text(mf(4).Parameters(2),1.1,mf(4).Type,...
    'horizon','center');
text((mf(5).Parameters(2)+mf(5).Parameters(4))/2,1.1,mf(5).Type,...
    'horizon','center');
```
text(mf(6).Parameters(2), 1.1,mf(6).Type,...
  'horizon','center');
h_gca = gca;
h_gca.XTick = [];

subplot(2,1,2);
plot(x,y(7:11,:)');
axis([min(x) max(x) 0 1.2]);
text(mf(7).Parameters(1),1.1,mf(7).Type,...
  'horizon','center');
text(((mf(8).Parameters(2)+mf(8).Parameters(4))/2,1.1,mf(8).Type,...
  'horizon','center');
text(((mf(9).Parameters(2)+mf(9).Parameters(4))/2,1.1,mf(9).Type,...
  'horizon','center');
text(((mf(10).Parameters(2)+mf(10).Parameters(3))/2,1.1,mf(10).Type,...
  'horizon','center');
text(mf(11).Parameters(2),1.1,mf(11).Type,...
  'horizon','center');
h_gca =  gca;
h_gca.XTick = [];

See Also

More About

- “Foundations of Fuzzy Logic” on page 1-10
- “Fuzzy Inference Process” on page 1-28
Defuzzification Methods

This example shows how to display five defuzzification methods supported in the Fuzzy Logic Toolbox™.

Problem Setup

Suppose you have the following region to be defuzzified. What are some of the methods you might choose?

```matlab
x = -10:0.1:10;

mf1 = trapmf(x,[-10 -8 -2 2]);
mf2 = trapmf(x,[-5 -3 2 4]);
mf3 = trapmf(x,[2 3 8 9]);
mf1 = max(0.5*mf2,max(0.9*mf1,0.1*mf3));

figure('Tag','defuzz');
plot(x,mf1,'LineWidth',3);
h_gca = gca;
h_gca.YTick = [0 .5 1] ;
ylim([-1 1]);
```
**Centroid**

Centroid defuzzification returns the center of area under the curve. If you think of the area as a plate of equal density, the centroid is the point along the x axis about which this shape would balance.

```markdown
x1 = defuzz(x,mf1,'centroid'); % ok<*NOPTS>
```

```markdown
h1 = line([x1 x1],[-0.2 1.2],Color,'k');
t1 = text(x1,-0.2,' centroid','FontWeight','bold');
```
Bisector

The bisector is the vertical line that will divide the region into two sub-regions of equal area. It is sometimes, but not always coincident with the centroid line.

```matlab
x2 = defuzz(x,mf1,'bisector');

gray = 0.7*[1 1 1];
h1.Color = gray;
t1.Color = gray;
h2 = line([x2 x2],[-0.4 1.2],'Color','k');
t2 = text(x2,-0.4,' bisector','FontWeight','bold');
```
**Middle, Smallest, and Largest of Maximum**

MOM, SOM, and LOM stand for Middle, Smallest, and Largest of Maximum, respectively. These three methods key off the maximum value assumed by the aggregate membership function. In this example, because there is a plateau at the maximum value, they are distinct. If the aggregate membership function has a unique maximum, then MOM, SOM, and LOM all take on the same value.

\[
x_3 = \text{defuzz}(x, \text{mf1, 'mom'})
\]
\[
x_3 = -5
\]

\[
x_4 = \text{defuzz}(x, \text{mf1, 'som'})
\]
\[
x_4 = -2
\]
x5 = defuzz(x,mf1,'lom')

x5 = -8

h2.Color = gray;
t2.Color = gray;

h3 = line([x3 x3],[-0.7 1.2],'Color','k');
t3 = text(x3,-0.7,'MOM','FontWeight','bold');
h4 = line([x4 x4],[-0.8 1.2],'Color','k');
t4 = text(x4,-0.8,'SOM','FontWeight','bold');
h5 = line([x5 x5],[-0.6 1.2],'Color','k');
t5 = text(x5,-0.6,'LOM','FontWeight','bold');
Picking a Method

Which of these methods is the right one? There's no simple answer. But if you want to get started quickly, generally the centroid method is good enough. Later you can always change your defuzzification method to see if another method works better.

```plaintext
h3.Color = gray;
t3.Color = gray;
h4.Color = gray;
t4.Color = gray;
h5.Color = gray;
t5.Color = gray;
h1.Color = 'red';
t1.Color = 'red';
```
See Also

More About

- “Foundations of Fuzzy Logic” on page 1-10
- “Fuzzy Inference Process” on page 1-28
Fuzzy vs. Nonfuzzy Logic

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.

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

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

Nonfuzzy Approach

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

```matlab
service = 0:.5:10;
tip = 0.15*ones(size(service));
plot(service,tip);
xlabel('Service')
ylabel('Tip')
ylim([0.05 0.25])
```
This relationship does not account for the quality of the service, so you must add a term to the equation. Since service is rated on a scale from 0 through 10, you the tip increase linearly from 5% if the service is bad to 25% if the service is excellent. Now the relation looks like the following plot:

```matlab
tip = (.20/10)*service+0.05;
plot(service,tip)
xlabel('Service')
ylabel('Tip')
ylim([0.05 0.25])
```
The formula does what you want it to do, and is straight forward. However, you may want the tip to reflect the quality of the food as well. This extension of the problem is defined as follows.

**Extended Tipping Problem.** Given two sets of numbers from 0 through 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.

```matlab
food = 0:.5:10;
[F,S] = meshgrid(food,service);
tip = (0.20/20).*(S+F)+0.05;
surf(S,F,tip)
```
In this case, the results look satisfactory, but when you look at them closely, they do not seem right. Suppose that 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%.

```matlab
servRatio = 0.8;
tip = servRatio*(0.20/10*S+0.05) + ...  
    (1-servRatio)*(0.20/10*F+0.05);
surf(S,F,tip)
xlabel('Service')
```

1-50
The response is still somehow too uniformly linear. Suppose that you want more of a flat response in the middle, that is, 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 create a simple conditional tip assignment using logical indexing.

```matlab
tip = zeros(size(service));
tip(service<3) = (0.10/3)*service(service<3)+0.05;
tip(service>=3 & service<7) = 0.15;
```
tip(service>=7 & service<=10) = ... 
   (0.10/3)*(service(service>=7 & service<=10)-7)+0.15;
plot(service,tip)
xlabel('Service')
ylabel('Tip')
ylim([0.05 0.25])

Suppose that you extend this approach to two dimensions, where you account for food 
quality again.

servRatio = 0.8;
tip = zeros(size(S));
tip(S<3) = ((0.10/3)*S(S<3)+0.05)*servRatio + ... 
   (1-servRatio)*(0.20/10*F(S<3)+0.05);
The plot looks good, but the function is surprisingly complicated. It is even not apparent how the algorithm works to someone who did not see the original design process.
Fuzzy Logic Approach

In general, you want 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 could 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. To include the effect of food quality 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 list of three rules like so.

Tipping Problem Rules - 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 and they correspond to 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?

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.

**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 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 list that influenced the shape of the overall output without needing to undo what had already been done.

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 can change from day to day.
day, city to city, country to country. However, 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 for difficult recalibration.

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

```matlab
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. 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

Fuzzy logic uses language that is clear to you and that also has meaning to the computer, which is why it is a 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 no longer the computer - it is your mental model of what the computer is doing. Fuzzy logic lets the machine work with your preferences rather than the other way around.

**See Also**

**Related Examples**

- “Build Fuzzy Systems at the Command Line” on page 2-41
- “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17
Fuzzy Inference System Modeling

- “Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2
- “Type-2 Fuzzy Inference Systems” on page 2-7
- “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17
- “Build Fuzzy Systems at the Command Line” on page 2-41
- “Build Fuzzy Systems Using Custom Functions” on page 2-52
- “Fuzzy Trees” on page 2-68
- “Fuzzy PID Control with Type-2 FIS” on page 2-78
- “Fuzzy Logic Image Processing” on page 2-97
Mamdani and Sugeno Fuzzy Inference Systems

Fuzzy Logic Toolbox software supports two types of fuzzy inference systems:
• Mamdani systems
• Sugeno systems

<table>
<thead>
<tr>
<th>Fuzzy Inference System</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mamdani</td>
<td>• Intuitive</td>
</tr>
<tr>
<td></td>
<td>• Well-suited to human input</td>
</tr>
<tr>
<td></td>
<td>• More interpretable rule base</td>
</tr>
<tr>
<td></td>
<td>• Have widespread acceptance</td>
</tr>
<tr>
<td>Sugeno</td>
<td>• Computationally efficient</td>
</tr>
<tr>
<td></td>
<td>• Work well with linear techniques, such as PID control</td>
</tr>
<tr>
<td></td>
<td>• Work well with optimization and adaptive techniques</td>
</tr>
<tr>
<td></td>
<td>• Guarantee output surface continuity</td>
</tr>
<tr>
<td></td>
<td>• Well-suited to mathematical analysis</td>
</tr>
</tbody>
</table>

Mamdani Fuzzy Inference Systems

Mamdani fuzzy inference was first introduced as a method to create a control system by synthesizing a set of linguistic control rules obtained from experienced human operators [1]. In a Mamdani system, the output of each rule is a fuzzy set.

Since Mamdani systems have more intuitive and easier to understand rule bases, they are well-suited to expert system applications where the rules are created from human expert knowledge, such as medical diagnostics.

The inference process of a Mamdani system is described in “Fuzzy Inference Process” on page 1-28 and summarized in the following figure.
The output of each rule is a fuzzy set derived from the output membership function and the implication method of the FIS. These output fuzzy sets are combined into a single fuzzy set using the aggregation method of the FIS. Then, to compute a final crisp output value, the combined output fuzzy set is defuzzified using one of the methods described in “Defuzzification Methods” on page 1-40.

**Sugeno Fuzzy Inference Systems**

Sugeno fuzzy inference, also referred to as Takagi-Sugeno-Kang fuzzy inference, uses *singleton* output membership functions that are either constant or a linear function of the input values. The defuzzification process for a Sugeno system is more computationally efficient compared to that of a Mamdani system, since it uses a weighted average or
weighted sum of a few data points rather than compute a centroid of a two-dimensional area. [2]

You can convert a Mamdani system into a Sugeno system using the `convertToSugeno` function. The resulting Sugeno system has constant output membership functions that correspond to the centroids of the Mamdani output membership functions.

Each rule in a Sugeno system operates as shown in the following diagram, which shows a two-input system with input values \( x \) and \( y \).

Each rule generates two values:

- \( z_i \) — Rule output level, which is either a constant value or a linear function of the input values:
  \[
  z_i = a_ix + b_iy + c_i
  \]
  Here, \( x \) and \( y \) are the values of input 1 and input 2, respectively, and \( a_i, b_i, \) and \( c_i \) are constant coefficients. For a zero-order Sugeno system, \( z_i \) is a constant \((a = b = 0)\).

- \( w_i \) — Rule firing strength derived from the rule antecedent
  \[
  w_i = AndMethod(F_1(x), F_2(y))
  \]
  Here, \( F_1(...) \) and \( F_2(...) \) are the membership functions for inputs 1 and 2, respectively.

The output of each rule is the weighted output level, which is the product of \( w_i \) and \( z_i \).

The easiest way to visualize first-order Sugeno systems \((a \text{ and } b \text{ are nonzero})\) 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 within the output space, depending on the input values. The rule firing strength then defines the size of the singleton spike.
The final output of the system is the weighted average over all rule outputs:

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

where \( N \) is the number of rules.

The following figure shows the fuzzy inference process for a Sugeno system.

**Note** Sugeno systems always use product implication and sum aggregation.
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 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 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.

References


See Also

More About

- “Foundations of Fuzzy Logic” on page 1-10
- “Fuzzy Inference Process” on page 1-28
- “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17
- “Build Fuzzy Systems at the Command Line” on page 2-41
- “Build Fuzzy Systems Using Custom Functions” on page 2-52
Type-2 Fuzzy Inference Systems

For any value in the universe of discourse, a traditional type-1 membership function has a single membership value. Therefore, while a type-1 membership function models the degree of membership in a given linguistic set, it does not model uncertainty in the degree of membership. To model such uncertainty, you can use interval type-2 membership functions. In such type-2 membership functions, the degree of membership can have a range of values.

For examples that use type-2 fuzzy inference systems, see “Fuzzy PID Control with Type-2 FIS” on page 2-78 and “Predict Chaotic Time Series Using Type-2 FIS” on page 3-62.

Interval Type-2 Membership Functions

An interval type-2 membership function is defined by an upper and lower membership function. The upper membership function (UMF) is equivalent to a traditional type-1 membership function. The lower membership function (LMF) is less than or equal to the upper membership function for all possible input values. The region between the UMF and LMF is the footprint of uncertainty (FOU). The following diagram shows the UMF (red) the LMF (blue), and the FOU (shaded) for a type-2 triangular membership function.

For each input value in the universe of discourse, the degree of membership is the range of values between the LMF and UMF values.
Type-2 Fuzzy Inference Systems

Using Fuzzy Logic Toolbox software, you can create both type-2 Mamdani and Sugeno fuzzy inference systems.

- In type-2 Mamdani systems, both the input and output membership functions are type-2 fuzzy sets.
- In type-2 Sugeno systems, only the input membership functions are type-2 fuzzy sets. The output membership functions are the same as for a type-1 Sugeno system — constant or a linear function of the input values.

To create type-2 Mamdani and Sugeno systems, use `mamfistype2` and `sugfistype2` objects, respectively. These objects have the same parameters as the type-1 `mamfis` and `sugfis` objects along with an additional `TypeReductionMethod` parameter.

You can also create a type-2 fuzzy inference system by converting an existing type-1 system, such as one created using the `genfis` function. To do so, use the `convertToType2` function.

Once you create a type-2 fuzzy inference system, you can:

- Evaluate the fuzzy system using the `evalfis` functions
- Simulate the fuzzy system using the Fuzzy Logic Controller block
- Tune the parameters of the fuzzy system using the `tunefis` function
- Deploy the fuzzy system as described in “Deploy Fuzzy Inference Systems” on page 6-2

Fuzzy Inference Process for Type-2 Fuzzy Systems

Antecedent Processing

For type-2 fuzzy inference systems, input values are fuzzified by finding the corresponding degree of membership in both the UMFs and LMFs from the rule antecedent. Doing so generates two fuzzy values for each type-2 membership function. For example, the fuzzification in the following figure shows the membership value in the upper membership function ($f_U$) and the lower membership function ($f_L$).
Next, a range of rule firing strengths is found by applying the fuzzy operator to the fuzzified values of the type-2 membership functions, as shown in the following figure. The maximum value of this range ($w_U$) is the result of applying the fuzzy operator to the fuzzy values from the UMFs. The minimum value ($w_L$) is the result of applying the fuzzy operator to the fuzzy values from the LMFs.
Antecedent processing is the same for both Mamdani and Sugeno systems.

**Consequent Processing**

For a Mamdani system, the implication method clips (min implication) or scales (prod implication) the UMF and LMF of the output type-2 membership function using the rule firing range limits. This process produces an output fuzzy set for each rule. The following figure shows the output fuzzy set (dark gray region) produced by applying min implication to the UMF (red) and LMF (blue).
For a type-2 Sugeno system, the output level $z_i$ for the $i$th rule is computed in the same manner as for a type-1 Sugeno system.

$$z_i = c_0^i + \sum_{j=1}^{M} c_j^i x_j$$

Here, $j$ is the input index, $x_j$ is the value of the $j$th input variable, and the $c$ terms are the upper membership function parameters.

Unlike a type-1 Sugeno system, the rule firing strengths are not used to process the consequent of each rule. Instead, the output level and rule firing strengths are used during the aggregation process.

**Aggregation**

The goal of the aggregation stage is to derive a single type-2 fuzzy set from the rule output fuzzy sets.

For a type-2 Mamdani system, the software finds an aggregate type-2 fuzzy set by applying the aggregation method to the UMFs and LMFs of the output fuzzy sets of all the rules. The following figure shows the aggregation of two type-2 fuzzy sets (the outputs for a two-rule system) using max aggregation.
For a type-2 Sugeno system, the aggregate fuzzy set is derived using the following steps:

1. Sort the rule output levels \((z_i)\) from all the rules into ascending order. These output level values define the universe of discourse for the aggregate type-2 fuzzy set.
2. For each output level, define the UMF value using the maximum firing range value from the corresponding rule.
3. For each output level, define the LMF value using the minimum firing range value from the corresponding rule.

For example, suppose you have a type-2 Sugeno system with seven rules. Further, assume these rules have the following output levels and firing range limits.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Output Level (z)</th>
<th>Minimum Firing Value</th>
<th>Maximum Firing Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.3</td>
<td>0.1</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>4.9</td>
<td>0.4</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>1.6</td>
<td>0.3</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>5.8</td>
<td>0.5</td>
<td>0.7</td>
</tr>
<tr>
<td>5</td>
<td>5.4</td>
<td>0.2</td>
<td>0.6</td>
</tr>
<tr>
<td>6</td>
<td>0.7</td>
<td>0.5</td>
<td>0.8</td>
</tr>
<tr>
<td>7</td>
<td>3.2</td>
<td>0.2</td>
<td>0.7</td>
</tr>
</tbody>
</table>

The following figure shows the aggregated type-2 fuzzy set with its associated UMF (red) and LMF (blue).
**Type Reduction and Defuzzification**

To find the final crisp output value for the inference process, the aggregate type-2 fuzzy set is first reduced to an interval type-1 fuzzy set, which is a range with lower limit $c_L$ and upper limit $c_R$. This interval type-1 fuzzy set is commonly referred to as the centroid of the type-2 fuzzy set. In theory, this centroid is the average of the centroids of all the type-1 fuzzy sets embedded in the type-2 fuzzy set. In practice, it is not possible to compute the exact values of $c_L$ and $c_R$. Instead, iterative type-reduction methods are used to estimate these values.

For a given aggregate type-2 fuzzy set, the approximate values of $c_L$ and $c_R$ are the centroids of the following type-1 fuzzy sets (green).
Mathematically, these centroids are [1]:

\[
C_L \approx \frac{\sum_{i=1}^{L} x_i \mu_{umf}(x_i) + \sum_{i=L+1}^{N} x_i \mu_{lmf}(x_i)}{\sum_{i=1}^{L} \mu_{umf}(x_i) + \sum_{i=L+1}^{N} \mu_{lmf}(x_i)}
\]

\[
C_R \approx \frac{\sum_{i=1}^{R} x_i \mu_{lmf}(x_i) + \sum_{i=R+1}^{N} x_i \mu_{umf}(x_i)}{\sum_{i=1}^{R} \mu_{lmf}(x_i) + \sum_{i=R+1}^{N} \mu_{umf}(x_i)}
\]

Here:

- \( N \) is the number of samples taken across the output variable range, specified using `evalfisOptions`.
- \( x_i \) is the \( i \)th output value sample.
- \( \mu_{umf} \) is the upper membership function.
- \( \mu_{lmf} \) is the lower membership function.
- \( L \) and \( R \) are switch points that are estimated by the various type-reduction methods.

For a list of supported methods, see “Type-Reduction Methods” on page 2-15.

For both Mamdani and Sugeno systems, the final defuzzified output value (\( y \)) is the average of the two centroid values from the type reduction process.
\[ y = \frac{c_L + c_R}{2} \]

**Type-Reduction Methods**

Fuzzy Logic Toolbox software supports four built-in type-reduction methods. These algorithms differ in their initialization methods, assumptions, computational efficiency, and terminating conditions.

To set the type-reduction method for a type-2 fuzzy system, set the `TypeReduction` property of the `mamfistype2` or `sugfistype2` object.

<table>
<thead>
<tr>
<th>Method</th>
<th>TypeReduction property Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karnik-Mendel (KM) [2]</td>
<td>&quot;karnikmendel&quot;</td>
<td>First type-reduction method developed</td>
</tr>
<tr>
<td>Enhanced Karnik-Mendel (EKM) [3]</td>
<td>&quot;ekm&quot;</td>
<td>Modification of the Karnik-Mendel algorithm with an improved initialization, modified termination condition, and improved computational efficiency</td>
</tr>
<tr>
<td>Iterative algorithm with stop condition (IASC) [4]</td>
<td>&quot;iasc&quot;</td>
<td>Iterative improvement to brute force methods</td>
</tr>
<tr>
<td>Enhanced iterative algorithm with stop condition (EIASC) [5]</td>
<td>&quot;eiasc&quot;</td>
<td>Improved version of the IASC algorithm</td>
</tr>
</tbody>
</table>

In general, the computational efficiency of these methods improve as you move down the table.

You can also use your own custom type-reduction method. For more information, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.
References


See Also
mamfistype2 | sugfistype2

More About

- “Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2
- “Fuzzy PID Control with Type-2 FIS” on page 2-78
- “Predict Chaotic Time Series Using Type-2 FIS” on page 3-62
Build Fuzzy Systems Using Fuzzy Logic Designer

Fuzzy Logic Toolbox Graphical User Interface Tools

This example shows how to build a fuzzy inference system (FIS) for the tipping example, described in “The Basic Tipping Problem” on page 2-19, using the Fuzzy Logic Toolbox UI tools. These tools support only type-1 fuzzy systems.

You use the following tools to build, edit, and view fuzzy inference systems:

• **Fuzzy Logic Designer** to handle the high-level issues for the system — How many input and output variables? What are their names?

  Fuzzy Logic Toolbox software 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 tools.

• **Membership Function Editor** on page 2-25 to define the shapes of all the membership functions associated with each variable

• **Rule Editor** on page 2-33 to edit the list of rules that defines the behavior of the system.

• **Rule Viewer** on page 2-36 to view the fuzzy inference diagram. Use this viewer as a diagnostic to see, for example, which rules are active, or how individual membership function shapes influence the results

• **Surface Viewer** on page 2-38 to view 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.

These UIs are dynamically linked, in that changes you make to the FIS using one of them, affect what you see on any of the other open UIs. For example, if you change the names of the membership functions in the Membership Function Editor, the changes are reflected in the rules shown in the Rule Editor. You can use the UIs to read and write variables both to the MATLAB workspace and to a file (the read-only viewers can still exchange plots with the workspace and save them to a file). You can have any or all of them open for any given system or have multiple editors open for any number of fuzzy systems.
The following figure shows how the main components of a FIS and the three editors fit together. The two viewers examine the behavior of the entire system.
In addition to these five primary UIs, the toolbox includes the graphical **Neuro-Fuzzy Designer**, which you use to build and analyze Sugeno-type adaptive neuro-fuzzy inference systems.

The Fuzzy Logic Toolbox UIs do not support building a FIS using data. If you want to use data to build a FIS, use one of the following techniques:

- **genfis** to generate a Sugeno-type FIS. Then, select **File > Import** in the **Fuzzy Logic Designer** to import the FIS and perform fuzzy inference, as described in “The Fuzzy Logic Designer” on page 2-20.
- Neuro-adaptive learning techniques to model the FIS, as described in “Neuro-Adaptive Learning and ANFIS” on page 3-92.

If you want to use MATLAB workspace variables, use the command-line interface instead of the **Fuzzy Logic Designer**. For an example, see “Build Fuzzy Systems at the Command Line” on page 2-41.

**The Basic Tipping Problem**

This example creates a Mamdani fuzzy inference system using a two-input, one-output tipping problem based on tipping practices in the U.S. While the example creates a Mamdani FIS, the methods used apply to creating Sugeno systems as well.

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:

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%.
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, use the UI tools to construct a fuzzy inference system for this decision process.

**The Fuzzy Logic Designer**

The **Fuzzy Logic Designer** displays information about a fuzzy inference system. To open the **Fuzzy Logic Designer**, type the following command at the MATLAB prompt:

```
fuzzyLogicDesigner
```

The **Fuzzy Logic Designer** opens and displays a diagram of the fuzzy inference system with the names of each input variable on the left, and those of each output variable on the right, as shown in the next figure. The sample membership functions shown in the boxes are just icons and do not depict the actual shapes of the membership functions.
Double-click the input variable icon to open the Membership Function Editor

Double-click the output variable icon to open the Membership Function Editor

Double-click the system diagram to open the Rule Editor

Double-click the output variable icon to open the Membership Function Editor

Below the diagram is the name of the system and the type of inference used.
In this example, you use the default Mamdani-type inference. Another type of inference, called Sugeno-type inference, is also available. For more information, see “Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2.

In the **Fuzzy Logic Designer**: 

- The drop-down lists let you modify the fuzzy inference functions.
- The **Current Variable** area displays the name of either an input or output variable, its type, and default range.
- A status line at the bottom displays information about the most recent operation.

To build the fuzzy inference system described in “The Basic Tipping Problem” on page 2-19 from scratch, type the following command at the MATLAB prompt:

```matlab
fuzzyLogicDesigner
```

The generic untitled **Fuzzy Logic Designer** opens, with one input `input1`, and one output `output1`. 
Tip To open the Fuzzy Logic Designer with the prebuilt fuzzy inference system stored in tipper.fis, enter

```matlab
fuzzyLogicDesigner('tipper.fis')
```

However, if you load the prebuilt system, you will not build rules or construct membership functions.

In this example, you construct a two-input, one output system. The two inputs are service and food. The one output is tip.

To add a second input variable and change the variable names to reflect these designations:

1. Select Edit > Add variable > Input.
A second yellow box labeled input2 appears.

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

**The Membership Function Editor**

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. The Membership Function Editor shares some features with the **Fuzzy Logic Designer**, as shown in the next figure. In fact, all of the five basic UI tools have similar menu options, status lines, and **Help** and **Close** buttons.
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 Fuzzy Logic Designer.

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 membership functions for the two-input tipping example, tipper, is as follows:

1. Double-click the input variable service to open the Membership Function Editor.

![Membership Function Editor for tipper example](image)
2 In the Membership Function Editor, enter $[0 \ 10]$ in the **Range** and the **Display Range** fields.

3 Create membership functions for the input variable **service**.

   a Select **Edit > Remove All MFs** to remove the default membership functions for the input variable **service**.

   b Select **Edit > Add MFs** to open the Membership Functions dialog box.

   c In the Membership Functions dialog box, select **gaussmf** as the **MF Type**.

   ![Membership Functions Dialog Box](image)

   d Verify that 3 is selected as the **Number of MFs**.

   e Click **OK** to add three Gaussian curves to the input variable **service**.

4 Rename the membership functions for the input variable **service**, and specify their parameters.

   a Click on the curve named **mf1** to select it, and specify the following fields in the **Current Membership Function (click on MF to select)** area:

      • In the **Name** field, enter **poor**.
      • In the **Params** field, enter $[1.5 \ 0]$.

      The two inputs of **Params** represent the standard deviation and center for the Gaussian curve.

      **Tip** To adjust the shape of the membership function, type in the desired parameters or use the mouse, as described previously.

   b Click on the curve named **mf2** to select it, and specify the following fields in the **Current Membership Function (click on MF to select)** area:

      • In the **Name** field, enter **good**.
• In the **Params** field, enter \([1.5 \ 5]\).

**c** Click on the curve named \(mf3\), and specify the following fields in the **Current Membership Function (click on MF to select)** area:

• In the **Name** field, enter excellent.
• In the **Params** field, enter \([1.5 \ 10]\).

The Membership Function Editor window looks similar to the following figure.

5 In the **FIS Variables** area, click the input variable \(food\) to select it.
6 Enter \([0 \ 10]\) in the **Range** and the **Display Range** fields.
7 Create the membership functions for the input variable \(food\).

   a Select **Edit > Remove All MFs** to remove the default Membership Functions for the input variable \(food\).
b Select Edit > Add MFs to open the Membership Functions dialog box.

c In the Membership Functions dialog box, select trapmf as the MF Type.

d Select 2 in the Number of MFs drop-down list.

e Click OK to add two trapezoidal curves to the input variable food.

8 Rename the membership functions for the input variable food, and specify their parameters:

a In the FIS Variables area, click the input variable food to select it.

b Click on the curve named mf1, and specify the following fields in the Current Membership Function (click on MF to select) area:

• In the Name field, enter rancid.
• In the Params field, enter [0 0 1 3].

c Click on the curve named mf2 to select it, and enter delicious in the Name field.

Reset the associated parameters if desired.

9 Click on the output variable tip to select it.

10 Enter [0 30] in the Range and the Display Range fields to cover the output range.

The inputs ranges from 0 to 10, but the output is a tip between 5% and 25%.

11 Rename the default triangular membership functions for the output variable tip, and specify their parameters.

a Click the curve named mf1 to select it, and specify the following fields in the Current Membership Function (click on MF to select) area:

• In the Name field, enter cheap.
• In the Params field, enter [0 5 10].

b Click the curve named mf2 to select it, and specify the following fields in the Current Membership Function (click on MF to select) area:

• In the Name field, enter average.
• In the Params field, enter [10 15 20].

c Click the curve named mf3 to select it, and specify the following:

• In the Name field, enter generous.
• In the **Params** field, enter \([20 \ 25 \ 30]\).

The Membership Function Editor looks similar to the following figure.

![Membership Function Editor](image)

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.
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 Fuzzy Logic Designer, the Rule Editor allows you to construct the rule statements automatically. You can:
Create rules by selecting an item in each input and output variable box, selecting 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 **Fuzzy Logic Designer** 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 UI 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 UI 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**.

Then, click **Add rule**.

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)
Tip 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 UI 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 UI 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 the five basic Fuzzy Logic Toolbox UI tools. To open the Surface Viewer, select **Surface** from the **View** menu.

**The Surface Viewer**

![Surface Viewer](image)
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.

By default, the surface plot updates automatically when you change the input or output variable selections or the number of grid points. To disable automatic plot updates, in the Options menu, clear the Always evaluate option. When this option is disabled, to update the plot, click Evaluate.

If you want to create a smoother plot, use the Plot points field to specify the number of points on which the membership functions are evaluated in the input or output range. This field defaults to the minimum number of plot plots, 101. If you specify fewer plot points, the field value automatically resets to 101. When you specify the number of plot points, the surface plot automatically updates.

By clicking on the plot axes and dragging the mouse, you can manipulate the surface so that you can view it from different angles.

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 NaNs holding the place of the varying inputs while numerical values indicates those values that remain fixed.

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

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. Do not manually edit the contents of a .fis file. Doing so can produce unexpected results when loading the file. When you save your fuzzy system to the MATLAB workspace, you are creating a variable that acts as a MATLAB object for the fuzzy system.

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

**See Also**
Fuzzy Logic Designer

**More About**
- “Build Fuzzy Systems at the Command Line” on page 2-41
- “Simulate Fuzzy Inference Systems in Simulink” on page 5-2
Build Fuzzy Systems at the Command Line

You can construct a fuzzy inference system (FIS) at the MATLAB® command line. This method is an alternative to interactively designing your FIS using Fuzzy Logic Designer.

This example shows you how to create a Mamdani fuzzy inference system. While you create a Mamdani FIS, the methods used apply to creating Sugeno systems as well.

**Tipping Problem at the Command Line**

To demonstrate the command-line functionality for creating and viewing fuzzy inference systems, this example uses the tipper FIS.

```matlab
fis = readfis('tipper.fis');
```

This command returns a `mamfis` object that contains the properties of the fuzzy system. For a Sugeno system, this command returns a `sugfis` object.

You can access the FIS properties using dot notation. For example, view the inputs of the fuzzy system.

```matlab
fis.Inputs
```

```matlab
ans = 1x2 fisvar array with properties:
        Name     Range     MembershipFunctions
    ___________________    ___________________
          _______    _______    ___________________
        Name       Range       MembershipFunctions
    _______    _______    ___________________
    "service"    0    10{1x3 fismf}
    "food"      0    10{1x2 fismf}
```

To set the properties of your fuzzy system, use dot notation. For example, set the name of the FIS.

```matlab
fis.Name = "gratuity";
```
FIS Object

You represent fuzzy inference systems using `mamfis` and `sugfis` objects. These objects contain all the fuzzy inference system information, including the variable names, membership function definitions, and fuzzy inference methods. Each FIS is itself a hierarchy of objects. The following objects are used within a fuzzy system:

- `fisvar` objects represent both input and output variables.
- `fismf` objects represent membership functions within each input and output variable.
- `fisrule` objects represent fuzzy rules that map inputs to outputs.

View all the information for a FIS by directly listing its properties.

```matlab
fis
```

```matlab
fis =
    mamfis with properties:
          Name: "gratuity"
        AndMethod: "min"
         OrMethod: "max"
   ImplicationMethod: "min"
     AggregationMethod: "max"
DefuzzificationMethod: "centroid"
    Inputs: [1x2 fisvar]
   Outputs: [1x1 fisvar]
      Rules: [1x3 fisrule]
DisableStructuralChecks: 0

See 'getTunableSettings' method for parameter optimization.
```

You can view the properties of the objects within a FIS object using dot notation. For example, view the `fisvar` object for first input variable.

```matlab
fis.Inputs(1)
```

```matlab
ans =
    fisvar with properties:
            Name: "service"
          Range: [0 10]
MembershipFunctions: [1x3 fismf]
```
Also, view the membership functions for this variable.

\[
\text{fis.Inputs(1).MembershipFunctions}
\]

\[
\text{ans} = \\
\text{1x3 fismf array with properties:}
\]

\[
\begin{array}{ccc}
\text{Type} & \text{Parameters} & \text{Name} \\
\text{gaussmf} & 1.5 & \text{poor} \\
\text{gaussmf} & 1.5 & \text{good} \\
\text{gaussmf} & 1.5 & \text{excellent}
\end{array}
\]

\textbf{System Display Functions}

To get a high-level view of your fuzzy system from the command line, use the \texttt{plotfis}, \texttt{plotmf}, and \texttt{gensurf} functions. \texttt{plotfis} displays the whole system as a block diagram, as shown in the \textbf{Fuzzy Logic Designer}.

\texttt{plotfis(fis)}
The `plotmf` function plots all the membership functions associated with a given variable. For example, view the membership functions for the first input variable.

`plotmf(fis,'input',1)`
Similarly, to view the membership functions for the first output, type:

```matlab
plotmf(fis,'output',1)
```
plotmf does not support viewing the output membership functions for Sugeno systems.

To view the rules of the fuzzy system, type:

fis.Rules

ans =
1x3 fisrule array with properties:

    Description    Antecedent    Consequent    Weight    Connection

2-46
The `gensurf` function plots the output of the FIS for any one or two input variables.

`gensurf(fis)`
Build Fuzzy Inference System

As an alternative to using the Fuzzy Logic Designer app, you can construct a FIS entirely from the command line.

First, create a Mamdani FIS, specifying its name.

```matlab
fis = mamfis('Name','tipper');
```

Add the first input variable for the service quality using `addInput`.

```matlab
fis = addInput(fis,[0 10],'Name','service');
```

Add membership functions for each of the service quality levels using `addMF`. In this case, use Gaussian membership functions. For more information on Gaussian membership function properties, see `gaussmf`.

```matlab
fis = addMF(fis,"service","gaussmf",[1.5 0],\'Name\',\'poor\');
fis = addMF(fis,"service","gaussmf",[1.5 5],\'Name\',\'good\');
fis = addMF(fis,"service","gaussmf",[1.5 10],\'Name\',\'excellent\');
```

Add the second input variable for the food quality, and add two trapezoidal membership functions. For information on trapezoidal membership functions, see `trapmf`.

```matlab
fis = addInput(fis,[0 10],\'Name\',\'food\');
fis = addMF(fis,"food","trapmf",[-2 0 1 3],\'Name\',\'rancid\');
fis = addMF(fis,"food","trapmf",[7 9 10 12],\'Name\',\'delicious\');
```

Add the output variable for the tip, and add three triangular membership functions. For more information on the triangular membership function, see `trimf`.

```matlab
fis = addOutput(fis,[0 30],\'Name\',\'tip\');
fis = addMF(fis,"tip","trimf",[0 5 10],\'Name\',\'cheap\');
fis = addMF(fis,"tip","trimf",[10 15 20],\'Name\',\'average\');
fis = addMF(fis,"tip","trimf",[20 25 30],\'Name\',\'generous\');
```

Specify the following three rules for the FIS as a numeric array:

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

Each row of the array contains one rule in the following format.
• Column 1 - Index of membership function for first input  
• Column 2 - Index of membership function for second input  
• Column 3 - Index of membership function for output  
• Column 4 - Rule weight (from 0 to 1)  
• Column 5 - Fuzzy operator (1 for AND, 2 for OR)

For the membership function indices, indicate a NOT condition using a negative value.  
For more information on fuzzy rule specification, see `addRule`.

```matlab
ruleList = [1 1 1 1 2;  
             2 0 2 1 1;  
             3 2 3 1 2];
```

Add the rules to the FIS.

```matlab
fis = addRule(fis, ruleList);
```

Alternatively, you can create the fuzzy inference system using a combination of dot notation and `fisvar`, `fismf`, and `fisrule` objects. This method is not a good practice for most applications. However, you can use this approach when your application requires greater flexibility in constructing and modifying your FIS.

Create the fuzzy inference system.

```matlab
fis = mamfis('Name', 'tipper');
```

Add and configure the first input variable. In this case, create a default `fisvar` object and specify its properties using dot notation.

```matlab
fis.Inputs(1) = fisvar;  
fis.Inputs(1).Name = "service";  
fis.Inputs(1).Range = [0 10];
```

Define the membership functions for the first input variable. For each MF, create a `fismf` object, and set the properties using dot notation.

```matlab
fis.Inputs(1).MembershipFunctions(1) = fismf;  
fis.Inputs(1).MembershipFunctions(1).Name = "poor";  
fis.Inputs(1).MembershipFunctions(1).Type = "gaussmf";  
fis.Inputs(1).MembershipFunctions(1).Parameters = [1.5 0];  
fis.Inputs(1).MembershipFunctions(2) = fismf;  
fis.Inputs(1).MembershipFunctions(2).Name = "good";  
fis.Inputs(1).MembershipFunctions(2).Type = "gaussmf";
```
fis.Inputs(1).MembershipFunctions(2).Parameters = [1.5 5];
fis.Inputs(1).MembershipFunctions(3) = fismf;
fis.Inputs(1).MembershipFunctions(3).Name = "excellent";
fis.Inputs(1).MembershipFunctions(3).Type = "gaussmf";
fis.Inputs(1).MembershipFunctions(3).Parameters = [1.5 10];

Add and configure the second input variable. For this variable, specify the name and range when you create the fisvar object.

fis.Inputs(2) = fisvar([0 10], 'Name', "food");

Specify the membership functions for the second input. For each MF, specify the name, type, and parameters when you create the fismf object.

fis.Inputs(2).MembershipFunctions(1) = fismf("trapmf", [-2 0 1 3],... 'Name', "rancid");
fis.Inputs(2).MembershipFunctions(2) = fismf("trapmf", [7 9 10 12],... 'Name', "delicious");

Similarly, add and configure the output variable and its membership functions.

fis.Outputs(1) = fisvar([0 30], 'Name', "tip");

In this case, specify the output membership functions using a vector of fismf objects.

mf1 = fismf("trimf", [0 5 10], 'Name', "cheap");
mf2 = fismf("trimf", [10 15 20], 'Name', "average");
mf3 = fismf("trimf", [20 25 30], 'Name', "generous");
fis.Outputs(1).MembershipFunctions = [mf1 mf2 mf3];

Create the rules for the fuzzy system. For each rule create a fisrule object. Then, specify the rules using a vector of these objects. When creating a fisrule object using numeric values, you must specify the number of inputs variables.

rule1 = fisrule([1 1 1 1 2],2);
rule2 = fisrule([2 0 2 1 1],2);
rule3 = fisrule([3 2 3 2 1],2);
rules = [rule1 rule2 rule3];

Before adding your rules to your fuzzy system, you must update them using the data in the FIS object. Update the rules using the update function, and add them the fuzzy system.

rules = update(rules, fis);
fis.Rules = rules;
When constructing your fuzzy system, you can also specify custom membership functions and inference functions. For more information, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

**Evaluate Fuzzy Inference System**

To evaluate the output of a fuzzy system for a given input combination, use the `evalfis` command. For example, evaluate `fis` using input variable values of 1 and 2.

```
evalfis(fis,[1 2])
ans =  5.5586
```

You can also evaluate multiple input combinations using an array where each row represents one input combination.

```
inputs = [3 5;  2 7;  3 1];
evalfis(fis,inputs)
ans = 3x1
    12.2184
     7.7885
     8.9547
```

**See Also**

`evalfis` | `gensurf` | `mamfis` | `plotfis` | `plotmf` | `sugfis`

**More About**

- “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17
Build Fuzzy Systems Using Custom Functions

Build Fuzzy Inference Systems Using Custom Functions in Fuzzy Logic Designer

When you build a fuzzy inference system, as described in “Fuzzy Inference Process” on page 1-28, you can replace the built-in membership functions, inference functions, or both with custom functions. In this section, you learn how to build a fuzzy inference system using custom functions in the Fuzzy Logic Designer app.

To build a fuzzy inference system using custom functions in the Fuzzy Logic Designer app:

1. Open Fuzzy Logic Designer. At the MATLAB command line, type:
   ```matlab
   fuzzyLogicDesigner
   ```

2. Specify the number of inputs and outputs of the fuzzy system, as described in “The Fuzzy Logic Designer” on page 2-20.

3. Create custom membership functions, and replace the built-in membership functions with them, as described in “Specify Custom Membership Functions” on page 2-53.

   Membership functions define how each point in the input space is mapped to a membership value between 0 and 1.

4. Create rules using the Rule Editor, as described in “The Rule Editor” on page 2-33.

   Rules define the logical relationship between the inputs and the outputs.

5. Create custom inference functions, and replace the built-in inference functions with them, as described in “Specify Custom Inference Functions” on page 2-59.

   Inference methods include the AND, OR, implication, aggregation, and defuzzification methods. This action generates the output values for the fuzzy system.

   The next figure shows the tipping problem example where the built-in Implication, Aggregation and Defuzzification functions are replaced with the custom functions, customimp, customagg, and customdefuzz, respectively.
Select **View > Surface** to view the output of the fuzzy inference system in the Surface Viewer, as described in “The Surface Viewer” on page 2-38.

### Specify Custom Membership Functions

You can create custom membership functions and use them in the fuzzy inference process. The values of these functions must lie between 0 and 1. For more information on the properties of membership functions, see “Membership Functions” on page 1-14.

To create a custom membership function, and replace the built-in membership function:

1. Create a MATLAB function, and save it in your current working folder.

   To learn how to create MATLAB functions, see “Scripts vs. Functions” (MATLAB).
The following code is an example of a multistep custom membership function, custmf1, that depends on eight parameters between 0 and 10.

```matlab
% Function to generate a multi-step custom membership function
% using 8 parameters for the input argument x
function out = custmf1(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 = 0.1*y'; % Scale the output to lie between 0 and 1.
end
```

2 Open the Fuzzy Logic Designer app.

fuzzyLogicDesigner

The Fuzzy Logic Designer opens with the default FIS name, Untitled, and contains one input, input1, and one output, output1.

3 In the Fuzzy Logic Designer, select Edit > Membership Functions to open the Membership Function Editor.

Three triangular-shaped membership functions for input1 are displayed by default.
To replace the default membership function with a custom function in the Membership Function Editor:

a. Select **Edit > Remove All MFs** to remove the default membership functions for **input1**.

b. Select **Edit > Add Custom MF** to open the Custom Membership Function dialog box.
To specify a custom function, in the Custom Membership Function dialog box:

a  In the **MF name** field, specify a name for the custom membership function.

**Note** When adding additional custom membership functions, specify a different **MF name** for each function.

b  In the **M-file function name** field, specify the name of the custom membership function file.

c  In the **Parameter list**, specify a vector of parameters.

These values determine the shape and position of the membership function, and the function is evaluated using these parameter values.

**Note** The length of the parameter vector must be greater than or equal to the number of parameters in the custom membership function.

Using the **custmf1** example in step 1, the Custom Membership Function dialog box looks similar to the following figure.
d  Click OK to add the custom membership function.

e  Specify both the **Range** and **Display Range** to be \([0 \ 10]\) to match the range of the custom membership function.

The Membership Function Editor displays the custom membership function plot.
This action also adds the custom membership function to the Rule Viewer, and makes it available for creating rules for the fuzzy inference process. To view the custom function in the Rule Viewer, select **Edit > Rules** in either the **Fuzzy Logic Designer** or the Membership Function Editor.
To add custom membership functions for output1, select it in the Membership Function Editor, and repeat steps 4 and 5.

You can also add a custom membership function to a FIS at the MATLAB command line. For example, to add custmf1 to the first input variable, input1 of the FIS, myFIS, and name it customMF1, type the following:

```matlab
myFIS = addMF(myFIS,"input1","custmf1",[0 1 2 4 6 8 9 10],"Name","customMF1");
```

**Specify Custom Inference Functions**

You can replace the built-in AND, OR, implication, aggregation, and defuzzification inference methods with custom functions. After you create the custom inference function, save it in your current working folder. To learn how to build fuzzy systems using custom inference functions, see the "Build Fuzzy Inference Systems Using Custom Functions in Fuzzy Logic Designer" on page 2-52 section.
The guidelines for creating and specifying the functions for building fuzzy inference systems are described in the following sections.

- “Create Custom AND and OR Functions” on page 2-60
- “Create Custom Implication Functions” on page 2-61
- “Create Custom Aggregation Functions” on page 2-62
- “Create Custom Defuzzification Functions” on page 2-63
- “Steps for Specifying Custom Inference Functions” on page 2-63

**Create Custom AND and OR Functions**

The custom AND and OR inference functions must operate column-wise on a matrix, in the same way as the MATLAB functions `max`, `min`, or `prod`.

For a row or column vector `x`, `min(x)` returns the minimum element.

```matlab
x = [1 2 3 4];
min(x)
```

```matlab
ans =
    1
```

For a matrix `x`, `min(x)` returns a row vector containing the minimum element from each column.

```matlab
x = [1 2 3 4; 5 6 7 8; 9 10 11 12];
min(x)
```

```matlab
ans =
    1   2   3   4
```

For N-D arrays, `min(x)` operates along the first non-singleton dimension.

The function `min(x,y)` returns an array that is same size as `x` and `y` with the minimum elements from `x` or `y`. Either of the input arguments can be a scalar. Functions such as `max`, and `prod` operate in a similar manner.

In the toolbox, the AND implication methods perform an element by element matrix operation, similar to the MATLAB function `min(x,y)`.

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

The OR implication methods perform an element by element matrix operation, similar to the MATLAB function max(x, y).

Create Custom Implication Functions

Custom implication functions must operate in the same way as the MATLAB functions max, min, or prod. Your custom implication function must be a T-norm fuzzy intersection operation. For more information, see “Additional Fuzzy Operators” on page 1-21.

An implication function must support either one or two inputs because the software calls the function in two ways:

- To calculate the output fuzzy set values using the firing strength of all the rules and the corresponding output membership functions. In this case, the software calls the implication function using two inputs, similar to the following example:

  impvals = customimp(w, outputmf)

  - \( w \) — Firing strength of multiple rules, specified as an \( nr \)-by-\( ns \) matrix. Here, \( nr \) is the number of rules and \( ns \) is the number of samples of the output membership functions.

    \( w(:,j) = w(:,1) \) for all \( j \). \( w(i,1) \) is the firing strength of the \( i \)th rule.

  - \( \text{outputmf} \) — Output membership function values, specified as an \( nr \)-by-\( ns \) matrix. Here, \( nr \) is the number of rules and \( ns \) is the number of samples of the output membership functions.

    \( \text{outputmf}(i,:) \) contains the data of the \( i \)th output membership function.

- To calculate the output fuzzy value using the firing strength of a single rule and the corresponding output membership function, for a given sample. In this case, the software calls the implication function using one input, similar to the following example:

  impval = customimp([w outputmf])

  \( w \) and \( \text{outputmf} \) are scalar values representing the firing strength of a rule and the corresponding output membership function value, for a given sample.
The following is an example of a bounded product custom implication function with binary mapping $T(a, b) = \max\{0, a + b - 1\}$. [1]

```
function y = customimp(x1,x2)

if nargin == 1
    % x1 assumed to be non-empty column vector or matrix.
    minVal = zeros(1,size(x1,2));
    y = ones(1,size(x1,2));

    for i = 1:size(x1,1)
        y = max(minVal,sum([y;x1(i,:)]-1));
    end
else
    % x1 and x2 assumed to be non-empty matrices.
    minVal = zeros(1,size(x1,2));
    y = zeros(size(x1));

    for i = 1:size(x1,1)
        y(i,:) = max(minVal,sum([x1(i,:);x2(i,:)]-1));
    end
end
end
```

**Note** Custom implication functions are not supported for Sugeno-type systems.

### Create Custom Aggregation Functions

The custom aggregation functions must operate in the same way as the MATLAB functions `max`, `min`, or `prod` and must be of the form $y = \text{customagg}(x)$. Your custom implication function must be a $T$-conorm (S-norm) fuzzy intersection operation. For more information, see “Additional Fuzzy Operators” on page 1-21.

$x$ is an $nv$-by-$nr$ matrix, which is the list of truncated output functions returned by the implication method for each rule. $nv$ is the number of output variables, and $nr$ is the number of rules. The output of the aggregation method is one fuzzy set for each output variable.

The following is an example of a bounded sum custom aggregation function with binary mapping $S(a, b) = \min\{a + b, 1\}$. [1]
function y = customagg(x)

maxVal = ones(1,size(x,2));
y = zeros(1,size(x,2));

for i = 1:size(x,1)
    y1muxVal, sum([y;x(i,:)]));
end
end

**Note** Custom aggregation functions are not supported for Sugeno-type systems.

**Create Custom Defuzzification Functions**

The custom defuzzification functions must be of the form \( y = \text{customdefuzz}(x, ymf) \), where \( x \) is the vector of values in the membership function input range, and \( ymf \) contains the values of the membership function for each \( x \) value.

The following is an example of a custom defuzzification function:

```matlab
function defuzzfun = customdefuzz(x,ymf)
    total_area = sum(ymf);
defuzzfun = sum(ymf.*x)/total_area;
end
```

**Note** Custom defuzzification functions are not supported for Sugeno-type systems.

**Steps for Specifying Custom Inference Functions**

After you create and save a custom inference function, specify the function in the fuzzy inference system using the following steps:

1. In the lower-left panel of the **Fuzzy Logic Designer**, select **Custom** from the dropdown menu corresponding to the inference method for which you want to specify the custom function.
Doing so opens a dialog box where you specify the name of the custom inference function.

2. In the **Method name** field, specify the name of the custom inference function, and click **OK**.
The custom function replaces the built-in function when building the fuzzy inference system.

**Note** In order to specify a custom inference function, you must first add at least one rule to your FIS.

3 To specify custom functions for other inference methods, repeat steps 1 and 2.

You can also specify custom inference functions for a FIS (myFIS) at the MATLAB command line. For example, to add a custom:

- Defuzzification method, type
  ```
  myFIS.DefuzzificationMethod = "customdefuzz";
  ```
  where customdefuzz is the name of the custom defuzzification function.

- Implication method, type
  ```
  myFIS.ImplicationMethod = "customimp";
  ```
  where customimp is the name of the custom implication function.

- Aggregation method, type
  ```
  myFIS.AggregationMethod = "customagg";
  ```
  where customagg is the name of the custom aggregation function.

### Specify Custom Type-Reduction Functions

For type-2 fuzzy inference systems, you can specify a custom type-reduction function. This function must be of the form \( y = \text{customtr}(x, \text{umf}, \text{lmf}) \), where \( x \) is the vector of values in the membership function input range. \( \text{umf} \) and \( \text{lmf} \) are the respective values of the upper and lower membership function for each \( x \) value. The output \( y \) is a two-element row vector of centroids \([c_L, c_R]\).

For more information on type reduction, see “Type-2 Fuzzy Inference Systems” on page 2-7.

By default, type-2 Sugeno systems support only a weighted average form of type reduction. The following custom type-reduction function implements a weighted sum form of type reduction for a Sugeno system.
function y = customtr(x,umf,ymf)
    y = zeros(1,2);
    y(1) = sum(x.*umf);
    y(2) = sum(x.*lmf);
end

To specify the custom type-reduction function for a FIS (myFIS) at the MATLAB command line, type

    myFIS.DTypeReductionMethod = "customtr";

where customtr is the name of the custom defuzzification function.

**Use Custom Functions in Code Generation**

You can use custom functions in fuzzy inference systems for which you generate code. For more information on code generation for fuzzy systems, see “Deploy Fuzzy Inference Systems” on page 6-2.

If you use a nondouble data type for your generated code, you must propagate the data type from the input arguments of your custom function to the output argument. For example, the following custom aggregation function maintains the data type of x in y using the ones and zeros with the 'like' argument.

    function y = customagg(x)
        maxVal = ones(1,size(x,2),'like',x);
        y = zeros(1,size(x,2),'like',x);
        for i = 1:size(x,1)
            y = min(maxVal,sum([y;x(i,:)]));
        end
    end

For more information on writing functions that support C/C++ code generation, see “MATLAB Programming for Code Generation” (MATLAB Coder).
References


See Also

*Fuzzy Logic Designer*

Related Examples

- “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17
- “Build Fuzzy Systems at the Command Line” on page 2-41
Fuzzy Trees

As the number of inputs to a fuzzy system increases, the number of rules increases exponentially. This large rule base reduces the computational efficiency of the fuzzy system. It also makes the operation of the fuzzy system harder to understand, and it makes the tuning of rule and membership function parameters more difficult. Because many applications have a limited amounts of training data, a large rule base reduces the generalizability of tuned fuzzy systems.

To overcome this issue, you can implement a fuzzy inference system (FIS) as a tree of smaller interconnected FIS objects rather than as a single monolithic FIS object. These fuzzy trees are also known as hierarchical fuzzy systems because the fuzzy systems are arranged in hierarchical tree structures. In a tree structure, the outputs of the low-level fuzzy systems are used as inputs to the high-level fuzzy systems. A fuzzy tree is more computationally efficient and easier to understand than a single FIS with the same number of inputs.

Types of Hierarchical Structures

There are several fuzzy tree structures that you can use for your application. The following figure shows commonly used fuzzy tree structures: an incremental, aggregated, or cascaded structure.
Incremental Structure

In an incremental structure, input values are incorporated in multiple stages to refine the output values in several levels. For example, the previous figure shows a three-level incremental fuzzy tree having fuzzy inference systems $FIS_i^n$, where $i$ indicates the index of a FIS in the $n$th level. In an incremental fuzzy tree, $i = 1$, meaning that each level has only one fuzzy inference system. In the previous figure, the $j$th input of the $i$th FIS in the $n$th level is shown as input $x_{ij}^n$, whereas the $k$th output of the $i$th FIS in the $n$th level is shown as input $x_{ik}^n$. In the figure, $n = 3$, $j = 1$ or 2, and $k = 1$. If each input has $m$ membership functions (MFs), each FIS has a complete set of $m^2$ rules. Hence, the total number of rules is $nm^2 = 3 \cdot 3^2 = 27$.

The following figure shows a monolithic ($n = 1$) FIS with four inputs ($j = 1, 2, 3, 4$) and three MFs ($m = 3$).
In the FIS of this figure, the total number of rules is $nm^4 = 1 \times 3^4 = 81$. Hence, the total number of rules in an incremental fuzzy tree is linear with the number of input pairs.

Input selection at different levels in an incremental fuzzy tree uses input rankings based on their contributions to the final output values. The input values that contribute the most are generally used at the lowest level, while the least influential ones are used at the highest level. In other words, low-rank input values are dependent on high-rank input values.

In an incremental fuzzy tree, each input value usually contributes to the inference process to a certain extent, without being significantly correlated with the other inputs. For example, a fuzzy system forecasts the possibility of buying an automobile using four inputs: color, number of doors, horse power, and autopilot. The inputs are four distinct automobile features, which can independently influence a buyer’s decision. Hence, the inputs can be ranked using the existing data to construct a fuzzy tree, as shown in the following figure.
For an example that illustrates creating an incremental fuzzy tree in MATLAB, see the example Create Incremental FIS Tree on the `fistree` reference page.

**Aggregated Structure**

In an aggregated structure, input values are incorporated as groups at the lowest level, where each input group is fed into a FIS. The outputs of the lower level fuzzy systems are combined (aggregated) using the higher level fuzzy systems. For example, the following shows a two-level aggregated fuzzy tree having fuzzy inference systems $FIS_{i_n}$, where $i_n$ indicates the index of a FIS in the $n$th level.

In this aggregated fuzzy tree, $i_1 = 1, 2$ and $i_2 = 1$. Hence, each level includes a different number of FIS. The $j$th input of the $i_n$th FIS is shown in the figure as input $x_{i_nj}$, and the $k$th output of the $i_n$th FIS is shown as output $y_{i_nk}$. In the figure, $j = 1, 2$ and $k = 1$. In other words, each FIS has two inputs and one output. If each input has $m$ MFs, then each FIS has a complete set of $m^2$ rules. Hence, the total number of rules for the three fuzzy systems is $3 \cdot m^2 = 3 \cdot 32 = 27$, which is the same as an incremental FIS for a similar configuration.

In an aggregated fuzzy tree, input values are naturally grouped together for specific decision-making. For example, an autonomous robot navigation task combines obstacle avoidance and target reaching subtasks for collision-free navigation. To achieve the navigation task, the fuzzy tree can use four inputs: distance to the closest obstacle, angle of the closest obstacle, distance to the target, and angle of the target. Distances and angles are measured with respect to the current position and heading direction of the
robot. In this case, at the lowest level, the inputs naturally group as shown in the following figure: obstacle distance and obstacle angle (group 1) and target distance and target angle (group 2). Two fuzzy systems separately process individual group inputs and then another fuzzy system combines their outputs to produce a collision-free heading for the robot.

For an example that illustrates creating an aggregated fuzzy tree in MATLAB, see the example Create Aggregated FIS Tree on the `fistree` reference page.

**Variation on Aggregated Structure**

In a variation of the aggregated structure known as parallel structure [1], the outputs of the lowest-level fuzzy systems are directly summed to generate the final output value. The following figure shows an example of a parallel fuzzy tree, where outputs of fis1 and fis2 are summed to produce the final output.
The `fistree` object does not provide the summing node Σ. Therefore, you must add a custom aggregation method to evaluate a parallel fuzzy tree. For an example, see "Create and Evaluate Parallel FIS Tree" on the `fistree` reference page.

**Cascaded or Combined Structure**

A cascaded structure, also known as combined structure, combines both incremental and aggregated structures to construct a fuzzy tree. This structure is suitable for a system that includes both correlated and uncorrelated inputs. The tree groups the correlated inputs in an aggregated structure, and adds uncorrelated inputs in an incremental structure. The following figure shows an example of a cascaded tree structure, where the first four inputs are grouped pairwise in an aggregated structure and the fifth input is added in an incremental structure.
For example, consider the robot navigation task discussed in “Aggregated Structure” on page 2-71. Suppose that task includes another input, the previous heading of the robot, taken into account to prevent large changes in the robot heading. You can add this input using the incremental structure of the following diagram.

For an example that illustrates creating an aggregated fuzzy tree in MATLAB, see the example "Create Cascaded FIS Tree" on the fistree reference page.
Add or Remove FIS Tree Outputs

When you evaluate a fistree object, it returns results for only the open outputs, which are not connected to any FIS inputs in the fuzzy tree. You can optionally access other outputs in the tree. For instance, in the following diagram of an aggregated fuzzy tree, you might want to obtain the output of fis2 when you evaluate the tree.

You can add such outputs to a fistree object. You can also remove outputs, provided that the fuzzy tree always has at least one output. For an example, see "Update FIS Tree Outputs" on the fistree reference page.

Use the Same Value for Multiple inputs of FIS Tree

A fistree object allows using the same value for multiple inputs. For instance, in the following figure, input2 of fis1 and input 1 of fis2 use the same value during evaluation.
For an example showing how to construct a FIS tree in this way, see the example "Use Same Value for Multiple Inputs of a FIS Tree" on the `fistree` reference page.

**Update Fuzzy Inference Systems in FIS Tree**

You can add or remove individual FIS elements from a `fistree` object. When you do so, the software automatically updates the Connections, Inputs, and Outputs properties of the `fistree` object. For an example, see "Update Fuzzy Inference Systems in a FIS Tree" in the `fistree` reference page.

**Tune a Fuzzy Tree**

Once you have configured the internal connections in your fuzzy tree, the next step is to tune the parameters of the tree. For an example, see “Tune FIS Tree for Gas Mileage Prediction” on page 3-45.

**References**


**See Also**

`fistree`
More About

- “Tuning Fuzzy Inference Systems” on page 3-2
- “Tune FIS Tree for Gas Mileage Prediction” on page 3-45
Fuzzy PID Control with Type-2 FIS

This example compares a type-2 fuzzy PID controller with both a type-1 fuzzy PID controller and conventional PID controller. This example is adapted from [1].

Fuzzy PID Control

This example uses the following fuzzy logic controller (FLC) structure as described in [1]. The output of the controller \( u \) is found using the error \( e \) and the derivative of the error \( \dot{e} \). Using scaling factors \( C_e \) and \( C_d \), inputs \( e \) and \( \dot{e} \) are normalized to \( E \) and \( \Delta E \), respectively. The normalized ranges for both inputs are in the range \([-1,1]\). The fuzzy logic controller also produces a normalized output in the range \([-1,1]\). Additional scaling factors \( C_0 \) and \( C_1 \) map the fuzzy logic controller output \( U \) into \( u \).

\[
G(s) = Ce^{-Ls} \\
\frac{T}{Ts + 1}
\]

Here, \( C, L, \text{ and } T \) are the gain, time delay, and time constant, respectively.

The scaling factors \( C_d, C_0, \text{ and } C_1 \) are defined as follows, where \( \tau_c \) is the closed-loop time constant.

\[
C_d = \min\left(T, \frac{L}{2}\right) \times C_e
\]

\[
C_0 = \frac{1}{C \times C_e (\tau_c + \frac{L}{2})}
\]

\[
C_1 = \max\left(T, \frac{L}{2}\right) \times C_0
\]
The input scaling factor $C_e$ is:

$$
C_e \equiv \frac{1}{r(t_r) - y(t_r)}
$$

where $r(t_r)$ and $y(t_r)$ are the reference and system output values at time $t = t_r$. These values correspond to the nominal operating point of the system.

This example compares the performance of type-1 and type-2 Sugeno fuzzy inference systems (FISs) using the Fuzzy Logic Controller Simulink® block.

**Construct Type-1 FIS**

Create a type-1 FIS using `sugfis`.

```matlab
fis1 = sugfis;
```

Add input variables to the FIS.

```matlab
fis1 = addInput(fis1,[-1 1], 'Name', 'E');
fis1 = addInput(fis1,[-1 1], 'Name', 'delE');
```

Add three uniformly distributed overlapping triangular membership functions (MFs) to each input. The MF names stand for negative (N), zero (Z), and positive (P).

```matlab
fis1 = addMF(fis1,'E','trimf',[-2 -1 0], 'Name', 'N');
fis1 = addMF(fis1,'E','trimf',[-1 0 1], 'Name', 'Z');
fis1 = addMF(fis1,'E','trimf',[0 1 2], 'Name', 'P');
fis1 = addMF(fis1,'delE','trimf',[-2 -1 0], 'Name', 'N');
fis1 = addMF(fis1,'delE','trimf',[-1 0 1], 'Name', 'Z');
fis1 = addMF(fis1,'delE','trimf',[0 1 2], 'Name', 'P');
```

Plot the input membership functions.

```matlab
figure
subplot(1,2,1)
plotmf(fis1,'input',1)
title('Input 1')
subplot(1,2,2)
plotmf(fis1,'input',2)
title('Input 2')
```
Add the output variable to the FIS.

\[
fis1 = \text{addOutput}(fis1, [-1 1], 'Name', 'U');
\]

Add uniformly distributed constant functions to the output. The MF names stand for negative big (NB), negative medium (NM), zero (Z), positive medium (PM), and positive big (PB).

\[
fis1 = \text{addMF}(fis1, 'U', 'constant', -1, 'Name', 'NB');
fis1 = \text{addMF}(fis1, 'U', 'constant', -0.5, 'Name', 'NM');
fis1 = \text{addMF}(fis1, 'U', 'constant', 0, 'Name', 'Z');
fis1 = \text{addMF}(fis1, 'U', 'constant', 0.5, 'Name', 'PM');
fis1 = \text{addMF}(fis1, 'U', 'constant', 1, 'Name', 'PB');
\]
Add rules to the FIS. These rules create a proportional control surface.

rules = [...
    "E==N & delE==N => U=NB"; ...
    "E==Z & delE==N => U=NM"; ...
    "E==P & delE==N => U=Z"; ...
    "E==N & delE==Z => U=NM"; ...
    "E==Z & delE==Z => U=Z"; ...
    "E==P & delE==Z => U=PM"; ...
    "E==N & delE==P => U=Z"; ...
    "E==Z & delE==P => U=PM"; ...
    "E==P & delE==P => U=PB" ...
    ];
fis1 = addRule(fis1,rules);

Plot the control surface.

figure
gensurf(fis1)
title('Control surface of type-1 FIS')
**Construct Type-2 FIS**

Convert the type-1 FIS, `fis1`, to a type-2 FIS.

```matlab
fis2 = convertToType2(fis1);
```

The type-2 Sugeno system, `fis2`, uses type-2 membership functions for the input variables and type-1 membership functions for the output variables.

Define the footprint of uncertainty (FOU) for the input MFs as defined in [1]. To do so, set the lower MF scaling factor for each MF. For this example, set the lower MF lag values to 0.

```matlab
scale = [0.2 0.9 0.2;0.3 0.9 0.3];
for i = 1:length(fis2.Inputs)
```

2-82
for j = 1:length(fis2.Inputs(i).MembershipFunctions)
    fis2.Inputs(i).MembershipFunctions(j).LowerLag = 0;
    fis2.Inputs(i).MembershipFunctions(j).LowerScale = scale(i,j);
end
end

Plot the type-2 input membership functions.

figure
subplot(1,2,1)
plotmf(fis2,'input',1)
title('Input 1')
subplot(1,2,2)
plotmf(fis2,'input',2)
title('Input 2')
The FOU adds additional uncertainty to the FIS and produces a nonlinear control surface.

```matlab
figure
gensurf(fis2)
title('Control surface of type-2 FIS')
```
Conventional PID Controller

This example compares the fuzzy logic controller performance with that of the following conventional PID controller:

\[
\text{PID}(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{\tau_f s + 1}
\]

Here, \(K_p\) is proportional gain, \(K_i\) is integrator gain, \(K_d\) is derivative gain, and \(\tau_f\) is the derivative filter time constant.

**Configure Simulation**

Define the nominal plant model.
C = 0.5;
L = 0.5;
T = 0.5;
G = tf(C,[T 1],'Outputdelay',L);

Generate the conventional PID controller parameters using \texttt{pidtune}.

\texttt{pidController = pidtune(G,'pidf');}

In this example, the reference ($r$) is a step signal and $t_r = 0$, which results in $C_e = 1$ as follows.

\[ C_e = \frac{1}{r(t_r) - y(t_r)} = \frac{1}{1-0} = 1. \]

\[ C_e = 1; \]

To configure the simulation, use the following nominal controller parameters.

\[ \text{tauC} = 0.2; \]

\[ C_d = \min(T,L/2)*C_e; \]
\[ C_0 = 1/(C*C_e*(\text{tauC}+L/2)); \]
\[ C_1 = \max(T,L/2)*C_0; \]

To simulate the controllers, use the \texttt{comparepidcontrollers} Simulink model.

\texttt{model = 'comparepidcontrollers';}
\texttt{load_system(model)
Simulate Nominal Process

Simulate the model at the nominal operating conditions.

```matlab
out1 = sim(model);
```

Plot the step response of the system for all three controllers.

```matlab
plotOutput(out1, ['Nominal: C=' num2str(C), ', L=' num2str(L), ', T=' num2str(T)]
```
Obtain the step-response characteristics of the system for each controller.

```
stepResponseTable(out1)
```

```
ans=3x4 table

<table>
<thead>
<tr>
<th></th>
<th>Rise Time (sec)</th>
<th>Overshoot (%)</th>
<th>Settling Time (sec)</th>
<th>Integral of Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>0.62412</td>
<td>11.234</td>
<td>4.5564</td>
<td>1.04</td>
</tr>
<tr>
<td>Type-1 FLC</td>
<td>1.4267</td>
<td>0</td>
<td>4.1023</td>
<td>1.1522</td>
</tr>
<tr>
<td>Type-2 FLC</td>
<td>1.8662</td>
<td>0</td>
<td>5.129</td>
<td>1.282</td>
</tr>
</tbody>
</table>
```

For the nominal process:
• Both the type-1 and type-2 fuzzy logic controllers outperform the conventional PID controller in terms of overshoot.
• The conventional PID controller, performs better with respect to rise-time and integral of absolute error (IAE).
• The type-1 FLC performs better than the type-2 FLC in terms of rise-time, settling-time, and IAE.

**Simulate Modified Process**

Modify the plant model by increasing the gain, time delay, and time constant values as compared to the nominal process.

```matlab
C = 0.85;
L = 0.6;
T = 0.6;
G = tf(C,[T 1],'Outputdelay',L);
```

Simulate the model using the updated plant parameters.

```matlab
out2 = sim(model);
```

Plot the step response of the system for all three controllers.

```matlab
plotOutput(out2,['Modified 1: C=' num2str(C) ',L=' num2str(L) ',T=' num2str(T)]
```
Obtain the step-response characteristics of the system for each controller.

\[
\text{stepResponseTable(out2)}
\]

\[
\text{ans=3×4 table}
\]

<table>
<thead>
<tr>
<th>Controller</th>
<th>Rise Time (sec)</th>
<th>Overshoot (%)</th>
<th>Settling Time (sec)</th>
<th>Integral of Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>0.38464</td>
<td>80.641</td>
<td>29.452</td>
<td>4.7486</td>
</tr>
<tr>
<td>Type-1 FLC</td>
<td>0.47262</td>
<td>24.877</td>
<td>4.6788</td>
<td>1.1137</td>
</tr>
<tr>
<td>Type-2 FLC</td>
<td>0.47262</td>
<td>22.787</td>
<td>3.4561</td>
<td>1.076</td>
</tr>
</tbody>
</table>

For this modified process:
• The conventional PID controller exhibits significant overshoot, larger settling-time, and higher IAE as compared to the fuzzy logic controllers.

• For all performance measures, the type-2 FLC produces the same or superior performance compared to the type-1 FLC.

**Conclusion**

Overall, the type-1 FLC produces superior performance for the nominal plant as compared to the conventional PID controller. The type-2 FLC shows more robust performance for the modified plant.

The robustness of the conventional PID controller can be improved using different methods, such as prediction or multiple PID controller configurations. On the other hand, the performance of a type-2 FLC can be improved by using a different:

• Rulebase
• Number of rules
• FOU

For example, you can create a type-2 FLC that defines the FOU using both the lower MF scaling factor and lower MF lag.

For `fis2`, set the lower MF scale and lag values to 0.7 and 0.1, respectively for all input membership functions.

```plaintext
for i = 1:length(fis2.Inputs)
    for j = 1:length(fis2.Inputs(i).MembershipFunctions)
        fis2.Inputs(i).MembershipFunctions(j).LowerScale = 0.7;
        fis2.Inputs(i).MembershipFunctions(j).LowerLag = 0.1;
    end
end
```

Plot the updated membership functions.

```plaintext
figure
subplot(1,2,1)
plotmf(fis2,'input',1)
title('Input 1')
subplot(1,2,2)
plotmf(fis2,'input',2)
title('Input 2')
```
Simulate the model using the nominal plant, and plot the step responses for the controllers.

\[
\begin{align*}
C &= 0.5; \\
L &= 0.5; \\
T &= 0.5; \\
G &= \text{tf}(C, [T 1], 'Outputdelay', L);
\end{align*}
\]

\[
\begin{align*}
\text{out4} &= \text{sim}(\text{model}); \\
\text{close_system}(\text{model}, 0) \\
\text{plotOutput}(\text{out4}, ['\text{Nominal: } C=' \text{num2str}(C) ', L=' \text{num2str}(L) ', T=' \text{num2str}(T)])
\end{align*}
\]
Obtain the step-response characteristics of the system for each controller.

\[
\text{stepResponseTable(out4)}
\]

\[
\text{ans}=3\times4 \text{ table}
\]

<table>
<thead>
<tr>
<th></th>
<th>Rise Time (sec)</th>
<th>Overshoot (%)</th>
<th>Settling Time (sec)</th>
<th>Integral of Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>0.62412</td>
<td>11.234</td>
<td>4.5564</td>
<td>1.04</td>
</tr>
<tr>
<td>Type-1 FLC</td>
<td>1.4267</td>
<td>0</td>
<td>4.1023</td>
<td>1.1522</td>
</tr>
<tr>
<td>Type-2 FLC</td>
<td>1.2179</td>
<td>0</td>
<td>3.8746</td>
<td>1.1087</td>
</tr>
</tbody>
</table>

In this case, the updated FOU of type-2 FLC improves the rise-time of the step response.
However, the lower MF lag values also increase the overshoot in the case of the modified plant.

\[
\begin{align*}
C &= 0.85; \\
L &= 0.6; \\
T &= 0.6; \\
G &= \text{tf}(C, [T 1], 'Outputdelay', L);
\end{align*}
\]

\[
\text{out5} = \text{sim}(\text{model}); \\
\text{plotOutput(out5, ['Nominal: C=' num2str(C) ', L=' num2str(L) ', T=' num2str(T)])}
\]

\[
\begin{tabular}{|c|c|c|c|}
\hline
Rise Time (sec) & Overshoot (%) & Settling Time (sec) & Integral of Absolute Error \\
\hline
2 & Fuzzy Inference System Modeling & 2-94 & \\
\hline
\end{tabular}
\]
<table>
<thead>
<tr>
<th>PID</th>
<th>0.38464</th>
<th>80.641</th>
<th>29.452</th>
<th>4.7486</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-1 FLC</td>
<td>0.47262</td>
<td>24.877</td>
<td>4.6788</td>
<td>1.1137</td>
</tr>
<tr>
<td>Type-2 FLC</td>
<td>0.47262</td>
<td>26.699</td>
<td>4.6812</td>
<td>1.1278</td>
</tr>
</tbody>
</table>

Therefore, to obtain desired step response characteristics, you can vary the lower MF scale and lag values to find a suitable combination.

You can further improve the fuzzy logic controller outputs using a Mamdani type FIS since it also provides lower MF scale and lag parameters for output membership functions. However, a Mamdani type-2 FLC introduces additional computational delay due to the expensive type-reduction process.

**References**


**Local Functions**

```matlab
function plotOutput(out,plotTitle)
    figure
    plot([0 20],[1 1])
    hold on
    plot(out.yout{1}.Values)
    plot(out.yout{2}.Values)
    plot(out.yout{3}.Values)
    hold off
    grid minor
    xlabel('Time (sec)')
ylabel('Output')
title(plotTitle)
legend(["Reference","PID","Type-1 FLC","Type-2 FLC"],'Location','best')
end

function t = stepResponseTable(out)
s = stepinfo(out.yout{1}.Values.Data,out.yout{1}.Values.Time);
stepResponseInfo(1).RiseTime = s.RiseTime;
stepResponseInfo(1).Overshoot = s.Overshoot;
stepResponseInfo(1).SettlingTime = s.SettlingTime;
stepResponseInfo(1).IAE = out.yout{4}.Values.Data(end);
```
s = stepinfo(out.yout{2}.Values.Data,out.yout{2}.Values.Time);  
stepResponseInfo(2).RiseTime = s.RiseTime;  
stepResponseInfo(2).Overshoot = s.Overshoot;  
stepResponseInfo(2).SettlingTime = s.SettlingTime;  
stepResponseInfo(2).IAE = out.yout{5}.Values.Data(end);  

s = stepinfo(out.yout{3}.Values.Data,out.yout{3}.Values.Time);  
stepResponseInfo(3).RiseTime = s.RiseTime;  
stepResponseInfo(3).Overshoot = s.Overshoot;  
stepResponseInfo(3).SettlingTime = s.SettlingTime;  
stepResponseInfo(3).IAE = out.yout{6}.Values.Data(end);  

end

t = struct2table(stepResponseInfo,"RowNames",["PID" "Type-1 FLC" "Type-2 FLC"]);  
t.Properties.VariableNames{1} = 'Rise Time (sec)';  
t.Properties.VariableNames{3} = 'Settling Time (sec)';  
t.Properties.VariableNames{4} = 'Integral of Absolute Error';
end

See Also
Fuzzy Logic Image Processing

This example shows how to use fuzzy logic for image processing. Specifically, this example shows how to detect edges in an image.

An edge is a boundary between two uniform regions. You can detect an edge by comparing the intensity of neighboring pixels. However, because uniform regions are not crisply defined, small intensity differences between two neighboring pixels do not always represent an edge. Instead, the intensity difference might represent a shading effect.

The fuzzy logic approach for image processing allows you to use membership functions to define the degree to which a pixel belongs to an edge or a uniform region.

Import RGB Image and Convert to Grayscale

Import the image.

```matlab
Irgb = imread('peppers.png');
```

`Irgb` is a 384 x 512 x 3 `uint8` array. The three channels of `Irgb` (third array dimension) represent the red, green, and blue intensities of the image.

Convert `Irgb` to grayscale so that you can work with a 2-D array instead of a 3-D array. To do so, use the `rgb2gray` function.

```matlab
Igray = rgb2gray(Irgb);
```

```matlab
figure
image(Igray,'CDataMapping','scaled')
colormap('gray')
title('Input Image in Grayscale')
```
Convert Image to Double-Precision Data

The `evalfis` function for evaluating fuzzy inference systems supports only single-precision and double-precision data. Therefore, convert `Igray` to a double array using the `im2double` function.

\[ I = \text{im2double}(Igray); \]

Obtain Image Gradient

The fuzzy logic edge-detection algorithm for this example relies on the image gradient to locate breaks in uniform regions. Calculate the image gradient along the x-axis and y-axis.
Gx and Gy are simple gradient filters. To obtain a matrix containing the x-axis gradients of I, you convolve I with Gx using the conv2 function. The gradient values are in the [-1 1] range. Similarly, to obtain the y-axis gradients of I, convolve I with Gy.

Gx = [-1 1];
Gy = Gx';
Ix = conv2(I,Gx,'same');
Iy = conv2(I,Gy,'same');

Plot the image gradients.

figure
image(Ix,'CDataMapping','scaled')
colormap('gray')
title('Ix')
```
figure
image(Iy,'CDataMapping','scaled')
colormap('gray')
title('Iy')
```
You can use other filters to obtain the image gradients, such as the Sobel operator or the Prewitt operator. For information about how you can filter an image using convolution, see “What Is Image Filtering in the Spatial Domain?” (Image Processing Toolbox)

Alternatively, if you have the Image Processing Toolbox software, you can use the `imfilter`, `imgradientxy`, or `imgradient` functions to obtain the image gradients.

**Define Fuzzy Inference System (FIS) for Edge Detection**

Create a fuzzy inference system (FIS) for edge detection, `edgeFIS`.

```
edgeFIS = mamfis('Name','edgeDetection');
```

Specify the image gradients, `Ix` and `Iy`, as the inputs of `edgeFIS`. 
edgeFIS = addInput(edgeFIS, [-1 1], 'Name', 'Ix');
edgeFIS = addInput(edgeFIS, [-1 1], 'Name', 'Iy');

Specify a zero-mean Gaussian membership function for each input. If the gradient value for a pixel is 0, then it belongs to the zero membership function with a degree of 1.

sx = 0.1;
sy = 0.1;
edgeFIS = addMF(edgeFIS, 'Ix', 'gaussmf', [sx 0], 'Name', 'zero');
edgeFIS = addMF(edgeFIS, 'Iy', 'gaussmf', [sy 0], 'Name', 'zero');

sx and sy specify the standard deviation for the zero membership function for the Ix and Iy inputs. To adjust the edge detector performance, you can change the values of sx and sy. Increasing the values makes the algorithm less sensitive to the edges in the image and decreases the intensity of the detected edges.

Specify the intensity of the edge-detected image as an output of edgeFIS.

edgeFIS = addOutput(edgeFIS, [0 1], 'Name', 'Iout');

Specify the triangular membership functions, white and black, for Iout.

wa = 0.1;
wb = 1;
wc = 1;
ba = 0;
bb = 0;
bc = 0.7;
edgeFIS = addMF(edgeFIS, 'Iout', 'trimf', [wa wb wc], 'Name', 'white');
edgeFIS = addMF(edgeFIS, 'Iout', 'trimf', [ba bb bc], 'Name', 'black');

As you can with sx and sy, you can change the values of wa, wb, wc, ba, bb, and bc to adjust the edge detector performance. The triplets specify the start, peak, and end of the triangles of the membership functions. These parameters influence the intensity of the detected edges.

Plot the membership functions of the inputs and outputs of edgeFIS.

figure
subplot(2,2,1)
plotmf(edgeFIS, 'input', 1)
title('Ix')
subplot(2,2,2)
plotmf(edgeFIS, 'input', 2)
Specify FIS Rules

Add rules to make a pixel white if it belongs to a uniform region and black otherwise. A pixel is in a uniform region when the image gradient is zero in both directions. If either direction has a nonzero gradient, then the pixel is on an edge.

\[
\begin{align*}
\text{r1} & = \text{"If Ix is zero and Iy is zero then Iout is white";} \\
\text{r2} & = \text{"If Ix is not zero or Iy is not zero then Iout is black";}
\end{align*}
\]
edgeFIS = addRule(edgeFIS,[r1 r2]);
edgeFIS.Rules

ans =
1x2 fisrule array with properties:

    Description
    Antecedent
    Consequent
    Weight
    Connection

Details:

    Description
    _________________________________
    1    "Ix==zero & Iy==zero => Iout=white (1)"
    2    "Ix~zero | Iy~zero => Iout=black (1)"

**Evaluate FIS**

Evaluate the output of the edge detector for each row of pixels in I using corresponding rows of Ix and Iy as inputs.

Ieval = zeros(size(I));
for ii = 1:size(I,1)
    Ieval(ii,:) = evalfis(edgeFIS,[(Ix(ii,:));(Iy(ii,:))']);
end

**Plot Results**

Plot the original grayscale image.

figure
image(I,'CDataMapping','scaled')
colormap('gray')
title('Original Grayscale Image')
Plot the detected edges.

```matlab
figure
image(Ieval,'CDataMapping','scaled')
colormap('gray')
title('Edge Detection Using Fuzzy Logic')
```
See Also
evalfis

More About
- “Build Fuzzy Systems at the Command Line” on page 2-41
Fuzzy Inference System Tuning

- “Tuning Fuzzy Inference Systems” on page 3-2
- “Tune Mamdani Fuzzy Inference System” on page 3-33
- “Tune FIS Tree for Gas Mileage Prediction” on page 3-45
- “Predict Chaotic Time Series Using Type-2 FIS” on page 3-62
- “Tune Fuzzy Systems using Custom Cost Function” on page 3-78
- “Neuro-Adaptive Learning and ANFIS” on page 3-92
- “Train Adaptive Neuro-Fuzzy Inference Systems” on page 3-100
- “Save Training Error Data to MATLAB Workspace” on page 3-117
- “Predict Chaotic Time-Series using ANFIS” on page 3-125
- “Modeling Inverse Kinematics in a Robotic Arm” on page 3-135
- “Adaptive Noise Cancellation Using ANFIS” on page 3-146
- “Nonlinear System Identification” on page 3-156
- “Gas Mileage Prediction” on page 3-171
Tuning Fuzzy Inference Systems

Designing a complex fuzzy inference system (FIS) with large number of inputs and membership functions (MFs) is a challenging problem due to large number of MF parameters and increasing number of rules. A promising solution to this problem is to design FIS trees, which include hierarchically connected modular FISs having small number of inputs. Designing a FIS tree with appropriate MF parameters and rules, however, is a difficult task when adequate expert knowledge is not available for a complex system. Hence, the data-driven approach of tuning and optimizing MF and rule parameters offers an elegant solution to this problem.

You can tune a fuzzy system using the `tunefis` command. The following figure shows how a fuzzy system is tuned with input/output training data. The tuning process uses an optimization method on an optimization problem created from a fuzzy system.

In each iteration, the optimization method generates multiple sets of solutions, which are values for the selected parameters of the fuzzy system. The fuzzy system is updated with...
each solution and then evaluated using the input training data. The evaluated output is compared with the output training data to generate costs of the solutions. This process continues for multiple iterations until the stop condition is met, and then it returns the minimum cost solution with the optimized fuzzy system parameters. For an example that uses this approach, see “Tune Mamdani Fuzzy Inference System” on page 3-33.

If input/output training data is not available, you can use a custom model in place of the training data to evaluate a fuzzy system for cost measurement. This approach is shown in the following figure.

In this case, the custom model uses the fuzzy system to minimize the cost of achieving specific performance goals. The parameter solution that produces the best performance of the custom model is returned as the optimization result. For example, in a robot navigation model, the performance goal is to minimize the travelled distances to target positions without colliding with any of the obstacles. Hence, the navigation model uses the fuzzy system to control the robot’s heading direction to achieve the goal. For an
example that uses this approach, see “Tune Fuzzy Systems using Custom Cost Function” on page 3-78.

You can select individual MF and rule parameters of a fuzzy system for optimization.

**Tune Membership Function Parameters**

This example shows how to specify parameter settings for each input and output MF of a FIS and tune it.

Create a FIS.

```matlab
fis = mamfis;
fis = addInput(fis,[0 10], NumMFs,3);
fis = addOutput(fis,[0 1], NumMFs,3);
fis = addRule(fis,[1 1 1 1;1 1 1 1;1 1 1 1]);
```

Extract input and output parameter settings from the FIS.

```matlab
[in,out] = getTunableSettings(fis)
```

```matlab
in = VariableSettings with properties:

Type: "input"
VariableName: "input1"
MembershipFunctions: [1×3 fuzzy.tuning.MembershipFunctionSettings]
FISName: "fis"

out = VariableSettings with properties:

Type: "output"
VariableName: "output1"
MembershipFunctions: [1×3 fuzzy.tuning.MembershipFunctionSettings]
FISName: "fis"
```

The parameter settings are represented by `VariableSettings` objects that include the FIS name, variable type, variable name, and MF parameter settings. Examine the parameter settings of MF 1 of input 1.

```matlab
in(1).MembershipFunctions(1).Parameters
```
ans =
    NumericParameters with properties:

    Minimum: [-Inf -Inf -Inf]
    Maximum: [Inf Inf Inf]
    Free: [1 1 1]

For each parameter value of an input/output MF, you can specify whether it is available for tuning and its minimum and maximum values. By default, all MF parameters are free for tuning and their ranges are set to [-Inf, Inf]. Make MF 1 of input 1 nontunable.

    in(1).MembershipFunctions(1) = setTunable(in(1).MembershipFunctions(1),false);

Similarly, make the first parameter of MF 2 of input 1 nontunable.

    in(1).MembershipFunctions(2).Parameters.Free(1) = false;

Set minimum ranges for second and third parameters of MF 3 of input 1 to 0.

    in(1).MembershipFunctions(3).Parameters.Minimum(2:3) = 0;

Set maximum ranges for second and third parameters of MF 3 of input 1 to 15.

    in(1).MembershipFunctions(3).Parameters.Maximum(2:3) = 15;

Note that the default minimum and maximum range values of tunable MF parameters are set to corresponding input/output ranges in the tuning process.

Finally, make the output nontunable.

    out = setTunable(out,false);

Now that you have configured the parameters, specify input and output training data. Generate some data for this example.

    x = (0:0.1:10)';
    y = abs(sin(2*x)./exp(x/5));

Specify options for `tunefis`. Use genetic algorithm for optimization.

    options = tunefisOptions("Method","ga");

Specify maximum 5 generations for optimization.

    options.MethodOptions.MaxGenerations = 5;
If you have Parallel Computing Toolbox™ software, you can improve the speed of the tuning process by setting `options.UseParallel` to `true`. If you do not have Parallel Computing Toolbox software, set `options.UseParallel` to `false`.

By default, `tunefis` uses root mean squared error (RMSE) for cost calculation. You can change the cost function to `norm1` or `norm2` by setting `options.DistanceMetric`.

```matlab
options.DistanceMetric = "norm1";
```

Tune `fis` using the parameter settings, training data, and tuning options.

```matlab
rng('default')  % for reproducibility
[fisout,optimout] = tunefis(fis,[in;out],x,y,options);
```

<table>
<thead>
<tr>
<th>Generation</th>
<th>Func-count</th>
<th>Best f(x)</th>
<th>Mean f(x)</th>
<th>Stall Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>32.84</td>
<td>32.84</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>147</td>
<td>32.84</td>
<td>32.84</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>32.84</td>
<td>32.84</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>241</td>
<td>32.84</td>
<td>32.84</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>288</td>
<td>32.84</td>
<td>32.84</td>
<td>4</td>
</tr>
</tbody>
</table>

Optimization terminated: maximum number of generations exceeded.

`fisout` includes the updated parameter values. `optimout` provides additional outputs of the optimization method and any error message that are returned during the update process of the input fuzzy system using the optimized parameter values.

```matlab
optimout
optimout = struct with fields:
    tuningOutputs: [1x1 struct]
    errorMessage: []
```

`optimout.tuningOutputs`

```matlab
ans = struct with fields:
    x: [5 9.1667 5.8333 10 14.1667]
    fval: 32.8363
    exitflag: 0
    output: [1x1 struct]
    population: [50x5 double]
    scores: [50x1 double]
```
You can optionally tune fis using either input or output parameter settings. In this example, output parameter settings are set to nontunable. Therefore, tuning fis with only input parameter settings produces the same results.

```matlab
rng('default')
tunefis(fis,in,x,y,options);
```

<table>
<thead>
<tr>
<th>Generation</th>
<th>Func-count</th>
<th>f(x)</th>
<th>f(x)</th>
<th>Stall Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>32.84</td>
<td>32.84</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>147</td>
<td>32.84</td>
<td>32.84</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>32.84</td>
<td>32.84</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>241</td>
<td>32.84</td>
<td>32.84</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>288</td>
<td>32.84</td>
<td>32.84</td>
<td>4</td>
</tr>
</tbody>
</table>

Optimization terminated: maximum number of generations exceeded.

Note that the best optimization costs shown in column 3 of both tuning results are the same, which indicates that the optimization results are the same in both cases.

**Tune Fuzzy Rules**

You can specify only rule parameters of a fuzzy system for tuning.

Use `getTunableSettings` to get rule parameter settings from a fuzzy system. Rule parameter settings are returned as the third output argument.

```matlab
[~,~,rule] = getTunableSettings(fis)
```

`rule` is a 3×1 object

3×1 RuleSettings array with properties:

- Index
- Antecedent
- Consequent
- FISName

Each rule parameter settings include FIS name, index of the rule in the FIS, and parameter settings for the rule antecedent and consequent (the *rule clause*).

The parameter settings for a rule clause include three options:

- Whether the input/output MF indices are available for tuning. By default, clause parameters are free for tuning.
• Whether the clause allows use of NOT logic, in other words, whether it allows negative MF indices. By default, rules do not allow NOT logic.

• Whether the clause allows absence of input/output variables, in other words, if it allows zero MF indices. By default, absence of a variable is allowed.

rule(1).Antecedent(1)

ans =
    ClauseParameters with properties:
        AllowNot: 0
        AllowEmpty: 1
        Free: 1

Allow NOT logic in the antecedent of rule 1.

rule(1).Antecedent.AllowNot = true;

Make the consequent of rule 1 not available for tuning.

rule(1).Consequent.Free = 0;

Do not allow absence of a variable in the consequent of rule 2.

rule(2).Consequent.AllowEmpty = false;

Set rule 3 nontunable.

rule(3) = setTunable(rule(3),false);

Set options.DistanceMetric to norm2.

options.DistanceMetric = "norm2";

Tune fis with the rule parameter settings.

rng('default')  % for reproducibility
fisout = tunefis(fis,rule,x,y,options);

<table>
<thead>
<tr>
<th>Generation</th>
<th>Func-count</th>
<th>f(x)</th>
<th>Mean f(x)</th>
<th>Stall Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>1.648</td>
<td>2.575</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>147</td>
<td>1.648</td>
<td>2.448</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>1.648</td>
<td>2.212</td>
<td>2</td>
</tr>
</tbody>
</table>
Because you made rule 3 nontunable, you can exclude rule 3 when you tune `fis`.

```matlab
rng('default')  % for reproducibility
tunefis(fis,rule(1:2),x,y,options);
```

<table>
<thead>
<tr>
<th>Generation</th>
<th>Func-count</th>
<th>Best f(x)</th>
<th>Mean f(x)</th>
<th>Stall Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>1.648</td>
<td>2.575</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>147</td>
<td>1.648</td>
<td>2.448</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>1.648</td>
<td>2.212</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>241</td>
<td>1.648</td>
<td>2.052</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>288</td>
<td>1.648</td>
<td>1.874</td>
<td>4</td>
</tr>
</tbody>
</table>

Optimization terminated: maximum number of generations exceeded.

Note that the best optimization costs shown in column 3 of both tuning results are the same, which indicates the optimization results are the same in both cases.

### Learn Fuzzy Rules

You can tune a fuzzy system without any rules in `tunefis`. In this case, set the `OptimizationType` option of `tunefisOptions` to `learning` so that `tunefis` learns rules for the FIS.

```matlab
fisin = fis;  fisin.Rules = [];  options.OptimizationType = 'learning';
```

Specify maximum size of the rule base to 3. This value specifies the maximum number of rules in the tuned FIS.

```matlab
options.NumMaxRules = 3;
```

Note that the size of the tuned rule base may be less than `NumMaxRules`, because `tunefis` removes duplicate rules from the tuned FIS. If you do not specify `NumMaxRules`, then `tunefis` adds the maximum number of rules determined by the combinations of input MFs. The default input MF combinations include zero MF indices, which allow absence of variables. The default combinations exclude negative MF indices, so that NOT logic is not allowed.

Set `options.DistanceMetric` to `rmse` and tune the FIS.
options.DistanceMetric = "rmse";
rng('default')  % for reproducibility
fisout = tunefis(fisin,[],x,y,options);

<table>
<thead>
<tr>
<th>Generation</th>
<th>Func-count</th>
<th>f(x)</th>
<th>f(x)</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>400</td>
<td>0.165</td>
<td>0.2973</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>590</td>
<td>0.165</td>
<td>0.2891</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>780</td>
<td>0.165</td>
<td>0.2685</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>970</td>
<td>0.165</td>
<td>0.2548</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1160</td>
<td>0.165</td>
<td>0.2378</td>
<td>4</td>
</tr>
</tbody>
</table>

Optimization terminated: maximum number of generations exceeded.

During the tuning process, the FIS automatically learns rules after cost optimization with the training data. Examine the tuned rules.

fisout.Rules

ans =
1x3 fisrule array with properties:
   Description
   Antecedent
   Consequent
   Weight
   Connection

Details:

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 &quot;input1==mf3 =&gt; output1=mf1 (1)&quot;</td>
</tr>
<tr>
<td>2 &quot;input1==mf1 =&gt; output1=mf2 (1)&quot;</td>
</tr>
<tr>
<td>3 &quot;input1==mf2 =&gt; output1=mf1 (1)&quot;</td>
</tr>
</tbody>
</table>

You can remove some of the existing rules and learn additional rules.

fisout.Rules(2:end) = [];
rng('default')  % for reproducibility
fisout = tunefis(fisin,[],x,y,options);

<table>
<thead>
<tr>
<th>Generation</th>
<th>Func-count</th>
<th>f(x)</th>
<th>f(x)</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>400</td>
<td>0.165</td>
<td>0.2973</td>
<td>0</td>
</tr>
</tbody>
</table>
fisout.Rules

ans =
1×3 fisrule array with properties:

   Description
   Antecedent
   Consequent
   Weight
   Connection

Details:

   Description
   ______________________________
    1    "input1==mf3 ⇒ output1=mf1 (1)"
    2    "input1==mf1 ⇒ output1=mf2 (1)"
    3    "input1==mf2 ⇒ output1=mf1 (1)"

You can also tune existing rules and learn new rules.

fisout.Rules(2:end) = [];
fisout.Rules(1).Antecedent = 1;
fisout.Rules(1).Consequent = 1;
 [~,~,rule] = getTunableSettings(fisout);
rng('default')
fisout = tunefis(fisin,rule,x,y,options);

<table>
<thead>
<tr>
<th>Generation</th>
<th>Func-count</th>
<th>Best f(x)</th>
<th>Mean f(x)</th>
<th>Stall Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>400</td>
<td>0.165</td>
<td>0.3075</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>590</td>
<td>0.165</td>
<td>0.2738</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>780</td>
<td>0.165</td>
<td>0.2548</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>970</td>
<td>0.165</td>
<td>0.2271</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1160</td>
<td>0.165</td>
<td>0.2083</td>
<td>4</td>
</tr>
</tbody>
</table>

Optimization terminated: maximum number of generations exceeded.

fisout.Rules
ans =
1×3 fisrule array with properties:

Description
Antecedent
Consequent
Weight
Connection

Details:

| Description                                                                 |
|                                                                           |
| 1    "input1==mf1 => output1=mf2 (1)"                                    |
| 2    "input1==mf2 => output1=mf1 (1)"                                    |
| 3    "input1==mf3 => output1=mf1 (1)"                                    |

Tune MF and Rule Parameters

You can tune all FIS parameters together.

[in,out,rule] = getTunableSettings(fis);
options = tunefisOptions('Method','ga');
options.MethodOptions.MaxGenerations = 5;
rng('default')  % for reproducibility
fisout = tunefis(fis,[in;out;rule],x,y,options);

<table>
<thead>
<tr>
<th>Generation</th>
<th>Func-count</th>
<th>Best</th>
<th>Mean</th>
<th>Stall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>400</td>
<td>0.1624</td>
<td>0.2997</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>590</td>
<td>0.1624</td>
<td>0.2776</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>780</td>
<td>0.1624</td>
<td>0.2653</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>970</td>
<td>0.1592</td>
<td>0.2486</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1160</td>
<td>0.1592</td>
<td>0.2342</td>
<td>1</td>
</tr>
</tbody>
</table>

Optimization terminated: maximum number of generations exceeded.

For a large fuzzy system, tuning all FIS parameters in the same tuning process may take several iterations to obtain the expected results. Hence, you can tune parameters in two steps:

1. Tune or learn rule parameters only.
2. Tune both MF and rule parameters.
The first step is less computationally expensive due to the small number of rule parameters. It quickly converges to a fuzzy rule base during training. In the second step, using the rule base from the first step as an initial condition provides fast convergence of the parameter tuning process.

**Tune FIS Tree Parameters**

You can tune the parameters of a FIS tree using a similar two-step process to the one described above for tuning a FIS.

Create a FIS tree to model $\frac{\sin(x) + \cos(x)}{\exp(x)}$ as shown in the following figure.

![FIS Tree Diagram](image)

Create `fis1` as a Sugeno type FIS, which results in faster tuning process due to computationally efficient defuzzification method. Add two inputs with range [0 10] having three MFs each. Use a smooth differentiable MF, such as `gaussmf`, to match characteristics of the data type you are modeling.

```matlab
fis1 = sugfis('Name','fis1');
fis1 = addInput(fis1,[0 10],'NumMFs',3,'MFType','gaussmf');
fis1 = addInput(fis1,[0 10],'NumMFs',3,'MFType','gaussmf');
```

Add an output with range [-1.5 1.5] having nine MFs to provide maximum granularity corresponding to each combination of the input MFs. The output range is set according to the values of $\sin(x) + \cos(x)$.

```matlab
fis1 = addOutput(fis1,[-1.5 1.5],'NumMFs',9);
```
Create \( \text{fis2} \) as a Sugeno type FIS as well. Add two inputs. Use \([-1.5 1.5]\) as the range of the first input, which is the output of \( \text{fis1} \). The second input is the same as the inputs of \( \text{fis1} \), so it also uses range \([0 10]\). Add three MFs for each of the inputs.

\[
\text{fis2} = \text{sugfis('Name','fis2');}
\]
\[
\text{fis2} = \text{addInput(fis2,[-1.5 1.5],'NumMFs',3,'MFType','gaussmf');}
\]
\[
\text{fis2} = \text{addInput(fis2,[0 10],'NumMFs',3,'MFType','gaussmf');}
\]

Add an output with range \([0 1]\) having nine MFs to provide maximum granularity corresponding to each combination of the input MFs. The output range is set according to the values of \( \frac{\sin(x) + \cos(x)}{\exp(x)} \).

\[
\text{fis2} = \text{addOutput(fis2,[0 1],'NumMFs',9);}
\]

Connect the inputs and the outputs as shown in the diagram. Output 1 of \( \text{fis1} \) connects to input 1 of \( \text{fis2} \), inputs 1 and 2 of \( \text{fis1} \) connect to each other, and input 2 of \( \text{fis1} \) connects to input 2 of \( \text{fis2} \).

\[
\text{con1} = \{"fis1/output1" "fis2/input1"\};
\]
\[
\text{con2} = \{"fis1/input1" "fis1/input2"\};
\]
\[
\text{con3} = \{"fis1/input2" "fis2/input2"\};
\]

Finally, create a FIS tree using \( \text{fis1}, \text{fis2}, \text{con1}, \text{con2}, \) and \( \text{con3} \).

\[
\text{fisT} = \text{fistree([fis1 fis2],[con1;con2;con3]);}
\]

Specify an additional output to the FIS tree to access the output of \( \text{fis1} \).

\[
\text{fisT.Outputs} = \{"fis1/output1";fisT.Outputs\};
\]

Generate input and output training data.

\[
\text{x} = (0:0.1:10)';
\]
\[
\text{y1} = \sin(x)+\cos(x);
\]
\[
\text{y2} = \text{y1}./\exp(x);
\]
\[
\text{y} = [\text{y1} \text{y2}];
\]

Tune the FIS tree parameters in two steps. First, use a global optimization method such as particle swarm or genetic algorithm to learn the rules of the FIS tree. Create \text{tunefis} options for learning with the \text{particleswarm} method.

\[
\text{options} = \text{tunefisOptions('Method','particleswarm','OptimizationType','learning');}
\]

This tuning step uses small number of iterations to learn a rule base without overfitting the training data. The rule base provides an educated initial condition for the second step.
to optimize all the FIS tree parameters together. Set the maximum iteration number to 5, and learn the rule base.

```matlab
options.MethodOptions.MaxIterations = 5;
rng('default') % for reproducibility
fisTout1 = tunefis(fisT,[],x,y,options);
```

<table>
<thead>
<tr>
<th>Iteration</th>
<th>f-count</th>
<th>Best f(x)</th>
<th>Mean f(x)</th>
<th>Stall Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>0.6682</td>
<td>0.9395</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>0.6682</td>
<td>1.023</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td>0.6652</td>
<td>0.9308</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0.6259</td>
<td>0.918</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>0.6259</td>
<td>0.9179</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>600</td>
<td>0.5969</td>
<td>0.9179</td>
<td>0</td>
</tr>
</tbody>
</table>

Optimization ended: number of iterations exceeded OPTIONS.MaxIterations.

To tune all the FIS tree parameters, the second step uses a local optimization method such as pattern search or simulation annealing method. Local optimization is generally faster than global optimization and can produce better results when the input fuzzy system parameters are already consistent with the training data.

Use the `patternsearch` method for optimization. Set the number of iterations to 25.

```matlab
options.Method = 'patternsearch';
options.MethodOptions.MaxIterations = 25;
```

Use `getTunableSettings` to get input, output, and rule parameter settings from `fisTout1`.

```matlab
[in,out,rule] = getTunableSettings(fisTout1);
```

Tune all FIS tree parameters.

```matlab
rng('default') % for reproducibility
fisTout2 = tunefis(fisTout1,[in;out;rule],x,y,options);
```

<table>
<thead>
<tr>
<th>Iter</th>
<th>Func-count</th>
<th>f(x)</th>
<th>MeshSize</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0.596926</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0.551284</td>
<td>2</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>0.548551</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0.546331</td>
<td>8</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>0.527482</td>
<td>16</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>5</td>
<td>33</td>
<td>0.527482</td>
<td>8</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>6</td>
<td>61</td>
<td>0.511532</td>
<td>16</td>
<td>Successful Poll</td>
</tr>
</tbody>
</table>
Tune FIS Tree with Selected Fuzzy Systems

You can tune specific fuzzy systems in a FIS tree. To get parameter settings of the specific fuzzy systems, use getTunableSettings. For example, after learning the rule base of the previous FIS tree, separately tune fis1 and fis2 parameters. First, get the parameter settings for fis1.

\[
\text{[in, out, rule]} = \text{getTunableSettings}(\text{fisTout1}, \text{"FIS"}, \text{"fis1"});
\]

Tune fis1 parameters of the tree.

\[
\text{rng('default')} \\
\text{fisTout2} = \text{tunefis}(\text{fisTout1}, [\text{in}; \text{out}; \text{rule}], x, y, \text{options});
\]
In this case, the optimization cost is improved by tuning only \texttt{fis1} parameter values.

Next, get the parameter settings for \texttt{fis2} and tune the \texttt{fis2} parameters.

\begin{verbatim}
[in,out,rule] = getTunableSettings(fisTout2,"FIS","fis2");
rng('default')
fisTout3 = tunefis(fisTout2,[in;out;rule],x,y,options);
\end{verbatim}
The optimization cost is further reduced by tuning fis2 parameter values. To avoid overfitting of individual FIS parameter values, you can further tune both fis1 and fis2 parameters together.

```
[in,out,rule] = getTunableSettings(fisTout3);
rng('default')
fisTout4 = tunefis(fisTout3,[in;out;rule],x,y,options);
```

<table>
<thead>
<tr>
<th>Iter</th>
<th>Func-count</th>
<th>f(x)</th>
<th>MeshSize</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0.333193</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>0.326804</td>
<td>2</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>2</td>
<td>91</td>
<td>0.326432</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>3</td>
<td>116</td>
<td>0.326261</td>
<td>8</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>4</td>
<td>154</td>
<td>0.326261</td>
<td>4</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>5</td>
<td>205</td>
<td>0.326261</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>6</td>
<td>302</td>
<td>0.326092</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>7</td>
<td>352</td>
<td>0.326092</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>8</td>
<td>391</td>
<td>0.325964</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>9</td>
<td>441</td>
<td>0.325964</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>10</td>
<td>478</td>
<td>0.32578</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>11</td>
<td>528</td>
<td>0.32578</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>12</td>
<td>562</td>
<td>0.325691</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>13</td>
<td>612</td>
<td>0.325691</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>14</td>
<td>713</td>
<td>0.229273</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>15</td>
<td>763</td>
<td>0.229273</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>16</td>
<td>867</td>
<td>0.22891</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>17</td>
<td>917</td>
<td>0.22891</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>18</td>
<td>1036</td>
<td>0.228688</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>19</td>
<td>1086</td>
<td>0.228688</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
</tbody>
</table>
Overall, the optimization cost reduces from 0.59 to 0.22 in three steps.

**Tune with Custom Cost Function**

Suppose you want to modify the previous FIS tree as shown in the following diagram.

Create the FIS tree.

```matlab
fis1 = sugfis('Name','fis1');
fis1 = addInput(fis1,[0 10],'NumMFs',3,'MFType','gaussmf');
fis1 = addOutput(fis1,[-1 1],'NumMFs',3);

fis2 = sugfis('Name','fis2');
fis2 = addInput(fis2,[0 10],'NumMFs',3,'MFType','gaussmf');
fis2 = addOutput(fis2,[-1 1],'NumMFs',3);

fis3 = sugfis('Name','fis3');
fis3 = addInput(fis3,[0 10],'NumMFs',3,'MFType','gaussmf');
fis3 = addOutput(fis3,[0 1]);
```
con = ["fis1/input1" "fis2/input1";"fis2/input1" "fis3/input1"];  
fisT = fistree([fis1 fis2 fis3],con);

To implement the addition and multiplication operations, use a custom cost function. For this example, use the function `customcostfcn`, included at the end of the example. Learn a rule base with this cost function.

```matlab
options.Method = 'particleswarm';
options.MethodOptions.MaxIterations = 5;
rng('default')
fisTout1 = tunefis(fisT,[],@(fis)customcostfcn(fis,x,y),options);
```

<table>
<thead>
<tr>
<th>Iteration</th>
<th>f-count</th>
<th>Best f(x)</th>
<th>Mean f(x)</th>
<th>Stall Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>0.746</td>
<td>1.318</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>0.5787</td>
<td>1.236</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td>0.5787</td>
<td>1.104</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0.5787</td>
<td>1.097</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>0.5171</td>
<td>1.155</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>600</td>
<td>0.5171</td>
<td>1.067</td>
<td>1</td>
</tr>
</tbody>
</table>

Optimization ended: number of iterations exceeded OPTIONS.MaxIterations.

Tune all parameters of the FIS tree.

```matlab
options.Method = 'patternsearch';
options.MethodOptions.MaxIterations = 25;
[in,out,rule] = getTunableSettings(fisTout1);
rng('default')
fisTout2 = tunefis(fisTout1,[in;out;rule],@(fis)customcostfcn(fis,x,y),options);
```

<table>
<thead>
<tr>
<th>Iter</th>
<th>Func-count</th>
<th>f(x)</th>
<th>MeshSize</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0.51705</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>0.514884</td>
<td>2</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>0.512873</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>0.512873</td>
<td>8</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>4</td>
<td>56</td>
<td>0.512873</td>
<td>4</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>5</td>
<td>79</td>
<td>0.512873</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>6</td>
<td>106</td>
<td>0.512869</td>
<td>4</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>7</td>
<td>129</td>
<td>0.512869</td>
<td>2</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>8</td>
<td>174</td>
<td>0.512869</td>
<td>1</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>9</td>
<td>197</td>
<td>0.512862</td>
<td>2</td>
<td>Successful Poll</td>
</tr>
<tr>
<td>10</td>
<td>242</td>
<td>0.512862</td>
<td>1</td>
<td>Refine Mesh</td>
</tr>
<tr>
<td>11</td>
<td>314</td>
<td>0.512862</td>
<td>0.5</td>
<td>Refine Mesh</td>
</tr>
</tbody>
</table>
Maximum number of iterations exceeded: increase `options.MaxIterations`.

You can add more input/output MFs and specify additional FIS tree outputs to improve the tuning performance. Using additional MF parameters and more training data for additional FIS tree outputs can further fine tune the outputs of `fis1`, `fis2`, and `fis3`.

### Custom Optimization Method

You can implement your own FIS parameter optimization method using `getTunableSettings`, `getTunableValues`, and `setTunableValues`. This example uses these functions to tune a rule base of a fuzzy system.

Create a FIS to approximate \( \sin(\theta) \), where \( \theta \) varies from 0 to \( 2\pi \).

```
fisin = mamfis;
```

Add an input with range \([0 \ 2\pi]\) having five MFs of Gaussian type, and an output with range \([-1 \ 1]\) having five MFs of Gaussian type.

```
fisin = addInput(fisin,[0 2*pi],'NumMFs',5,'MFType','gaussmf');
fisin = addOutput(fisin,[-1 1],'NumMFs',5,'MFType','gaussmf');
```

Add 5 rules.

```
fisin = addRule(fisin,[1 1 1 1;2 2 1 1;3 3 1 1;4 4 1 1;5 5 1 1]);
fisin.Rules
```

```
ans =
1×5 fisrule array with properties:
```
### Details:

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

Set `DisableStructuralChecks` to true for faster FIS update.

```matlab
fisin.DisableStructuralChecks = true;
```

Get the rule parameter settings.

```matlab
 [~,~,rule] = getTunableSettings(fisin);
```

Make the antecedent nontunable. In the consequent, do not allow NOT logic (negative MF indices) or empty variables (zero MF indices) in the rules.

```matlab
for i = 1:numel(rule)
    rule(i).Antecedent.Free = false;
    rule(i).Consequent.AllowNot = false;
    rule(i).Consequent.AllowEmpty = false;
end
```

Generate data for tuning.

```matlab
x = (0:0.1:2*pi)';
y = sin(x);
```

To tune the rule parameters, use the `customtunefis` function defined at the end of this example. Set the number of iterations to 2, and do not allow invalid parameter values when updating the FIS using `setTunableValues`. 

---

3-22
numite = 2;
ignoreinvp = false;
fisout = customtunefis(fisin,rule,x,y,numite,ignoreinvp);

Initial cost = 1.170519
Iteration 1: Cost = 0.241121
Iteration 2: Cost = 0.241121

Display tuned rules.

fisout.Rules

ans =
1×5 fisrule array with properties:

    Description
    Antecedent
    Consequent
    Weight
    Connection

Details:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;input1==mf1 =&gt; output1=mf4 (1)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;input1==mf2 =&gt; output1=mf5 (1)&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;input1==mf3 =&gt; output1=mf3 (1)&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;input1==mf4 =&gt; output1=mf1 (1)&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;input1==mf5 =&gt; output1=mf2 (1)&quot;</td>
</tr>
</tbody>
</table>

Allow NOT logic in the rules, and optimize the FIS again.

for i = 1:numel(rule)
    rule(i).Consequent.AllowNot = true;
end
fisout = customtunefis(fisin,rule,x,y,numite,ignoreinvp);

Initial cost = 1.170519
Iteration 1: Cost = 0.357052
Iteration 2: Cost = 0.241121

fisout.Rules

ans =
1×5 fisrule array with properties:
With NOT logic, there are more combinations of rule parameters, and it generally takes more iterations to tune a FIS.

Next, reset AllowNot to false and set AllowEmpty to true, in other words, allow absence of variables (zero output MF indices) in the consequent. Tune the FIS with the updated rule parameter settings.

```matlab
for i = 1:numel(rule)
    rule(i).Consequent.AllowNot = false;
    rule(i).Consequent.AllowEmpty = true;
end
```

try
    fisout = customtunefis(fisin,rule,x,y,numite,ignoreinvp);
catch me
    disp("Error: "+me.message)
end

Initial cost = 1.170519
Error: Rule consequent must have at least one nonzero membership function index.

The tuning process fails since the FIS only contains one output, which must be nonzero (nonempty) in the rule consequent. To ignore invalid parameter values, specify IgnoreInvalidParameters with setTunableValues.

Set ignoreinvp to true, which specifies IgnoreInvalidParameters value in the call to setTunableValues used in customtunefis.
ignoreinvp = true;
fisout = customtunefis(fisin,rule,x,y,numite,ignoreinvp);

Initial cost = 1.170519
Iteration 1: Cost = 0.241121
Iteration 2: Cost = 0.241121

fisout.Rules

ans =
  1×5 fisrule array with properties:

    Description     Antecedent      Consequent    Weight    Connection
    _______________________________________________________________
       1    "input1==mf1 => output1=mf4 (1)"
       2    "input1==mf2 => output1=mf5 (1)"
       3    "input1==mf3 => output1=mf3 (1)"
       4    "input1==mf4 => output1=mf1 (1)"
       5    "input1==mf5 => output1=mf2 (1)"

In this case, the tuning process bypasses the invalid values and uses only valid parameter values for optimization.

By default, tunefis ignores invalid values when updating fuzzy system parameters. You can change this behavior by setting tunefisOptions.IgnoreInvalidParameters to false.

**Generate FIS from Data and Tune**

You can generate a FIS from the training data using genfis and then optimize the FIS with tunefis. In this approach, the tuning process can employ a local optimization method because the rule base is derived from the training data.

This example describes the tuning steps to approximate the function
\[
\frac{\sin(2x)}{\exp \left( \frac{x}{5} \right)},
\]

where the input \( x \) varies from 0 to 10.

Generate training data.

\[
x = (0:0.1:10)';
y = \sin(2*x)./\exp(x/5);
\]

Create options for \texttt{genfis} that specify five MFs, a Gaussian MF for the input, and a constant MF for the output.

\[
goptions = \texttt{genfisOptions('GridPartition','NumMembershipFunctions',5, ...}
\]
\[
\text{'InputMembershipFunctionType','gaussmf', ...}
\]
\[
\text{'OutputMembershipFunctionType','constant');}
\]

Generate the initial FIS, and get its parameter settings.

\[
fisin = \texttt{genfis(x,y,goptions)};
\]
\[
\text{[in,out,rule]} = \texttt{getTunableSettings(fisin)};
\]

Use pattern search method for optimization, setting the maximum number of iterations to 25, and tune the FIS.

\[
toptions = \texttt{tunefisOptions('Method','patternsearch');}
toptions.MethodOptions.MaxIterations = 25;
\]
\[
rng('default')
fisout = \texttt{tunefis(fisin,[in;out],x,y,toptions)};
\]
You can increase the number of iterations to further optimize the cost.

**Validate Training Results**

To avoid overfitting of training data sets, check the performance of a trained fuzzy system with validation data. One common approach is to divide available data into training and validation data sets so that both data sets include similar characteristics.

If training and validation performances differ significantly, you can:

- Change the maximum number of tuning iterations. For example, if the training performance is better than validation performance, you can reduce the number of tuning iterations.
- Resample the training and validation data sets to maintain homogeneity.

To see how the validation performance can vary based on nonhomogeneous sampling, create a single-input-single-output fuzzy inference system to model the function

\[
\frac{\sin(2x)}{\exp^{\frac{x}{5}}},
\]

where the input varies from 0 to 10 and the output range is [0 1]. Add five default MFs to the input and output. Get the input and output parameter settings.

```matlab
fisin = sugfis;
fisin = addInput(fisin,[0 10],'NumMFs',5,'MFType','gaussmf');
fisin = addOutput(fisin,[0 1],'NumMFs',5);
[in,out] = getTunableSettings(fisin);
```
Create `tunefis` options for learning with the `particleswarm` optimization method, and set the maximum number of iterations to 5.

```matlab
options = tunefisOptions('Method','particleswarm','OptimizationType','learning');
options.MethodOptions.MaxIterations = 5;
```

Divide the available data into two sets without maintaining homogeneity in the data sets.

```matlab
x = (0:0.1:10)';
n = numel(x);
midn = floor(n/2);
trnX = x(1:midn);
vldX = x(midn+1:end);
f = @(x)(sin(2*x)./exp(x/5));
trnY = f(trnX);
vldY = f(vldX);
```

Tune the FIS parameters.

```matlab
rng('default')
fisout = tunefis(fisin,[in;out],trnX,trnY,options);
```

<table>
<thead>
<tr>
<th>Iteration</th>
<th>f-count</th>
<th>Best f(x)</th>
<th>Mean f(x)</th>
<th>Stall Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>0.4279</td>
<td>0.5932</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>0.3846</td>
<td>0.6183</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td>0.3751</td>
<td>0.5675</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0.3606</td>
<td>0.568</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>0.3606</td>
<td>0.5596</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>600</td>
<td>0.3598</td>
<td>0.5307</td>
<td>0</td>
</tr>
</tbody>
</table>

Optimization ended: number of iterations exceeded OPTIONS.MaxIterations.

Find the optimization cost for validation data using the `findcost` function defined at the end of this example.

```matlab
vldCost = findcost(fisout,vldX,vldY)
```

`vldCost = 0.1780`

Note that the validation cost differs from the best training cost shown in column 3 of the tuning result.

Next, resample the available data, and create two homogeneous data sets for training and validation.
trnX = x(1:2:end);
vldX = x(2:2:end);
f = @(x)(sin(2*x)./exp(x/5));
trnY = f(trnX);
vldY = f(vldX);

Tune the FIS with the new training data.

rng('default')
fisout = tunefis(fisin,[in;out],trnX,trnY,options);

<table>
<thead>
<tr>
<th>Iteration</th>
<th>f-count</th>
<th>Best f(x)</th>
<th>Mean f(x)</th>
<th>Stall Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>0.3445</td>
<td>0.5852</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>0.2996</td>
<td>0.5616</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td>0.2907</td>
<td>0.5381</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0.2878</td>
<td>0.5334</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>0.2878</td>
<td>0.5624</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>600</td>
<td>0.2877</td>
<td>Inf</td>
<td>0</td>
</tr>
</tbody>
</table>

Optimization ended: number of iterations exceeded OPTIONS.MaxIterations.

Find the optimization cost for the new validation data.

vldCost = findcost(fisout,vldX,vldY)

vldCost = 0.2803

In this case, homogeneous sampling reduces the difference between the training and validation costs.

**Ways to Improve Tuning Results**

You can improve the training error of a tuned fuzzy systems by following these guidelines.

- **Use multiple steps in a tuning process.** For example, first learn the rules of a fuzzy system, and then tune input/output MF parameters using the learned rule base. You can also separately tune individual FIS parameters in a FIS tree and then tune all the fuzzy systems together to generalize the parameter values.

- **Increase the number of iterations in both the rule-learning and parameter-tuning phases.** Doing so increases the duration of the optimization process. It can also increase validation error due to overtuned system parameters with the training data.

- **Use global optimization methods, such as ga and particleswarm, in both rule-learning and parameter-tuning phases.** Global optimizers ga and particleswarm
perform better for large parameter tuning ranges than local optimizers. The local optimizers `patternsearch` and `simulannealbnd` perform better for small parameter ranges. If rules are already added to a FIS tree using training data, then `patternsearch` and `simulannealbnd` can produce faster convergence compared to `ga` and `particleswarm`. For more information on these optimization methods and their options, see `ga`, `particleswarm`, `patternsearch`, and `simulannealbnd`.

- **Change the clustering technique used by `genfis`**. Depending on the clustering technique, the generated rules can differ in their representation of the training data. Hence, the use of different clustering techniques can affect the performance of `tunefis`.

- **Change FIS properties**. Try changing properties such as the type of FIS, number of inputs, number of input/output MFs, MF types, and number of rules. A Sugeno system has fewer output MF parameters (assuming constant MFs) and faster defuzzification. Therefore, for fuzzy systems with a large number of inputs, a Sugeno FIS generally converges faster than a Mamdani FIS. Small numbers of MFs and rules reduce the number of parameters to tune, producing a faster tuning process. Furthermore, a large number of rules might overfit the training data.

- **Modify tunable parameter settings for MFs and rules**. For example, you can tune the support of a triangular MF without changing its peak location. Doing so reduces the number of tunable parameters and can produce a faster tuning process for specific applications. For rules, you can exclude zero MF indices by setting the `AllowEmpty` tunable setting to `false`, which reduces the overall number of rules during the learning phase.

- **Change FIS tree properties**, such as the number of fuzzy systems and the connections between the fuzzy systems.

- **Use different ranking and grouping of the inputs to the FIS tree**. For more information about creating FIS trees, see “Fuzzy Trees” on page 2-68.

**Local functions**

```matlab
function cost = customcostfcn(fis,x,y)

tY = evalfis(fis,x);
sincosx = tY(:,1)+tY(:,2);
sincosexpx = sincosx.*tY(:,3);
actY = [sincosx;sincosexpx];
d = y(:)-actY;
cost = sqrt(mean(d.*d));
end
```

3-30
function fis = customtunefis(fis,rule,x,y,n,ignore)

% Show initial cost.
cost = findcost(fis,x,y);
fprintf('Initial cost = %f\n',cost);

% Optimize rule parameters.
umMFs = numel(fis.Outputs.MembershipFunctions);
for ite = 1:n
  for i = 1:numel(rule)
    % Get consequent value.
    pval = getTunableValues(fis,rule(i));
    % Loop through output MF indices to minimize the cost.
    % Use output indices according to AllowNot and AllowEmpty.
    allowNot = rule(i).Consequent.AllowNot;
    allowEmpty = rule(i).Consequent.AllowEmpty;
    if allowNot && allowEmpty
      mfID = -numMFs:numMFs;
    elseif allowNot && ~allowEmpty
      mfID = [-numMFs:-1 1:numMFs];
    elseif ~allowNot && allowEmpty
      mfID = 0:numMFs;
    else
      mfID = 1:numMFs;
    end
    cost = 1000;
    minCostFIS = fis;
    for j = 1:length(mfID)
      % Update consequent value.
      pval(1) = mfID(j);
      % Set updated consequent value to the FIS.
      fis = setTunableValues(fis,rule(i),pval,'IgnoreInvalidParameters',ignore);
      % Evaluate cost.
      rmse = findcost(fis,x,y);
      % Update FIS with the minimum cost.
      if rmse<cost
        cost = rmse;
        minCostFIS = fis;
      end
    end
  end
fis = minCostFIS;
fprintf('Iteration %d: Cost = %f\n',ite,cost);
function cost = findcost(fis,x,y)

actY = evalfis(fis,x);
d = y - actY;
cost = sqrt(mean(d.*d));

end

See Also

genfis | getTunableSettings | tunefis

More About

- “Tune Mamdani Fuzzy Inference System” on page 3-33
- “Tune FIS Tree for Gas Mileage Prediction” on page 3-45
Tune Mamdani Fuzzy Inference System

This example shows how to tune membership function (MF) and rule parameters of a Mamdani fuzzy inference system (FIS). This example uses particle swarm and pattern search optimization, which require Global Optimization Toolbox™ software.

Automobile fuel consumption prediction in miles per gallon (MPG) is a typical nonlinear regression problem. It uses several automobile profile attributes to predict fuel consumption. The training data is available in the University of California at Irvine Machine Learning Repository and contains data collected from automobiles of various makes and models.

This example uses the following six input data attributes to predict the output data attribute MPG with a FIS:

1. Number of cylinders
2. Displacement
3. Horsepower
4. Weight
5. Acceleration
6. Model year

Prepare Data

Load the data. Each row of the dataset obtained from the repository represents a different automobile profile.

```matlab
[data,name] = loadgas;
```

Remove leading and trailing whitespace from the attribute names.

```matlab
name = strtrim(string(name));
```

data contains 7 columns, where the first six columns contain the following input attributes.

- Number of cylinders
- Displacement
- Horsepower
- Weight
- Acceleration
- Model year

The seventh column contains the output attribute, MPG.

Create separate input and output data sets, X and Y, respectively.

\[
X = \text{data}(:,1:6); \\
Y = \text{data}(:,7);
\]

Partition the input and output data sets into training data (odd-indexed samples) and validation data (even-indexed samples).

\[
\text{trnX} = X(1:2:end,:); \quad \% \text{Training input data set} \\
\text{trnY} = Y(1:2:end,:); \quad \% \text{Training output data set} \\
\text{vldX} = X(2:2:end,:); \quad \% \text{Validation input data set} \\
\text{vldY} = Y(2:2:end,:); \quad \% \text{Validation output data set}
\]

Extract the range of each data attribute, which you will use for input/output range definition during FIS construction.

\[
\text{dataRange} = [\min(\text{data})' \; \max(\text{data})'];
\]

**Construct FIS using Data Attribute Ranges**

Create a Mamdani FIS for tuning.

\[
fisin = \text{mamfis};
\]

Add input and output variables to the FIS, where each variable represents one of the data attributes. For each variable, use the corresponding attribute name and range.

To reduce the number of rules, use two MFs for each input variable, which results in \(2^6 = 64\) input MF combinations. Therefore, the FIS uses a maximum of 64 rules corresponding to the input MF combinations.

To improve data generalization beyond the training data, use 64 MFs for the output variable. Doing so allows the FIS to use a different output MF for each rule.

Both input and output variables use default triangular MFs, which are uniformly distributed over the variable ranges.
for i = 1:6
    fisin = addInput(fisin,dataRange(i,:),'Name',name(i),'NumMFs',2);
end
fisin = addOutput(fisin,dataRange(7,:),'Name',name(7),'NumMFs',64);

View the FIS structure. Initially, the FIS has zero rules. The rules of the system are found during the tuning process.

figure
plotfis(fisin)

Tune FIS with Training Data

Tuning is performed in two steps.
Learn the rule base while keeping the input and output MF parameters constant.

Tune the parameters of the input/output MFs and rules.

The first step is less computationally expensive due to the small number of rule parameters, and it quickly converges to a fuzzy rule base during training. In the second step, using the rule base from the first step as an initial condition provides fast convergence of the parameter tuning process.

**Learn Rules**

To learn a rule base, first specify tuning options using a `tunefisOptions` object. Since the FIS allows a large number of output MFs (used in rule consequents), use a global optimization method (genetic algorithm or particle swarm). Such methods perform better in large parameter tuning ranges as compared to local optimization methods (pattern search and simulation annealing). For this example, tune the FIS using the particle swarm optimization method (`'particleswarm'`).

To learn new rules, set the `OptimizationType` to `'learning'`. Restrict the maximum number of rules to 64. The number of tuned rules can be less than this limit, since the tuning process removes duplicate rules.

```matlab
options = tunefisOptions('Method','particleswarm',...    
                        'OptimizationType','learning', ... 
                        'NumMaxRules',64);
```

If you have Parallel Computing Toolbox™ software, you can improve the speed of the tuning process by setting `options.UseParallel` to `true`. If you do not have Parallel Computing Toolbox software, set `options.UseParallel` to `false`.

Set the maximum number of iterations to 20. To reduce training error in the rule learning process, you can increase the number of iterations. However, using too many iterations can overtune the FIS to the training data, increasing the validation errors.

```matlab
options.MethodOptions.MaxIterations = 20;
```

Since particle swarm optimization uses random search, to obtain reproducible results, initialize the random number generator to its default configuration.

```matlab
rng('default')
```

Tune the FIS using the specified tuning data and options.
Learning rules using the `tunefis` function takes approximately 5 minutes. For this example, enable tuning by setting `runtunefis` to `true`. To load pretrained results without running `tunefis`, you can set `runtunefis` to `false`.

```matlab
runtunefis = false;
```

Parameter settings can be empty when learning new rules. For more information, see `tunefis`.

```matlab
if runtunefis
    fisout1 = tunefis(fisin,[],trnX,trnY,options);
#else
    tunedfis = load('tunedfismpgprediction.mat');
    fisout1 = tunedfis.fisout1;
    fprintf('Training RMSE = %.3f MPG\n',calculateRMSE(fisout1,trnX,trnY));
end
```

Training RMSE = 4.452 MPG

The **Best f(x)** column shows the training root-mean-squared-error (RMSE).

View the structure of the tuned FIS, `fisout1`.

```matlab
plotfis(fisout1)
```
The learning process produces a set of new rules for the FIS. For example, view the descriptions of the first three rules.

[fisout1.Rules(1:3).Description]'

ans =

0x0 empty string array

The learned system should have similar RMSE performance for both the training and validation data sets. To calculate the RMSE for the validation data set, evaluate fisout1 using validation input data set vldX. To hide run-time warnings during evaluation, set all the warning options to none.
Calculate the RMSE between the generated output data and the validation output data set `vldY`.

```
plotActualAndExpectedResultsWithRMSE(fisout1,vldX,vldY)
```

Since the training and validation errors are similar, the learned system does not overfit the training data.

**Tune All Parameters**

After learning the new rules, tune the input/output MF parameters along with the parameters of the learned rules. To obtain the tunable parameters of the FIS, use the `getTunableSettings` function.
To tune the existing FIS parameter settings without learning new rules, set the OptimizationType to 'tuning'.

options.OptimizationType = 'tuning';

Since the FIS already learned rules using the training data, use a local optimization method for fast convergence of the parameter values. For this example, use the pattern search optimization method ('patternsearch').

options.Method = 'patternsearch';

Tuning the FIS parameters takes more iterations than the previous rule-learning step. Therefore, increase the maximum number of iterations of the tuning process to 60. As in the first tuning stage, you can reduce training errors by increasing the number of iterations. However, using too many iterations can overtune the parameters to the training data, increasing the validation errors.

options.MethodOptions.MaxIterations = 60;

Tune the FIS parameters using the specified tunable settings, training data, and tuning options.

Tuning parameter values with tunefis function takes approximately 5 minutes. To load pretrained results without running tunefis, you can set runtunefis to false.

if runtunefis
  rng('default') %#ok<UNRCH>
  fisout = tunefis(fisout1,[in;out;rule],trnX,trnY,options);
else
  fisout = tunedfis.fisout;
  fprintf('Training RMSE = %.3f MPG\n',calculateRMSE(fisout,trnX,trnY));
end

Training RMSE = 3.272 MPG

At the end of the tuning process, some of the tuned MF shapes are different than the original ones.

figure
plotfis(fisout)
Check Performance

Validate the performance of the tuned FIS, fisout, using the validation input data set vldX.

Compare the expected MPG obtained from the validation output data set vldY and actual MPG generated using fisout. Compute the RMSE between these results.

plotActualAndExpectedResultsWithRMSE(fisout,vldX,vldY);
Tuning the FIS parameters improves the RMSE compared to the results from the initial learned rule base. Since the training and validation errors are similar, the parameters values are not overtuned.

**Conclusion**

You can further improve the training error of the tuned FIS by:

- Increasing number of iterations in both the rule-learning and parameter-tuning phases. Doing so increases the duration of the optimization process and can also increase validation error due to overtuned system parameters with the training data.

- Using global optimization methods, such as ga and particleswarm, in both rule-learning and parameter-tuning phases. ga and particleswarm perform better for
large parameter tuning ranges since they are global optimizers. On the other hand, *patternsearch* and *simulannealbnd* perform better for small parameter ranges since they are local optimizers. If a FIS is generated from training data with *genfis* or a rule base is already added to a FIS using training data, then *patternsearch* and *simulannealbnd* may produce faster convergence as compared to *ga* and *particleswarm*. For more information on these optimization methods and their options, see *ga*, *particleswarm*, *patternsearch*, and *simulannealbnd*.

- Changing the FIS properties, such as the type of FIS, number of inputs, number of input/output MFs, MF types, and number of rules. For fuzzy systems with a large number of inputs, a Sugeno FIS generally converges faster than a Mamdani FIS since a Sugeno system has fewer output MF parameters (if constant MFs are used) and faster defuzzification. Small numbers of MFs and rules reduce the number of parameters to tune, producing a faster tuning process. Furthermore, a large number of rules may overfit the training data. In general, for larger fuzzy systems, a FIS tree can produce similar performance with a smaller number of rules as compared to a single FIS. For an example, see “Tune FIS Tree for Gas Mileage Prediction” on page 3-45.

- Modifying tunable parameter settings for MFs and rules. For example, you can tune the support of a triangular MF without changing its peak location. Doing so reduces the number of tunable parameters and can produce a faster tuning process for specific applications. For rules, you can exclude zero MF indices by setting the *AllowEmpty* tunable setting to `false`, which reduces the overall number of rules during the learning phase.

**Local Functions**

```matlab
function plotActualAndExpectedResultsWithRMSE(fis,x,y)

% Calculate RMSE between actual and expected results
[rmse,actY] = calculateRMSE(fis,x,y);

% Plot results
figure
subplot(2,1,1)
hold on
bar(actY)
bar(y)
bar(min(actY,y),'FaceColor',[0.5 0.5 0.5])
hold off
axis([0 200 0 60])
xlabel("Validation input dataset index"),ylabel("MPG")
```

3-43
legend(["Actual MPG" "Expected MPG" "Minimum of actual and expected values"],...
'Location','NorthWest')
title("RMSE = " + num2str(rmse) + " MPG")

subplot(2,1,2)
bar(actY-y)
xlabel("Validation input dataset index"),ylabel("Error (MPG)"
)title("Difference Between Actual and Expected Values")

end

function [rmse,actY] = calculateRMSE(fis,x,y)

% Specify options for FIS evaluation
persistent evalOptions
if isempty(evalOptions)
    evalOptions = evalfisOptions("EmptyOutputFuzzySetMessage","none", ... 
        "NoRuleFiredMessage","none","OutOfRangeInputValueMessage","none");
end

% Evaluate FIS
actY = evalfis(fis,x,evalOptions);

% Calculate RMSE
del = actY - y;
rmse = sqrt(mean(del.^2));

end

See Also
getTunableSettings|mamfis|tunefis

More About
• “Tuning Fuzzy Inference Systems” on page 3-2
• “Tune FIS Tree for Gas Mileage Prediction” on page 3-45
This example shows how to tune parameters of a FIS tree, which is a collection of connected fuzzy inference systems. This example uses particle swarm and pattern search optimization, which require Global Optimization Toolbox™ software.

Automobile fuel consumption prediction in miles per gallon (MPG) is a typical nonlinear regression problem. It uses several automobile profile attributes to predict fuel consumption. The training data is available in the University of California at Irvine Machine Learning Repository and contains data collected from automobiles of various makes and models.

This example uses the following six input data attributes to predict the output data attribute MPG with a FIS tree:

1. Number of cylinders
2. Displacement
3. Horsepower
4. Weight
5. Acceleration
6. Model year

Prepare Data

Load the data. Each row of the dataset obtained from the repository represents a different automobile profile.

data = loadgas;

data contains 7 columns, where the first six columns contain the following input attributes.

- Number of cylinders
- Displacement
- Horsepower
- Weight
- Acceleration
- Model year
The seventh column contains the output attribute, MPG.

Create separate input and output data sets, \( X \) and \( Y \), respectively.

\[
X = \text{data}(:,1:6); \newline
Y = \text{data}(:,7);
\]

Partition the input and output data sets into training data (odd-indexed samples) and validation data (even-indexed samples).

\[
\text{trnX} = X(1:2:end,:); \quad \% \text{Training input data set} \newline
\text{trnY} = Y(1:2:end,:); \quad \% \text{Training output data set} \newline
\text{vldX} = X(2:2:end,:); \quad \% \text{Validation input data set} \newline
\text{vldY} = Y(2:2:end,:); \quad \% \text{Validation output data set}
\]

Extract the range of each data attribute, which you will use for input/output range definition during FIS construction.

\[
data\text{Range} = [\text{min(data)}' \max(\text{data})'];
\]

**Construct a FIS Tree**

For this example, construct a FIS tree using the following steps:

1. Rank the input attributes based on their correlations with the output attribute.
2. Create multiple FIS objects using the ranked input attributes.
3. Construct a FIS tree from the FIS objects.

**Rank Inputs According to Correlation Coefficients**

Calculate the correlation coefficients for the training data. In the final row of the correlation matrix, the first six elements show the correlation coefficients between the six input data attributes and the output attribute.

\[
c1 = \text{corrcoef(data)}; \\
c1(end,:)
\]

\[
\text{ans} = 1\times7 \\
\quad -0.7776 \quad -0.8051 \quad -0.7784 \quad -0.8322 \quad 0.4233 \quad 0.5805 \quad 1.0000
\]

The first four input attributes have negative values, and the last two input attributes have positive values.
Rank the input attributes that have negative correlations in descending order by the absolute value of their correlation coefficients.

1. Weight
2. Displacement
3. Horsepower
4. Number of cylinders

Rank the input attributes that have positive correlations in descending order by the absolute value of their correlation coefficients.

1. Model year
2. Acceleration

These rankings show that the weight and model year have the highest negative and positive correlations with MPG, respectively.

**Create Fuzzy Inference Systems**

For this example, implement a FIS tree with the following structure.

The FIS tree uses multiple two-input-one-output FIS objects to reduce the total number of rules used in the inference process. fis1, fis2, and fis3 directly take the input values and generate intermediate MPG values, which are further combined using fis4 and fis5.

Input attributes with negative and positive correlation values are paired up to combine both positive and negative effects on the output for prediction. The inputs are grouped according to their ranks as follows:
• Weight and model year
• Displacement and acceleration
• Horsepower and number of cylinders

The last group includes only inputs with negative correlation values since there are only two inputs with positive correlation values.

This example uses Sugeno-type FIS objects for faster evaluation during the tuning process as compared to Mamdani systems. Each FIS includes two inputs and one output, where each input contains two default triangular membership functions (MFs), and the output includes 4 default constant MFs. Specify the input and output ranges using the corresponding data attribute ranges.

The first FIS combines the weight and model year attributes.

```matlab
fis1 = sugfis('Name','fis1');
fis1 = addInput(fis1,dataRange(4,:),'NumMFs',2,'Name','weight');
fis1 = addInput(fis1,dataRange(6,:),'NumMFs',2,'Name','year');
fis1 = addOutput(fis1,dataRange(7,:),'NumMFs',4);
```

The second FIS combines the displacement and acceleration attributes.

```matlab
fis2 = sugfis('Name','fis2');
fis2 = addInput(fis2,dataRange(2,:),'NumMFs',2,'Name','displacement');
fis2 = addInput(fis2,dataRange(5,:),'NumMFs',2,'Name','acceleration');
fis2 = addOutput(fis2,dataRange(7,:),'NumMFs',4);
```

The third FIS combines the horsepower and number of cylinder attributes.

```matlab
fis3 = sugfis('Name','fis3');
fis3 = addInput(fis3,dataRange(3,:),'NumMFs',2,'Name','horsepower');
fis3 = addInput(fis3,dataRange(1,:),'NumMFs',2,'Name','cylinders');
fis3 = addOutput(fis3,dataRange(7,:),'NumMFs',4);
```

The fourth FIS combines the outputs of the first and second FIS.

```matlab
fis4 = sugfis('Name','fis4');
fis4 = addInput(fis4,dataRange(7,:),'NumMFs',2);
fis4 = addInput(fis4,dataRange(7,:),'NumMFs',2);
fis4 = addOutput(fis4,dataRange(7,:),'NumMFs',4);
```

The final FIS combines the outputs of third and fourth FIS and generates the estimated MPG. This FIS has the same input and output ranges as the fourth FIS.
fis5 = fis4;
fis5.Name = 'fis5';
fis5.Outputs(1).Name = "mpg";

**Construct FIS Tree**

Connect the fuzzy systems (fis1, fis2, fis3, fis4, and fis5) according to the FIS tree diagram.

```matlab
fisTin = fistree([fis1 fis2 fis3 fis4 fis5],
    ["fis1/output1" "fis4/input1"; ...
     "fis2/output1" "fis4/input2"; ...
     "fis3/output1" "fis5/input2"; ...
     "fis4/output1" "fis5/input1"])
```

fisTin = fistree with properties:
  
  FIS: [1x5 sugfis]
  Connections: [4x2 string]
  Inputs: [6x1 string]
  Outputs: "fis5/mpg"
  DisableStructuralChecks: 0

See 'getTunableSettings' method for parameter optimization.

**Tune FIS Tree with Training Data**

Tuning is performed in two steps.

1. Learn the rule base while keeping the input and output MF parameters constant.
2. Tune the parameters of the input/output MFs and rules.

The first step is less computationally expensive due to the small number of rule parameters, and it quickly converges to a fuzzy rule base during training. In the second step, using the rule base from the first step as an initial condition provides fast convergence of the parameter tuning process.

**Learn Rules**

To learn a rule base, first specify tuning options using a `tunefisOptions` object. Global optimization methods (genetic algorithm or particle swarm) are suitable for initial
training when all the parameters of a fuzzy system are untuned. For this example, tune
the FIS tree using the particle swarm optimization method ('particleswarm').

To learn new rules, set the OptimizationType to 'learning'. Restrict the maximum
number of rules to 4. The number of tuned rules of each FIS can be less than this limit,
since the tuning process removes duplicate rules.

```matlab
options = tunefisOptions('Method','particleswarm',
    'OptimizationType','learning', ...
    'NumMaxRules',4);
```

If you have Parallel Computing Toolbox™ software, you can improve the speed of the
tuning process by setting options.UseParallel to true. If you do not have Parallel
Computing Toolbox software, set options.UseParallel to false.

Set the maximum number of iterations to 50. To reduce training error in the rule learning
process, you can increase the number of iterations. However, using too many iterations
can overtune the FIS tree to the training data, increasing the validation errors.

```matlab
options.MethodOptions.MaxIterations = 50;
```

Since particle swarm optimization uses random search, to obtain reproducible results,
initialize the random number generator to its default configuration.

```matlab
rng('default')
```

Tune the FIS tree using the specified tuning data and options. Set the input order of the
training data according to the FIS tree connections as follows: weight, year,
displacement, acceleration, horsepower, and cylinders.

```matlab
inputOrders1 = [4 6 2 5 3 1];
orderedTrnX1 = trnX(:,inputOrders1);
```

Learning rules with tunefis function takes approximately 4 minutes. For this example,
enable tuning by setting runtunefis to true. To load pretrained results without running
tunefis, you can set runtunefis to false.

```matlab
runtunefis = false;
```

Parameter settings can be empty when learning new rules. For more information, see
tunefis.

```matlab
if runtunefis
    fisTout1 = tunefis(fisTin,[],orderedTrnX1,trnY,options); %#ok<UNRCH>
```
else
    tunedfis = load('tunedfistreempgprediction.mat');
    fisTout1 = tunedfis.fisTout1;
    fprintf('Training RMSE = %.3f MPG\n',calculateRMSE(fisTout1,orderedTrnX1,trnY));
end

Training RMSE = 3.399 MPG

The Best \( f(x) \) column shows the training root-mean-squared-error (RMSE).

The learning process produces a set of new rules for the FIS tree.

fprintf("Total number of rules = %d\n",numel([fisTout1.FIS.Rules]));

Total number of rules = 17

The learned system should have similar RMSE performance for both the training and validation data sets. To calculate the RMSE for the validation data set, evaluate \( \text{fisout1} \) using validation input data set \( \text{vldX} \). To hide run-time warnings during evaluation, set all the warning options to none.

Calculate the RMSE between the generated output data and the validation output data set \( \text{vldY} \). Since the training and validation errors are similar, the learned system does not overfit the training data.

orderedVldX1 = vldX(:,inputOrders1);
plotActualAndExpectedResultsWithRMSE(fisTout1,orderedVldX1,vldY)
Tune All Parameters

After learning the new rules, tune the input/output MF parameters along with the parameters of the learned rules. To obtain the tunable parameters of the FIS tree, use the `getTunableSettings` function.

\[
\text{[in, out, rule]} = \text{getTunableSettings}(\text{fisTout1});
\]

To tune the existing FIS tree parameter settings without learning new rules, set the `OptimizationType` to 'tuning'.

\[
\text{options.OptimizationType} = \text{'tuning'};
\]
Since the FIS tree already learned rules using the training data, use a local optimization method for fast convergence of the parameter values. For this example, use the pattern search optimization method ('patternsearch').

```plaintext
options.Method = 'patternsearch';
```

Tuning the FIS tree parameters takes more iterations than the previous rule-learning step. Therefore, increase the maximum number of iterations of the tuning process to 75. As in the first tuning stage, you can reduce training errors by increasing the number of iterations. However, using too many iterations can overtune the parameters to the training data, increasing the validation errors.

```plaintext
options.MethodOptions.MaxIterations = 75;
```

Tune the FIS tree parameters using the specified tunable settings, training data, and tuning options.

Tuning parameter values with `tunefis` function takes several minutes. To load pretrained results without running `tunefis`, you can set `runtunefis` to `false`.

```plaintext
rng('default')
if runtunefis
    fisTout2 = tunefis(fisTout1,[in;out;rule],orderedTrnX1,trnY,options); %#ok<UNRCH>
else
    fisTout2 = tunedfis.fisTout2;
    fprintf('Training RMSE = %.3f MPG
',calculateRMSE(fisTout2,orderedTrnX1,trnY));
end
```

Training RMSE = 3.261 MPG

At the end of the tuning process, the training error reduces compared to the previous step.

**Check Performance**

Validate the performance of the tuned FIS tree, `fisout2`, using the validation input data set `vldX`.

Compare the expected MPG obtained from the validation output data set `vldY` and actual MPG generated using `fisout2`. Compute the RMSE between these results.

```plaintext
plotActualAndExpectedResultsWithRMSE(fisTout2,orderedVldX1,vldY)
```
Tuning the FIS tree parameters improves the RMSE compared to the results from the initial learned rule base. Since the training and validation errors are similar, the parameters values are not overtuned.

**Analyze Intermediate Data**

To gain insight into the operation of your fuzzy tree, you can add the outputs of the component fuzzy systems as outputs of your FIS tree. For this example, to access the intermediate FIS outputs, add three additional outputs to the tuned FIS tree.

```matlab
fisTout3 = fisTout2;
fisTout3.Outputs(end+1) = "fis1/output1";
fisTout3.Outputs(end+1) = "fis2/output1";
fisTout3.Outputs(end+1) = "fis3/output1";
```
To generate the additional outputs, evaluate the augmented FIS tree, fisTout3.

```matlab
t = evaluateFIS(fisTout3, orderedVldX1);
figure, plot(t(:,[2 3 4 1])), xlabel("Input dataset index"), ylabel("MPG"), axis([1 200]), legend(["Output of fis1" "Output of fis2" "Output of fis3" "Output of fis5"],...

'Location','NorthEast', 'NumColumns',2)
title("Intermediate and Final Outputs")
```

The final output of the FIS tree (fis5 output) appears to be highly correlated with the outputs of fis1 and fis3. To validate this assessment, check the correlation coefficients of the FIS outputs.

```matlab
c2 = corrcoef(t(:,[2 3 4 1]));
c2(end,:)```
ans = 1x4

0.9724   0.7935   -0.8156   1.0000

The last row of the correlation matrix shows that the outputs of \textit{fis1} and \textit{fis3} (first and third column, respectively) have higher correlations with the final output as compared to the output of \textit{fis2} (second column). This result indicates that simplifying the FIS tree by removing \textit{fis2} and \textit{fis4} and can potentially produce similar training results compared to the original tree structure.

\textbf{Simplify and Retrain FIS Tree}

Remove \textit{fis2} and \textit{fis4} from the FIS tree and connect the output of \textit{fis1} to the first input of \textit{fis5}. When you remove a FIS from a FIS tree, any existing connections to that FIS are also removed.

![FIS Tree Diagram](image)

\begin{verbatim}
fisTout3.FIS([2 4]) = [];
fisTout3.Connections(end+1,:) = ["fis1/output1" "fis5/input1"]; fis5.Inputs(1).Name = "fis1out";
\end{verbatim}

To make the number of FIS tree outputs match the number of outputs in the training data, remove the FIS tree outputs from \textit{fis1} and \textit{fis3}.

\begin{verbatim}
fisTout3.Outputs(2:end) = [];
\end{verbatim}

Update the input training data order according to the new FIS tree input configuration.

\begin{verbatim}
inputOrders2 = [4 6 3 1]; orderedTrnX2 = trnX(:,inputOrders2);
\end{verbatim}
Since the FIS tree configuration is changed, you must rerun both the learning and tuning steps. In the learning phase, the existing rule parameters are also tuned to fit the new configuration of the FIS tree.

```matlab
options.Method = "particleswarm";
options.OptimizationType = "learning";
options.MethodOptions.MaxIterations = 50;
 [~,~,rule] = getTunableSettings(fisTout3);

rng('default')
if runtunefis
    fisTout4 = tunefis(fisTout3,rule,orderedTrnX2,trnY,options); %#ok<UNRCH>
else
    fisTout4 = tunedfis.fisTout4;
    fprintf('Training RMSE = %.3f MPG\n',calculateRMSE(fisTout4,orderedTrnX2,trnY));
end

Training RMSE = 2.929 MPG

In the training phase, the parameters of the membership function and rules are tuned.

```matlab
options.Method = "patternsearch";
options.OptimizationType = "tuning";
options.MethodOptions.MaxIterations = 75;
 [in,out,rule] = getTunableSettings(fisTout4);
 rng('default')
 if runtunefis
     fisTout5 = tunefis(fisTout4,[in;out;rule],orderedTrnX2,trnY,options); %#ok<UNRCH>
 else
     fisTout5 = tunedfis.fisTout5;
     fprintf('Training RMSE = %.3f MPG\n',calculateRMSE(fisTout5,orderedTrnX2,trnY));
 end

Training RMSE = 2.828 MPG
```

At the end of the tuning process, the FIS tree contains updated MF and rule parameter values. The rule base size of the new FIS tree configuration is smaller than the previous configuration.

```matlab
fprintf("Total number of rules = %d\n",numel([fisTout5.FIS.Rules]));

Total number of rules = 11
Check Performance of the Simplified FIS Tree

Evaluate the updated FIS tree using the four input attributes of the checking dataset.

```matlab
orderedVldX2 = vldX(:,inputOrders2);
plotActualAndExpectedResultsWithRMSE(fisTout5,orderedVldX2,vldY)
```

The simplified FIS tree with four input attributes produces better results in terms of RMSE as compared to the first configuration, which uses six input attributes. Therefore, it shows that a FIS tree can be represented with fewer number of inputs and rules to generalize the training data.
Conclusion

You can further improve the training error of the tuned FIS tree by:

- Increasing number of iterations in both the rule-learning and parameter-tuning phases. Doing so increases the duration of the optimization process and can also increase validation error due to overtuned system parameters with the training data.

- Using global optimization methods, such as `ga` and `particleswarm`, in both rule-learning and parameter-tuning phases. `ga` and `particleswarm` perform better for large parameter tuning ranges since they are global optimizers. On the other hand, `patternsearch` and `simulannealbnd` perform better for small parameter ranges since they are local optimizers. If rules are already added to a FIS tree using training data, then `patternsearch` and `simulannealbnd` may produce faster convergence as compared to `ga` and `particleswarm`. For more information on these optimization methods and their options, see `ga`, `particleswarm`, `patternsearch`, and `simulannealbnd`.

- Changing the FIS properties, such as the type of FIS, number of inputs, number of input/output MFs, MF types, and number of rules. For fuzzy systems with a large number of inputs, a Sugeno FIS generally converges faster than a Mamdani FIS since a Sugeno system has fewer output MF parameters (if constant MFs are used) and faster defuzzification. Small numbers of MFs and rules reduce the number of parameters to tune, producing a faster tuning process. Furthermore, a large number of rules may overfit the training data.

- Modifying tunable parameter settings for MFs and rules. For example, you can tune the support of a triangular MF without changing its peak location. Doing so reduces the number of tunable parameters and can produce a faster tuning process for specific applications. For rules, you can exclude zero MF indices by setting the `AllowEmpty` tunable setting to `false`, which reduces the overall number of rules during the learning phase.

- Changing FIS tree properties, such as number of fuzzy systems and connections between the fuzzy systems.

- Using different ranking and grouping of the inputs to the FIS tree.

Local Functions

```matlab
function plotActualAndExpectedResultsWithRMSE(fis,x,y)

% Calculate RMSE between actual and expected results
[rms,actY] = calculateRMSE(fis,x,y);
```
% Plot results
figure
subplot(2,1,1)
hold on
bar(actY)
bar(y)
bar(min(actY,y), 'FaceColor', [0.5 0.5 0.5])
hold off
axis([0 200 0 60])
xlabel("Validation input dataset index"), ylabel("MPG")
legend(["Actual MPG" "Expected MPG" "Minimum of actual and expected values"],...
'Location', 'NorthWest')
title("RMSE = " + num2str(rmse) + " MPG")

subplot(2,1,2)
bar(actY-y)
xlabel("Validation input dataset index"), ylabel("Error (MPG)")
title("Difference Between Actual and Expected Values")
end

function [rmse,actY] = calculateRMSE(fis,x,y)
% Evaluate FIS
actY = evaluateFIS(fis,x);

% Calculate RMSE
del = actY - y;
rmse = sqrt(mean(del.^2));
end

function y = evaluateFIS(fis,x)
% Specify options for FIS evaluation
persistent evalOptions
if isempty(evalOptions)
    evalOptions = evalfisOptions("EmptyOutputFuzzySetMessage","none", ... 
        "NoRuleFiredMessage","none", "OutOfRangeInputValueMessage","none");
end

% Evaluate FIS
y = evalfis(fis,x,evalOptions);
See Also
fistree | getTunableSettings | sugfis | tunefis

More About
• “Tuning Fuzzy Inference Systems” on page 3-2
• “Tune Mamdani Fuzzy Inference System” on page 3-33
Predict Chaotic Time Series Using Type-2 FIS

This example shows chaotic time series prediction using a tuned type-2 fuzzy inference system (FIS). This example tunes the FIS using particle swarm optimization, which requires Global Optimization Toolbox™ software.

Time Series Data

This example simulates times-series data using the following form of the Mackey-Glass (MG) nonlinear delay differential equation.

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

Simulate the time series for 1200 samples using the following configuration:

- Sample time \( t_s = 1 \) sec
- Initial condition \( x(0) = 1.2 \)
- \( \tau = 20 \)
- \( x(t - \tau) = 0 \) for \( t < \tau \).

```matlab
ts = 1;
numSamples = 1200;
tau = 20;
x = zeros(1,numSamples+tau+1);
x(tau+1) = 1.2;
for t = 1+tau:numSamples+tau
    x_dot = 0.2*x(t-tau)/(1+(x(t-tau))^10)-0.1*x(t);
    x(t+1) = x(t) + ts*x_dot;
end
```

Plot the simulated MG time-series data.

```matlab
figure(1)
plot(x(tau+2:end))
title('Mackey-Glass Chaotic Time Series')
xlabel('Time (sec)')
ylabel('x(t)')
```
Generate Training and Validation Data

Time-series prediction uses known time-series values up to time $t$ to predict a future value at time $t + P$. The standard method for this type of prediction is to create a mapping from $D$ sample data points, sampled every $\Delta$ units in time $(x(t - (D - 1)\Delta), \ldots, x(t - \Delta), x(t))$ to a predicted future value $x = (t + P)$. For this example, set $D = 4$ and $\Delta = P = 1$. Hence, for each $t$, the input and output training data sets are $[x(t - 3), x(t - 2), x(t - 1), x(t)]$ and $x(t + 1)$, respectively. In other words, use four successive known time-series values to predict the next value.

Create 1000 input/output data sets from samples $x(100 + D - 1)$ to $x(1100 + D - 2)$.

```matlab
D = 4;
inputData = zeros(1000,D);
```
outputData = zeros(1000,1);
for t = 100+D-1:1:1100+D-2
    for i = 1:D
        inputData(t-100-D+2,i) = x(t-D+i);
    end
    outputData(t-100-D+2,:) = x(t+1);
end

Use the first 500 data sets as training data (trnX and trnY) and the second 500 sets as validation data (vldX and vldY).

trnX = inputData(1:500,:);
trnY = outputData(1:500,:);
vldX = inputData(501:end,:);
vldY = outputData(501:end,:);

Construct FIS

This example uses a type-2 Sugeno FIS. Since a Sugeno FIS has fewer tunable parameters than a Mamdani FIS, a Sugeno system generally converges faster during optimization.

fisin = sugfistype2;

Add three inputs, each with three default triangular membership functions (MFs). Initially, eliminate the footprint of uncertainty (FOU) for each input MF by setting each lower MF equal to its corresponding upper MF. To do so, set the scale and lag values of each lower MF to 1 and 0, respectively. By eliminating the FOU for all input membership functions, you configure the type-2 FIS to behave like a type-1 FIS.

numInputs = D;
numInputMFs = 3;
range = [min(x) max(x)];
for i = 1:numInputs
    fisin = addInput(fisin,range,'NumMFs',numInputMFs);
    for j = 1:numInputMFs
        fisin.Inputs(i).MembershipFunctions(j).LowerScale = 1;
        fisin.Inputs(i).MembershipFunctions(j).LowerLag = 0;
    end
end

For prediction, add an output to the FIS. The output contains default constant membership functions. To provide maximum resolution for the input-output mapping, set the number of output MFs equal to the number of input MF combinations.
numOutputMFs = numInputMFs^numInputs;
fisin = addOutput(fisin,range,'NumMFs',numOutputMFs);

View the FIS structure. Initially, the FIS has zero rules. The rules of the system are found during the tuning process.

plotfis(fisin)

**Tune FIS with Training Data**

To tune the FIS, you use the following three steps.

1. Learn the rule base while keeping the input and output MF parameters constant.
2 Tune the output MF parameters and the upper MF parameters of the inputs while keeping the rule and lower MF parameters constant.

3 Tune the lower MF parameters of the inputs while keeping the rule, output MF, and upper MF parameters constant.

The first step is less computationally expensive due to the small number of rule parameters, and it quickly converges to a fuzzy rule base during training. After the second step, the system is a trained type-1 FIS. The third step produces a tuned type-2 FIS.

**Learn Rules**

To learn a rule base, first specify tuning options using a `tunefisOptions` object.

```matlab
options = tunefisOptions;
```

Since the FIS does not contain any pretuned fuzzy rules, use a global optimization method (genetic algorithm or particle swarm) to learn the rules. Global optimization methods perform better in large parameter tuning ranges as compared to local optimization methods (pattern search and simulated annealing). For this example, tune the FIS using particle swarm optimization (`'particleswarm'`).

```matlab
options.Method = 'particleswarm';
```

To learn new rules, set the `OptimizationType` to `'learning'`.

```matlab
options.OptimizationType = 'learning';
```

Restrict the maximum number of rules to the number of input MF combinations. The number of tuned rules can be less than this limit, since the tuning process removes duplicate rules.

```matlab
options.NumMaxRules = numInputMFs^numInputs;
```

If you have Parallel Computing Toolbox™ software, you can improve the speed of the tuning process by setting `UseParallel` to `true`. If you do not have Parallel Computing Toolbox software, set `UseParallel` to `false`.

```matlab
options.UseParallel = false;
```

Set the maximum number of iterations to `10`. Increasing the number of iterations can reduce training error. However, the larger number of iterations increases the duration of the tuning process and can overtrain the rule parameters to the training data.
options.MethodOptions.MaxIterations = 10;

Since particle swarm optimization uses random search, to obtain reproducible results, initialize the random number generator to its default configuration.

rng('default')

Tuning a FIS using the tunefis function takes several minutes. For this example, enable tuning by setting runtunefis to true. To load pretrained results without running tunefis, you can set runtunefis to false.

runtunefis = false;

Tune the FIS using the specified training data and options.

if runtunefis
    fisout1 = tunefis(fisin,[],trnX,trnY,options);
else
    tunedfis = load('tunedfischaotictimeseriesestype2.mat');
    fisout1 = tunedfis.fisout1;
end

View the structure of the trained FIS, which contains the new learned rules.

plotfis(fisout1)
Check the individual input-output relationships tuned by the learned rulebase. For example, the following figure shows the relationship between the second input and the output.

gensurf(fisout1,gensurfOptions('InputIndex',2))
Evaluate the tuned FIS using the input validation data. Plot the actual generated output with the expected validation output, and compute the root-mean-square-error (RMSE).

plotActualAndExpectedResultsWithRMSE(fisout1,vldX,vldY)
Tune Upper Membership Function Parameters

Tune the upper membership function parameters. A type-2 Sugeno FIS supports only crisp output functions. Therefore, this step tunes input upper MFs and crisp output functions.

Obtain the input and output parameter settings using `getTunableSettings`. Since the FIS uses triangular input MFs, you can tune the input MFs using asymmetric lag values.

```
[in,out] = getTunableSettings(fisout1,'AsymmetricLag',true);
```

Disable the tuning of lower MF parameters.

```
for i = 1:length(in)
    for j = 1:length(in(i).MembershipFunctions)
```

RMSE = 0.12635
To optimize the existing tunable MF parameters while keeping the rulebase constant, set `OptimizationType` to 'tuning'.

```matlab
options.OptimizationType = 'tuning';
```

Tune the FIS using the specified tuning data and options. To load pretrained results without running `tunefis`, you can set `runtunefis` to `false`.

```matlab
rng('default')
if runtunefis
    fisout2 = tunefis(fisout1,[in;out],trnX,trnY,options);
else
    tunefis = load('tunedfischaotictimeseriestype2.mat');
    fisout2 = tunefis.fisout2;
end
```

View the structure of the trained FIS, which now contains tuned upper MF parameters.

```matlab
plotfis(fisout2)
```
Evaluate the tuned FIS using the validation data, compute the RMSE, and plot the actual generated output with the expected validation output.

plotActualAndExpectedResultsWithRMSE(fisout2, vldX, vldY)
Tuning the upper MF parameters improves the performance of the FIS. This result is equivalent to tuning a type-1 FIS.

**Tune Lower Membership Function Parameters**

Tune only the input lower MF parameters. To do so, set the lower scale and lag values tunable, and disable tuning of the upper MF parameters.

```matlab
for i = 1:length(in)
    for j = 1:length(in(i).MembershipFunctions)
        in(i).MembershipFunctions(j).UpperParameters.Free = false;
        in(i).MembershipFunctions(j).LowerScale.Free = true;
        in(i).MembershipFunctions(j).LowerLag.Free = true;
```
Tune the FIS using the specified tuning data and options. To load pretrained results without running `tunefis`, you can set `runtunefis` to `false`.

```matlab
rng('default')
if runtunefis
    fisout3 = tunefis(fisout2,in,trnX,trnY,options);
else
    tunedfis = load('tunedfisiaotchotictimeseriestype2.mat');
    fisout3 = tunedfis.fisout3;
end
```

View structure of the trained FIS, which now contains tuned lower MF parameters.

```matlab
plotfis(fisout3)
```
Evaluate the tuned FIS using the validation data, compute the RMSE, and plot the actual generated output with the expected validation output.

`plotActualAndExpectedResultsWithRMSE(fisout3, vldX, vldY)`
Tuning both the upper and lower MF values improves the FIS performance. The RMSE improves when the trained FIS includes both tuned upper and lower parameter values.

**Conclusion**

Type-2 MFs provides additional tunable parameters as compared to type-1 MFs. Therefore, with adequate training data, a tuned type-2 FIS can fit the training data better than a tuned type-1 FIS.

Overall, you can produce different tuning results by modifying any of the following FIS properties or tuning options:

- Number of inputs
- Number of MFs
- Type of MFs
- Optimization method
- Number of tuning iterations

**Local Functions**

```matlab
function [rmse,actY] = calculateRMSE(fis,x,y)
% Specify options for FIS evaluation
evalOptions = evalfisOptions("EmptyOutputFuzzySetMessage","none", ...
  "NoRuleFiredMessage","none","OutOfRangeInputValueMessage","none");
% Evaluate FIS
actY = evalfis(fis,x,evalOptions);
% Calculate RMSE
del = actY - y;
rmse = sqrt(mean(del.^2));
end
```

```matlab
function plotActualAndExpectedResultsWithRMSE(fis,vldX,vldY)
[rmse,actY] = calculateRMSE(fis,vldX,vldY);
figure
plot([actY vldY])
axis([0 length(vldY) min(vldY)-0.01 max(vldY)+0.13])
xlabel('sample index')
ylabel('signal value')
title(['RMSE = ' num2str(rmse)])
legend(['Actual output' 'Expected output'],'Location','northeast')
end
```

**See Also**
Tune Fuzzy Systems using Custom Cost Function

This example shows how to tune a fuzzy inference system (FIS) using a custom cost function. The example requires Global Optimization Toolbox™ software.

**Problem Description**

You use a custom cost function to learn robot navigation in a simulation environment. The goal of the navigation task is to reach a specified target while avoiding obstacles. The direction to the target is represented as a unit force vector \( \vec{F}_t \) directed from the robot to a target location. The obstacle avoidance direction is represented by a unit force vector \( \vec{F}_o \) directed towards the robot from the closest obstacle location.
The robot, target, and obstacle are shown as circles with 0.5 m radius in the 25 m x 25 m simulation environment. The navigation task is to combine the force vectors such that the direction $\theta$ of the resultant force vector $F$ provides a collision-free direction for the robot.

$$F = wF_o + (1 - w)F_t,$$ where $0 \leq w \leq 1$

$$\theta = \angle F$$

This example assumes a robot with differential kinematics for the simulation. In other words, the robot can rotate on its center without any constraints. However, to avoid sharp turns, the change per time step in the robot direction is limited to $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$. Therefore, if the current robot heading direction is $\theta_r(k)$, the next heading direction is calculated as:

$$\theta_r(k + 1) = \theta_r(k) + \min\left(\max\left(\theta - \theta_r(k), \frac{\pi}{4}\right), -\frac{\pi}{4}\right).$$

The weight $w$ of the force vector $F_o$ is calculated using function $f_w$:

$$w = f_w(\alpha, \theta_{t, o})$$

where

- $\alpha = \frac{d_o}{d_t}$ is the ratio of the robot-to-obstacle distance ($d_o$) and the robot-to-target-distance ($d_t$)
- $\theta_{t, o}$ is the absolute difference between the target and obstacle directions with respect to the robot

To achieve the navigation task, the function $f_w$ must generate high $w$ values, that is, focus on avoiding the obstacle when:

- Both the target and obstacle directions from the robot are similar ($\theta_{t, o}$ is low)
- The obstacle is closer to the robot than the target ($\alpha$ is low).

Otherwise, $f_w$ must generate low $w$ values, that is, focus on reaching the target.
The goal of this example is to design a FIS that learns fuzzy rules and optimizes the FIS parameters to model the function $f_w$ for collision-free robot navigation in the simulation environment.

**Assumptions**

The following assumptions apply for the robot simulation:

- The robot can perfectly localize in the simulation environment; that is, the robot knows its current position in the simulation environment.
- The robot is equipped with perfect sensors to identify the obstacle and determine its location.
- The robot has no dynamic constraints; that is, the robot can rotate and move as commanded without any mechanical constraints. To avoid sharp turns, a soft constraint is imposed on rotation, which limits the change per time step in the robot heading to $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$.
- The robot runs with a fixed speed. You can include additional fuzzy systems to control the robot speed. For simplicity, this example uses a fixed speed for the robot.

**Construct Fuzzy Systems**

To model function $f_w$, construct a FIS as shown in the following figure. For this example, use a Mamdani FIS.

\[
\begin{align*}
\alpha & \rightarrow \text{FIS} \\
\theta_{t,o} & \rightarrow \text{FIS} \\
\rightarrow & \ w
\end{align*}
\]

\[
\text{fisin} = \text{mamfis};
\]

Add the following two inputs as shown in the previous figure.

- $\alpha$ — Ratio of distances, robot-to-obstacle / robot-to-target
- $\theta_{t,o}$ — Difference between target and obstacle directions

Set the range of the first input to $[0, 2]$, which indicates that $\alpha$ contributes to obstacle avoidance when the obstacle distance is less than or equal to twice the target distance.
Set the range of the second input to \([\theta, \pi/2]\), which indicates that \(\theta_{t,o}\) contributes to obstacle avoidance when the difference between the target and obstacle directions is less than or equal to \(\pi/2\).

\[
\text{fisin} = \text{addInput}(\text{fisin},[0 \ 2], 'Name','alpha'); \\
\text{fisin} = \text{addInput}(\text{fisin},[0 \ \pi/2], 'Name','\theta_{t,o}');
\]

To minimize the number of rules, which corresponds to the number of combinations of input membership functions, add two membership functions (MFs) to each input. To generate similar membership values beyond the input ranges, use \(\text{zmf}\) (Z-shaped curve membership function) and \(\text{smf}\) (S-shaped curve membership function) MFs. The tuning process optimizes the input MF parameter values.

Add membership functions to the first input.

\[
\text{fisin} = \text{addMF}(\text{fisin},'alpha','\text{zmf}',[0 \ 2], 'Name','low'); \\
\text{fisin} = \text{addMF}(\text{fisin},'alpha','\text{smf}',[0 \ 2], 'Name','high');
\]

Add membership functions to the second input.

\[
\text{fisin} = \text{addMF}(\text{fisin},'\theta_{t,o}','\text{zmf}',[0 \ \pi/2], 'Name','low'); \\
\text{fisin} = \text{addMF}(\text{fisin},'\theta_{t,o}','\text{smf}',[0 \ \pi/2], 'Name','high');
\]

Add an output to the FIS or the obstacle force vector weight, restricting the weight values to the range \([0, 1]\).

\[
\text{fisin} = \text{addOutput}(\text{fisin},[0 \ 1], 'Name','w');
\]

Add two MFs to the output. You can add more MFs to the output for finer granularity of output values. However, doing so increases the number of tuning parameters. The output MFs also use \(\text{zmf}\) and \(\text{smf}\) to generate similar membership values beyond the input ranges. The tuning process optimizes the output MF parameter values.

\[
\text{fisin} = \text{addMF}(\text{fisin},'w','\text{zmf}',[0 \ 1], 'Name','low'); \\
\text{fisin} = \text{addMF}(\text{fisin},'w','\text{smf}',[0 \ 1], 'Name','high');
\]

View the FIS structure. Initially, the FIS has zero rules. The tuning process finds rules for the fuzzy system.

\[
\text{figure} \\
\text{plotfis(fisin)}
\]
Learn Rules and Optimize FIS Parameters

Since you do not have training data for this example, you simulate the robot navigation using a custom cost function. The tuning process uses this custom cost function when optimizing the FIS parameters.

For parameter optimization, obtain the parameter settings from the FIS.

\[ \text{[in, out]} = \text{getTunableSettings(fisin)}; \]

Next, create tuning options with OptimizationType set to learning. This example uses the genetic algorithm (ga) optimization method for the tuning process. To improve the speed of the tuning process, set the UseParallel option to true, which requires
Parallel Computing Toolbox™ software. If you do not have Parallel Computing Toolbox software, set UseParallel to false.

```matlab
options = tunefisOptions('Method','ga','OptimizationType','learning');
```

Set the population size of the genetic algorithm to 200. The larger population size increases the probability of generating a better solution in fewer generations.

```matlab
options.MethodOptions.PopulationSize = 200;
```

Set the maximum number of generations to 25. To tune the parameters further, you can set a higher number of generations. However, doing so increases the duration of the tuning process and can overtune the parameter values.

```matlab
options.MethodOptions.MaxGenerations = 25;
```

Set the locations of the obstacle, target, and robot for the training environments. Set the initial heading of the robot to \( \pi/2 \) for the training environment. To learn navigation both with and without obstacle avoidance on the way to the target location, use two training tasks with different obstacle locations.

```matlab
trnObstacle = [3 12;13 18];
trnTarget = [13 22;13 22];
trnRobot = [13 2 pi/2;13 2 pi/2;];
```

```matlab
showSimulationEnvironmentsForTraining(trnObstacle,trnTarget,trnRobot)
```

Specify the custom cost function using a function handle.

```matlab
costFunction = @(fis)navigationCostFcn(fis,trnObstacle,trnTarget,trnRobot);
```
In the cost function, the robot navigation is simulated in the training environments using each FIS from the population. Each navigation task is run for 100 iterations, where each iteration is equivalent to a decision cycle of length 1 s. The robot uses a fixed speed of 0.5 m/s throughout the navigation task. For more simulation details, see the `getNavigationResults` function.

```matlab
function cost = navigationCostFcn(fis,obstacle,target,robot)
    cost = 0;
    for i = 1:size(obstacle,1)
        results = getNavigationResults(fis,obstacle(i,:),target(i,:),robot(i,:));
        cost = cost + getNavigationCost(results);
    end
end
```

The cost of each navigation task is the total distance traveled by the robot. If the robot does not reach the target or collides with the obstacle, a high cost value (200) is assigned for the simulation.

```matlab
function cost = getNavigationCost(results)
    if results.notSafe || ~results.reachedTarget
        cost = 200;
    else
        cost = results.travelledDistance;
    end
end
```

Since genetic algorithm optimization uses random search, to obtain reproducible results, initialize the random number generator to its default configuration.

```matlab
rng('default')
```
Learning rules using the `tunefis` function takes approximately 10 minutes. For this example, enable tuning by setting `runtunefis` to `true`. To load pretrained results without running `tunefis`, you can set `runtunefis` to `false`.

```matlab
runtunefis = false;
```

Tune the FIS using the specified training environments and tuning options.

```matlab
if runtunefis
    fisout = tunefis(fisin,[in;out],costFunction,options); %#ok<UNRCH>
else
    tunedfis = load('tunedfisnavigation.mat');
    fisout = tunedfis.fisout;
end
```

The tuned FIS produces the following robot trajectories in the simulation environments.

```matlab
showNavigationTrajectories(fisout,trnObstacle,trnTarget,trnRobot)
```

In the first case, the robot reaches the target with minimum distance since the obstacle is not on the path to the target. In the second case, the robot successfully avoids the obstacle and reaches the target.

View the structure of the tuned FIS, `fisout`.

```matlab
figure
plotfis(fisout)
```
The tuning process produces a set of new rules for the FIS.

\texttt{fisout.Rules}

The rules are described as follows with respect to the expected behaviors of $f_w$:

- Rule 1 is consistent with the expected behavior of $f_w$. When the obstacle is not located in front of the robot on the way to the target ($\theta_{t,o}$ is high) and the obstacle is close ($\alpha$ is low), this rule produces low weight values.
- Rule 4 is also consistent with the expected behavior of $f_w$. When the obstacle is not located in front of the robot on the way to the target ($\theta_{t,o}$ is high) and the obstacle is farther away ($\alpha$ is high), this rule produces low weight values.
• Rule 3 generates low weight values when the obstacle is not located in front of the robot ($\theta_{t,o}$ is high), irrespective of the obstacle distance. This rule covers the conditions for both rule 1 and rule 4. Therefore, rules 1 and 4 are redundant and can be removed.

• Rule 2 is also consistent with the expected behavior of $f_w$. When the obstacle is close to the robot ($\alpha$ is low) and is located in front of the robot on the way to the target ($\theta_{t,o}$ is low), this rule produces high weight values for the obstacle avoidance task.

• Rule 5 generates low weight values when the obstacle distance is low. This rule contradicts rule 2 when $\theta_{t,o}$ is low. In this case, the output of rule 5 does not contribute to the final output due to the high output values of rule 2. Therefore, rule 5 can also be removed.

Remove the redundant rules.

```matlab
fisoutpruned = fisout;
fisoutpruned.Rules([1 4 5]) = [];
fisoutpruned.Rules
```

`fisout` and `fisoutpruned` generate similar control surfaces. Therefore, only two rules are necessary for obstacle avoidance in the simulation environment.

```matlab
figure
subplot(1,2,1)
gensurf(fisout)
title('Output surface of fisout')
subplot(1,2,2)
gensurf(fisoutpruned)
title('Output surface of fisoutpruned')
```
Check Performance

Validate the performance of the tuned FIS with different positions of the obstacle, robot, and target. In the following validation cases, the robot successfully avoids the obstacle to reach the target position using the tuned FIS.

```
vldObstacle = [13 5; 10 10; 8 8];
vldRobot = [5 12; 0 5 20 -pi/2; 19 19 -pi];
vldTarget = [23 12; 15 4; 5 5];

showNavigationTrajectories(fisoutpruned, vldObstacle, vldTarget, vldRobot)
```
Conclusion

This example uses a custom cost function that simulates robot navigation in different training environments to learn fuzzy rules and optimize the FIS parameter values for collision-free navigation. You can include more training environments to learn obstacle avoidance in other scenarios, for example narrow corridor and U-shape obstacles. In these scenarios, the robot may need additional navigation subtasks, such as wall following and subtarget (intermediate landmark) reaching, for successful collision-free navigation to the target. Complex environments also require additional terms in the cost calculation for safe navigation. For example, in a narrow corridor the robot should stay in the middle; that is, the distances to the obstacles on the left and right should be the same.

Using a custom cost function with `tunefis` provides the flexibility of simulating a custom system with custom cost calculation. However, the tradeoff is a lengthy tuning process due to the number of simulations required (for each set of optimized parameter values). Therefore, if possible, to expedite the tuning process, use training data. For instance, the tuning process in this example would run faster if input/output decision data of a human operator was available for tuning the FIS.

Local Functions

```matlab
function showSimulationEnvironment(obstacle,target,robot,navigationResults)
% Show the robot trajectory in the simulation environment.
%
% Radius of the robot, target, and obstacle.
radius = 1; % 1m
%
% Use 25mx25m simulation environment.
axis([0 25 0 25]);
```
% Set equal aspect ratio.
pbaspect([1 1 1])

% Temporary plots to enable legends.
hold on
plot(robot(1),robot(2)+radius,'ob','LineWidth',radius*1,'MarkerFaceColor','b')
plot(robot(1),robot(2)+radius,'or','LineWidth',radius*1,'MarkerFaceColor','r')
plot(robot(1),robot(2)+radius,'og','LineWidth',radius*1,'MarkerFaceColor','g')
hold off

% Draw obstacle.
rectangle('Position',[obstacle(1)-0.5*radius obstacle(2)+0.5*radius radius radius],'Curvature',[1 1], ...  
    'FaceColor','b','EdgeColor','b')

% Draw target.
rectangle('Position',[target(1)-0.5*radius target(2)+0.5*radius radius radius],'Curvature',[1 1], ...  
    'FaceColor','r','EdgeColor','r')

% Draw robot.
rectangle('Position',[robot(1)-0.5*radius robot(2)+0.5*radius radius radius],'Curvature',[1 1], ...  
    'FaceColor','g','EdgeColor','g')

% Add labels, title, and legends.
xlabel('x (m)'),ylabel('y (m)')
title('Simulation Environment for Robot Navigation')
legend(["Obstacle" "Target" "Robot"])

% Plot the robot trajectory if specified.
if nargin == 4
    x = navigationResults.x;
y = navigationResults.y;
    for i = 1:numel(x)
        rectangle('Position',[x(i)-0.5*radius y(i)+0.5*radius radius radius], ...  
            'Curvature',[1 1],'FaceColor','g','EdgeColor','g')
    end
end
function showNavigationTrajectories(fis, obstacle, target, robot)

% Show robot trajectories in the simulation environments.
drawEnvironmentAndShowTrajectory(obstacle, target, robot, 'Navigation Task', fis)
end

function drawEnvironmentAndShowTrajectory(obstacle, target, robot, plotTitle, varargin)

% Expand figure horizontally to tile the simulation environments.
h = figure;
h.Position = [h.Position(1:2) 3*h.Position(3) h.Position(4)];
numTasks = size(target, 1);

% Draw each simulation environment.
for i = 1:numTasks
    o = obstacle(i, :);
    t = target(i, :);
    r = robot(i, :);
    subplot(1, numTasks, i)
    if ~isempty(varargin)
        results = getNavigationResults(varargin{1}, o, t, r);
        showSimulationEnvironment(o, t, r, results)
    else
        showSimulationEnvironment(o, t, r)
    end
    title([plotTitle ': ' num2str(i)])
extend
end
end

See Also
getTunableSettings | mamfis | tunefis

More About
• “Tuning Fuzzy Inference Systems” on page 3-2
• “Tune Mamdani Fuzzy Inference System” on page 3-33
Neuro-Adaptive Learning and ANFIS

Suppose that 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. Also, assume that you do not necessarily have a predetermined model structure based on the characteristics of variables in your system. In some modeling situations, discerning membership functions parameters by looking at data can be difficult or impossible. In these cases, rather than choosing the parameters associated with a given membership function arbitrarily, you can tailor the membership function parameters to the input/output data. Using Fuzzy Logic Toolbox software, you can tune Sugeno fuzzy inference systems using neuro-adaptive learning techniques similar to those used for training neural networks.

Fuzzy Logic Toolbox software provides a command-line function (anfis) and an interactive app (Neuro-Fuzzy Designer) for training an adaptive neuro-fuzzy inference system (ANFIS).

FIS Structure

Using ANFIS training methods, you can train Sugeno systems with the following properties:

- Single output
- Weighted average defuzzification
- First or zeroth order system; that is, all output membership functions must be the same type, either 'linear' or 'constant'.
- No rule sharing. Different rules cannot use the same output membership function; that is, the number of output membership functions must equal the number of rules.
- Unity weight for each rule.
- No custom membership functions or defuzzification methods.

To create such a fuzzy system in the MATLAB workspace, you can:

- Use the genfis function. When using this method, you can create your system using either grid partitioning or subtractive clustering. Grid partitioning can produce a large number of rules when the number of inputs reaches four or five. To reduce the number of rules, consider using the subtractive clustering method.
- Use the Fuzzy Logic Designer app, and export the FIS to the MATLAB workspace.
• Use the sugfis function.
• Load a system from a file using the readfis function.

When training your system using the anfis function, specify the initial structure by creating an anfisOptions option set and setting the InitialFIS property. If you do not specify this property, the anfis function derives the FIS structure using grid partitioning.

When using the Neuro-Fuzzy Designer, in the Generate FIS section, you can create your FIS by:
• Loading from a file (select Load from file)
• Loading from the MATLAB workspace (select Load from worksp)
• Using grid partitioning (select Grid partition)
• Using subtractive clustering (select Sub. clustering)

**Training Data**

To train a fuzzy system using neuro-adaptive methods, you must collect input/output training data using experiments or simulations of the system you want to model. In general, ANFIS training works well if the training data is fully representative of the features of the data that the trained FIS is intended to model.

To specify your training data, you can:

• Create an array in the MATLAB workspace. Each row contains a data point, with the final column containing the output value and the remaining columns containing input values. You can then pass this data to the trainingData input argument of the anfis function or load it into the Neuro-Fuzzy Designer app.
• Load the data from a .dat file. Each line of the file contains a data point with values separated by white space. The final value on each line is the output, and the remaining values are the inputs.

When using the anfis function, create or load the input data and pass it to the trainingData input argument.

When using Neuro-Fuzzy Designer, in the Load data section, select Training, and then:
• To load data from a file, select **file**.
• To load data from the MATLAB workspace, select **worksp**.

### Training Options

Both `anfis` and the **Neuro-Fuzzy Designer** allow you to adjust the optimization method, number of training epochs, and training error goal. However, `anfis` provides additional training options to control the training step size.

<table>
<thead>
<tr>
<th>Option</th>
<th><code>anfisOptions Property</code></th>
<th>Neuro-Fuzzy Designer Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Method</td>
<td><code>OptimizationMethod</code></td>
<td>In the <strong>Train FIS</strong> section, specify <strong>Optim. Method</strong>.</td>
</tr>
<tr>
<td>Number of training epochs</td>
<td><code>EpochNumber</code></td>
<td>In the <strong>Train FIS</strong> section, specify <strong>Epochs</strong>.</td>
</tr>
<tr>
<td>Training error goal</td>
<td><code>ErrorGoal</code></td>
<td>In the <strong>Train FIS</strong> section, specify <strong>Error Tolerance</strong>.</td>
</tr>
<tr>
<td>Initial step size</td>
<td><code>InitialStepSize</code></td>
<td>Not available</td>
</tr>
<tr>
<td>Step-size decrease rate</td>
<td><code>StepSizeDecreaseRate</code></td>
<td></td>
</tr>
<tr>
<td>Step-size increase rate</td>
<td><code>StepSizeIncreaseRate</code></td>
<td></td>
</tr>
</tbody>
</table>

#### Optimization Method

To train a fuzzy system using ANFIS, the Fuzzy Logic Toolbox software uses a back-propagation algorithm either alone or in combination with a least-squares algorithm. This training process tunes the membership function parameters of a FIS such that the system models your input/output data.

The following table shows the two methods that both `anfis` and the **Neuro-Fuzzy Designer** use for updating membership function parameters.

<table>
<thead>
<tr>
<th>Optimization Method</th>
<th><code>anfisOptions Setting</code></th>
<th>Neuro-Fuzzy Designer Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backpropagation for all parameters (a steepest-descent method)</td>
<td><code>OptimizationMethod = 'backpropagation'</code></td>
<td>In the <strong>Train FIS</strong> section, under <strong>Optim. Method</strong>, select backpropa.</td>
</tr>
</tbody>
</table>
Optimization Method | anfisOptions Setting | Neuro-Fuzzy Designer Setting
--- | --- | ---
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 | OptimizationMethod = 'hybrid' | In the Train FIS section, under Optim. Method, select hybrid.

### Step Size

When training using the `anfis` function, you can adjust the training step size options. During training, the software updates the step-size according to the following rules:

- If the error undergoes four consecutive reductions, increase the step size by multiplying it by a constant (StepSizeIncreaseRate) 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 (StepSizeDecreaseRate) less than one.

Ideally, the step size increases at the start of training, reaches a maximum, and then decreases for the remainder of the training. To achieve this step size profile, adjust the initial step size (InitialStepSize), step-size increase rate, and step-size decrease rate.

### Display Options

When training using the `anfis` function, you can specify what training progress information to display in the MATLAB Command Window. Using an `anfisOptions` option set, you can set the following display options.

- **DisplayANFISInformation** — Display ANFIS information at the start of training
- **DisplayErrorValues** — Display the training error at each epoch
- **DisplayStepSize** — Display the step-size each time it changes.
- **DisplayFinalResults** — Display the final training error and validation error

The **Neuro-Fuzzy Designer** does not provide user-specified display options. Instead, it displays the training progress as a plot.
Training Validation

Validation data lets you check the generalization capability of your trained fuzzy inference system. The validation data should fully represent the features of the data the FIS is intended to model, while also being sufficiently different from the training data to test training generalization. The software uses this data set to cross-validate the fuzzy inference model by applying the validation data to the model and seeing how well the model responds to this data.

Model validation is useful in the following situations:

- Noisy data — In some cases, data is collected using noisy measurements, and the training data is unable to represent all the features of the data the FIS is intended to model.
- Overfitting — Since the model structure used for ANFIS is fixed with a large number of parameters, there is a tendency for the model to overfit the data on which it is trained, especially when using a large number of training epochs. If overfitting does occur, the trained FIS may not generalize well to other independent data sets.

The idea behind using a checking data set for model validation is that, after a certain point in the training process, the model begins overfitting the training data set. In principle, the model error for the checking data set decreases up to the point that overfitting begins. After this point, the model error for the checking data increases. Overfitting is accounted for by testing the trained FIS 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, the training and checking data sets are collected based on observations of the target system and are then stored in separate files. To specify validation data when using the:

- `anfis` function, create an `anfisOptions` object, and set the `ValidationData` option.
- **Neuro-Fuzzy Designer**, in the Load data section, select **Checking**.

The array and file formats for the checking data are the same as those for the training data.
Training Results

When you train your fuzzy system using the anfis function, you can obtain the following trained fuzzy systems:

- The fis output argument is the fuzzy system for which the training error is minimum. This system is always returned by the anfis function, and corresponds to the FIS returned by Neuro-Fuzzy Designer when you do not specify checking data.
- The chkFIS output argument is the fuzzy system for which the validation error is minimum. This system is returned only when you specify validation data using anfisOptions, and corresponds to the FIS returned by Neuro-Fuzzy Designer when you specify checking data. This FIS object is the one that you should use for further calculation if checking data is used for cross-validation.

You can obtain the error associated with each of the trained fuzzy systems. In each case, the returned error is the root mean squared error (RMSE), and is returned as a vector. Each element of the vector is the RMSE error value at each training epoch.

- Training error — Difference between the training data output value and the output of the fuzzy inference system for the corresponding training data input values.
- Validation error — Difference between the checking data output value and the output of the fuzzy inference system for the corresponding checking data input values. This error is returned only when you specify validation data using anfisOptions.

During training, the Neuro-Fuzzy Designer app plots the training and checking error for each training epoch. Exporting training and checking error from the app is not supported. To obtain the training error, you must retrain the system from the command-line. For an example, see “Save Training Error Data to MATLAB Workspace” on page 3-117.

To further test your trained fuzzy system, you can use an additional set of testing data that you did not use for training or validation. To do so:

- When training a system at the command-line, use the evalfis function.
- When using Neuro-Fuzzy Designer, in the Load data section, select Testing, and click Load Data. To evaluate the trained system for any loaded data set, in the Test FIS section, select a data set, and click Test Now.
Training Algorithm Differences

The Neuro-Fuzzy Designer app manages training epochs in a manner different from the anfis function. This difference produces variations in the training results.

To train a system for \( N \) epochs at the command line, you call the anfis function one time, specifying the number of epochs as \( N \). However, the Neuro-Fuzzy Designer app calls the anfis function \( N \) times, specifying the number of epochs as 2 each time.

For a command-line example that demonstrates the Neuro-Fuzzy Designer training algorithm, see “Save Training Error Data to MATLAB Workspace” on page 3-117.

References


See Also

Apps
Neuro-Fuzzy Designer

Functions
anfis

More About
• “Train Adaptive Neuro-Fuzzy Inference Systems” on page 3-100
• “Save Training Error Data to MATLAB Workspace” on page 3-117
• “Predict Chaotic Time-Series using ANFIS” on page 3-125
Train Adaptive Neuro-Fuzzy Inference Systems

This example shows how to create, train, and test Sugeno-type fuzzy systems using the Neuro-Fuzzy Designer. For more information on:

- Neuro-adaptive fuzzy systems, see “Neuro-Adaptive Learning and ANFIS” on page 3-92.
- Training neuro-adaptive fuzzy systems at the command line, see anfis.

Load Training Data

Training and validating systems using the Neuro-Fuzzy Designer app requires data. Import the training data (fuzex1trnData) and validation data (fuzex1chkData) to the MATLAB workspace.

```matlab
load fuzex1trnData.dat
load fuzex1chkData.dat
```

Open the Neuro-Fuzzy Designer app.

```matlab
neuroFuzzyDesigner
```
Load the training data set from the workspace. In the **Load data** section, select **Training** and **worksp**.

Click **Load Data**. In the Load from workspace dialog box, enter the variable name **fuzexltrnData**.
Click **OK**. The **Neuro-Fuzzy Designer** displays the training data in the plot as a set of circles.
Load the checking data from the MATLAB workspace into the Neuro-Fuzzy Designer. In the **Load data** section, select **Checking**.

Load the checking data in the same manner as the training data, specifying the variable name `fuzex1chkData`. The **Neuro-Fuzzy Designer** displays the checking data using plus signs superimposed on the training data.
To clear a specific data set from the app, in the Load data area, select the data Type, and click Clear Data.

**Generate or Load FIS Structure**

Before you start the FIS training, you must specify an initial FIS model structure. To specify the model structure, you perform one of the following tasks:
• Load a previously saved single-output Sugeno-type FIS object from a file or the MATLAB workspace.
• Generate the initial FIS model using grid partitioning.
• Generate the initial FIS model using subtractive clustering.

For this example, generate the initial FIS using grid partitioning. In the Neuro-Fuzzy Designer, in the Generate FIS section, select Grid partition.

Click Generate FIS.

In the Add Membership Functions dialog box:

• In the Input section, in Number of MFs, specify the number of input membership functions. For this example, use 4 membership functions for all input variables.
• In MF Type, select gbellmf as the input membership function type.
• In the Output section, in MF Type, select linear as the output membership function type.
Interactively Specify FIS Structure

Alternatively, you can interactively specify your own FIS structure with specified membership functions and rules. The system you define must be a Sugeno system with the following properties:

- Single output
- Weighted average defuzzification
- First or zeroth order system; that is, all output membership functions must be the same type, either 'linear' or 'constant'.
- No rule sharing. Different rules cannot use the same output membership function; that is, the number of output membership functions must equal the number of rules.
- Unity weight for each rule.
- No custom membership functions or defuzzification methods.

To define the:

- Membership functions for each variable, in the **Neuro-Fuzzy Designer**, select `Edit > Membership Functions`. Then, in the Membership Function Editor window, define the membership functions.

- Rules, in the **Neuro-Fuzzy Designer**, select `Edit > Rules`. Then, in the Rule Editor window, define the rules.

These tools are the same as those used by the **Fuzzy Logic Designer** app. For more information, see “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17.

**View FIS Structure**

After you load or generate the FIS, you can view the model structure. To do so, in the **Neuro-Fuzzy Designer**, click **Structure**.
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. To view information about the structure, click on each node.

Also, to view the FIS:

- Membership functions, in the **Neuro-Fuzzy Designer**, select **Edit > Membership Functions**.
• Rules, in the **Neuro-Fuzzy Designer**, select **Edit > Rules**.

### Train FIS

After loading the training data and generating the initial FIS structure, you can train your FIS. To do so, in the **Neuro-Fuzzy Designer**, in the **Train FIS** section, specify the following parameters:

- **Optim. Method** — Optimization method. For this example, select the hybrid method, which uses a combination of backpropagation and least-squares regression to tune the FIS parameters.
- **Epochs** — Number of training epochs. For this example, specify 40 epochs.
- **Error Tolerance** — Error tolerance stopping condition. For this example, specify a value of 0, which indicates that the training will stop when the number of training epochs is reached.

To train the FIS, click **Train Now**.
The app trains the FIS and plots the training error (as stars) and checking error (as dots) for each training epoch.

The checking error decreases up to a certain point in the training, and then it increases. This increase occurs at the point where the training starts overfitting the training data. The app selects the FIS associated with this overfitting point as the trained ANFIS model.
Validate Trained FIS

After the FIS is trained, validate the model using a Testing or Checking data set that differs from the training data. For this example, use the previously loaded checking data.

To test your FIS against the checking data, in the Test FIS section, select Checking data. Then, click Test Now.
The app plots the output values of the testing data set (using blue +'s) and the output of the trained FIS for the corresponding testing data input values (using red *'s). The FIS output values correlate well with the expected output.

**Importance of Checking Data**

It is important to have checking data that fully represents the features of the data the FIS is intended to model. If your checking data is significantly different from your training data and does not cover the same data features to model as the training data, then the training results will be poor.

For example, load new training and checking data into the **Neuro-Fuzzy Designer**. This data has significantly different training and checking sets.

1. At the MATLAB command line, load the training and checking data.

   ```matlab
   load fuzex2trnData.dat
   load fuzex2chkData.dat
   ```

2. Clear the previously loaded training and checking data. In the **Load data** section, select each data type, click **Clear Data**.

3. Load the training data (`fuzex2trnData`) and checking data (`fuzex2chkData`), as you did previously.
Generate a FIS structure and train the FIS as you did previously, except now select 60 training epochs.
In this case, the checking error is large, with the minimum occurring in the first epoch. Since the app chooses the trained FIS parameters associated with the minimum checking error, the trained FIS does not sufficiently capture the features of this data set. 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, if you think the training data sufficiently captures the features you are trying to represent, the system can be retrained without the checking data.

To verify the poor training results, test the trained FIS model against the checking data.
As expected, there are significant differences between the checking data output values and the FIS output.

**See Also**

**Neuro-Fuzzy Designer**

**More About**

- “Neuro-Adaptive Learning and ANFIS” on page 3-92
- “Save Training Error Data to MATLAB Workspace” on page 3-117
Save Training Error Data to MATLAB Workspace

When using **Neuro-Fuzzy Designer**, you can export your initial FIS structure to the MATLAB workspace and then generate ANFIS training error values. Since exporting the training and validation error profiles from the **Neuro-Fuzzy Designer** app is not supported, use this method to generate such error plots.

The following example shows how to save the training error generated during ANFIS training to the MATLAB workspace.

1. Load your training data (`fuzex1trnData`) and validation data (`fuzex1chkData`) to the MATLAB workspace.
   ```matlab
   load fuzex1trnData.dat
   load fuzex1chkData.dat
   ```

2. Open the **Neuro-Fuzzy Designer** app.
   ```matlab
   neuroFuzzyDesigner
   ```
Load the training data from the MATLAB workspace into the **Neuro-Fuzzy Designer**.

a In the **Load data** section, select **Training**.

b Select **worksp**.

c Click **Load Data**. In the Load from workspace dialog box, enter the variable name **fuzex1trnData**.
d Click **OK**. The **Neuro-Fuzzy Designer** displays the training data in the plot as a set of circles.
Load the checking data from the MATLAB workspace into the Neuro-Fuzzy Designer. In the **Load data** section, select **Checking**.

Load the checking data in the same manner as the training data, specifying the variable name `fuzex1chkData`. The **Neuro-Fuzzy Designer** displays the checking data using plus signs superimposed on the training data.
5 Generate an initial FIS.

a In the Generate FIS section, select Grid partition.
b Click Generate FIS.
c In the Add Membership Functions dialog box:
   • In the Input section, in Number of MFs, specify the number of input membership functions. For this example use 4 for all input variables.
• In MF Type, select gbellmf as the input membership function type.
• In the Output section, in MF Type, select linear as the output membership function type.

d Click OK.
6 Export the initial FIS to the MATLAB workspace.

a In the Neuro-Fuzzy Designer, select File > Export > To Workspace.

This action opens a dialog box where you specify the MATLAB variable name.

b In the Export To Workspace dialog box, in the Workspace variable field, enter initfis.
c  Click **OK**. The app exports the FIS structure to the MATLAB workspace.

7  Train the FIS for 40 epochs. Instead of using a single call to the `anfis` function, call the function inside a loop using 2 epochs for each call. This training method replicates the training process used by the **Neuro-Fuzzy Designer** app.

At each training epoch, save the training and validation errors.

```matlab
fis = initfis;
opt = anfisOptions('EpochNumber',2,'ValidationData',fuzex1chkData);
trainError = zeros(1,40);
checkError = zeros(1,40);
for ct = 1:40
    opt.InitialFIS = fis;
    [fis,error,~,~,chkError] = anfis(fuzex1trnData,opt);
    trainError(ct) = error(1);
    checkError(ct) = chkError(1);
end
```

8  Plot the training and validation errors over the training process. These error values are the root mean squared errors at each training epoch.

```matlab
epochNum = 1:40;
plot(epochNum,trainError,'b*',epochNum,checkError,'ro')
xlabel('Epoch Number')
ylabel('Error')
legend('Training Error','Validation Error')
```
These error profiles are similar to the error profiles when the same initial FIS structure is trained in the **Neuro-Fuzzy Designer** app.

### See Also

**Neuro-Fuzzy Designer**

### More About

- “Neuro-Adaptive Learning and ANFIS” on page 3-92
- “Train Adaptive Neuro-Fuzzy Inference Systems” on page 3-100
Predict Chaotic Time-Series using ANFIS

This example shows how to do chaotic time-series prediction using ANFIS.

**Time Series Data**

This example uses `anfis` to predict a time series 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 with 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, the fourth-order Runge-Kutta method was used to find the numerical solution to the previous MG equation. It was assumed that \( x(0) = 1.2 \), \( \tau = 17 \), and \( x(t) = 0 \) for \( t < 0 \). The result was saved in the file `mgdata.dat`.

Load and plot the MG time series.

```matlab
load mgdata.dat
time = mgdata(:,1);
x = mgdata(:, 2);
figure(1)
plot(time,x)
title('Mackey-Glass Chaotic Time Series')
xlabel('Time (sec)')
ylabel('x(t)')
```
Preprocess Data

In time-series prediction, you use known values of the time series up to point in time, \( t \), to predict the value at some point in the future, \( t + P \). The standard method for this type of prediction is to create a mapping from \( D \) sample data points, sampled every \( \Delta \) units in time \((x(t - (D - 1)\Delta), \ldots, x(t - \Delta), x(t))\) 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 \texttt{anfis} is a four-column vector of the following form.

\[
    w(t) = [x(t - 19), x(t - 12), x(t - 6), x(t)]
\]

The output training data corresponds to the trajectory prediction.

\[
    s(t) = x(t + 6)
\]
For each $t$, ranging in values from 118 to 1117, there are 1000 input/output training samples. For this example, use the first 500 samples as training data (`trnData`) and the second 500 values as checking data for validation (`chkData`). Each row of the training and checking data arrays contains one sample point where the first four columns contain the four-dimensional input $w$ and the fifth column contains the output $s$.

Construct the training and checking data arrays.

```matlab
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,:);
```

**Build Initial Fuzzy System**

Create an initial Sugeno FIS object for training using the `genfis` function with grid partitioning.

```matlab
fis = genfis(trnData(:,1:end-1),trnData(:,end),...
            genfisOptions('GridPartition'));
```

The number of FIS inputs and outputs corresponds to the number of columns in the input and output training data, four and one, respectively.

By default, `genfis` creates two generalized bell membership functions for each of the four inputs. The initial membership functions for each variable are equally spaced and cover the whole input space.

```matlab
figure
subplot(2,2,1)
plotmf(fis,'input',1)
subplot(2,2,2)
plotmf(fis,'input',2)
subplot(2,2,3)
plotmf(fis,'input',3)
subplot(2,2,4)
plotmf(fis,'input',4)
```
The generated FIS object contains $2^4 = 16$ fuzzy rules with 104 parameters (24 nonlinear parameters and 80 linear 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 approximately five (500/104), which is a good balance between fitting parameters and training sample points.

**Train ANFIS Model**

To configure training options, create an `anfisOptions` option set, specifying the initial FIS and validation data.

```plaintext
options = anfisOptions('InitialFIS',fis,'ValidationData',chkData);
```
Train the FIS using the specified training data and options.

\[
[fis1, error1, ss, fis2, error2] = \text{anfis(trnData, options)};
\]

**ANFIS info:**
- Number of nodes: 55
- Number of linear parameters: 80
- Number of nonlinear parameters: 24
- Total number of parameters: 104
- Number of training data pairs: 500
- Number of checking data pairs: 500
- Number of fuzzy rules: 16

Start training ANFIS ...

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Training RMSE</th>
<th>Checking RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00296046</td>
<td>0.00292488</td>
</tr>
<tr>
<td>2</td>
<td>0.00290346</td>
<td>0.0028684</td>
</tr>
<tr>
<td>3</td>
<td>0.00285048</td>
<td>0.00281544</td>
</tr>
<tr>
<td>4</td>
<td>0.00280117</td>
<td>0.00276566</td>
</tr>
<tr>
<td>5</td>
<td>0.00275517</td>
<td>0.00271874</td>
</tr>
</tbody>
</table>

Step size increases to 0.011000 after epoch 5.

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Training RMSE</th>
<th>Checking RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.00271214</td>
<td>0.00267438</td>
</tr>
<tr>
<td>7</td>
<td>0.002666783</td>
<td>0.00262818</td>
</tr>
<tr>
<td>8</td>
<td>0.00262626</td>
<td>0.00258435</td>
</tr>
<tr>
<td>9</td>
<td>0.00258702</td>
<td>0.00254254</td>
</tr>
</tbody>
</table>

Step size increases to 0.012100 after epoch 9.

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Training RMSE</th>
<th>Checking RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00254972</td>
<td>0.00250247</td>
</tr>
</tbody>
</table>

Designated epoch number reached --> ANFIS training completed at epoch 10.

Minimal training RMSE = 0.002550
Minimal checking RMSE = 0.00250247

**fis1** is the trained fuzzy inference system for the training epoch where the training error is smallest. Since you specified validation data, the fuzzy system with the minimum checking error, **fis2**, is also returned. The FIS with the smallest checking error shows the best generalization beyond the training data.

Plots the membership functions for the trained system.

```matlab
figure
subplot(2,2,1)
plotmf(fis2,'input',1)
subplot(2,2,2)
```
plotmf(fis2, 'input', 2)
subplot(2,2,3)
plotmf(fis2, 'input', 3)
subplot(2,2,4)
plotmf(fis2, 'input', 4)

Plot Errors Curves
Plot the training and checking error signals.

figure
plot([error1 error2])
hold on
plot([error1 error2], 'o')
The training error is higher than the checking error in all epochs. This phenomenon is not uncommon in ANFIS learning or nonlinear regression in general; it could indicate that additional training could produce better training results.

**Compare Original and Predicted Series**

To check prediction capability of the trained system, evaluate the fuzzy system using the training and checking data, and plot the result alongside the original
anfis_output = evalfis(fis2,[trnData(:,1:4); chkData(:,1:4)]);

figure
index = 125:1124;
plot(time(index),[x(index) anfis_output])
xlabel('Time (sec)')
title('MG Time Series and ANFIS Prediction')

The predicted series is similar to the original series.

Calculate and plot the prediction error.

diff = x(index) - anfis_output;
plot(time(index),diff)
xlabel('Time (sec)')
title('Prediction Errors')

The scale of the prediction error plot is about one-hundredth of the scale of the time-series plot. In this example, you trained the system for only 10 epoch. Training for additional epochs can improve the training results.

See Also
anfis | evalfis | genfis
More About

- “Neuro-Adaptive Learning and ANFIS” on page 3-92
Modeling Inverse Kinematics in a Robotic Arm

This example shows how to use a fuzzy system to model the inverse kinematics in a two-joint robotic arm.

What Is Inverse Kinematics?

Kinematics is the science of motion. In a two-joint robotic arm, given the angles of the joints, the kinematics equations give the location of the tip of the arm. Inverse kinematics refers to the reverse process. Given a desired location for the tip of the robotic arm, what should the angles of the joints be so as to locate the tip of the arm at the desired location. There is usually more than one solution and can at times be a difficult problem to solve.

This is a typical problem in robotics that needs to be solved to control a robotic arm to perform tasks it is designated to do. In a 2-dimensional input space, with a two-joint robotic arm and given the desired coordinate, the problem reduces to finding the two angles involved. The first angle is between the first arm and the ground (or whatever it is attached to). The second angle is between the first arm and the second arm.

Figure 1: Illustration showing the two-joint robotic arm with the two angles, theta1 and theta2
**Why Use Fuzzy Logic?**

For simple structures like the two-joint robotic arm, it is possible to mathematically deduce the angles at the joints given the desired location of the tip of the arm. However, with more complex structures (for example: n-joint robotic arms operating in a 3-dimensional input space) deducing a mathematical solution for the inverse kinematics may prove challenging.

Using fuzzy logic, we can construct a fuzzy inference system that deduces the inverse kinematics if the forward kinematics of the problem is known, hence sidestepping the need to develop an analytical solution. Also, the fuzzy solution is easily understandable and does not require special background knowledge to comprehend and evaluate it.

In the following section, a broad outline for developing such a solution is described, and later, the detailed steps are elaborated.

**Overview of Fuzzy Solution**

Since the forward kinematics formulae for the two-joint robotic arm are known, x and y coordinates of the tip of the arm are deduced for the entire range of angles of rotation of the two joints. The coordinates and the angles are saved to be used as training data to train an ANFIS (adaptive neuro-fuzzy inference system) network.

During training, the ANFIS network learns to map the coordinates (x, y) to the angles (\(\theta_1, \theta_2\)). The trained ANFIS network is then used as a part of a larger control system to control the robotic arm. Knowing the desired location of the robotic arm, the control system uses the trained ANFIS network to deduce the angular positions of the joints and applies force to the joints of the robotic arm accordingly to move it to the desired location.

**What Is ANFIS?**

ANFIS stands for adaptive neuro-fuzzy inference system. It is a hybrid neuro-fuzzy technique that brings learning capabilities of neural networks to fuzzy inference systems. The learning algorithm tunes the membership functions of a Sugeno-type fuzzy inference system using the training input/output data.

In this case, the input/output data refers to the "coordinates/angles" dataset. The coordinates act as input to the ANFIS and the angles act as the output. The learning algorithm teaches the ANFIS to map the coordinates to the angles through a process called training. At the end of training, the trained ANFIS network would have learned the input-output map and be ready to be deployed into the larger control system solution.
Data Generation

Let $\theta_1$ be the angle between the first arm and the ground. Let $\theta_2$ be the angle between the second arm and the first arm (Refer to Figure 1 for illustration). Let the length of the first arm be $l_1$ and that of the second arm be $l_2$.

Assume that the first joint has limited freedom to rotate and it can rotate between 0 and 90 degrees. Similarly, assume that the second joint has limited freedom to rotate and can rotate between 0 and 180 degrees. (This assumption takes away the need to handle some special cases which will confuse the discourse.) Hence, $0 \leq \theta_1 \leq \pi/2$ and $0 \leq \theta_2 \leq \pi$. 

\[ 0 \leq \theta_2 \leq \pi \]

\[ 0 \leq \theta_1 \leq \pi/2 \]
Figure 2: Illustration showing all possible theta1 and theta2 values.

Now, for every combination of theta1 and theta2 values the x and y coordinates are deduced using forward kinematics formulae.

The following code snippet shows how data is generated for all combination of theta1 and theta2 values and saved into a matrix to be used as training data. The reason for saving the data in two matrices is explained in the following section.

```matlab
l1 = 10; % length of first arm
l2 = 7; % length of second arm
theta1 = 0:0.1:pi/2; % all possible theta1 values
theta2 = 0:0.1:pi; % all possible theta2 values

[THETA1,THETA2] = meshgrid(theta1,theta2); % generate a grid of theta1 and theta2 values
X = l1 * cos(THETA1) + l2 * cos(THETA1 + THETA2); % compute x coordinates
Y = l1 * sin(THETA1) + l2 * sin(THETA1 + THETA2); % compute y coordinates

data1 = [X(:) Y(:) THETA1(:)]; % create x-y-theta1 dataset
data2 = [X(:) Y(:) THETA2(:)]; % create x-y-theta2 dataset
```

The following plot shows all the X-Y data points generated by cycling through different combinations of theta1 and theta2 and deducing x and y coordinates for each. The plot can be generated by using the code-snippet shown below. The plot is illustrated further for easier understanding.

```matlab
plot(X(:),Y(:),'r.);
axis equal;
xlabel('X','fontsize',10)
ylabel('Y','fontsize',10)
title('X-Y coordinates generated for all theta1 and theta2 combinations using forward kinematics','fontsize',10)
```
Building ANFIS Networks

One approach to building an ANFIS solution for this problem, is to build two ANFIS networks, one to predict theta1 and the other to predict theta2.

In order for the ANFIS networks to be able to predict the angles they have to be trained with sample input-output data. The first ANFIS network will be trained with X and Y coordinates as input and corresponding theta1 values as output. The matrix data1

**Figure 3**: X-Y coordinates generated for all theta1 and theta2 combinations using forward kinematics formulae

Modeling Inverse Kinematics in a Robotic Arm
contains the \textit{x-y-theta1} dataset required to train the first ANFIS network. Therefore \texttt{data1} will be used as the dataset to train the first ANFIS network.

Similarly, the second ANFIS network will be trained with X and Y coordinates as input and corresponding \textit{theta2} values as output. The matrix \texttt{data2} contains the \textit{x-y-theta2} dataset required to train the second ANFIS network. Therefore \texttt{data2} will be used as the dataset to train the second ANFIS network.

To train an ANFIS network, first specify the training options using the \texttt{anfisOptions} command. For this example, specify an FIS object with 7 membership functions for each input variable. Train the system for 150 epochs and suppress the Command Window display of training information.

\begin{verbatim}
opt = anfisOptions;
opt.InitialFIS = 7;
opt.EpochNumber = 150;
opt.DisplayANFISInformation = 0;
opt.DisplayErrorValues = 0;
opt.DisplayStepSize = 0;
opt.DisplayFinalResults = 0;
\end{verbatim}

Train an ANFIS system using the first set of training data, \texttt{data1}.

\begin{verbatim}
disp('--- Training first ANFIS network.');
--> Training first ANFIS network.
anfis1 = anfis(data1,opt);
\end{verbatim}

Change the number of input membership functions and train an ANFIS system using the second set of training data, \texttt{data2}.

\begin{verbatim}
disp('--- Training second ANFIS network.');
--> Training second ANFIS network.
opt.InitialFIS = 6;
anfis2 = anfis(data2,opt);
\end{verbatim}

For this example, the number of input membership functions and training epochs were selected based on experimentation with different potential values.

\texttt{anfis1} and \texttt{anfis2} represent the two trained ANFIS networks that will be deployed in the larger control system.
Once the training is complete, the two ANFIS networks have learned to approximate the angles (\( \theta_1 \), \( \theta_2 \)) as a function of the coordinates (\( x \), \( y \)). One advantage of using the fuzzy approach is that the ANFIS network can now approximate the angles for coordinates that are similar but not exactly the same as it was trained with. For example, the trained ANFIS networks are now capable of approximating the angles for coordinates that lie between two points that were included in the training dataset. This will allow the final controller to move the arm smoothly in the input space.

We now have two trained ANFIS networks which are ready to be deployed into the larger system that will utilize these networks to control the robotic arms.

**Validating ANFIS Networks**

Having trained the networks, an important follow up step is to validate the networks to determine how well the ANFIS networks would perform inside the larger control system.

Since this example problem deals with a two-joint robotic arm whose inverse kinematics formulae can be derived, it is possible to test the answers that the ANFIS networks produce with the answers from the derived formulae.

Assume that it is important for the ANFIS networks to have low errors within the operating range \( 0 < x < 2 \) and \( 8 < y < 10 \).

\[
\begin{align*}
  x &= 0:0.1:2; \quad \text{% x coordinates for validation} \\
  y &= 8:0.1:10; \quad \text{% y coordinates for validation}
\end{align*}
\]

The \( \theta_1 \) and \( \theta_2 \) values are deduced mathematically from the \( x \) and \( y \) coordinates using inverse kinematics formulae.

\[
\begin{align*}
  [X,Y] &= \text{meshgrid}(x,y) \\
  c2 &= (X.^2 + Y.^2 - l1^2 - l2^2)/(2*l1*l2); \\
  s2 &= \text{sqrt}(1 - c2.^2); \\
  \text{THETA2D} &= \text{atan2}(s2,c2); \quad \% \ \text{theta2 is deduced} \\
  k1 &= l1 + l2.*c2; \\
  k2 &= l2*s2; \\
  \text{THETA1D} &= \text{atan2}(Y,X) - \text{atan2}(k2,k1); \quad \% \ \text{theta1 is deduced}
\end{align*}
\]

\( \text{THETA1D} \) and \( \text{THETA2D} \) are the variables that hold the values of \( \theta_1 \) and \( \theta_2 \) deduced using the inverse kinematics formulae.

The \( \theta_1 \) and \( \theta_2 \) values predicted by the trained ANFIS networks are obtained by using the command \text{evalfis} which evaluates a FIS for the given inputs.
Here, `evalfis` is used to find out the FIS outputs for the same x-y values used earlier in the inverse kinematics formulae.

```matlab
XY = [X(:) Y(:)];
THETA1P = evalfis(anfis1,XY); % theta1 predicted by anfis1
THETA2P = evalfis(anfis2,XY); % theta2 predicted by anfis2
```

Now, we can see how close the FIS outputs are with respect to the deduced values.

```matlab
theta1diff = THETA1D(:) - THETA1P;
theta2diff = THETA2D(:) - THETA2P;
```

```matlab
subplot(2,1,1);
plot(theta1diff);
ylabel('THETA1D - THETA1P','fontsize',10)
title('Deduced theta1 - Predicted theta1','fontsize',10)

subplot(2,1,2);
plot(theta2diff);
ylabel('THETA2D - THETA2P','fontsize',10)
title('Deduced theta2 - Predicted theta2','fontsize',10)
```
The errors are in the $1e^{-3}$ range which is a fairly good number for the application it is being used in. However this may not be acceptable for another application, in which case the parameters to the `anfis` function may be tweaked until an acceptable solution is arrived at. Also, other techniques like input selection and alternate ways to model the problem may be explored.

**Building a Solution Around the Trained ANFIS Networks**

Now given a specific task, such as robots picking up an object in an assembly line, the larger control system will use the trained ANFIS networks as a reference, much like a lookup table, to determine what the angles of the arms must be, given a desired location for the tip of the arm. Knowing the desired angles and the current angles of the joints, the
system will apply force appropriately on the joints of the arms to move them towards the desired location.

The invkine command launches a GUI that shows how the two trained ANFIS networks perform when asked to trace an ellipse.

![GUI for Inverse Kinematics Modeling](image)

**Figure 4:** GUI for Inverse Kinematics Modeling.

The two ANFIS networks used in the example have been pretrained and are deployed into a larger system that controls the tip of the two-joint robot arm to trace an ellipse in the input space.

The ellipse to be traced can be moved around. Move the ellipse to a slightly different location and observe how the system responds by moving the tip of the robotic arm from
its current location to the closest point on the new location of the ellipse. Also observe that the system responds smoothly as long as the ellipse to be traced lies within the 'x' marked spots which represent the data grid that was used to train the networks. Once the ellipse is moved outside the range of data it was trained with, the ANFIS networks respond unpredictably. This emphasizes the importance of having relevant and representative data for training. Data must be generated based on the expected range of operation to avoid such unpredictability and instability issues.

See Also

anfis | evalfis

More About

- “Neuro-Adaptive Learning and ANFIS” on page 3-92
Adaptive Noise Cancellation Using ANFIS

This example shows how to do adaptive nonlinear noise cancellation using the anfis and genfis commands.

Signal and Noise

Define a hypothetical information signal, \( x \), sampled at 100Hz over 6 seconds.

```matlab
time = (0:0.01:6)';
x = sin(40./(time+0.01));
plot(time,x)
title('Information Signal x','fontsize',10)
xlabel('time','fontsize',10)
ylabel('x','fontsize',10)
```
Assume that $x$ cannot be measured without an interference signal, $n_2$, which is generated from another noise source, $n_1$, via a certain unknown nonlinear process.

The plot below shows noise source $n_1$.

```matlab
n1 = randn(size(time));
plot(time,n1)
title('Noise Source n_1','fontsize',10)
xlabel('time','fontsize',10)
ylabel('n_1','fontsize',10)
```
Assume that the interference signal, $n_2$, that appears in the measured signal is generated via an unknown nonlinear equation:

$$n_2(k) = \frac{4\sin(n_1(k)) \cdot n_1(k - 1)}{1 + n_1(k - 1)^2}$$

Plot this nonlinear characteristic as a surface.

```matlab
domain = linspace(min(n1),max(n1),20);
[xx,yy] = meshgrid(domain,domain);
zz = 4*sin(xx).*yy./(1+yy.^2);
surf(xx,yy,zz);
```
Compute the interference signal, $n_2$, from the noise source, $n_1$, and plot both signals.

```matlab
n1d0 = n1;                          % n1 with delay 0
n1d1 = [0; n1d0(1:length(n1d0)-1)];  % n1 with delay 1
n2 = 4*sin(n1d0).*n1d1./(1+n1d1.^2); % interference

subplot(2,1,1)
plot(time,n1)
ylabel('noise n_1','fontsize',10);
```

Adaptive Noise Cancellation Using ANFIS

```
xlabel('n_1(k)','fontsize',10);
ylabel('n_1(k-1)','fontsize',10);
zlabel('n_2(k)','fontsize',10);
title('Unknown Interference Channel Characteristics','fontsize',10);
```
$n_2$ is related to $n_1$ via the highly nonlinear process shown previously; from the plots, it is hard to see if these two signals are correlated in any way.

The measured signal, $m$, is the sum of the original information signal, $x$, and the interference, $n_2$. However, we do not know $n_2$. The only signals available to us are the noise signal, $n_1$, and the measured signal $m$.

\[ m = x + n_2; \]  
\% measured signal

subplot(2,1,1)  
plot(time, m)
You can recover the original information signal, $x$, using adaptive noise cancellation via ANFIS training.

**Build the ANFIS Model**

Use the `anfis` command to identify the nonlinear relationship between $n_1$ and $n_2$. While $n_2$ is not directly available, you can assume that $m$ is a "contaminated" version of $n_2$ for training. This assumption treats $x$ as "noise" in this kind of nonlinear fitting.
Assume the order of the nonlinear channel is known (in this case, 2), so you can use a 2-input ANFIS model for training.

Define the training data. The first two columns of data are the inputs to the ANFIS model, \( n_1 \) and a delayed version of \( n_1 \). The final column of data is the measured signal, \( m \).

\[
\text{delayed}_n = [0; n_1(1:length(n1)-1)];
\]
\[
data = [\text{delayed}_n \; n_1 \; m];
\]

Generate the initial FIS object. By default, the grid partitioning algorithm uses two membership functions for each input variable, which produces four fuzzy rules for learning.

\[
genOpt = \text{genfisOptions}('GridPartition');
\]
\[
inFIS = \text{genfis}(data(:,1:end-1),data(:,end),genOpt);
\]

Tune the FIS using the \text{anfis} command with an initial training step size of 0.2.

\[
\text{trainOpt} = \text{anfisOptions('InitialFIS',inFIS,'InitialStepSize',0.2)};
\]
\[
\text{outFIS} = \text{anfis(data,trainOpt)};
\]

ANFIS info:
- Number of nodes: 21
- Number of linear parameters: 12
- Number of nonlinear parameters: 12
- Total number of parameters: 24
- Number of training data pairs: 601
- Number of checking data pairs: 0
- Number of fuzzy rules: 4

Start training ANFIS ...

\[
\begin{array}{ll}
1 & 0.761817 \\
2 & 0.748426 \\
3 & 0.739315 \\
4 & 0.733993 \\
5 & 0.729492 \\
\end{array}
\]

Step size increases to 0.220000 after epoch 5.

\[
\begin{array}{ll}
6 & 0.725382 \\
7 & 0.721269 \\
8 & 0.717621 \\
9 & 0.714474 \\
\end{array}
\]

Step size increases to 0.242000 after epoch 9.

\[
\begin{array}{ll}
10 & 0.71207 \\
\end{array}
\]
Designated epoch number reached --> ANFIS training completed at epoch 10.

Minimal training RMSE = 0.712070

The tuned FIS, outFIS, models the second-order relationship between $n_1$ and $n_2$.

**Evaluate Model**

Calculate the estimated interference signal, estimated_n2, by evaluating the tuned FIS using the original training data.

\[
\text{estimated}_n2 = \text{evalfis(} \text{outFIS}, \text{data}(:,1:2))\;
\]

Plot the and actual $n_2$ signal and the estimated version from the ANFIS output.

```matlab
subplot(2,1,1)
plot(time, n2)
ylabel('n_2 (unknown)');

subplot(2,1,2)
plot(time, estimated_n2)
ylabel('Estimated n_2');
```
The estimated information signal is equal to the difference between the measured signal, \( m \), and the estimated interference (ANFIS output).

\[
\text{estimated}_x = m - \text{estimated}_n2;
\]

Compare the original information signal, \( x \), and the estimate, \( \text{estimated}_x \).

```plaintext
figure
plot(time,estimated_x,'b',time,x,'r')
legend('Estimated x','Actual x (unknown)','Location','SouthEast')
```
Without extensive training, the ANFIS produces a good estimate of the information signal.

**See Also**

anfis | evalfis | genfis

**More About**

- “Neuro-Adaptive Learning and ANFIS” on page 3-92
Nonlinear System Identification

This example shows how to use anfis command for nonlinear dynamic system identification.

This example requires System Identification Toolbox™, as a comparison is made between a nonlinear ANFIS and a linear ARX model.

Problem Setup

Exit if System Identification Toolbox is not available.

```matlab
if ~fuzzychecktoolboxinstalled('ident')
    errordlg('DRYDEMO needs the System Identification Toolbox.');
    return;
end
```

The data set for ANFIS and ARX modeling was obtained from a laboratory device called Feedback's Process Trainer PT 326, as described in Chapter 17 of Prof. Lennart Ljung's book "System Identification, Theory for the User", Prentice-Hall, 1987. The device functions like a hair dryer: air is fanned through a tube and heated at the inlet. The air temperature is measured by a thermocouple at the outlet. The input u(k) is the voltage over a mesh of resistor wires to heat incoming air; the output y(k) is the outlet air temperature.

Here are the results of the test.

```matlab
load drydemodata
data_n = length(y2);
output = y2;
input = [[0; y2(1:data_n-1)] ...
[0; 0; y2(1:data_n-2)] ...
[0; 0; 0; y2(1:data_n-3)] ...
[0; 0; 0; 0; y2(1:data_n-4)] ...
[0; u2(1:data_n-1)] ...
[0; 0; u2(1:data_n-2)] ...
```
```matlab
[0; 0; 0; u2(1:data_n-3)] ...
[0; 0; 0; 0; u2(1:data_n-4)] ...
[0; 0; 0; 0; 0; u2(1:data_n-5)] ...
[0; 0; 0; 0; 0; 0; u2(1:data_n-6)]

data = [input output];
data(1:6, :) = [];
input_name = char('y(k-1)','y(k-2)','y(k-3)','y(k-4)','u(k-1)','u(k-2)','u(k-3)','u(k-4)',
index = 1:100;
subplot(2,1,1)
plot(index,y2(index),'-',index,y2(index),'o')
ylabel('y(k)','fontsize',10)
subplot(2,1,2)
plot(index,u2(index),'-',index,u2(index),'o')
ylabel('u(k)','fontsize',10)
```
The data points were collected at a sampling time of 0.08 seconds. One thousand input-output data points were collected from the process as the input \( u(k) \) was chosen to be a binary random signal shifting between 3.41 and 6.41 V. The probability of shifting the input at each sample was 0.2. The data set is available from the System Identification Toolbox, and the above plots show the output temperature \( y(k) \) and input voltage \( u(t) \) for the first 100 time steps.

**ARX Model Identification**

A conventional method is to remove the means from the data and assume a linear model of the form:

\[
y(k) + a_1 y(k-1) + \ldots + a_m y(k-m) = b_1 u(k-d) + \ldots + b_n u(k-d-n+1)
\]

where \( a_i \) (\( i = 1 \) to \( m \)) and \( b_j \) (\( j = 1 \) to \( n \)) are linear parameters to be determined by least-squares methods. This structure is called the ARX model and it is exactly specified by three integers \([m, n, d]\). To find an ARX model for the dryer device, the data set was divided into a training (\( k = 1 \) to 300) and a checking (\( k = 301 \) to 600) set. An exhaustive search was performed to find the best combination of \([m, n, d]\), where each of the integer is allowed to changed from 1 to 10 independently. The best ARX model thus found is specified by \([m, n, d] = [5, 10, 2]\), with a training RMSE of 0.1122 and a checking RMSE of 0.0749. The above figure shows the fitting results of the best ARX model.

```matlab
trn_data_n = 300;
total_data_n = 600;
z = [y2 u2];
z = dtrend(z);
ave = mean(y2);
ze = z(1:trn_data_n,:);
zv = z(trn_data_n+1:total_data_n,:);
T = 0.08;

% Run through all different models
V = arxstruc(ze,zv,struc(1:10,1:10,1:10));
% Find the best model
nn = selstruc(V,0);
% Time domain plot
th = arx(ze,nn);
th.Ts = 0.08;
u = z(:,2);
y = z(:,1)+ave;
yp = sim(u,th)+ave;
```
xlbl = 'Time Steps';

subplot(2,1,1)
index = 1:trn_data_n;
plot(index, y(index), index, yp(index), '.')
rmse = norm(y(index)-yp(index))/sqrt(length(index));
title(sprintf(['(a) Training Data (Solid Line) and ARX Prediction (Dots)\nwith RMSE = ' num2str(rmse)]), 'fontsize', 10)
xlabel(xlbl, 'fontsize', 10)

subplot(2,1,2)
index = (trn_data_n+1):(total_data_n);
plot(index, y(index), index, yp(index), '.')
rmse = norm(y(index)-yp(index))/sqrt(length(index));
title(sprintf(['(b) Checking Data (Solid Line) and ARX Prediction (Dots)\nwith RMSE = ' num2str(rmse)]), 'fontsize', 10)
xlabel(xlbl, 'fontsize', 10)

[na nb d] = 5 10 2
ANFIS Model Identification

The ARX model is inherently linear and the most significant advantage is that we can perform model structure and parameter identification rapidly. The performance in the above plots appears to be satisfactory. However, if a better performance level is desired, we might want to resort to a nonlinear model. In particular, we are going to use a neuro-fuzzy modeling approach, ANFIS, to see if we can push the performance level with a fuzzy inference system.

To use ANFIS for system identification, the first thing we need to do is select the input. That is, to determine which variables should be the input arguments to an ANFIS model. For simplicity, we suppose that there are 10 input candidates \( (y(k-1), y(k-2), y(k-3), y(k-4), u(k-1), u(k-2), u(k-3), u(k-4), u(k-5), u(k-6)) \), and the output to be predicted is \( y(k) \). A
A heuristic approach to input selection is called sequential forward search, in which each input is selected sequentially to optimize the total squared error. This can be done by the function `seqsrch`; the result is shown in the above plot, where 3 inputs \((y(k-1), u(k-3), \text{and} u(k-4))\) are selected with a training RMSE of 0.0609 and checking RMSE of 0.0604.

```matlab
trn_data_n = 300;
trn_data = data(1:trn_data_n,:);
chk_data = data(trn_data_n+1:trn_data_n+300,:);
[~,elapsed_time] = seqsrch(3,trn_data,chk_data,input_name);  % #ok<*ASGLU>
fprintf('
Elapsed time = %f
',elapsed_time);
winH1 = gcf;

Selecting input 1 ...
ANFIS model 1: \(y(k-1)\) \(\rightarrow\) trn=0.2043, chk=0.1888
ANFIS model 2: \(y(k-2)\) \(\rightarrow\) trn=0.3819, chk=0.3541
ANFIS model 3: \(y(k-3)\) \(\rightarrow\) trn=0.5245, chk=0.4903
ANFIS model 4: \(y(k-4)\) \(\rightarrow\) trn=0.6308, chk=0.5977
ANFIS model 5: \(u(k-1)\) \(\rightarrow\) trn=0.8271, chk=0.8434
ANFIS model 6: \(u(k-2)\) \(\rightarrow\) trn=0.7976, chk=0.8087
ANFIS model 7: \(u(k-3)\) \(\rightarrow\) trn=0.7266, chk=0.7349
ANFIS model 8: \(u(k-4)\) \(\rightarrow\) trn=0.6215, chk=0.6346
ANFIS model 9: \(u(k-5)\) \(\rightarrow\) trn=0.5419, chk=0.5650
ANFIS model 10: \(u(k-6)\) \(\rightarrow\) trn=0.5304, chk=0.5601
Currently selected inputs: \(y(k-1)\)

Selecting input 2 ...
ANFIS model 11: \(y(k-1)\) \(y(k-2)\) \(\rightarrow\) trn=0.1085, chk=0.1024
ANFIS model 12: \(y(k-1)\) \(y(k-3)\) \(\rightarrow\) trn=0.1339, chk=0.1283
ANFIS model 13: \(y(k-1)\) \(y(k-4)\) \(\rightarrow\) trn=0.1542, chk=0.1461
ANFIS model 14: \(y(k-1)\) \(u(k-1)\) \(\rightarrow\) trn=0.1892, chk=0.1734
ANFIS model 15: \(y(k-1)\) \(u(k-2)\) \(\rightarrow\) trn=0.1663, chk=0.1574
ANFIS model 16: \(y(k-1)\) \(u(k-3)\) \(\rightarrow\) trn=0.1082, chk=0.1077
ANFIS model 17: \(y(k-1)\) \(u(k-4)\) \(\rightarrow\) trn=0.0925, chk=0.0948
ANFIS model 18: \(y(k-1)\) \(u(k-5)\) \(\rightarrow\) trn=0.1533, chk=0.1531
ANFIS model 19: \(y(k-1)\) \(u(k-6)\) \(\rightarrow\) trn=0.1952, chk=0.1853
Currently selected inputs: \(y(k-1)\) \(u(k-4)\)

Selecting input 3 ...
ANFIS model 20: \(y(k-1)\) \(u(k-4)\) \(y(k-2)\) \(\rightarrow\) trn=0.0808, chk=0.0822
ANFIS model 21: \(y(k-1)\) \(u(k-4)\) \(y(k-3)\) \(\rightarrow\) trn=0.0806, chk=0.0836
ANFIS model 22: \(y(k-1)\) \(u(k-4)\) \(y(k-4)\) \(\rightarrow\) trn=0.0817, chk=0.0855
ANFIS model 23: \(y(k-1)\) \(u(k-4)\) \(u(k-1)\) \(\rightarrow\) trn=0.0886, chk=0.0912
ANFIS model 24: \(y(k-1)\) \(u(k-4)\) \(u(k-2)\) \(\rightarrow\) trn=0.0835, chk=0.0843
```
For input selection, another more computationally intensive approach is to do an exhaustive search on all possible combinations of the input candidates. The function that performs exhaustive search is exhsrch, which selects 3 inputs from 10 candidates. However, exhsrch usually involves a significant amount of computation if all combinations are tried. For instance, if 3 is selected out of 10, the total number of ANFIS models is $\binom{10}{3} = 120$. 
Fortunately, for dynamic system identification, we do know that the inputs should not come from either of the following two sets of input candidates exclusively:

\[ Y = \{y(k-1), y(k-2), y(k-3), y(k-4)\} \]

\[ U = \{u(k-1), u(k-2), u(k-3), u(k-4), u(k-5), u(k-6)\} \]

A reasonable guess would be to take two inputs from \( Y \) and one from \( U \) to form the inputs to ANFIS; the total number of ANFIS models is then \( C(4,2) \times 6 = 36 \), which is much less. The above plot shows that the selected inputs are \( y(k-1), y(k-2) \) and \( u(k-3) \), with a training RMSE of 0.0474 and checking RMSE of 0.0485, which are better than ARX models and ANFIS via sequential forward search.

```matlab
% group1 = [1 2 3 4];    % y(k-1), y(k-2), y(k-3), y(k-4)
group2 = [1 2 3 4];    % y(k-1), y(k-2), y(k-3), y(k-4)
group3 = [5 6 7 8 9 10];    % u(k-1) through y(k-6)

anfis_n = 6*length(group3);
index = zeros(anfis_n,3);
trn_error = zeros(anfis_n,1);
chk_error = zeros(anfis_n,1);

% ====== Training options
% Create option set for generating initial FIS.
genOpt = genfisOptions('GridPartition','NumMembershipFunctions',2, ...'
    'InputMembershipFunctionType','gbellmf');

% Create option set for |anfis| command and set options that remain constant
% for different training scenarios.
anfisOpt = anfisOptions('EpochNumber',1,...
    'InitialStepSize',0.1,...
    'StepSizeDecreaseRate',0.5,...
    'StepSizeIncreaseRate',1.5,...
    'DisplayANFISInformation',0,...
    'DisplayErrorValues',0,...
    'DisplayStepSize',0,...
    'DisplayFinalResults',0);

% ====== Train ANFIS with different input variables
fprintf('
Train %d ANFIS models, each with 3 inputs selected from 10 candidates...
', anfis_n);
model = 1;
for i = 1:length(group1)
    for j = i+1:length(group2)
        for k = 1:length(group3)
            in1 = deblank(input_name(group1(i),:));
            in2 = deblank(input_name(group2(j),:));
```

3-163
in3 = deblank(input_name(group3(k),:));
index(model, :) = [group1(i) group2(j) group3(k)];
trn_data = data(1:trn_data_n, [group1(i) group2(j) group3(k) size(data,2)]);
chk_data = data(trn_data_n+1:trn_data_n+300, [group1(i) group2(j) group3(k)]);
in_fismat = genfis(trn_data(:,1:end-1),trn_data(:,end),genOpt);
% Set initial FIS and validation data in option set for ANFIS training.
anfisOpt.InitialFIS = in_fismat;
anfisOpt.ValidationData = chk_data;
[~, t_err, ~, ~, c_err] = anfis(trn_data,anfisOpt);
trn_error(model) = min(t_err);
chk_error(model) = min(c_err);
fprintf('ANFIS model = %d: %s %s %s',model,in1,in2,in3);
fprintf(' --> trn=%.4f,',trn_error(model));
fprintf(' chk=%.4f',chk_error(model));
fprintf('
');
model = model+1;
end
end
end

% ====== Reordering according to training error
[~, b] = sort(trn_error);
b = flipud(b);            % List according to decreasing trn error
trn_error = trn_error(b);
chk_error = chk_error(b);
index = index(b,:);

% ====== Display training and checking errors
x = (1:anfis_n)';
subplot(2,1,1)
plot(x, trn_error,'-',x,chk_error,'-', ...
     x,trn_error,'o',x,chk_error,'*')
tmp = x(:, ones(1,3))';
X = tmp(:,);
tmp = [zeros(anfis_n,1) max(trn_error,chk_error) nan*ones(anfis_n,1)]';
Y = tmp(:,);
hold on
plot(X,Y,'g')
hold off
axis([1 anfis_n -inf inf])
h_gca = gca;
h_gca.XTickLabel = [];

% ====== Add text of input variables
for k = 1:anfis_n
    text(x(k), 0, ...
        [input_name(index(k,1),:) ' ' ...
        input_name(index(k,2),:) ' ' ...
        input_name(index(k,3),:)]);
end
h = findobj(gcf,'type','text');
set(h,'rot',90,'fontsize',11,'hori','right');
drawnow

% ====== Generate input_index for bjtrain.m
[a, b] = min(trn_error);
input_index = index(b,:);
title('Training (Circles) and Checking (Asterisks) Errors','fontsize',10)
ylabel('RMSE','fontsize',10)

Train 36 ANFIS models, each with 3 inputs selected from 10 candidates...

ANFIS model = 1: y(k-1) y(k-2) u(k-1) --> trn=0.0990, chk=0.0962
ANFIS model = 2: y(k-1) y(k-2) u(k-2) --> trn=0.0852, chk=0.0862
ANFIS model = 3: y(k-1) y(k-2) u(k-3) --> trn=0.0474, chk=0.0485
ANFIS model = 4: y(k-1) y(k-2) u(k-4) --> trn=0.0808, chk=0.0822
ANFIS model = 5: y(k-1) y(k-2) u(k-5) --> trn=0.1023, chk=0.0991
ANFIS model = 6: y(k-1) y(k-2) u(k-6) --> trn=0.1021, chk=0.0974
ANFIS model = 7: y(k-1) y(k-3) u(k-1) --> trn=0.1231, chk=0.1206
ANFIS model = 8: y(k-1) y(k-3) u(k-2) --> trn=0.1047, chk=0.1085
ANFIS model = 9: y(k-1) y(k-3) u(k-3) --> trn=0.0587, chk=0.0626
ANFIS model = 10: y(k-1) y(k-3) u(k-4) --> trn=0.0806, chk=0.0836
ANFIS model = 11: y(k-1) y(k-3) u(k-5) --> trn=0.1261, chk=0.1311
ANFIS model = 12: y(k-1) y(k-3) u(k-6) --> trn=0.1210, chk=0.1151
ANFIS model = 13: y(k-1) y(k-4) u(k-1) --> trn=0.1420, chk=0.1353
ANFIS model = 14: y(k-1) y(k-4) u(k-2) --> trn=0.1224, chk=0.1229
ANFIS model = 15: y(k-1) y(k-4) u(k-3) --> trn=0.0700, chk=0.0765
ANFIS model = 16: y(k-1) y(k-4) u(k-4) --> trn=0.0817, chk=0.0855
ANFIS model = 17: y(k-1) y(k-4) u(k-5) --> trn=0.1337, chk=0.1405
ANFIS model = 18: y(k-1) y(k-4) u(k-6) --> trn=0.1421, chk=0.1333
ANFIS model = 19: y(k-2) y(k-3) u(k-1) --> trn=0.2393, chk=0.2264
ANFIS model = 20: y(k-2) y(k-3) u(k-2) --> trn=0.2104, chk=0.2077
ANFIS model = 21: y(k-2) y(k-3) u(k-3) --> trn=0.1452, chk=0.1497
ANFIS model = 22: y(k-2) y(k-3) u(k-4) --> trn=0.0958, chk=0.1047
ANFIS model = 23: y(k-2) y(k-3) u(k-5) --> trn=0.2048, chk=0.2135
ANFIS model = 24: y(k-2) y(k-3) u(k-6) --> trn=0.2388, chk=0.2326
ANFIS model = 25: y(k-2) y(k-4) u(k-1) --> trn=0.2756, chk=0.2574
This window shows ANFIS predictions on both training and checking data sets. Obviously the performance is better than those of the ARX model.
if ishghandle(winH1),delete(winH1);
end

trn_data = data(1:trn_data_n,[input_index, size(data,2)]);
chk_data = data(trn_data_n+1:600,[input_index, size(data,2)]);

% generate FIS matrix
in_fismat = genfis(trn_data(:,1:end-1),trn_data(:,end));
anfisOpt = anfisOptions('InitialFIS',in_fismat,...
    'EpochNumber',1,...
    'InitialStepSize',0.01,...
    'StepSizeDecreaseRate',0.5,...
    'StepSizeIncreaseRate',1.5,...
    'ValidationData',chk_data);
[trn_out_fismat,trn_error,step_size,chk_out_fismat,chk_error] = ...
    anfis(trn_data,anfisOpt);

subplot(2,1,1)
index = 1:trn_data_n;
plot(index,y(index),index,yp(index),'.')
rmse = norm(y(index)-yp(index))/sqrt(length(index));
title(sprintf(['(a) Training Data (Solid Line) and ARX Prediction (Dots)
with RMSE = ' num2str(rmse)]),...'
    xlabel('Time Steps','fontsize',10)
subplot(2,1,2)
index = (trn_data_n+1):(total_data_n);
plot(index, y(index),index,yp(index),'.')
rmse = norm(y(index)-yp(index))/sqrt(length(index));
title(sprintf(['(b) Checking Data (Solid Line) and ARX Prediction (Dots)
with RMSE = ' num2str(rmse)]),...'
    xlabel('Time Steps','fontsize',10)

ANFIS info:
    % Number of nodes: 34
    Number of linear parameters: 32
    Number of nonlinear parameters: 18
    Total number of parameters: 50
    Number of training data pairs: 300
    Number of checking data pairs: 300
    Number of fuzzy rules: 8

Start training ANFIS …

1      0.0474113      0.0485325

3-167
Designated epoch number reached --> ANFIS training completed at epoch 1.

Minimal training RMSE = 0.047411
Minimal checking RMSE = 0.0485325
[na nb d] = 5  10   2

\[ y_{\text{hat}} = \text{evalfis}(\text{chk\_out\_fismat}, \text{data}(1:600,\text{input\_index})); \]

\[ \text{subplot}(2,1,1) \]
\[ \text{index} = 1:\text{trn\_data\_n}; \]
\[ \text{plot}(	ext{index}, \text{data}(\text{index},\text{size(data,2)}), '-', ',...\]
\[ \text{index}, y_{\text{hat}}(\text{index}), '.') \]
\[ \text{rmse} = \text{norm}(y_{\text{hat}}(\text{index})-\text{data}(\text{index},\text{size(data,2)}))/\sqrt{\text{length}(	ext{index})}; \]
Training Data (Solid Line) and ANFIS Prediction (Dots) with RMSE = 0.047411

Checking Data (Solid Line) and ANFIS Prediction (Dots) with RMSE = 0.048532
Conclusion

The table above is a comparison among various modeling approaches. The ARX modeling spends the least amount of time to reach the worst precision, and the ANFIS modeling via exhaustive search takes the most amount of time to reach the best precision. In other words, if fast modeling is the goal, then ARX is the right choice. But if precision is the utmost concern, then we should go with ANFIS, which is designed for nonlinear modeling and higher precision.

<table>
<thead>
<tr>
<th>Models</th>
<th>ARX Model</th>
<th>ANFIS Model (Sequential Search)</th>
<th>ANFIS Model (Exhaustive Search)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Input Arguments</td>
<td>14</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Training RMSE</td>
<td>0.1122</td>
<td>0.0609</td>
<td>0.0474</td>
</tr>
<tr>
<td>Checking RMSE</td>
<td>0.0749</td>
<td>0.0604</td>
<td>0.0485</td>
</tr>
<tr>
<td>No. of Linear Parameters</td>
<td>15</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>No. of Nonlinear Parameters</td>
<td>0</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Computation Time (Pentium-Pro 200 MHz, 64 MB RAM)</td>
<td>1.6 $s$</td>
<td>10.3 $s$</td>
<td>21.5 $s$</td>
</tr>
</tbody>
</table>

See Also

anfis | evalfis | genfis

More About

- “Neuro-Adaptive Learning and ANFIS” on page 3-92
Gas Mileage Prediction

This example shows how to predict of fuel consumption (miles per gallon) for automobiles, using data from previously recorded observations.

Introduction

Automobile MPG (miles per gallon) prediction is a typical nonlinear regression problem, in which several attributes of an automobile's profile information are used to predict another continuous attribute, the fuel consumption in MPG. The training data is available in the UCI (Univ. of California at Irvine) Machine Learning Repository and contains data collected from automobiles of various makes and models.

The table shown above is several observations or samples from the MPG data set. The six input attributes are no. of cylinders, displacement, horsepower, weight, acceleration, and model year. The output variable to be predicted is the fuel consumption in MPG. (The automobile's manufacturers and models in the first column of the table are not used for prediction).

Partitioning Data

The data set is obtained from the original data file 'auto-gas.dat'. The dataset is then partitioned into a training set (odd-indexed samples) and a checking set (even-indexed samples).

\[
\begin{align*}
[data, input\_name] &= \text{loadgas}; \\
\text{trn\_data} &= data(1:2:end,:); \\
\text{chk\_data} &= data(2:2:end,:);
\end{align*}
\]

Input Selection

The function exhsrch performs an exhaustive search within the available inputs to select the set of inputs that most influence the fuel consumption. The first parameter to the function specifies the number of input combinations to be tried during the search. Essentially, exhsrch builds an ANFIS model for each combination and trains it for one epoch and reports the performance achieved. In the following example, exhsrch is used to determine the one most influential input attribute in predicting the output.

\[
exhsrch(1, \text{trn\_data}, \text{chk\_data}, \text{input\_name});
\]

Train 6 ANFIS models, each with 1 inputs selected from 6 candidates...

ANFIS model 1: Cylinder --> trn=4.6400, chk=4.7255
Figure 1: Every input variable's influence on fuel consumption

The left-most input variable in Figure 1 has the least error or in other words the most relevance with respect to the output.

The plot and results from the function clearly indicate that the input attribute 'Weight' is the most influential. The training and checking errors are comparable, which implies that
there is no overfitting. This means we can push a little further and explore if we can select more than one input attribute to build the ANFIS model.

Intuitively, we can simply select Weight and Disp directly since they have the least errors as shown in the plot. However, this will not necessarily be the optimal combination of two inputs that result in the minimal training error. To verify this, we can use \texttt{exhsrch} to search for the optimal combination of 2 input attributes.

\begin{verbatim}
input_index = exhsrch(2,trn_data,chk_data,input_name);
Train 15 ANFIS models, each with 2 inputs selected from 6 candidates...

ANFIS model 1: Cylinder Disp --> trn=3.9320, chk=4.7920
ANFIS model 2: Cylinder Power --> trn=3.7364, chk=4.8683
ANFIS model 3: Cylinder Weight --> trn=3.8741, chk=4.6763
ANFIS model 4: Cylinder Acceler --> trn=4.3287, chk=5.9625
ANFIS model 5: Cylinder Year --> trn=3.7129, chk=4.5946
ANFIS model 6: Disp Power --> trn=3.8087, chk=3.8594
ANFIS model 7: Disp Weight --> trn=4.0271, chk=4.6350
ANFIS model 8: Disp Acceler --> trn=4.0782, chk=4.4890
ANFIS model 9: Disp Year --> trn=2.9565, chk=3.3905
ANFIS model 10: Power Weight --> trn=3.9310, chk=4.2976
ANFIS model 11: Power Acceler --> trn=4.2740, chk=3.8738
ANFIS model 12: Power Year --> trn=3.3796, chk=3.3505
ANFIS model 13: Weight Acceler --> trn=4.0875, chk=4.0095
ANFIS model 14: Weight Year --> trn=2.7657, chk=2.9953
ANFIS model 15: Acceler Year --> trn=5.6242, chk=5.6481
\end{verbatim}
**Figure 2**: All two input variable combinations and their influence on fuel consumption

The results from `exhsrch` indicate that Weight and Year form the optimal combination of two input attributes. The training and checking errors are getting distinguished, indicating the outset of overfitting. It may not be prudent to use more than two inputs for building the ANFIS model. We can test this premise to verify its validity.

```
exhsrch(3,trn_data,chk_data,input_name);
```

Train 20 ANFIS models, each with 3 inputs selected from 6 candidates...

- ANFIS model 1: Cylinder Disp Power --> trn=3.4446, chk=11.5329
- ANFIS model 2: Cylinder Disp Weight --> trn=3.6686, chk=4.8922
- ANFIS model 3: Cylinder Disp Acceler --> trn=3.6610, chk=5.2384
ANFIS model 4: Cylinder Disp Year --> trn=2.5463, chk=4.9001
ANFIS model 5: Cylinder Power Weight --> trn=3.4797, chk=9.3761
ANFIS model 6: Cylinder Power Acceler --> trn=3.5432, chk=4.4804
ANFIS model 7: Cylinder Power Year --> trn=2.6300, chk=3.6300
ANFIS model 8: Cylinder Weight Acceler --> trn=3.5708, chk=4.8379
ANFIS model 9: Cylinder Weight Year --> trn=2.4951, chk=4.0435
ANFIS model 10: Cylinder Acceler Year --> trn=3.2698, chk=6.2616
ANFIS model 11: Disp Power Weight --> trn=3.5879, chk=7.4942
ANFIS model 12: Disp Power Acceler --> trn=3.5395, chk=3.9953
ANFIS model 13: Disp Power Year --> trn=2.4607, chk=3.3563
ANFIS model 14: Disp Weight Acceler --> trn=3.6075, chk=4.2318
ANFIS model 15: Disp Weight Year --> trn=2.5617, chk=3.7866
ANFIS model 16: Disp Acceler Year --> trn=2.4149, chk=3.2480
ANFIS model 17: Power Weight Acceler --> trn=3.7884, chk=4.0480
ANFIS model 18: Power Weight Year --> trn=2.4371, chk=3.2852
ANFIS model 19: Power Acceler Year --> trn=2.7276, chk=3.2580
ANFIS model 20: Weight Acceler Year --> trn=2.3603, chk=2.9152
Figure 3: All three input variable combinations and their influence on fuel consumption

The plot shows the result of selecting three inputs, in which Weight, Year, and Acceler are selected as the best combination of three input variables. However, the minimal training (and checking) error do not reduce significantly from that of the best two-input model, which indicates that the newly added attribute Acceler does not improve the prediction much. For better generalization, we always prefer a model with a simple structure. Therefore we will stick to the two-input ANFIS for further exploration.

We then extract the selected input attributes from the original training and checking datasets.
close all;
new_trn_data = trn_data(:,[input_index, size(trn_data,2)]);
new_chk_data = chk_data(:,[input_index, size(chk_data,2)]);

**Training ANFIS Model**

The function `exhsrch` only trains each ANFIS for a single epoch in order to be able to quickly find the right inputs. Now that the inputs are fixed, we can spend more time on ANFIS training (100 epochs).

The `genfis` function generates a initial FIS from the training data, which is then fine-tuned by ANFIS to generate the final model.

```matlab
in_fismat = genfis(new_trn_data(:,1:end-1),new_trn_data(:,end));
anfisOpt = anfisOptions('InitialFIS',in_fismat,'EpochNumber',100,...
    'StepSizeDecreaseRate',0.5,...
    'StepSizeIncreaseRate',1.5,...
    'ValidationData',new_chk_data,...
    'DisplayANFISInformation',0,...
    'DisplayErrorValues',0,...
    'DisplayStepSize',0,...
    'DisplayFinalResults',0);
[trn_out_fismat,trn_error,step_size,chk_out_fismat,chk_error] = ...
    anfis(new_trn_data,anfisOpt);
```

ANFIS returns the error with respect to training data and checking data in the list of its output parameters. The plot of the errors provides useful information about the training process.

```matlab
[a,b] = min(chk_error);
plot(1:100,trn_error,'g-',1:100,chk_error,'r-',b,a,'ko')
title('Training (green) and checking (red) error curve','fontsize',10)
xlabel('Epoch numbers','fontsize',10)
ylabel('RMS errors','fontsize',10)
```

Gas Mileage Prediction
Figure 4: ANFIS training and checking errors

The plot above shows the error curves for 100 epochs of ANFIS training. The green curve gives the training errors and the red curve gives the checking errors. The minimal checking error occurs at about epoch 45, which is indicated by a circle. Notice that the checking error curve goes up after 50 epochs, indicating that further training over fits the data and produces worse generalization.

ANFIS vs Linear Regression

A good exercise at this point would be to check the performance of the ANFIS model with a linear regression model.
The ANFIS prediction can be compared against a linear regression model by comparing their respective RMSE (Root mean square) values against checking data.

```matlab
% Performing Linear Regression
N = size(trn_data,1);
A = [trn_data(:,1:6) ones(N,1)];
B = trn_data(:,7);
coef = A\B; % Solving for regression parameters from training data

Nc = size(chk_data,1);
A_ck = [chk_data(:,1:6) ones(Nc,1)];
B_ck = chk_data(:,7);
lr_rmse = norm(A_ck*coef-B_ck)/sqrt(Nc);

% Printing results
fprintf('
RMSE against checking data
ANFIS : %1.3f	Linear Regression : %1.3f
',a,lr_rmse);

RMSE against checking data
ANFIS : 2.978    Linear Regression : 3.444
```

It can be seen that the ANFIS model outperforms the linear regression model.

**Analyzing ANFIS Model**

The variable `chk_out_fismat` represents the snapshot of the ANFIS model at the minimal checking error during the training process. The input-output surface of the model is shown in the plot below.

```matlab
chk_out_fismat.Inputs(1).Name = "Weight";
chk_out_fismat.Inputs(2).Name = "Year";
chk_out_fismat.Outputs(1).Name = "MPG";

% Generating the FIS output surface plot
gensurf(chk_out_fismat);
```
Figure 5: Input-Output surface for trained FIS

The input-output surface shown above is a nonlinear and monotonic surface and illustrates how the ANFIS model will respond to varying values of 'weight' and 'year'.

Limitations and Cautions

We can see some spurious effects at the far-end corner of the surface. The elevated corner says that the heavier an automobile is, the more gas-efficient it will be. This is totally counter-intuitive, and it is a direct result from lack of data.

```matlab
plot(new_trn_data(:,1),new_trn_data(:, 2),'bo', ... new_chk_data(:,1),new_chk_data(:, 2),'rx')
xlabel('Weight','fontsize',10)
```
Figure 6: Weight vs Year plot showing lack of data in the upper-right corner

This plot above shows the data distribution. The lack of training data at the upper right corner causes the spurious ANFIS surface mentioned earlier. Therefore the prediction by ANFIS should always be interpreted with the data distribution in mind.

See Also
anfis | evalfis | genfis
More About

- “Neuro-Adaptive Learning and ANFIS” on page 3-92
Data Clustering

- “Fuzzy Clustering” on page 4-2
- “Cluster Quasi-Random Data Using Fuzzy C-Means Clustering” on page 4-4
- “Adjust Fuzzy Overlap in Fuzzy C-Means Clustering” on page 4-8
- “Fuzzy C-Means Clustering” on page 4-12
- “Fuzzy C-Means Clustering for Iris Data” on page 4-17
- “Model Suburban Commuting Using Subtractive Clustering” on page 4-22
- “Modeling Traffic Patterns using Subtractive Clustering” on page 4-34
- “Data Clustering Using Clustering Tool” on page 4-48
**Fuzzy Clustering**

**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 tools 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. To automatically generate this type of FIS, use the `genfis` command.

**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 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. To generate a Sugeno-type fuzzy inference system that models the behavior of input/output data, you can configure the `genfis` command to use FCM clustering.
Subtractive Clustering

If you do not have a clear idea how many clusters there should be for a given set of data, *subtractive clustering* is a fast, one-pass algorithm for estimating the number of clusters and the cluster centers for a set of data [2]. 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 using the subtractive clustering method.

To generate a Sugeno-type fuzzy inference system that models the behavior of input/output data, you can configure the `genfis` command to use subtractive clustering.

References


See Also

`fcm | genfis | subclust`

More About

- “Cluster Quasi-Random Data Using Fuzzy C-Means Clustering” on page 4-4
- “Model Suburban Commuting Using Subtractive Clustering” on page 4-22
- “Data Clustering Using Clustering Tool” on page 4-48
Cluster Quasi-Random Data Using Fuzzy C-Means Clustering

This example shows how FCM clustering works using quasi-random two-dimensional data.

Load the data set and plot it.

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

Next, invoke the command-line function, `fcm`, to find two clusters in this data set until the objective function is no longer decreasing much at all.
[center,U,objFcn] = fcm(fcmdData,2);

Iteration count = 1, obj. fcn = 8.970479
Iteration count = 2, obj. fcn = 7.197402
Iteration count = 3, obj. fcn = 6.325579
Iteration count = 4, obj. fcn = 4.586142
Iteration count = 5, obj. fcn = 3.893114
Iteration count = 6, obj. fcn = 3.810804
Iteration count = 7, obj. fcn = 3.799801
Iteration count = 8, obj. fcn = 3.797862
Iteration count = 9, obj. fcn = 3.797508
Iteration count = 10, obj. fcn = 3.797444
Iteration count = 11, obj. fcn = 3.797432
Iteration count = 12, obj. fcn = 3.797430

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 the following routines:

- initfcm - initializes the problem
- distfcm - performs Euclidean distance calculation
- stepfcm - performs one iteration of clustering

To view the progress of the clustering, plot the objective function.

figure
plot(objFcn)
title('Objective Function Values')
xlabel('Iteration Count')
ylabel('Objective Function Value')
Finally, plot the two cluster centers found by the fcm function. The large characters in the plot indicate cluster centers.

```matlab
maxU = max(U);
index1 = find(U(1,:) == maxU);
index2 = find(U(2,:) == maxU);
figure
line(fcmldata(index1,1), fcmldata(index1,2), 'linestyle', ...
    'none','marker', 'o','color','g')
line(fcmldata(index2,1),fcmldata(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)
```
Note: Every time you run this example, the `fcm` function initializes with different initial conditions. This behavior swaps the order in which the cluster centers are computed and plotted.

**See Also**

- `fcm`

**More About**

- “Fuzzy Clustering” on page 4-2
Adjust Fuzzy Overlap in Fuzzy C-Means Clustering

This example shows how to adjust the amount of fuzzy overlap when performing fuzzy c-means clustering.

Create a random data set. For reproducibility, initialize the random number generator to its default value.

```matlab
rng('default')
data = rand(100,2);
```

Specify fuzzy partition matrix exponents.

```matlab
M = [1.1 2.0 3.0 4.0];
```

The exponent values in \( M \) must be greater than 1, with smaller values specifying a lower degree of fuzzy overlap. In other words, as \( M \) approaches 1, the boundaries between the clusters become more crisp.

For each overlap exponent:

- Cluster the data.
- Classify each data point into the cluster for which it has the highest degree of membership.
- Find the data points with maximum membership values below 0.6. These points have a more fuzzy classification.
- To quantify the degree of fuzzy overlap, calculate the average maximum membership value across all data points. A higher average maximum membership value indicates that there is less fuzzy overlap.
- Plot the clustering results.

```matlab
for i = 1:4
    % Cluster the data.
    options = [M(i) NaN NaN 0];
    [centers,U] = fcm(data,2,options);
    
    % Classify the data points.
    maxU = max(U);
    index1 = find(U(1,:)==maxU);
    index2 = find(U(2,:)==maxU);
```
% Find data points with lower maximum membership values.
index3 = find(maxU < 0.6);

% Calculate the average maximum membership value.
averageMax = mean(maxU);

% Plot the results.
subplot(2,2,i)
plot(data(index1,1),data(index1,2),'ob')
hold on
plot(data(index2,1),data(index2,2),'or')
plot(data(index3,1),data(index3,2),'xk','LineWidth',2)
plot(centers(1,1),centers(1,2),'xb','MarkerSize',15,'LineWidth',3)
plot(centers(2,1),centers(2,2),'xr','MarkerSize',15,'LineWidth',3)
hold off
title(['M = ' num2str(M(i)) ', Ave. Max. = ' num2str(averageMax,3)])
end
A given data point is classified into the cluster for which it has the highest membership value, as indicated by maxU. A maximum membership value of 0.5 indicates that the point belongs to both clusters equally. The data points marked with a black x have maximum membership values below 0.6. These points have a greater degree of uncertainty in their cluster membership.

More data points with low maximum membership values indicate a greater degree of fuzzy overlap in the clustering result. The average maximum membership value, averageMax, provides a quantitative description of the overlap. An averageMax value of 1 indicates crisp clusters, with smaller values indicating more overlap.
See Also

fcn

More About

- “Fuzzy Clustering” on page 4-2
- “Cluster Quasi-Random Data Using Fuzzy C-Means Clustering” on page 4-4
Fuzzy C-Means Clustering

This example shows how to perform fuzzy c-means clustering on 2-dimensional data. For an example that clusters higher-dimensional data, see “Fuzzy C-Means Clustering for Iris Data” on page 4-17.

Fuzzy c-means (FCM) is a data clustering technique in which a data set is grouped into $N$ clusters with every data point in the dataset belonging to every cluster to a certain degree. For example, a data point that lies close to the center of a cluster will have a high degree of membership in that cluster, and another data point that lies far away from the center of a cluster will have a low degree of membership to that cluster.

The `fcm` function performs FCM clustering. It starts with a random initial guess for the cluster centers; that is the mean location of each cluster. Next, `fcm` assigns every data point a random membership grade for each cluster. By iteratively updating the cluster centers and the membership grades for each data point, `fcm` moves the cluster centers to the correct location within a data set and, for each data point, finds the degree of membership in each cluster. This iteration minimizes an objective function that represents the distance from any given data point to a cluster center weighted by the membership of that data point in the cluster.

Load Data

Load the five sample data sets, and select a data set to cluster. These data sets have different numbers of clusters and data distributions.

```matlab
load fcmdata
dataset = ;
```

Specify FCM Settings

Configure the clustering algorithm settings. For more information on these settings, see `fcm`. To obtain accurate clustering results for each data set, try different clustering options.

Specify the number of clusters to compute, which must be greater than 1.

```matlab
N = ;
```

Specify the exponent the fuzzy partition matrix, which controls the degree of fuzzy overlap between clusters. This value must be greater than 1, with smaller values creating more crisp cluster boundaries. For more information, see “Adjust Fuzzy Overlap in Fuzzy C-Means Clustering” on page 4-8.
exponent = ;

Specify the maximum number of optimization iterations.

maxIterations = ;

Specify the minimum improvement in the objective function between successive iterations. When the objective function improves by a value below this threshold, the optimization stops. A smaller value produces more accurate clustering results, but the clustering can take longer to converge.

minImprovement = ;

Specify whether to display the objective function value after each iteration.

displayObjectiveFunction = ;

Create an option vector for the fcm function using these settings.

options = [exponent maxIterations minImprovement displayObjectiveFunction];

Cluster Data

Cluster the data into N clusters.

[C,U] = fcm(dataset,N,options);

C contains the computed centers for each cluster. U contains the computed fuzzy partition matrix, which indicates the degree of membership of each data point within each cluster.

Classify each data point into the cluster for which it has the highest degree of membership.

maxU = max(U);
index = cell(N,1);
for i=1:N
    index{i} = find(U(i,:) == maxU);
end

Plot Clustering Results

Plot the clustering results.

figure
hold on
for i=1:N
    plot(dataset(index{i},1),dataset(index{i},2), 'o')
    plot(C(i,1),C(i,2), 'xk', 'MarkerSize', 15, 'LineWidth', 3)
end
hold off

The data points in each cluster are shown in a different colors. The center for each cluster is shown as a black X.

**Plot Data Point Membership Values**

Select a cluster for which to plot a membership function surface.

cluster = ;
Obtain the membership function for the selected cluster by fitting a surface to the cluster membership values for all data points. For more information on interpolating scattered 3-D data, see griddata.

```matlab
[X,Y] = meshgrid(0:0.05:1, 0:0.05:1);
Z = griddata(dataset(:,1),dataset(:,2),U(cluster,:),X,Y);
surf(X,Y,Z)
```
When you decrease the exponent value, the transition from maximum full cluster membership to zero cluster membership becomes more steep; that is, the cluster boundary becomes more crisp.

See Also
fcm

More About
• “Fuzzy Clustering” on page 4-2
Fuzzy C-Means Clustering for Iris Data

This example shows how to use fuzzy c-means clustering for the iris data set. This dataset was collected by botanist Edgar Anderson and contains random samples of flowers belonging to three species of iris flowers: *setosa*, *versicolor*, and *virginica*. For each of the species, the data set contains 50 observations for sepal length, sepal width, petal length, and petal width.

**Load Data**

Load the data set from the `iris.dat` data file.

```
load iris.dat
```

Partition the data into three groups named *setosa*, *versicolor*, and *virginica*.

```
setosaIndex = iris(:,5)==1;
versicolorIndex = iris(:,5)==2;
virginicaIndex = iris(:,5)==3;

setosa = iris(setosaIndex,:);
versicolor = iris(versicolorIndex,:);
virginica = iris(virginicaIndex,:);
```

**Plot Data in 2-D**

The iris data contains four dimensions representing sepal length, sepal width, petal length, and petal width. Plot the data points for each combination of two dimensions.

```
Characteristics = {'sepal length','sepal width','petal length','petal width'};
pairs = [1 2; 1 3; 1 4; 2 3; 2 4; 3 4];

for i = 1:6
    x = pairs(i,1);
y = pairs(i,2);
    subplot(2,3,i)
    plot([setosa(:,x) versicolor(:,x) virginica(:,x)], ...
         [setosa(:,y) versicolor(:,y) virginica(:,y)], '.')
xlabel(Characteristics{x})
ylabel(Characteristics{y})
end
```
Setup Parameters

Specify the options for clustering the data using fuzzy c-means clustering. These options are:

- **Nc** — Number of clusters
- **M** — Fuzzy partition matrix exponent, which indicates the degree of fuzzy overlap between clusters. For more information, see “Adjust Fuzzy Overlap in Fuzzy C-Means Clustering” on page 4-8.
- **maxIter** — Maximum number of iterations. The clustering process stops after this number of iterations.
• **minImprove** — Minimum improvement. The clustering process stops when the objective function improvement between two consecutive iterations is less than this value.

\[ \text{Nc} = 3; \]
\[ \text{M} = 2.0; \]
\[ \text{maxIter} = 100; \]
\[ \text{minImprove} = 1 \times 10^{-6}; \]

For more information about these options and the fuzzy c-means algorithm, see `fcm`.

**Compute Clusters**

Fuzzy c-means clustering is an iterative process. Initially, the `fcm` function generates a random fuzzy partition matrix. This matrix indicates the degree of membership of each data point in each cluster.

In each clustering iteration, `fcm` calculates the cluster centers and updates the fuzzy partition matrix using the calculated center locations. It then computes the objective function value.

Cluster the data, displaying the objective function value after each iteration.

```matlab
clusteringOptions = [M maxIter minImprove true];
[centers,U] = fcm(iris,Nc,clusteringOptions);
```

<table>
<thead>
<tr>
<th>Iteration count</th>
<th>obj. fcn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28838.424340</td>
</tr>
<tr>
<td>2</td>
<td>21010.880067</td>
</tr>
<tr>
<td>3</td>
<td>15272.280943</td>
</tr>
<tr>
<td>4</td>
<td>11029.756194</td>
</tr>
<tr>
<td>5</td>
<td>10550.015503</td>
</tr>
<tr>
<td>6</td>
<td>10301.776800</td>
</tr>
<tr>
<td>7</td>
<td>9283.793786</td>
</tr>
<tr>
<td>8</td>
<td>7344.379868</td>
</tr>
<tr>
<td>9</td>
<td>6575.117093</td>
</tr>
<tr>
<td>10</td>
<td>6295.215539</td>
</tr>
<tr>
<td>11</td>
<td>6107.998500</td>
</tr>
<tr>
<td>12</td>
<td>6080.461019</td>
</tr>
<tr>
<td>13</td>
<td>6068.116247</td>
</tr>
<tr>
<td>14</td>
<td>6062.713326</td>
</tr>
<tr>
<td>15</td>
<td>6060.390433</td>
</tr>
<tr>
<td>16</td>
<td>6059.403978</td>
</tr>
<tr>
<td>17</td>
<td>6058.988494</td>
</tr>
</tbody>
</table>
Iteration count = 19, obj. fcn = 6058.814438
Iteration count = 20, obj. fcn = 6058.741777
Iteration count = 21, obj. fcn = 6058.711512
Iteration count = 22, obj. fcn = 6058.698925
Iteration count = 23, obj. fcn = 6058.693695
Iteration count = 24, obj. fcn = 6058.691523
Iteration count = 25, obj. fcn = 6058.690622
Iteration count = 26, obj. fcn = 6058.690247
Iteration count = 27, obj. fcn = 6058.690092
Iteration count = 28, obj. fcn = 6058.690028
Iteration count = 29, obj. fcn = 6058.690001
Iteration count = 30, obj. fcn = 6058.689990
Iteration count = 31, obj. fcn = 6058.689985
Iteration count = 32, obj. fcn = 6058.689983
Iteration count = 33, obj. fcn = 6058.689983

The clustering stops when the objective function improvement is below the specified minimum threshold.

Plot the computed cluster centers as bold numbers.

```matlab
for i = 1:6
    subplot(2,3,i);
    for j = 1:Nc
        x = pairs(i,1);
        y = pairs(i,2);
        text(centers(j,x),centers(j,y),int2str(j),'FontWeight','bold');
    end
end
```
See Also

fcm

More About

• “Fuzzy Clustering” on page 4-2
Model Suburban Commuting Using Subtractive Clustering

This example shows how to model the relationship between the number of automobile trips generated from an area and the demographics of the area using the \texttt{genfis} function. 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 and plot the data.

\begin{verbatim}
mytripdata
subplot(2,1,1)
plot(datin)
ylabel('input')
subplot(2,1,2)
plot(datout)
ylabel('output')
\end{verbatim}
The `mytripdata` command 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 pair variables are `chkdatin` and `chkdatout`.

Generate a model from the data using subtractive clustering using the `genfis` command.

First, create a `genfisOptions` option set for subtractive clustering, specifying the `ClusterInfluenceRange` range property. The `ClusterInfluenceRange` property 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, and results in many rules. Specifying a large cluster radius usually yields a few large clusters in the data, and results in fewer rules.
opt = genfisOptions('SubtractiveClustering','ClusterInfluenceRange',0.5);

Generate the FIS model using the training data and the specified options.

fismat = genfis(datin,datout,opt);

The `genfis` command uses a one-pass method that does not perform any iterative optimization. The model type for the generated FIS object is a first order Sugeno model with three rules.

Verify the model. Here, `trnRMSE` is the root mean squared error of the system generated by the training data.

\[
\text{fuzout} = \text{evalfis}(\text{fismat},\text{datin});
\]

\[
\text{trnRMSE} = \frac{\text{norm}(\text{fuzout}-\text{datout})}{\sqrt{\text{length}(\text{fuzout})}}
\]

\[
\text{trnRMSE} = 0.5276
\]

Next, apply the test data to the FIS to validate the model. In this example, the validation data is used for both checking and testing the FIS parameters. Here, `chkRMSE` is the root mean squared error of the system generated by the validation data.

\[
\text{chkfuzout} = \text{evalfis}(\text{fismat},\text{chkdatin});
\]

\[
\text{chkRMSE} = \frac{\text{norm}(\text{chkfuzout}-\text{chkdatout})}{\sqrt{\text{length}(\text{chkfuzout})}}
\]

\[
\text{chkRMSE} = 0.6179
\]

Plot the output of the model, `chkfuzout`, against the validation data, `chkdatout`.

```matlab
figure
plot(chkdatout)
hold on
plot(chkfuzout,'o')
hold off
```
The model output and validation data are shown as circles and solid blue line, respectively. The plot shows that the model does not perform well on the validation data.

At this point, you can use the optimization capability of anfis to improve the model. First, try using a relatively short training period (20 epochs) without using validation data, and then test the resulting FIS model against the testing data.

```matlab
anfisOpt = anfisOptions('InitialFIS',fismat,'EpochNumber',20,...
                          'InitialStepSize',0.1);
fismat2 = anfis([datin datout],anfisOpt);
```

ANFIS info:
  Number of nodes: 44
  Number of linear parameters: 18
Number of nonlinear parameters: 30
Total number of parameters: 48
Number of training data pairs: 75
Number of checking data pairs: 0
Number of fuzzy rules: 3

Start training ANFIS ...

1  0.527607
2  0.513727
3  0.492996
4  0.499985
5  0.490585
6  0.492924
7  0.48733
Step size decreases to 0.090000 after epoch 7.
8  0.485036
9  0.480813
10  0.475097
Step size increases to 0.099000 after epoch 10.
11  0.469759
12  0.462516
13  0.451177
14  0.447856
Step size increases to 0.108900 after epoch 14.
15  0.444357
16  0.433904
17  0.433739
18  0.420408
Step size increases to 0.119790 after epoch 18.
19  0.420512
20  0.420275

Designated epoch number reached --> ANFIS training completed at epoch 20.

Minimal training RMSE = 0.420275

After the training is complete, validate the model.

fuzout2 = evalfis(fismat2,datin);
trnRMSE2 = norm(fuzout2-datout)/sqrt(length(fuzout2))

trnRMSE2 = 0.4203
chkfuzout2 = evalfis(fismat2,chkdatin);
chkRMSE2 = norm(chkfuzout2-chkdatout)/sqrt(length(chkfuzout2))

chkRMSE2 = 0.5894

The model has improved a lot with respect to the training data, but only a little with respect to the validation data. Plot the improved model output obtained using anfis against the testing data.

```matlab
figure
plot(chkdatout)
hold on
plot(chkfuzout2,'o')
hold off
```
The model output and validation data are shown as circles and solid blue line, respectively. This plot shows that subtractive clustering with `genfis` can be used as a standalone, fast method for generating a fuzzy model from data, or as a preprocessor to determine the initial rules for `anfis` training. 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 result reduces the problem of an excessive propagation of rules when the input data has a high dimension.

Overfitting can be detected when the checking error starts to increase while the training error continues to decrease.

To check the model for overfitting, use `anfis` with validation data to train the model for 200 epochs.

First configure the ANFIS training options by modifying the existing `anfisOptions` option set. Specify the epoch number and validation data. Since the number of training epochs is larger, suppress the display of training information to the Command Window.

```matlab
anfisOpt.EpochNumber = 200;
anfisOpt.ValidationData = [chkdatin chkdatout];
anfisOpt.DisplayANFISInformation = 0;
anfisOpt.DisplayErrorValues = 0;
anfisOpt.DisplayStepSize = 0;
anfisOpt.DisplayFinalResults = 0;
```

Train the FIS.

```matlab
[fismat3, trnErr, stepSize, fismat4, chkErr] = anfis([datin datout], anfisOpt);
```

Here,

- `fismat3` is the FIS object when the training error reaches a minimum.
- `fismat4` is the snapshot FIS object when the validation data error reaches a minimum.
- `stepSize` is a history of the training step sizes.
- `trnErr` is the RMSE using the training data
- `chkErr` is the RMSE using the validation data for each training epoch.

After the training completes, validate the model.

```matlab
fuzout4 = evalfis(fismat4, datin);
trnRMSE4 = norm(fuzout4 - datout) / sqrt(length(fuzout4))
```
trnRMSE4 = 0.3393
chkfuzout4 = evalfis(fismat4, chkdatin);
chkRMSE4 = norm(chkfuzout4 - chkdatout) / sqrt(length(chkfuzout4))
chkRMSE4 = 0.5834

The error with the training data is the lowest thus far, and the error with the validation data is also slightly lower than before. This result suggests possible overfitting, which occurs when you fit the fuzzy system to the training data so well that it no longer does a good job of fitting the validation data. The result is a loss of generality.

View the improved model output. Plot the model output against the checking data.

    figure
    plot(chkdatout)
    hold on
    plot(chkfuzout4, 'o')
    hold off
The model output and validation data are shown as circles and solid blue line, respectively.

Next, plot the training error, `trnErr`.

```matlab
figure
plot(trnErr)
title('Training Error')
xlabel('Number of Epochs')
ylabel('Training Error')
```
This plot shows that the training error settles at about the 60th epoch point.

Plot the checking error, chkErr.

```matlab
figure
plot(chkErr)
title('Checking Error')
xlabel('Number of Epochs')
ylabel('Checking Error')
```
The plot shows that the smallest value of the validation data error occurs at the 52nd epoch. After this point 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 ability of the model to generalize the test data.

You can also compare the output of fismat2 and fistmat4 against the validation data, chkdatout.

```matlab
figure
plot(chkdatout)
hold on
plot(chkfuzout4,'ob')
plot(chkfuzout2,'+r')
```
See Also

anfis | subclust

More About
  • “Fuzzy Clustering” on page 4-2
Modeling Traffic Patterns using Subtractive Clustering

This example shows how to use subtractive clustering to model traffic patterns in an area based on the demographics of the area.

**The Problem: Understanding Traffic Patterns**

In this example we attempt to understand the relationship between the number of automobile trips generated from an area and the area's demographics. Demographic and trip data were collected from 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.

Hereon, the demographic factors will be addressed as inputs and the trips generated will be addressed as output. Hence our problem has five input variables (five demographic factors) and one output variable (number of trips generated).

**The Data**

Load the input and output variables used for this example into the workspace.

```matlab
tripdata
```

Two variables are loaded in the workspace, `datin` and `datout`. `datin` has 5 columns representing the 5 input variables and `datout` has 1 column representing the 1 output variable.

```matlab
subplot(2,1,1) plot(datin) legend('population','num. of dwelling units','vehicle ownership',... 'median household income','total employment') title('Input Variables','fontsize',10)

subplot(2,1,2) plot(datout) legend('num of trips') title('Output Variable','fontsize',10)
```
The number of rows in $\text{datin}$ and $\text{datout}$, 75, represent the number of observations or samples or datapoints available. A row in $\text{datin}$, say row 11, constitutes a set of observed values of the 5 input variables (population, number of dwelling units, vehicle ownership, median household income and total employment) and the corresponding row, row 11, in $\text{datout}$ represents the observed value for the number of trips generated given the observations made for the input variables.

We will model the relationship between the input variables (demographics) and the output variable (trips) by first clustering the data. The cluster centers will then be used as a basis to define a fuzzy inference system (FIS) which can then be used to explore and understand traffic patterns.
Why Clustering and Fuzzy Logic?

Clustering can be a very effective technique to identify natural groupings in data from a large data set, thereby allowing concise representation of relationships embedded in the data. In this example, clustering allows us to group traffic patterns into broad categories hence allowing for easier understandability.

Fuzzy logic is an effective paradigm to handle imprecision. It can be used to take fuzzy or imprecise observations for inputs and yet arrive at crisp and precise values for outputs. Also, a fuzzy inference system is a way to build systems without using complex analytical equations.

In this example, fuzzy logic is used to capture the broad categories identified during clustering into a Fuzzy Inference System (FIS). The FIS will then act as a model that will reflect the relationship between demographics and auto trips.

Clustering and fuzzy logic together provide a simple yet powerful means to model the traffic relationship that we want to study.

Clustering the Data

subclust is the function that implements a clustering technique called subtractive clustering. Subtractive clustering, [1], is a fast, one-pass algorithm for estimating the number of clusters and the cluster centers in a dataset.

In this section, we will see how subtractive clustering is performed on a dataset and in the next section we will explore independently how clustering is used to build a Fuzzy Inference System (FIS).

[C,S] = subclust([datin datout],0.5);

The first argument to the subclust function is the data to be clustered. The second argument to the function is the radii which marks a cluster's radius of influence in the input space.

The variable C now holds all the centers of the clusters that have been identified by subclust. Each row of C contains the position of a cluster.

C

C =
In this case, C has 3 rows representing 3 clusters with 6 columns representing the positions of the clusters in each dimension.

subclust has hence identified 3 natural groupings in the demographic-trip dataset being considered. The following plot shows how the clusters have been identified in the 'total employment' and 'trips' dimensions of the input space.

clf
plot(datin(:,5),datout(:,1),'.',C(:,5),C(:,6),'r*')
legend('Data points','Cluster centers','Location','SouthEast')
xlabel('total employment','fontsize',10)
ylabel('num of trips','fontsize',10)
title('Data and Clusters in selected two dimensions of the input space','fontsize',10)
Figure 2: Cluster centers in the 'total employment' and 'trips' dimensions of the input space

The variable $S$ 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.

$S$

$$S = \begin{bmatrix} 1.1621 & 0.4117 & 0.6555 & 7.6139 & 2.8931 & 1.4395 \end{bmatrix}$$
S in this case has 6 columns representing the influence of the cluster centers on each of the 6 dimensions.

**Generating the Fuzzy Inference System (FIS)**

genfis is the function that creates a FIS using subtractive clustering. genfis employs subclust behind the scenes to cluster the data and uses the cluster centers and their range of influences to build a FIS which will then be used to explore and understand traffic patterns.

```matlab
myfis=genfis(datin,datout, ...
    genfisOptions('SubtractiveClustering','ClusterInfluenceRange',0.5));
```

The first argument is the input variables matrix datin, the second argument is the output variables matrix datout and the third argument is the radii that should be used while using subclust.

genfis assigns default names for inputs, outputs and membership functions. For our understanding it is beneficial to rename the inputs and outputs meaningfully.

Assign names to the inputs and outputs.

```matlab
myfis.Inputs(1).Name = "population";
myfis.Inputs(2).Name = "dwelling units";
myfis.Inputs(3).Name = "num vehicles";
myfis.Inputs(4).Name = "income";
myfis.Inputs(5).Name = "employment";
myfis.Outputs(1).Name = "num of trips";
```

**Understanding the Clusters-FIS Relationship**

An FIS is composed of inputs, outputs, and rules. Each input and output can have any number of membership functions. The rules dictate the behavior of the fuzzy system based on inputs, outputs and membership functions. genfis constructs the FIS in an attempt to capture the position and influence of each cluster in the input space.

myfis is the FIS that genfis has generated. Since the dataset has 5 input variables and 1 output variable, genfis constructs a FIS with 5 inputs and 1 output. Each input and output has as many membership functions as the number of clusters that subclust has identified. As seen previously, for the current dataset subclust identified 3 clusters. Therefore each input and output will be characterized by 3 membership functions. Also, the number of rules equals the number of clusters and hence 3 rules are created.
We can now probe the FIS to understand how the clusters got converted internally into membership functions and rules using the Fuzzy Logic Designer app.

```
fuzzyLogicDesigner(myfis)
```

**Figure 3:** The graphical editor for building Fuzzy Inference Systems (FIS)

As can be seen, the FIS has 5 inputs and 1 output with the inputs mapped to the outputs through a rule base (white box in the figure).
Let's now try to analyze how the cluster centers and the membership functions are related.

```matlab
mfedit(myfis)
```

**Figure 4:** The graphical membership function editor
mfedit(myfis) launches the graphical membership function editor. It can also be launched by clicking on the inputs or the outputs in the FIS editor launched by fuzzyLogicDesigner.

Notice that all the inputs and outputs have exactly 3 membership functions. The 3 membership functions represent the 3 clusters that were identified by subclust.

Each input in the FIS represents an input variable in the input dataset datin and each output in the FIS represents an output variable in the output dataset datout.

By default, the first membership function, in1cluster1, of the first input population would be selected in the membership function editor. Notice that the membership function type is gaussmf (Gaussian type membership function) and the parameters of the membership function are \([1.162 \ 1.877]\), where 1.162 represents the spread coefficient of the Gaussian curve and 1.877 represents the center of the Gaussian curve. in1cluster1 captures the position and influence of the first cluster for the input variable population. \((C(1,1)=1.877, \ S(1)=1.1621\ )\)

Similarly, the position and influence of the other 2 clusters for the input variable population are captured by the other two membership functions in1cluster2 and in1cluster3.

The rest of the 4 inputs follow the exact pattern mimicking the position and influence of the 3 clusters along their respective dimensions in the dataset.

Now, let's explore how the fuzzy rules are constructed.

ruleedit(myfis)
Figure 5: The graphical rule editor

ruleedit is the graphical fuzzy rule editor. As you can notice, there are exactly three rules. Each rule attempts to map a cluster in the input space to a cluster in the output space.

The first rule can be explained simply as follows. If the inputs to the FIS, population, dwelling units, num vehicles, income, and employment, strongly belong to their respective cluster1 membership functions then the output, num of trips, must
strongly belong to its cluster1 membership function. The (1) at the end of the rule is to indicate that the rule has a weight or an importance of "1". Weights can take any value between 0 and 1. Rules with lesser weights will count for less in the final output.

The significance of the rule is that it succinctly maps cluster 1 in the input space to cluster 1 in the output space. Similarly, the other two rules map cluster 2 and cluster 3 in the input space to cluster 2 and cluster 3 in the output space.

If a datapoint closer to the first cluster, or in other words having strong membership to the first cluster, is fed as input to myfis then rule1 will fire with more firing strength than the other two rules. Similarly, an input with strong membership to the second cluster will fire the second rule will with more firing strength than the other two rules and so on.

The output of the rules (firing strengths) are then used to generate the output of the FIS through the output membership functions.

The one output of the FIS, num of trips, has 3 linear membership functions representing the 3 clusters identified by subclust. The coefficients of the linear membership functions though are not taken directly from the cluster centers. Instead, they are estimated from the dataset using least squares estimation technique.

All 3 membership functions in this case will be of the form a*population + b*dwelling units + c*num vehicles + d*income + e*employment + f, where a, b, c, d, e and f represent the coefficients of the linear membership function. Click on any of the num of trips membership functions in the membership function editor to observe the parameters of these linear membership functions.

**Using the FIS for Data Exploration**

You can now use the FIS that has been constructed to understand the underlying dynamics of relationship being modeled.

surfview(myfis)
surfview is the surface viewer that helps view the input-output surface of the fuzzy system. In other words, this tool simulates the response of the fuzzy system for the entire range of inputs that the system is configured to work for. Thereafter, the output or the response of the FIS to the inputs are plotted against the inputs as a surface. This visualization is very helpful to understand how the system is going to behave for the entire range of values in the input space.

In the plot above the surface viewer shows the output surface for two inputs population and num of dwelling units. As you can see the number of auto trips increases with increase in population and dwelling units, which sounds very rational. You can change the...
inputs in the X and Y drop-down boxes to observe the output surface with respect to the inputs you choose.

```matlab
ruleview(myfis)
```

**Figure 7:** Rule viewer that simulates the entire fuzzy inference process

`ruleview` is the graphical simulator for simulating the FIS response for specific values of the input variables. Now, having built the fuzzy system, if we want to understand how
many trips will occur for a demographic setup, say an area with a particular population, a
certain number of dwelling units and so on, this tool will help you simulate the FIS
response for the input of your choice.

Another feature of this GUI tool is, it gives you a snapshot of the entire fuzzy inference
process, right from how the membership functions are being satisfied in every rule to how
the final output is being generated through defuzzification.

**Conclusion**

This example has attempted to convey how clustering and fuzzy logic can be employed as
effective techniques for data modeling and analysis.

Fuzzy logic has also found various applications in other areas of technology like non-
linear control, automatic control, signal processing, system identification, pattern
recognition, time series prediction, data mining, financial applications etc.

**Reference**


**See Also**

*anfis|subclust*

**More About**

- “Fuzzy Clustering” on page 4-2
Data Clustering Using Clustering Tool

The Clustering tool implements the fuzzy data clustering functions `fcm` and `subclust`, and lets you perform clustering on data. For more information on the clustering methods, see “Fuzzy Clustering” on page 4-2.

To open the tool, at the MATLAB command line, type:

`findcluster`
Use the Clustering tool to perform the following tasks:

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

Access the online help topics by clicking Info or using the Help menu.

**Load and Plot Data**

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

- Click **Load Data**, and select the file containing the data.
- Open the Clustering Tool with a data set directly by calling `findcluster` with the data set as an input argument.

For example, enter:

```markdown
findcluster('clusterdemo.dat')
```

The data set file must have the extension `.dat`. Each line of the data set file contains one data point. For example, if you have 5-dimensional data with 100 data points, the file contains 100 lines, and each line contains five values.

The Clustering 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**.

**Cluster Data**

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:

   - Fuzzy c-means clustering using the **Cluster Num**, **Max Iteration**, **Min**, and **Exponent** fields. For information on these options, see `fcm`.
   - Subtractive clustering using the **Influence Range**, **Squash**, **AspectRatio**, and **Reject Ratio** fields. For information on these options, see `subclust`.
3 Cluster the data by clicking **Start**.

Once the clustering is complete, the cluster centers appear in black as shown in the next figure.

**Tip** Using the Clustering tool, you can obtain only the computed cluster centers. To obtain additional information for:
• Fuzzy c-means clustering, such as the fuzzy partition matrix, cluster the data using \texttt{fcm}.
• Subtractive clustering, such as the range of influence in each data dimension, cluster the data using \texttt{subclust}.

To use the same clustering data with either \texttt{fcm} or \texttt{subclust}, first load the data file into the MATLAB workspace. For example, at the MATLAB command line, type:

\begin{verbatim}
load clusterdemo.dat
\end{verbatim}

\section*{Save Cluster Centers}

To save the cluster centers, click \textbf{Save Center}.

\section*{See Also}

\texttt{fcm} | \texttt{findcluster} | \texttt{subclust}

\section*{More About}

• “Fuzzy Clustering” on page 4-2
Fuzzy Logic in Simulink

- “Simulate Fuzzy Inference Systems in Simulink” on page 5-2
- “Water Level Control in a Tank” on page 5-15
- “Temperature Control in a Shower” on page 5-23
- “Implement Fuzzy PID Controller in Simulink Using Lookup Table” on page 5-30
Simulate Fuzzy Inference Systems in Simulink

You can simulate a fuzzy inference system (FIS) in Simulink using either the Fuzzy Logic Controller or Fuzzy Logic Controller with Ruleviewer blocks. Alternatively, you can evaluate fuzzy systems at the command line using `evalfis`.

Using the Fuzzy Logic Controller, you can simulate traditional type-1 fuzzy inference systems (mamfis and sugfis) and type-2 fuzzy inference systems (mamfistype2 and sugfistype2). The Fuzzy Logic Controller with Ruleviewer block supports only type-1 systems.

For more information on creating fuzzy inference systems, see “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17 and “Build Fuzzy Systems at the Command Line” on page 2-41.

Simulate Fuzzy Inference System

Once you have implemented a fuzzy inference system using Fuzzy Logic Designer, using Neuro-Fuzzy Designer, or at the command line, you can simulate the system in Simulink.

For this example, you control the level of water in a tank using a fuzzy inference system implemented using a Fuzzy Logic Controller block. Open the sltank model.

`open_system('sltank')`
For this system, you control the water that flows into the tank using a valve. The outflow rate depends on the diameter of the output pipe, which is constant, and the pressure in the tank, which varies with water level. Therefore, the system has nonlinear characteristics.

The two inputs to the fuzzy system are the water level error, level, and the rate of change of the water level, rate. The output of the fuzzy system is the rate at which the control valve is opening or closing, valve.

To implement a fuzzy inference system, specify the **FIS name** parameter of the Fuzzy Logic Controller block as the name of a FIS object in the MATLAB® workspace. In this example, the block uses the `tank.mamfis` object.

For more information on this system, see “Water Level Control in a Tank” on page 5-15.

As a first attempt to control the water level, set the following rules in the FIS. These rules adjust the valve based on only the water level error.
• If the water level is okay, then do not adjust the valve.
• If the water level is low, then open the valve quickly.
• If the water level is high, then close the valve quickly.

Specify the rules by creating a vector of `fisrule` objects and assigning it to the `Rules` property of the `tank` FIS object.

```matlab
rule1 = "If level is okay then valve is no_change";
rule2 = "If level is low then valve is open_fast";
rule3 = "If level is high then valve is close_fast";
rules = [rule1 rule2 rule3];
tank.Rules = fisrule(rules);
```

Simulate the model, and view the water level.

```matlab
open_system('sltank/Comparison')
sim('sltank',100)
```
These rules are insufficient for controlling the system, since the water level oscillates around the setpoint.

To reduce the oscillations, add two more rules to the system. These rules adjust the valve based on the rate of change of the water level when the water level is near the setpoint.

- If the water level is okay and increasing, then close the valve slowly.
- If the water level is okay and decreasing, then open the valve slowly.

To add these rules, use the `addRule` function.

```matlab
rule4 = "If level is okay and rate is positive then valve is close_slow";
rule5 = "If level is okay and rate is negative then valve is open_slow";
newRules = [rule4 rule5];
tank = addRule(tank,newRules);
```

Simulate the model.

```matlab
sim('sltank',100)
```
The water level now tracks the setpoint without oscillating.

You can also simulate fuzzy systems using the Fuzzy Logic Controller with RuleViewer block. The `sltankrule` model is the same as the `sltank` model, except that it uses the Fuzzy Logic Controller with RuleViewer block.

```matlab
open_system('sltankrule')
```
During simulation, this block displays the Rule Viewer from the **Fuzzy Logic Designer** app.

```matlab
sim('sltankrule',100)
```
If you pause the simulation, you can examine the FIS behavior by manually adjusting the input variable values in the Rule Viewer, and observing the inference process and output.

You can also access the Fuzzy Logic Designer editors from the Rule Viewer. From the Rule Viewer, you can then adjust the parameters of your fuzzy system using these editors, and export the updated system to the MATLAB workspace. To simulate the updated FIS, restart the simulation. For more information on using these editors, see “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17.

**Access Intermediate Fuzzy Inference Results**

You can access intermediate fuzzy inference results using the Fuzzy Logic Controller block. You can use this data to visualize the fuzzy inference process or troubleshoot the performance of your FIS. To access this data, enable the corresponding parameters in the block, and connect signals to the corresponding output ports.
<table>
<thead>
<tr>
<th>Block Parameter</th>
<th>Description</th>
<th>Output Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzified Inputs</td>
<td>Fuzzified input values, obtained by evaluating the input membership functions of each rule at the current input values.</td>
<td>fi</td>
</tr>
<tr>
<td>Rule firing strengths</td>
<td>Rule firing strengths, obtained by evaluating the antecedent of each rule.</td>
<td>rfs</td>
</tr>
<tr>
<td>Rule outputs</td>
<td>Rule outputs, obtained by evaluating the consequent of each rule.</td>
<td>ro</td>
</tr>
<tr>
<td>Aggregated outputs</td>
<td>Aggregate output for each output variable, obtained by combining the corresponding outputs from all the rules.</td>
<td>ao</td>
</tr>
</tbody>
</table>

For more information, see Fuzzy Logic Controller.

## Simulation Modes

The Fuzzy Logic Controller block has the following two simulation modes:

- **Interpreted execution** — Simulate fuzzy systems using precompiled MEX files. Using this option reduces the initial compilation time of the model.

- **Code generation** — Simulate fuzzy system without precompiled MEX files. Use this option when simulating fuzzy systems for code generation applications. Doing so simulates your system using the same code path used for generated code.

To select a simulation mode, set the **Simulate using** parameter of the block. By default, the block uses **Interpreted execution** mode for simulation.

## Map Command-Line Functionality to Fuzzy Logic Controller Block

The parameters and ports of the Fuzzy Logic Controller block map to the input and output arguments of `evalfis` or the properties of `evalfisOptions`. The following table shows the block parameters and ports that map to `evalfis` arguments.

<table>
<thead>
<tr>
<th>evalfis Argument</th>
<th>Description</th>
<th>Block Parameter or Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>fis</td>
<td>Fuzzy inference system</td>
<td>FIS name</td>
</tr>
<tr>
<td>evalfis Argument</td>
<td>Description</td>
<td>Block Parameter or Port</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>input, when a single row</td>
<td>Input variable values</td>
<td>in</td>
</tr>
<tr>
<td>output, when a single row</td>
<td>Output variable values</td>
<td>out</td>
</tr>
<tr>
<td>fuzzifiedIn</td>
<td>Fuzzified inputs</td>
<td>fi</td>
</tr>
<tr>
<td>ruleOut</td>
<td>Rule outputs</td>
<td>ro</td>
</tr>
<tr>
<td>aggregateOut</td>
<td>Aggregated outputs</td>
<td>ao</td>
</tr>
<tr>
<td>ruleFiring</td>
<td>Rule firing strengths</td>
<td>rfs</td>
</tr>
</tbody>
</table>

The following table shows the block parameters that map to evalfisOptions properties.

<table>
<thead>
<tr>
<th>evalfisOptions Property</th>
<th>Description</th>
<th>Block Parameter or Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumSamplePoints</td>
<td>Number of points in output fuzzy sets</td>
<td>Number of samples for output discretization</td>
</tr>
<tr>
<td>OutOfRangeInputValueMessage</td>
<td>Diagnostic message behavior when an input is out of range</td>
<td>Out of range input value</td>
</tr>
<tr>
<td>NoRuleFiredMessage</td>
<td>Diagnostic message behavior when no rules fire</td>
<td>No rule fired</td>
</tr>
<tr>
<td>EmptyOutputFuzzySetMessage</td>
<td>Diagnostic message behavior when an output fuzzy set is empty</td>
<td>Empty output fuzzy set</td>
</tr>
</tbody>
</table>

The remaining parameters of the Fuzzy Logic Controller block do not map to arguments of evalfis. Also, unlike the Fuzzy Logic Controller block, evalfis does not support fixed-point data for simulation or code generation.

**See Also**

**Blocks**
Fuzzy Logic Controller | Fuzzy Logic Controller with Ruleviewer

**More About**

- “Temperature Control in a Shower” on page 5-23
• “Water Level Control in a Tank” on page 5-15
Water Level Control in a Tank

This model shows how to implement a fuzzy inference system (FIS) in a Simulink® model.

Simulink Model

This model controls the level of water in a tank using a fuzzy inference system implemented using a Fuzzy Logic Controller block. Open the sltank model.

```
open_system('sltank')
```

For this system, you control the water that flows into the tank using a valve. The outflow rate depends on the diameter of the output pipe, which is constant, and the pressure in the tank, which varies with water level. Therefore, the system has nonlinear characteristics.
**Fuzzy Inference System**

The fuzzy system is defined in a FIS object, `tank`, in the MATLAB® workspace. For more information on how to specify a FIS in a Fuzzy Logic Controller block, see Fuzzy Logic Controller.

The two inputs to the fuzzy system are the water level error, `level`, and the rate of change of the water level, `rate`. Each input has three membership functions.

```matlab
g = plotmf(tank, 'input', 1)
g2 = plotmf(tank, 'input', 2)
```

![Membership functions for input 'level'](image)
The output of the fuzzy system is the rate at which the control valve is opening or closing, valve, which has five membership functions.

plotmf(tank,'output',1)
Due to the diameter of the outflow pipe, the water tank in this system empties more slowly than it fills up. To compensate for this imbalance, the close_slow and open_slow valve membership functions are not symmetrical. A PID controller does not support such asymmetry.

The fuzzy system has five rules. The first three rules adjust the valve based on only the water level error.

- If the water level is okay, then do not adjust the valve.
- If the water level is low, then open the valve quickly.
- If the water level is high, then close the valve quickly.
The other two rules adjust the valve based on the rate of change of the water level when the water level is near the setpoint.

- If the water level is okay and increasing, then close the valve slowly.
- If the water level is okay and decreasing, then open the valve slowly.

```
ans =

1x5 fisrule array with properties:

    Description
    Antecedent
    Consequent
    Weight
    Connection

Details:

    Description

1    "level==okay => valve=no_change (1)"
2    "level==low => valve=open_fast (1)"
3    "level==high => valve=close_fast (1)"
4    "level==okay & rate==positive => valve=close_slow (1)"
5    "level==okay & rate==negative => valve=open_slow (1)"
```

In this model, you can also control the water level using a PID controller. To switch to the PID controller, set the const block to a value greater than or equal to zero.

**Simulation**

The model simulates the controller with periodic changes in the setpoint of the water level. Run the simulation.

```
sim('sltank',100)
open_system('sltank/Comparison')
```
Water Level Control
The water level tracks the setpoint well. You can adjust the performance of the controller by modifying the rules of the tank FIS. For example, if you remove the last two rules, which are analogous to a derivative control action, the controller performs poorly, with large oscillations in the water level.

**See Also**

**Blocks**
Fuzzy Logic Controller | Fuzzy Logic Controller with Ruleviewer
More About

- “Simulate Fuzzy Inference Systems in Simulink” on page 5-2
- “Temperature Control in a Shower” on page 5-23
Temperature Control in a Shower

This model shows how to implement a fuzzy inference system (FIS) in a Simulink® model.

**Simulink Model**

The model controls the temperature of a shower using a fuzzy inference system implemented using a Fuzzy Logic Controller block. Open the `shower` model.

```matlab
open_system('shower')
```

For this system, you control the flow rate and temperature of a shower by adjusting hot and cold water valves.

Since there are two inputs for the fuzzy system, the model concatenates the input signals using a Mux block. The output of the Mux block is connected to the input of the Fuzzy Logic Controller block. Similarly, the two output signals are obtained using a Demux block connected to the controller.
**Fuzzy Inference System**

The fuzzy system is defined in a FIS object, `fisMatrix`, in the MATLAB® workspace. For more information on how to specify a FIS in a Fuzzy Logic Controller block, see Fuzzy Logic Controller.

The two inputs to the fuzzy system are the temperature error, `temp`, and the flow rate error, `flow`. Each input has three membership functions.

```matlab
figure
plotmf(fisMatrix, 'input', 1)
figure
plotmf(fisMatrix, 'input', 2)
```
The two outputs of the fuzzy system are the rate at which the cold and hot water valves are opening or closing, cold and hot respectively. Each output has five membership functions.

```matlab
figure
plotmf(fisMatrix,'output',1)
figure
plotmf(fisMatrix,'output',2)
```
The fuzzy system has nine rules for adjusting the hot and cold water valves based on the flow and temperature errors. The rules adjust the total flow rate based on the flow error, and adjust the relative hot and cold flow rates based on the temperature error.

fisMatrix.Rules

ans =

1x9 fisrule array with properties:

Description
Antecedent
Consequent
Weight Connection

Details:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;temp==cold &amp; flow==soft =&gt; cold=openSlow, hot=openFast (1)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;temp==cold &amp; flow==good =&gt; cold=closeSlow, hot=openSlow (1)&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;temp==cold &amp; flow==hard =&gt; cold=closeFast, hot=closeSlow (1)&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;temp==good &amp; flow==soft =&gt; cold=openSlow, hot=openSlow (1)&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;temp==good &amp; flow==good =&gt; cold=steady, hot=steady (1)&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;temp==good &amp; flow==hard =&gt; cold=closeSlow, hot=closeSlow (1)&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;temp==hot &amp; flow==soft =&gt; cold=openFast, hot=openSlow (1)&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;temp==hot &amp; flow==good =&gt; cold=openSlow, hot=closeSlow (1)&quot;</td>
</tr>
<tr>
<td>9</td>
<td>&quot;temp==hot &amp; flow==hard =&gt; cold=closeSlow, hot=closeFast (1)&quot;</td>
</tr>
</tbody>
</table>

Simulation

The model simulates the controller with periodic changes in the setpoints of the water temperature and flow rate.

```matlab
set_param('shower/flow scope','Open','on','Ymin','0','Ymax','1')
set_param('shower/temp scope','Open','on','Ymin','15','Ymax','30')
sim('shower',50)
```
The flow rate tracks the setpoint well. The temperature also tracks its setpoint, though there are temperature deviations when the controller adjusts to meet a new flow setpoint.

```matlab
bdclose('shower') % Closing model also clears its workspace variables.
```

**See Also**

**Blocks**
Fuzzy Logic Controller

**More About**
- “Simulate Fuzzy Inference Systems in Simulink” on page 5-2
- “Water Level Control in a Tank” on page 5-15
Implement Fuzzy PID Controller in Simulink Using Lookup Table

This example shows how to implement a fuzzy inference system for nonlinear PID control using a 2-D Lookup Table block.

Overview

A fuzzy inference system (FIS) maps given inputs to outputs using fuzzy logic. For example, a typical mapping of a two-input, one-output fuzzy controller can be depicted in a 3-D plot. The plot is often referred to as a control surface plot.
For control applications, typical FIS inputs are the error \((e(k))\) and change of error \((e(k) - e(k-1))\), \(E\) and \(CE\) respectively in the control surface plot. The FIS output is the control action inferred from the fuzzy rules, \(u\) in the surface plot. Fuzzy Logic Toolbox™ provides commands and apps for designing a FIS for a desired control surface. You can then simulate the designed FIS using the Fuzzy Logic Controller block in Simulink®.

You can often approximate nonlinear control surfaces using lookup tables to simplify the generated code and improve execution speed. For example, you can replace a Fuzzy Logic Controller block in Simulink with a set of Lookup Table blocks, one table for each output defined in the FIS. You can compute the data used in the lookup table using the \texttt{evalfis}\ command.

For this example, you design a nonlinear fuzzy PID controller for a plant in Simulink. The plant is a single-input, single-output system in discrete time. The design goal is to achieve good reference tracking performance.

\[ Ts = 0.1; \]
\[ Plant = \text{c2d}(\text{zp}([],[1 \ -3 \ -5],1),Ts); \]

You also implement the fuzzy inference system using a 2-D lookup table that approximates the control surface and achieves the same control performance.

**Fuzzy PID Controller Structure**

The fuzzy controller in this example is in the feedback loop and computes PID-like actions using fuzzy inference. Open the Simulink model.

\texttt{open_system('sllookuptable')}
The fuzzy PID controller uses a parallel structure as shown in the Fuzzy PID subsystem. For more information, see [1]. The controller is a combination of fuzzy PI control and fuzzy PD control.

`open_system('sllookuptable/Fuzzy PID')`
The fuzzy PID controller uses the change of the output \(-(y(k) - y(k-1))\), instead of change of error \(e(k) - e(k-1)\), as the second input signal to the FIS. Doing so prevents the step change in reference signal from directly triggering the derivative action. The two gain blocks, GCE and GCU, in the feed forward path from \(r\) to \(u\), ensure that the error signal \(e\) is used in proportional action when the fuzzy PID controller is linear.

**Design Conventional PID Controller**

The conventional PID controller in this example is a discrete-time PID controller with Backward Euler numerical integration in both the integral and derivative actions. The controller gains are \(K_p\), \(K_i\), and \(K_d\).

```
open_system('sllookuptable/Conventional PID')
```

Similar to the fuzzy PID controller, the input signal to the derivative action is \(-y(k)\), instead of \(e(k)\).

You can tune the PID controller gains manually or using tuning formulas. In this example, obtain the initial PID design using the `pidtune` command from Control System Toolbox™.

Define the PID structure, tune the controller, and extract the PID gains.
C0 = pid(1,1,1,'Ts',Ts,'IF','B','DF','B');  
C = pidtune(Plant,C0)  
[Kp,Ki,Kd] = piddata(C);  

C =  
\[
\frac{Ts*z}{z-1} + \frac{z-1}{Kp + Ki * \frac{z-1}{Ts*z} + Kd * \frac{Ts*z}{z-1}}
\]

with Kp = 30.6, Ki = 25.2, Kd = 9.02, Ts = 0.1  
Sample time: 0.1 seconds  
Discrete-time PID controller in parallel form.  

**Design Equivalent Linear Fuzzy PID Controller**  

By configuring the FIS and selecting the four scaling factors, you can obtain a linear fuzzy PID controller that reproduces the control performance of the conventional PID controller.  

First, configure the fuzzy inference system so that it produces a linear control surface from inputs E and CE to output u. The FIS settings are based on design choices described in [2]:  

- Use a Sugeno style fuzzy inference system with default inference methods.  
- Normalize the ranges of both inputs to [-10 10].  
- Use triangular input membership functions that overlap their neighbor functions at a membership value of 0.5.  
- Use an output range of [-20 20].  
- Use constant output membership functions.  

Construct the fuzzy inference system.  

FIS = sugfis;  
Define input variable E.  

FIS = addInput(FIS,[-10 10],'Name','E');  
FIS = addMF(FIS,'E','trimf',[-20 -10 0],'Name','Negative');
Define input \( CE \).

\[
\text{FIS} = \text{addInput}(\text{FIS}, [-10 10], 'Name', 'CE'); \\
\text{FIS} = \text{addMF}(\text{FIS}, 'CE', 'trimf', [-20 -10 0], 'Name', 'Negative'); \\
\text{FIS} = \text{addMF}(\text{FIS}, 'CE', 'trimf', [-10 0 10], 'Name', 'Zero'); \\
\text{FIS} = \text{addMF}(\text{FIS}, 'CE', 'trimf', [0 10 20], 'Name', 'Positive');
\]

Define output variable \( u \) with constant membership functions.

\[
\text{FIS} = \text{addOutput}(\text{FIS}, [-20 20], 'Name', 'u'); \\
\text{FIS} = \text{addMF}(\text{FIS}, 'u', 'constant', -20, 'Name', 'LargeNegative'); \\
\text{FIS} = \text{addMF}(\text{FIS}, 'u', 'constant', -10, 'Name', 'SmallNegative'); \\
\text{FIS} = \text{addMF}(\text{FIS}, 'u', 'constant', 0, 'Name', 'Zero'); \\
\text{FIS} = \text{addMF}(\text{FIS}, 'u', 'constant', 10, 'Name', 'SmallPositive'); \\
\text{FIS} = \text{addMF}(\text{FIS}, 'u', 'constant', 20, 'Name', 'LargePositive');
\]

Define the following fuzzy rules:

1. If \( E \) is negative and \( CE \) is negative, then \( u \) is -20.
2. If \( E \) is negative and \( CE \) is zero, then \( u \) is -10.
3. If \( E \) is negative and \( CE \) is positive then \( u \) is 0.
4. If \( E \) is zero and \( CE \) is negative, then \( u \) is -10.
5. If \( E \) is zero and \( CE \) is zero, then \( u \) is 0.
6. If \( E \) is zero and \( CE \) is positive, then \( u \) is 10.
7. If \( E \) is positive and \( CE \) is negative, then \( u \) is 0.
8. If \( E \) is positive and \( CE \) is zero, then \( u \) is 10.
9. If \( E \) is positive and \( CE \) is positive, then \( u \) is 20.

\[
\text{ruleList} = [1 1 1 1 1; \quad \% \text{Rule 1} \\
1 2 2 1 1; \quad \% \text{Rule 2} \\
1 3 3 1 1; \quad \% \text{Rule 3} \\
2 1 2 1 1; \quad \% \text{Rule 4} \\
2 2 3 1 1; \quad \% \text{Rule 5} \\
2 3 4 1 1; \quad \% \text{Rule 6} \\
3 1 3 1 1; \quad \% \text{Rule 7} \\
3 2 4 1 1; \quad \% \text{Rule 8} \\
3 3 5 1 1]; \quad \% \text{Rule 9}
\]

\[
\text{FIS} = \text{addRule}(\text{FIS}, \text{ruleList});
\]
While you implement your FIS from the command line in this example, you can alternatively build your FIS using the **Fuzzy Logic Designer** app.

Plot the linear control surface.

```matlab
gensurf(FIS)
```

Determine scaling factors $GE$, $GCE$, $GCU$, and $GU$ from the $Kp$, $Ki$, and $Kd$ gains of by the conventional PID controller. Comparing the expressions of the traditional PID and the linear fuzzy PID, the variables are related as follows:

- $Kp = GCU \times GCE + GU \times GE$
- $Ki = GCU \times GE$
\[ K_d = G_U * G_C E \]

Assume that the maximum reference step is 1, and thus the maximum error \( e \) is 1. Since the input range of \( E \) is \([-10, 10]\), set \( G_E \) to 10. You can then solve for \( G_C E \), \( G_C U \), and \( G_U \).

\[
G_E = 10; \\
G_C E = G_E * (K_p - \sqrt{K_p^2 - 4 * K_i * K_d}) / 2 / K_i; \\
G_C U = K_i / G_E; \\
G_U = K_d / G_C E;
\]

**Implement Fuzzy Inference System Using 2-D Lookup Table**

The fuzzy controller block has two inputs (\( E \) and \( C_E \)) and one output (\( u \)). Therefore, you can replace the fuzzy system using a 2-D lookup table.

To generate a 2-D lookup table from your FIS, loop through the input universe, and compute the corresponding output values using `evalfis`. Since the control surface is linear, you can use a few sample points for each input variable.

\[
\text{Step} = 10; \\
E = -10: \text{Step}:10; \\
C_E = -10: \text{Step}:10; \\
N = \text{length}(E); \\
\text{LookUpTableData} = \text{zeros}(N); \\
\text{for} \ i=1:N \\
\quad \text{for} \ j=1:N \\
\quad \quad \% \ Compute \ output \ u \ for \ each \ combination \ of \ sample \ points. \\
\quad \quad \text{LookUpTableData}(i,j) = \text{evalfis}(\text{FIS},[E(i) \ C_E(j)]); \\
\quad \text{end} \\
\text{end} \\
\]

View the fuzzy PID controller using 2-D lookup table.

```
open_system('sllookuptable/Fuzzy PID using Lookup Table')
```
The only difference compared to the Fuzzy PID controller is that the Fuzzy Logic Controller block is replaced with a 2-D Lookup Table block.

When the control surface is linear, a fuzzy PID controller using the 2-D lookup table produces the same result as one using the Fuzzy Logic Controller block.

**Simulate Closed-Loop Response in Simulink**

The Simulink model simulates three different controller subsystems, namely Conventional PID, Fuzzy PID, and Fuzzy PID using Lookup Table, to control the same plant.

Run the simulation. To compare the closed-loop responses to a step reference change, open the scope. As expected, all three controllers produce the same result.

```matlab
sim('sllookuptable')
open_system('sllookuptable/Scope')
```
Design Fuzzy PID Controller with Nonlinear Control Surface

Once you have a linear fuzzy PID controller, you can obtain a nonlinear control surface by adjusting your FIS settings, such as its style, membership functions, and rule base.

For this example, design a steep control surface using a Sugeno-type FIS. Each input set has two terms (Positive and Negative), and the number of rules is reduced to four.

Construct the FIS.

\[
FIS = 	ext{sugfis};
\]

Define input E.

\[
FIS = \text{addInput}(FIS,[-10 10],'\text{Name}','E');
FIS = \text{addMF}(FIS,'E',\text{gaussmf},[7 -10],\text{Name}','Negative');
FIS = \text{addMF}(FIS,'E',\text{gaussmf},[7 10],\text{Name}','Positive');
\]

Define input CE.
FIS = addInput(FIS,[-10 10],'Name','CE');
FIS = addMF(FIS,'CE','gaussmf',[7 -10],'Name','Negative');
FIS = addMF(FIS,'CE','gaussmf',[7 10],'Name','Positive');

Define output u.

FIS = addOutput(FIS,[-20 20],'Name','u');
FIS = addMF(FIS,'u','constant',-20,'Name','Min');
FIS = addMF(FIS,'u','constant',0,'Name','Zero');
FIS = addMF(FIS,'u','constant',20,'Name','Max');

Define the following rules:

1 If E is negative and CE is negative, then u is -20.
2 If E is negative and CE is positive, then u is 0.
3 If E is positive and CE is negative, then u is 0.
4 If E is positive and CE is positive, then u is 20.

ruleList = [1 1 1 1 1; % Rule 1
1 2 2 1 1; % Rule 2
2 1 2 1 1; % Rule 3
2 2 3 1 1]; % Rule 4
FIS = addRule(FIS,ruleList);

View the 3-D nonlinear control surface. This surface has a higher gain near the center of the E and CE plane than the linear surface has, which helps reduce the error more quickly when the error is small. When the error is large, the controller becomes less aggressive to avoid possible saturation.

gensurf(FIS)
Before starting the simulation, update the lookup table with the new control surface data. Since the surface is nonlinear, to obtain a sufficient approximation, add more sample points.

Step = 1;
E = -10:Step:10;
CE = -10:Step:10;
N = length(E);
LookUpTableData = zeros(N);
for i=1:N
    for j=1:N
        % Compute output u for each combination of sample points.
        LookUpTableData(i,j) = evalfis(FIS,[E(i) CE(j)]);
Compared with the traditional linear PID controller (the response curve with large overshoot), the nonlinear fuzzy PID controller reduces the overshoot by 50%. The two response curves from the nonlinear fuzzy controllers almost overlap, which indicates that the 2-D lookup table approximates the fuzzy system well.

\[
\text{bdclose('sllookuptable')} \quad \text{% Closing model also clears its workspace variables.}
\]

**Conclusion**

You can approximate a nonlinear fuzzy PID controller using a lookup table. By replacing a Fuzzy Logic Controller block with Lookup Table blocks in Simulink, you can deploy a fuzzy controller with simplified generated code and improved execution speed.
References


See Also

Blocks
2-D Lookup Table | Fuzzy Logic Controller

More About

•  “Simulate Fuzzy Inference Systems in Simulink” on page 5-2
Deployment

- “Deploy Fuzzy Inference Systems” on page 6-2
- “Generate Code for Fuzzy System Using Simulink Coder” on page 6-4
- “Generate Structured Text for Fuzzy System Using Simulink PLC Coder” on page 6-9
- “Generate Code for Fuzzy System Using MATLAB Coder” on page 6-11
Deploy Fuzzy Inference Systems

You can deploy a fuzzy inference system (FIS) by generating code in either Simulink or MATLAB. You can generate code for both type-1 (mamfis, sugfis) and type-2 fuzzy (mamfistype2, sugfistype2) inference systems. All fuzzy inference system options, including custom inference functions, support code generation.

Generate Code in Simulink

You can generate code for evaluating fuzzy inference systems in Simulink using the Fuzzy Logic Controller block. You can generate code for double-precision, single-precision, or fixed-point data using Simulink Coder™ or Simulink PLC Coder™.

For more information, see “Generate Code for Fuzzy System Using Simulink Coder” on page 6-4 and “Generate Structured Text for Fuzzy System Using Simulink PLC Coder” on page 6-9.

Generate Code in MATLAB

You can generate code for evaluating fuzzy inference systems in MATLAB. You can generate code for double-precision or single-precision data using MATLAB Coder.

Code generation in MATLAB does not support fuzzy inference system objects. Instead, convert your fuzzy system into a homogeneous structure using getFISCodeGenerationData, and pass the resulting structure to evalfis.

For more information, see “Generate Code for Fuzzy System Using MATLAB Coder” on page 6-11.

**Note** Code generation does not support the construction of fuzzy systems at the command line.

See Also

Functions

evalfis | mamfis | sugfis
Blocks
Fuzzy Logic Controller

More About

- “Build Fuzzy Systems at the Command Line” on page 2-41
- “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17
Generate Code for Fuzzy System Using Simulink Coder

You can generate code for a Fuzzy Logic Controller block using Simulink® Coder™. For more information on generating code, see “Generate Code Using Simulink® Coder™” (Simulink Coder).

While this example generates code for a type-1 Sugeno fuzzy inference system, the workflow also applies to Mamdani and type-2 fuzzy systems.

Generate Code for Fuzzy Inference System

By default, the Fuzzy Logic Controller block uses double-precision data for simulation and code generation. The fuzzyPID model is configured to use double-precision data. For more information on configuring your fuzzy inference system for code generation, see Fuzzy Logic Controller.

```matlab
mdl = 'fuzzyPID';
open_system(mdl)
```

It is good practice to validate the performance of the system in Simulink. Run the simulation. The model saves the output response, u, to the MATLAB® workspace.

```matlab
sim(mdl)
```

To generate code for the model, use the `rtwbuild` function. For this example, suppress the Command Window output for the build process.

```matlab
set_param(mdl,'RTWVerbose','off')
rtwbuild(mdl)
```
### Starting build procedure for model: fuzzyPID
### Successful completion of build procedure for model: fuzzyPID

By default, Simulink Coder generates C code for a generic real-time target. To select a different target file and language, in the Configuration Parameters dialog box, modify the **System target file** and **Language** parameters, respectively.

The generated code is stored in a new `fuzzyPID_grt_rtw` folder in your current working folder. The name of this folder depends on the selected target file.

On a Windows® system, by default, an executable file named `fuzzyPID.exe` is also added to the current working folder. To generate code without compilation, in the Configuration parameters dialog box, select the **Generate code only** parameter before generating code.

Run the executable.

```matlab
if ispc
    status = system(mdl);
else
    disp('The example only runs the executable on Windows system.');</end
```

** starting the model **
** created fuzzyPID.mat **

After the executable completes successfully (`status = 0`), the software creates a `fuzzyPID.mat` data file that contains the simulation results.

You can compare the output response from the generated code, `rtw_y`, with the output from the Simulink simulation, `y`, using the following code:

```matlab
load fuzzyPID.mat
plot(tout,y,'b-',rt_tout,rt_y,'ro')
legend('Simulink','Executable','Location','Southeast')
```
The result from the generated code matches the Simulink simulation.

You can also generate code for just the controller subsystem in this model. To do so, specify the subsystem when calling the `rtwbuild` function.

```
rtwbuild([mdl '/Fuzzy PID'])
```

### Starting build procedure for model: Fuzzy0
### Successful completion of build procedure for model: Fuzzy0

You can deploy generated code according to your application needs. For example, you can configure the properties of executable files and create static or dynamic libraries. For more information, see “Build Process Workflow for Real-Time Systems” (Simulink Coder).
Generate Code for Other Data Types

The Fuzzy Logic Controller block also supports single-precision and fixed-point data for simulation and code generation. In both cases, your resulting fuzzy system has decreased accuracy compared to an equivalent double-precision fuzzy system. Use:

- Single-precision data to reduce the memory footprint of your system.
- Fixed-point data if your target platform only supports fixed-point arithmetic.

To use one of these data types, set the Data type property of the block, and configure the other components in the model to use the same data type.

The fuzzyPID_single model is configured for single-precision data. Open the model.

```matlab
mdl2 = 'fuzzyPID_single';
open_system(mdl2)
```

In this model, the Data type parameter of the Fuzzy Logic Controller block is set to single. The Fuzzy Logic Controller block automatically converts input signals to the specified data type. Also, the Simulate using parameter is set to Code Generation. The Simulate using option does not affect the code generation process. Instead, setting this option simulates your fuzzy system using the same code path used by generated code.

Generate code for this model.

```matlab
set_param(mdl2,'RTWVerbose','off')
rtwbuild(mdl2)
```
Setting the **Data type** parameter of a Fuzzy Logic Controller block ensures that all the inference steps use the specified data type. However, depending on the configuration of other blocks in the model, some of the generated code can still use double-precision data.

**See Also**
Fuzzy Logic Controller

**More About**
- “Deploy Fuzzy Inference Systems” on page 6-2
- “Generate Structured Text for Fuzzy System Using Simulink PLC Coder” on page 6-9
- “Generate Code for Fuzzy System Using MATLAB Coder” on page 6-11
Generate Structured Text for Fuzzy System Using Simulink PLC Coder

You can generate Structured Text for a Fuzzy Logic Controller block using Simulink® PLC Coder™. For more information on generating Structured Text, see “Code Generation” (Simulink PLC Coder).

While this example generates Structured Text for a type-1 Sugeno fuzzy inference system, the workflow also applies to Mamdani and type-2 fuzzy systems.

By default, the Fuzzy Logic Controller block uses double-precision data for simulation and code generation. The fuzzyPID model is configured to use double-precision data. You can also use either single-precision or fixed-point data. For more information on configuring your fuzzy inference system for code generation, see Fuzzy Logic Controller.

mdl = 'fuzzyPID';
open_system(mdl)

It is good practice to validate the performance of the system in Simulink before generating code. Run the simulation.

sim(mdl)
open_system([mdl '/Output'])

To generate Structured Text for the model, use the plcgeneratecode function, which generates code for an atomic subsystem in a model. To generate code for the Fuzzy PID controller, configure the subsystem as an atomic subsystem by selecting the Treat as atomic unit parameter for the subsystem.

subsys = [mdl '/Fuzzy_PID'];
set_param(subsys,'TreatAsAtomicUnit','on')

When generating code for just a Fuzzy Logic Controller block, place the block inside a subsystem, and set the Treat as atomic unit parameter of that subsystem.

Generate Structured Text for the Fuzzy PID subsystem.

plcgeneratecode(subsys);

By default, the software saves the generated code in the following location:
plcsrc/fuzzy_PID.exp

See Also
Fuzzy Logic Controller

More About
- “Deploy Fuzzy Inference Systems” on page 6-2
- “Generate Code for Fuzzy System Using Simulink Coder” on page 6-4
Generate Code for Fuzzy System Using MATLAB Coder

You can generate code for evaluating a fuzzy inference system using MATLAB® Coder™. For more information on generating code, see “Code Generation” (MATLAB Coder).

To generate code for evaluating fuzzy systems, you must first create a fuzzy inference system (FIS). For more information, see “Build Fuzzy Systems at the Command Line” on page 2-41 and “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17.

While this example generates code for a type-1 Mamdani fuzzy inference system, the workflow also applies to Sugeno and type-2 fuzzy systems.

Generating code using MATLAB Coder does not support fuzzy FIS objects (mamfis, sugfis, mamfistype2, sugfistype2). To generate code for evaluating fuzzy systems, you must convert your fuzzy inference system objects into homogeneous structures using the getFISCodeGenerationData function.

**Embed FIS Data in Generated Code**

You can embed the data for your fuzzy inference system within the generated code. Use this option if you do not want to change the FIS data after compilation.

First, create a fuzzy system, or load a fuzzy system from a .fis file. For this example, load the fuzzy system from tipper.fis.

```matlab
fisObject = readfis("tipper.fis");
```

To use this FIS for code generation, convert it to a homogeneous structure.

```matlab
fis = getFISCodeGenerationData(fisObject);
```

By default, getFISCodeGenerationData assumes that the FIS object is a type-1 system. To generate code for a type-2 system, you must indicate the system type using getFISCodeGenerationData(fisObject,"type2").

Create a function for evaluating the fuzzy system fis for a given input vector x. Within this function, you can specify options for the evalfis function using evalfisOptions.

```matlab
function y = evaluatefis1(fis,x)
    %#codegen
    opt = evalfisOptions('NumSamplePoints',51);
    y = evalfis(fis,x,opt);
end
```
Generate code for `evaluatefis1`, specifying that the `fis` input argument is constant. You can specify different targets for your build, such as a static library, an executable, or a MEX file. For this example, generate a MEX file.

```matlab
codegen('evaluatefis1','-args',{coder.Constant(fis),[0 0]},'-config:mex')
```

To verify the execution of the MEX file:

1. Evaluate the MEX file for one or more input values. When you call the MEX file, specify the same FIS structure that you used at compile time.
2. Evaluate the original FIS for the same input values using `evalfis`. When evaluating using `evalfis`, use the same homogeneous FIS structure.
3. Compare the evaluation results.

```matlab
mexOutput1 = evaluatefis1_mex(fis,[7 9])
mexOutput1 = 21.0327
opt = evalfisOptions('NumSamplePoints',51);
evalfisOutput = evalfis(fis,[7 9],opt)
evalfisOutput = 21.0327
```

The MEX file output matches the `evalfis` output.

Alternatively, you can embed the FIS data in the generated code by reading the FIS data from a file at code generation time. Specify a function for evaluating a fuzzy system for given input vector `x`. Within this function, read the FIS data from the file `tipper.fis`.

```matlab
function y = evaluatefis2(x)
    %#codegen
    fis = getFISCodeGenerationData('tipper.fis');
    opt = evalfisOptions('NumSamplePoints',51);
    y = evalfis(fis,x,opt);
end
```

Generate code for `evaluatefis2`.

```matlab
codegen('evaluatefis2','-args',{[0 0]},'-config:mex')
```

Verify the execution of the MEX file using the same input values for `x`. In this case, you do not have to specify the original FIS structure used at compile time.

```matlab
mexOutput2 = evaluatefis2_mex([7 9])
```
Generate Code for Loading FIS Data at Run Time

You can generate code for evaluating a FIS that is read from a .fis file specified at run time. In this case, the FIS data is not embedded in the generated code. Specify a function for evaluating the fuzzy system defined in the specified file fileName for a given input vector x.

```matlab
function y = evaluatefis3(fileName,x)
    %#codegen
    fis = getFISCodeGenerationData(fileName);
    opt = evalfisOptions('NumSamplePoints',51);
    y = evalfis(fis,x,opt);
end
```

Define input data types for this function.

```matlab
fileName = coder.newtype('char',[1 Inf],[false true]);
x = coder.newtype('double', [1 Inf],[false true]);
```

Generate code for evaluatefis3.

```matlab
codegen('evaluatefis3','-args',{fileName,x},'config:mex')
```

Verify the execution of the MEX file using the same input values for x. In this case, you specify the name of the .fis file.

```matlab
mexOutput3 = evaluatefis3_mex('tipper.fis',[7 9])
mexOutput3 = 21.0327
```

evalfisOutput

evalfisOutput = 21.0327

Each time you run evaluatefis3, it reloads the fuzzy system from the file. For computational efficiency, you can create a function that only loads the FIS when a new file name is specified.

```matlab
function y = evaluatefis4(fileName,x)
    %#codegen
```
persistent fisName fis
if isempty(fisName)
    [fisName,fis] = loadFIS(fileName);
elseif ~strcmp(fisName,fileName)
    [fisName,fis] = loadFIS(fileName);
end

opt = evalfisOptions('NumSamplePoints',51);
y = evalfis(fis,x,opt);
end

function [fisName,fis] = loadFIS(fileName)
    fisName = fileName;
    fis = getFISCodeGenerationData(fisName);
end

Generate code `evaluatefis4`. The input data types for this function are the same as for `evaluatefis3`.

codegen('evaluatefis4','-args',{fileName,x},'--config:mex')

Verify the execution of the MEX file using the same input values file name.

mexOutput4 = evaluatefis4_mex('tipper.fis',[7 9])
mexOutput4 = 21.0327

evalfisOutput
evalfisOutput = 21.0327

**Generate Code for Single-Precision Data**

The preceding examples generated code for double-precision data. To generate code for single-precision data, specify the data type of the input values as `single`. For example, generate code for `evaluatefis2` using single-precision data.

codegen('evaluatefis2','-args',{single([0 0])},'--config:mex')

Verify the MEX file execution, passing in single-precision input values.

mexOutputSingle = evaluatefis2_mex(single([7 9]))
mexOutputSingle = single
   21.0327

evalfisOutput

evalfisOutput = 21.0327

See Also
evalfis | getFISCodeGenerationData

More About
• "Deploy Fuzzy Inference Systems" on page 6-2
• "Generate Code for Fuzzy System Using Simulink Coder" on page 6-4
Apps — Alphabetical List
Fuzzy Logic Designer

Design and test fuzzy inference systems

Description
The Fuzzy Logic Designer app lets you design and test fuzzy inference systems for modeling complex system behaviors.

Using this app, you can:

- Design Mamdani and Sugeno fuzzy inference systems.
- Add or remove input and output variables.
- Specify input and output membership functions.
- Define fuzzy if-then rules.
- Select fuzzy inference functions for:
  - And operations
  - Or operations
  - Implication
  - Aggregation
  - Defuzzification
- Adjust input values and view associated fuzzy inference diagrams.
- View output surface maps for fuzzy inference systems.
- Export fuzzy inference systems to the MATLAB workspace.

Limitations
The Fuzzy Logic Designer app does not support type-2 fuzzy systems.

Open the Fuzzy Logic Designer App

- MATLAB Toolstrip: On the Apps tab, under Control System Design and Analysis, click the app icon.
• MATLAB command prompt: Enter `fuzzyLogicDesigner`.

Examples
• “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17

Programmatic Use

`fuzzyLogicDesigner` opens the **Fuzzy Logic Designer** app.

`fuzzyLogicDesigner(fis)` opens the app and loads the fuzzy inference system `fis`. `fis` can be any `mamfis` or `sugfis` object in the MATLAB workspace.

`fuzzyLogicDesigner(fileName)` opens the app and loads a fuzzy inference system from a file. `fileName` is the name of a `.fis` file on the MATLAB path.

To save a fuzzy inference system to a `.fis` file:

• In **Fuzzy Logic Designer**, select **File > Export > To File**.
• At the command line, use `writeFIS`.

Compatibility Considerations

**Support for representing fuzzy inference systems as structures will be removed**

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:

• Object property names that differ from the corresponding structure fields.
• Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.
To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

**See Also**

**Apps**

Neuro-Fuzzy Designer

**Functions**

`evalfis` | `mfedit` | `newfis` | `plotfis` | `ruleedit` | `ruleview` | `surfview`

**Topics**

“Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17
“What Is Fuzzy Logic?” on page 1-3
“Foundations of Fuzzy Logic” on page 1-10
“Fuzzy Inference Process” on page 1-28

**Introduced in R2014b**
Neuro-Fuzzy Designer

Design, train, and test Sugeno-type fuzzy inference systems

Description

The Neuro-Fuzzy Designer app lets you design, train, and test adaptive neuro-fuzzy inference systems (ANFIS) using input/output training data.

Using this app, you can:

- Tune membership function parameters of Sugeno-type fuzzy inference systems.
- Automatically generate an initial inference system structure based on your training data.
- Modify the inference system structure before tuning.
- Prevent overfitting to the training data using additional checking data.
- Test the generalization ability of your tuned system using testing data.
- Export your tuned fuzzy inference system to the MATLAB workspace.

You can use the Neuro-Fuzzy Designer to train a type-1 Sugeno-type fuzzy inference system that:

- Has a single output.
- Uses weighted average defuzzification.
- Has output membership functions all of the same type, for example linear or constant.
- Has complete rule coverage with no rule sharing; that is, the number of rules must match the number of output membership functions, and every rule must have a different consequent.
- Has unity weight for each rule.
- Does not use custom membership functions.
Open the Neuro-Fuzzy Designer App

- MATLAB Toolstrip: On the Apps tab, under Control System Design and Analysis, click the app icon.
- MATLAB command prompt: Enter neuroFuzzyDesigner.

Examples

- “Train Adaptive Neuro-Fuzzy Inference Systems” on page 3-100

Programmatic Use

neuroFuzzyDesigner opens the Neuro-Fuzzy Designer app.

neuroFuzzyDesigner(fis) opens the app and loads the fuzzy inference system fis. fis can be any valid sugfis object in the MATLAB workspace.

You can create an initial Sugeno-type fuzzy inference system from training data using the genfis command.

neuroFuzzyDesigner(fileName) opens the app and loads a fuzzy inference system. fileName is the name of a .fis file on the MATLAB path.

To save a fuzzy inference system to a .fis file:

- In the Fuzzy Logic Designer, select File > Export > To File
- At the command line, use writeFIS.

Compatibility Considerations

Support for representing fuzzy inference systems as structures will be removed

Warns starting in R2019b
Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

**See Also**

**Apps**

Fuzzy Logic Designer

**Functions**

`anfis` | `genfis`

**Topics**

“Train Adaptive Neuro-Fuzzy Inference Systems” on page 3-100
“Neuro-Adaptive Learning and ANFIS” on page 3-92

**Introduced in R2014b**
Functions — Alphabetical List
addInput

Add input variable to fuzzy inference system

Syntax

fisOut = addInput(fisIn)
fisOut = addInput(fisIn,range)
fisOut = addInput(____,Name,Value)

Description

fisOut = addInput(fisIn) adds a default input variable to fisIn and returns the resulting fuzzy system in fisOut. This input variable has a default name, default range, and no membership functions.

fisOut = addInput(fisIn,range) adds an input variable with the specified range.

fisOut = addInput(____,Name,Value) configures the input variable using one or more name-value pair arguments.

Examples

Add Input Variable to Fuzzy Inference System

Create a Sugeno fuzzy inference system.

fis = sugfis('Name','tipper');

Add an input variable with default specifications.

fis = addInput(fis);

You can configure the input variable properties using dot notation. For example, specify the name and range for the variable.
fis.Inputs(1).Name = "service";
fis.Inputs(1).Range = [0 10];

View the input variable.

fis.Inputs(1)

ans =
    fisvar with properties:
        Name: "service"
        Range: [0 10]
    MembershipFunctions: [0x0 fismf]

You can also specify a variable name and range when you add it to the fuzzy system.

fis2 = sugfis('Name','tipper');
fis2 = addInput(fis2,[0 10],'Name','service');

Add Input Variable with Membership Functions

Create a fuzzy inference system.

fis = mamfis('Name','tipper');

Add an input variable with three Gaussian membership functions distributed over the input range.

fis = addInput(fis,'NumMFs',3,'MFType','gaussmf');

View the membership functions.

plotmf(fis,'input',1)
**Input Arguments**

**fisIn — Fuzzy inference system**

mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
• sugfistype2 object — Type-2 Sugeno fuzzy inference system

**range — Variable range**

[0 1] (default) | two-element vector

Variable range, specified as a two-element element vector where the first element is less than the second element. The first element specifies the lower bound of the range, and the second element specifies the upper bound of the range.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1,...,NameN, ValueN.

Example: 'NumMFs', 3 configures the variable to use three membership functions.

**Name — Variable name**

string | character vector

Variable name, specified as the comma-separated pair consisting of 'Name' and a string or character vector. The default variable name is "input<uniqueIndex>“, where uniqueIndex is automatically generated based on the current number of inputs in fisIn.

**NumMFs — Number of membership functions**

0 (default) | nonnegative integer

Number of membership functions, specified as the comma-separated pair consisting of 'NumMFs' and a nonnegative integer.

**MFType — Membership function type**

"trimf" (default) | "gaussmf"

Membership function type, specified as the comma-separated pair consisting of 'MFType' and one of the following:

• "trimf" — Triangular membership functions
• "gaussmf" — Gaussian membership functions
The membership functions are uniformly distributed over the input variable range with approximately 80% overlap in the membership function supports.

**Output Arguments**

`fisOut — Fuzzy inference system`  
`mamfis object | sugfis object | mamfistype2 object | sugfistype2 object`

Fuzzy inference system, specified as one of the following:

- `mamfis object` — Mamdani fuzzy inference system
- `sugfis object` — Sugeno fuzzy inference system
- `mamfistype2 object` — Type-2 Mamdani fuzzy inference system
- `sugfistype2 object` — Type-2 Sugeno fuzzy inference system

`fisOut` contains the added input variable, with all other properties matching the properties of `fisIn`.

**See Also**

`addOutput | fisvar | removeInput`

**Topics**

“Build Fuzzy Systems at the Command Line” on page 2-41

**Introduced in R2018b**
addMF

Add membership function to fuzzy variable

**Syntax**

```matlab
fisOut = addMF(fisIn,varName)
fisOut = addMF(fisIn,varName,type,parameters)
fisOut = addMF(___,Name,Value)
```

```matlab
varOut = addMF(varIn)
varOut = addMF(varIn,type,parameters)
varOut = addMF(___,Name,Value)
```

**Description**

`fisOut = addMF(fisIn,varName)` adds a default membership function to the input or output variable `varName` in the fuzzy inference system `fisIn` and returns the resulting fuzzy system in `fisOut`.

`fisOut = addMF(fisIn,varName,type,parameters)` adds a membership function with the specified type and parameters.

`fisOut = addMF(___,Name,Value)` configures the membership function using one or more name-value pair arguments.

`varOut = addMF(varIn)` adds a default membership function to fuzzy variable `varIn` and returns the resulting fuzzy variable in `varOut`.

If `varIn` does not contain any membership functions, this syntax adds a default type-1 membership function. Otherwise, the type of the added membership function matches the type of the existing membership functions in `varIn`.

`varOut = addMF(varIn,type,parameters)` adds a membership function with the specified type and parameters.

`varOut = addMF(___,Name,Value)` specifies the name of the membership function using the `Name` name-value pair argument.
To add a type-2 membership function to a fuzzy variable with no existing membership functions, you must specify either the `LowerLag` or `LowerScale` name-value pair argument.

### Examples

**Add Membership Function to Fuzzy Inference System**

Create a Mamdani fuzzy system, and add three input variables and one output variable. For this example, give the second input variable and the output variable the same name.

```matlab
fis = mamfis;
fis = addInput(fis,[0 80],"Name","speed");
fis = addInput(fis,[0 100],"Name","throttle");
fis = addInput(fis,[0 10],"Name","distance");
fis = addOutput(fis,[0 100],"Name","throttle");
```

Add a membership function to the first input variable, specifying a trapezoidal membership function, and set the membership function parameters.

```matlab
fis = addMF(fis,"speed","trapmf",[-5 0 10 30]);
```

You can also specify the name of your membership when you add it to a fuzzy system. Add a membership function called "high" to the first input variable.

```matlab
fis = addMF(fis,"speed","trapmf",[50 70 80 85],"Name","high");
```

View the membership functions for the first input variable.

```matlab
plotmf(fis,"input",1)
```
If your system has an input variable with the same name as an output variable, you must specify the variable type when adding a membership function. For example, add a membership function to the output variable.

```matlab
fis = addMF(fis,"throttle","trimf",[0 20 40],"VariableType","output");
plotmf(fis,"output",1)
```
Alternatively, you can add a default membership function to a fuzzy system and set its parameters using dot notation. For example, add and configure a membership function for the third input variable.

```matlab
fis = addMF(fis,"distance");
fis.Inputs(3).MembershipFunctions(1).Type = "trapmf";
fis.Inputs(3).MembershipFunctions(1).Parameters = [-1 0 2 4];
plotmf(fis,"input",3)
```
Create a type-2 Sugeno fuzzy system, and add two input variables and one output variable.

```matlab
fis = sugfistype2;
fis = addInput(fis,[0 80],"Name","speed");
fis = addInput(fis,[0 10],"Name","distance");
fis = addOutput(fis,[0 100],"Name","braking");
```
Add a membership function to the first input variable, specifying a trapezoidal membership function, and set the membership function parameters. This type-2 membership function uses default lower membership function lag and scale parameters.

```matlab
fis = addMF(fis, "speed", "trapmf", [-5 0 10 30]);
```

You can also specify the configuration of the lower MF when adding a type-2 membership function.

```matlab
fis = addMF(fis, "speed", "trapmf", [10 30 50 70], 'LowerScale', 0.8, 'LowerLag', 0.1);
```

You can also specify the name of your membership function when you add it to a fuzzy system. Add a membership function called "high" to the first input variable.

```matlab
fis = addMF(fis, "speed", "trapmf", [50 70 80 85], 'Name', "high");
```

View the membership functions for the first input variable.

```matlab
plotmf(fis, "input", 1)
```
Add Membership Function to Fuzzy Variable

Create a fuzzy variable with a specified range.

\[ \text{var} = \text{fisvar}([0 \ 1]); \]

Add a membership function to the variable, specifying a trapezoidal membership function, and set the membership function parameters.

\[ \text{var} = \text{addMF}(\text{var}, \text{"trapmf"}, [-0.5 \ 0 \ 0.2 \ 0.4]); \]
You can also specify the name of your membership when you add it to a fuzzy variable. For example, add a membership function called "large".

```matlab
var = addMF(var,"trapmf",[0.6 0.8 1 1.5],'Name','large');
```

View the membership functions.

```matlab
var.MembershipFunctions
```

```
ans =
1x2 fismf array with properties:

Type
Parameters
Name

Details:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;mf1&quot;</td>
<td>&quot;trapmf&quot;</td>
<td>-0.5 0 0.2 0.4</td>
</tr>
<tr>
<td>&quot;large&quot;</td>
<td>&quot;trapmf&quot;</td>
<td>0.6 0.8 1 1.5</td>
</tr>
</tbody>
</table>
```

Alternatively, you can add a default membership function to a fuzzy variable and set its parameters using dot notation.

```matlab
var = fisvar([0 1]);
var = addMF(var);
var.MembershipFunctions(1).Type = "trapmf";
var.MembershipFunctions(1).Parameters = [-0.5 0 0.2 0.4];
```

### Add Type-2 Membership Function to Fuzzy Variable

Create a fuzzy variable with a specified range. By default, this variable has no membership functions.

```matlab
var = fisvar([0 9]);
```

To add a type-2 membership function to a variable with no existing membership functions, specify either a LowerLag or LowerScale value for the membership function. For example specify a lower scale value.
var = addMF(var,"trimf",[0 3 6], 'LowerScale',1);

Once a variable contains a type-2 membership function, you can add additional type-2 membership functions without specifying one of these parameters.

var = addMF(var,"trimf",[3 6 9]);

View the membership functions.

var.MembershipFunctions

ans =
    1x2 fismftype2 array with properties:
        Type    UpperParameters    LowerScale    LowerLag
        Name

    Details:
    Name Type UpperParameters Lower Scale Lower Lag
    --- ---- ---------- ---------- ---------- ----------
    1    "mf1" "trimf"      0    3    6 1       0.2    0.2
    2    "mf2" "trimf"      3    6    9 1       0.2    0.2

**Input Arguments**

*fisIn — Fuzzy inference system*

mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- *mamfis object* — Mamdani fuzzy inference system
- *sugfis object* — Sugeno fuzzy inference system
- *mamfistype2 object* — Type-2 Mamdani fuzzy inference system
- *sugfistype2 object* — Type-2 Sugeno fuzzy inference system
**varName — Variable name**

*string | character vector*

Variable name, specified as a string or character vector. You can specify the name of either an input or output variable in your FIS. If your system has an input variable with the same name as an output variable, specify the type of the variable you want to add a membership function to using the `VariableType` name-value pair.

**type — Membership function type**

"trimf" (default) | string | character vector | function handle

Membership function type, specified as a string or character vector that contains the name of a function in the current working folder or on the MATLAB path. You can also specify a handle to such a function. When you specify `type`, you must also specify parameters.

This table describes the values that you can specify for `type`.

<table>
<thead>
<tr>
<th>Membership Function Type</th>
<th>Description</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;gbellmf&quot;</td>
<td>Generalized bell-shaped membership function</td>
<td>gbellmf</td>
</tr>
<tr>
<td>&quot;gaussmf&quot;</td>
<td>Gaussian membership function</td>
<td>gaussmf</td>
</tr>
<tr>
<td>&quot;gauss2mf&quot;</td>
<td>Gaussian combination membership function</td>
<td>gauss2mf</td>
</tr>
<tr>
<td>&quot;trimf&quot;</td>
<td>Triangular membership function</td>
<td>trimf</td>
</tr>
<tr>
<td>&quot;trapmf&quot;</td>
<td>Trapezoidal membership function</td>
<td>trapmf</td>
</tr>
<tr>
<td>&quot;sigmf&quot;</td>
<td>Sigmoidal membership function</td>
<td>sigmf</td>
</tr>
<tr>
<td>&quot;dsigmf&quot;</td>
<td>Difference between two sigmoidal membership functions</td>
<td>dsigmf</td>
</tr>
<tr>
<td>&quot;psigmf&quot;</td>
<td>Product of two sigmoidal membership functions</td>
<td>psigmf</td>
</tr>
<tr>
<td>&quot;zmf&quot;</td>
<td>Z-shaped membership function</td>
<td>zmf</td>
</tr>
<tr>
<td>&quot;pimf&quot;</td>
<td>Pi-shaped membership function</td>
<td>pimf</td>
</tr>
<tr>
<td>&quot;smf&quot;</td>
<td>S-shaped membership function</td>
<td>smf</td>
</tr>
<tr>
<td>Membership Function Type</td>
<td>Description</td>
<td>For More Information</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>&quot;constant&quot;</td>
<td>Constant membership function (not supported for output variables of Mamdani systems or for any input variables)</td>
<td>“Sugeno Fuzzy Inference Systems” on page 2-3</td>
</tr>
<tr>
<td>&quot;linear&quot;</td>
<td>Linear membership function (not supported for output variables of Mamdani systems or for any input variables)</td>
<td></td>
</tr>
<tr>
<td>String or character vector</td>
<td>Name of a custom membership function in the current working folder or on the MATLAB path. Custom functions are not supported for output variables of Sugeno systems.</td>
<td>“Build Fuzzy Systems Using Custom Functions” on page 2-52</td>
</tr>
<tr>
<td>Function handle</td>
<td>Handle to a custom membership function in the current working folder or on the MATLAB path. Custom functions are not supported for output variables of Sugeno systems.</td>
<td></td>
</tr>
</tbody>
</table>

**parameters — Membership function parameters**

[0 0.5 1] (default) | vector

Membership function parameters, specified as a vector. The length of the parameter vector depends on the membership function type. When you specify parameters, you must also specify type.

When fisIn is a type-1 FIS or varIn contains type-1 membership functions, parameters sets the Parameters property of the added membership function.

When fisIn is a type-2 FIS or varIn contains type-2 membership functions, parameters sets the UpperParameters property of the added membership function.

**varIn — Fuzzy variable**

fisvar object

Fuzzy variable, specified as a fisvar object.
Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'Name','large' specifies the membership function name "large"

Name — Membership function name
string | character vector

Membership function name, specified as the comma-separated pair consisting of 'Name' and a string or character vector. The default membership function name is "mf<uniqueIndex>", where uniqueIndex is automatically generated based on the current number of membership functions in the associated variable.

VariableType — Variable type
"input" | "output"

Variable type, specified as the comma-separated pair 'VariableType' and one of the following:

- "input" — Input variable
- "output" — Output variable

If your system has an input variable with the same name as an output variable, specify which variable to add the membership function to VariableType.

This name-value pair does not apply when adding when adding a membership function to a fisvar object.

LowerScale — Lower membership function scaling factor
1 (default) | positive scalar less than or equal to 1

Lower membership function scaling factor for type-2 membership functions, specified as a positive scalar less than or equal to 1. Use LowerScale to define the maximum value of the lower membership function.

Depending on the value of LowerLag, the actual maximum lower membership function value can be less than LowerScale.

This name-value pair applies only when adding type-2 membership functions.
LowerLag — Lower membership function delay factor
[0.2 0.2] (default) | scalar value between 0 and 1 | vector of length 2

Lower membership function delay factor for type-2 membership functions, specified as a scalar value or a vector of length two. You can specify lag values between 0 and 1, inclusive.

This name-value pair applies only when adding type-2 membership functions.

The following membership function types support only a scalar LowerLag value:

- Symmetric MFs — gaussmf and gbellmf
- One-sided MFs — sigmf, smf, and zmf

All other built-in membership functions support either a scalar or vector LowerLag value. For these membership functions, when you specify a:

- Scalar value, the same lag value is used for both the left and right side of the membership function.
- Vector value, you can define different lag values for the left and right sides of the membership function.

The lag value defines the point at which the lower membership function value starts increasing from zero based on the value of the upper membership function. For example, a lag value of 0.1 indicates that the lower membership function becomes positive when the upper membership function has a membership value of 0.1.

When the lag value is zero, the lower membership function starts increasing at the same point as the upper membership function.

Some membership function types restrict the maximum lag value. For example, LowerLag must be less than 1 for the gaussmf, gauss2mf, gbellmf, sigmf, dsigmf, and psigmf membership functions.

Output Arguments

fisOut — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:
• mamfis object — Mamdani fuzzy inference system
• sugfis object — Sugeno fuzzy inference system
• mamfistype2 object — Type-2 Mamdani fuzzy inference system
• sugfistype2 object — Type-2 Sugeno fuzzy inference system

fisOut contains the added membership function, with all other properties matching the properties of fisIn

varOut — Fuzzy variable
fisvar object

Fuzzy variable, returned as a fisvar object. varOut contains the added membership function, with all other properties matching the properties of varIn.

Compatibility Considerations

addmf is now addMF and its function syntax has changed
Behavior changed in R2018b

The name and behavior of the addmf function has changed. Now:

• addmf is addMF
• You specify the variable to which you want to add the membership function by name rather than by index.
• You specify the name of the membership function using a name-value pair argument.

These changes require updates to your code.

Update Code

The following table shows some typical usages of addmf for adding membership functions to fuzzy variables and how to update your code. In this table, fis is a fuzzy inference system with two inputs, service and food, and one output, tip.
If your code has this form:

<table>
<thead>
<tr>
<th>fis = addmf(fis'input',1,...</th>
</tr>
</thead>
<tbody>
<tr>
<td>'poor',...</td>
</tr>
<tr>
<td>'gaussmf',[1.5 0])</td>
</tr>
</tbody>
</table>

Use this code instead:

<table>
<thead>
<tr>
<th>fis = addMF(fis,&quot;service&quot;,...</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;gaussmf&quot;,[1.5 0],</td>
</tr>
<tr>
<td>'Name','poor&quot;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fis = addmf(fis,'input',2,...</th>
</tr>
</thead>
<tbody>
<tr>
<td>'rancid',...</td>
</tr>
<tr>
<td>'trapmf',[-2 0 1 3])</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fis = addMF(fis,&quot;food&quot;,...</th>
</tr>
</thead>
</table>
| "trapmf",[-2 0 1 3],...
| 'Name','rancid")          |

<table>
<thead>
<tr>
<th>fis = addmf(fis,'output',1,...</th>
</tr>
</thead>
<tbody>
<tr>
<td>'cheap',...</td>
</tr>
<tr>
<td>'trimf',[0 5 10])</td>
</tr>
</tbody>
</table>

| fis = addMF(fis,"tip",...
|--------------------------------|
| "trimf",[0 5 10],...
| 'Name','cheap")          |

---

Support for representing fuzzy inference systems as structures will be removed

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

**See Also**

`addInput` | `addOutput` | `addRule` | `fisvar` | `mamfis` | `removeMF` | `sugfis`

**Topics**

“Build Fuzzy Systems at the Command Line” on page 2-41

**Introduced in R2018b**
addOutput

Add output variable to fuzzy inference system

Syntax

fisOut = addOutput(fisIn)
fisOut = addOutput(fisIn,range)
fisOut = addOutput(____,Name,Value)

Description

fisOut = addOutput(fisIn) adds a default output variable to fisIn, and returns the resulting fuzzy system in fisOut. This output variable has a default name, default range, and no membership functions.

fisOut = addOutput(fisIn,range) adds an output variable with the specified range.

fisOut = addOutput(____,Name,Value) configures the output variable using one or more name-value pair arguments.

Examples

Add Output Variable to Fuzzy Inference System

Create a Mamdani fuzzy inference system.

fis = mamfis('Name','tipper');

Add an output variable with default specifications.

fis = addOutput(fis);

You can configure the output variable properties using dot notation. For example, specify the name and range for the variable.
fis.Outputs(1).Name = "tip";
fis.Outputs(1).Range = [10 30];

View the output variable.

fis.Outputs(1)

ans =
   fisvar with properties:
     Name: "tip"
     Range: [10 30]
   MembershipFunctions: [0x0 fismf]

You can also specify the variable name and range when you add it to the fuzzy system.

fis2 = mamfis('Name','tipper');
fis2 = addOutput(fis2,[10 30], 'Name','tip');

Add Output Variable with Membership Functions

Create a Sugeno fuzzy inference system.

fis = sugfis('Name','tipper');

Add an output variable with three constant membership functions distributed over the output range.

fis = addOutput(fis,'NumMFs',3,'MFType','constant');

View the membership functions.

fis.Outputs(1).MembershipFunctions

ans =
   1x3 fismf array with properties:
         Type
         Parameters
         Name

Details:
### Input Arguments

**fisIn** — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
- sugfistype2 object — Type-2 Sugeno fuzzy inference system

**range** — Variable range
[0 1] (default) | two-element vector

Variable range, specified as a two-element element vector where the first element is less than the second element. The first element specifies the lower bound of the range, and the second element specifies the upper bound of the range.

### Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'NumMFs',3 configures the variable to use three membership functions

**Name** — Variable name
string | character vector
Variable name, specified as the comma-separated pair consisting of 'Name' and a string or character vector.

**NumMFs — Number of membership functions**

0 (default) | nonnegative integer

Number of membership functions, specified as the comma-separated pair consisting of 'NumMFs' and a nonnegative integer.

**MFType — Membership function type**

"trimf" (default) | "gaussmf"

Membership function type, specified as the comma-separated pair consisting of 'MFType' and one of the following:

- "trimf" — Triangular membership functions for the outputs of Mamdani system
- "gaussmf" — Gaussian membership functions for the outputs of Mamdani systems
- "constant" — Constant membership functions for the outputs of Sugeno systems
- "linear" — Linear membership functions for the outputs of Sugeno systems. To add an output variable with linear membership functions, your FIS must have at least one input variable.

The membership functions are uniformly distributed over the variable range with approximately 80% overlap in the membership function supports.

**Output Arguments**

**fisOut — Fuzzy inference system**

mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
- sugfistype2 object — Type-2 Sugeno fuzzy inference system

fisOut contains the added output variable, with all other properties matching the properties of fisIn.
See Also
addInput | fisvar | removeOutput

Topics
“Build Fuzzy Systems at the Command Line” on page 2-41

Introduced in R2018b
addRule

Add rule to fuzzy inference system

Syntax

fisOut = addRule(fisIn)
fisOut = addRule(fisIn,ruleDescription)

Description

fisOut = addRule(fisIn) adds a single fuzzy rule to fuzzy inference system fisIn with the default description "input1==mf1 => output1=mf1" and returns the resulting fuzzy system in fisOut.

fisOut = addRule(fisIn,ruleDescription) adds one or more fuzzy rules using the rule descriptions in ruleDescription.

Examples

Add Single Rule to Fuzzy Inference System

Load a fuzzy inference system (FIS), and clear the existing rules.

fis = readfis('tipper');
fis.Rules = [];

Add a rule to the FIS.

ruleTxt = 'If service is poor then tip is cheap';
fis2 = addRule(fis,ruleTxt);

fis2 is equivalent to fis, except that the specified rule is added to the rule base.

fis2.Rules
Add Rules Using Symbolic Expressions

Load a fuzzy inference system (FIS), and clear the existing rules.

```matlab
fis = readfis('tipper');
fis.Rules = [];
```

Specify the following rules using symbolic expressions:

- If service is poor or food is rancid then tip is cheap.
- If service is excellent and food is not rancid then tip is generous.

```matlab
rule1 = "service==poor | food==rancid => tip=cheap";
rule2 = "service==excellent & food~=rancid => tip=generous";
rules = [rule1 rule2];
```

Add the rules to the FIS.

```matlab
fis2 = addRule(fis,rules);
```

`fis2` is equivalent to `fis`, except that the specified rules are added to the rule base.

```matlab
fis2.Rules
ans =
1x2 fisrule array with properties:
    Description
    Antecedent
    Consequent
    Weight
    Connection
```
Add Rules Using Membership Function Indices

Load fuzzy inference system (FIS) and clear the existing rules.

```matlab
fis = readfis('mam22.fis');
fis.Rules = [];
```

Specify the following rules using membership function indices:

- **If** angle **is** small **and** velocity **is** big, **then** force **is** negBig and force2 **is** posBig2.
- **If** angle **is** not small **and** velocity **is** small, **then** force **is** posSmall and force2 **is** negSmall2.

```matlab
rule1 = [1 2 1 4 1 1];
rule2 = [-1 1 3 2 1 1];
rules = [rule1; rule2];
```

Add the rules to the FIS.

```matlab
fis2 = addRule(fis,rules);
```

`fis2` is equivalent to `fis`, except that the specified rules are added to the rule base.

```
fis2.Rules
```

ans =
```
1x2 fisrule array with properties:
   Description
   Antecedent
   Consequent
   Weight
```
### Input Arguments

**fisIn** — Fuzzy inference system

mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
- sugfistype2 object — Type-2 Sugeno fuzzy inference system

**ruleDescription** — Rule description

string | character vector | numeric row vector | string array | character array | numeric array

Rule description, specified using either a text or numeric rule definition

**Text Rule Description**

For a text rule description, specify `ruleDescription` as one of the following:

- String or character vector specifying a single rule

  ```
  rule = "If service is poor or food is rancid then tip is cheap";
  ```

- String array, where each element corresponds to a rule. For example:

  ```
  ruleList = ['"If service is poor or 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"];
  ```
• Character array where each row corresponds to a rule. For example:

```
rule1 = 'If service is poor or food is rancid then tip is cheap';
rule2 = 'If service is good then tip is average';
rule3 = 'If service is excellent or food is delicious then tip is generous';
ruleList = char(rule1,rule2,rule3);
```

For each rule, use one of the following rule text formats:

• Verbose — Linguistic expression in the following format, using the IF and THEN keywords:

"IF <antecedent> THEN <consequent> (<weight>)"

In <antecedent>, specify the membership function for each input variable using the IS or IS NOT keyword. Connect these conditions using the AND or OR keywords. If a rule does not use a given input variable, omit it from the antecedent.

In <consequent>, specify the condition for each output variable using the IS or IS NOT keyword, and separate these conditions using commas. The IS NOT keyword is not supported for Sugeno outputs. If a rule does not use a given output variable, omit it from the consequent.

Specify the weight using a positive numerical value.

For example:

"IF A IS a AND B IS NOT b THEN X IS x, Y IS NOT y (1)"

• Symbolic — Expression that uses the symbols in the following table instead of keywords. There is no symbol for the IF keyword.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>IS (in rule antecedent)</td>
</tr>
<tr>
<td>~=</td>
<td>IS NOT</td>
</tr>
<tr>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>=&gt;</td>
<td>THEN</td>
</tr>
<tr>
<td>=</td>
<td>IS (in rule consequent)</td>
</tr>
</tbody>
</table>

For example, the following symbolic rule is equivalent to the previous verbose rule.

"A==a & B~=b => X=x, Y~=y (1)"
**Numeric Rule Description**

For a numeric rule description, specify `ruleDescription` as one of the following:

- Row vector to specify a single fuzzy rule
- Array, where each row of `ruleValues` specifies one rule

For each row, the numeric rule description has $M+N+2$ columns, where $M$ is the number of input variables and $N$ is the number of output variables. Each column contains the following information:

- The first $M$ columns specify input membership function indices and correspond to the Antecedent property of the rule. To indicate a NOT condition, specify a negative value. If a rule does not use a given input, set the corresponding index to 0. For each rule, at least one input membership function index must be nonzero.
- The next $N$ columns specify output membership function indices and correspond to the Consequent property of the rule. To indicate a NOT condition for Mamdani systems, specify a negative value. NOT conditions are not supported for Sugeno outputs. If a rule does not use a given output, set the corresponding index to 0. For each rule, at least one output membership function index must be nonzero.
- Column $M+N+1$ specifies the rule weight and corresponds to the Weight property of the rule.
- The final column specifies the antecedent fuzzy operator and corresponds to the Connection property of the rule.

**Output Arguments**

**fisOut** — Fuzzy inference system

`mamfis` object | `sugfis` object | `mamfistype2` object | `sugfistype2` object

Fuzzy inference system, specified as one of the following:

- `mamfis` object — Mamdani fuzzy inference system
- `sugfis` object — Sugeno fuzzy inference system
- `mamfistype2` object — Type-2 Mamdani fuzzy inference system
- `sugfistype2` object — Type-2 Sugeno fuzzy inference system

`fisOut` contains the added output rules, with all other properties matching the properties of `fisIn`. 

8-32
Compatibility Considerations

**addrule is now addRule**

*Behavior changed in R2018b*

`addrule` is now `addRule`. To update your code, change the function name from `addrule` to `addRule`. The syntaxes are equivalent.

**Support for representing fuzzy inference systems as structures will be removed**

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

**See Also**

`addInput` | `addMF` | `addOutput`

**Topics**

“Build Fuzzy Systems at the Command Line” on page 2-41

**Introduced in R2018b**
addvar

(To be removed) Add variable to fuzzy inference system

**Note** addvar will be removed in a future release. Use addInput or addOutput instead. For more information, see “Compatibility Considerations”.

**Syntax**

```matlab
fis = addvar(fis,varType,varName,varBounds)
```

**Description**

addvar has four input arguments:

- **fis** — Fuzzy inference system in the MATLAB workspace, specified as a FIS structure.
- **varType** — Type of variable to add, specified as 'input' or 'output'.
- **varName** — Name of the variable to add, specified as a character vector or string.
- **varBounds** — Variable range, specified as a two-element vector, where the first element is the minimum value and the second element is the maximum value for the variable.

Indices are applied to variables in the order in which they are added. Therefore, 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**

**Add Variable to Fuzzy Inference System**

Create new FIS.

```matlab
fis = newfis('tipper');
```
Add new input variable.

\[
fis = \text{addvar}(fis,'\text{input}','\text{service}',[0 \ 10]);
\]

View new variable properties.

\[
\text{getfis}(fis,'\text{input'},1)
\]

\[
\text{ans} = \text{struct with fields:}
\begin{align*}
\text{Name}: & \ '\text{service}' \\
\text{NumMFs}: & \ 0 \\
\text{range}: & \ [0 \ 10]
\end{align*}
\]

### Compatibility Considerations

**addvar will be removed**

*Not recommended starting in R2018b*

addvar will be removed in a future release. Use addInput or addOutput instead. There are differences between these functions that require updates to your code.

To add input or output variables to a fuzzy system, use addInput or addOutput, respectively.

**Update Code**

This table shows some typical usages of addvar and how to update your code to use addInput or addOutput instead.

<table>
<thead>
<tr>
<th>If your code has this form:</th>
<th>Use this code instead:</th>
</tr>
</thead>
</table>
| \[
fis = \text{addvar}(fis,'\text{input'},...\nonumber\text{service}',[0 \ 10])
\] | \[
fis = \text{addInput}(fis,[0 \ 10],...\nonumber\text{Name}',"\text{service}")
\] |
| \[
fis = \text{addvar}(fis,'\text{output'},...\nonumber\text{tip}',[0 \ 30])
\] | \[
fis = \text{addOutput}(fis,[0 \ 30],...\nonumber\text{Name}',"\text{tip}")
\] |

**See Also**

addInput | addMF | addOutput | addRule | rmmf | rmvar
Introduced before R2006a
anfis

Tune Sugeno-type fuzzy inference system using training data

Syntax

fis = anfis(trainingData)
fis = anfis(trainingData,options)

[fis,trainError] = anfis(____)
[fis,trainError,stepSize] = anfis(____)

[fis,trainError,stepSize,chkFIS,chkError] = anfis(trainingData, options)

Description

fis = anfis(trainingData) generates a single-output Sugeno fuzzy inference system (FIS) and tunes the system parameters using the specified input/output training data. The FIS object is automatically generated using grid partitioning.

The training algorithm uses a combination of the least-squares and backpropagation gradient descent methods to model the training data set.

fis = anfis(trainingData,options) tunes an FIS using the specified training data and options. Using this syntax, you can specify:

- An initial FIS object to tune.
- Validation data for preventing overfitting to training data.
- Training algorithm options.
- Whether to display training progress information.

[fis,trainError] = anfis(____) returns the root mean square training error for each training epoch.

[fis,trainError,stepSize] = anfis(____) returns the training step size at each training epoch.
[fis,trainError,stepSize,chkFIS,chkError] = anfis(trainingData, options) returns the validation data error for each training epoch, chkError, and the tuned FIS object for which the validation error is minimum, chkFIS. To use this syntax, you must specify validation data using options.ValidationData.

Examples

Train Fuzzy Inference System Using ANFIS

Load training data. This data has a single input and a single output.

load fuzex1trnData.dat

Generate and train a fuzzy inference system. By default, the FIS structure is created using a grid partition of the input variable range with two membership functions.

fis = anfis(fuzex1trnData);

ANFIS info:
Number of nodes: 12
Number of linear parameters: 4
Number of nonlinear parameters: 6
Total number of parameters: 10
Number of training data pairs: 25
Number of checking data pairs: 0
Number of fuzzy rules: 2

Start training ANFIS ...

1 0.229709
2 0.22896
3 0.228265
4 0.227624
5 0.227036
Step size increases to 0.011000 after epoch 5.
6 0.2265
7 0.225968
8 0.225488
9 0.225052
Step size increases to 0.012100 after epoch 9.
10 0.22465
Designated epoch number reached --> ANFIS training completed at epoch 10.

Minimal training RMSE = 0.224650

Plot the ANFIS output and training data.

```matlab
x = fuzex1trnData(:,1);
anfisOutput = evalfis(fis,x);
plot(x,fuzex1trnData(:,2),'*r',x,anfisOutput,'.b')
legend('Training Data','ANFIS Output','Location','NorthWest')
```

The ANFIS data does not match the training data well. To improve the match:
• Increase the number of membership functions in the FIS structure to 4. Doing so adds fuzzy rules and tunable parameters to the system.
• Increase the number of training epochs.

```matlab
opt = anfisOptions('InitialFIS',4,'EpochNumber',40);

Suppress the error and step size Command Window display.

```matlab
opt.DisplayErrorValues = 0;
opt.DisplayStepSize = 0;

Train the FIS.

```matlab
fis = anfis(fuzex1trnData,opt);

ANFIS info:
   Number of nodes: 20
   Number of linear parameters: 8
   Number of nonlinear parameters: 12
   Total number of parameters: 20
   Number of training data pairs: 25
   Number of checking data pairs: 0
   Number of fuzzy rules: 4

Minimal training RMSE = 0.083385

Plot the ANFIS output and training data.

```matlab
figure
anfisOutput = evalfis(fis,x);
plot(x,fuzex1trnData(:,2),'r',x,anfisOutput,'b')
legend('Training Data','ANFIS Output','Location','NorthWest')
```
The match between the training data and ANFIS output has improved.

**Create Initial FIS for ANFIS Training**

Create single-input, single-output training data.

```matlab
x = (0:0.1:10)';
y = sin(2*x).\exp(x/5);
```

Define an initial FIS structure with five Gaussian input membership functions.
genOpt = genfisOptions('GridPartition');
genOpt.NumMembershipFunctions = 5;
genOpt.InputMembershipFunctionType = 'gaussmf';
inFIS = genfis(x,y,genOpt);

Configure the ANFIS training options. Set the initial FIS, and suppress the training progress display.

opt = anfisOptions('InitialFIS',inFIS);
opt.DisplayANFISInformation = 0;
opt.DisplayErrorValues = 0;
opt.DisplayStepSize = 0;
opt.DisplayFinalResults = 0;

Train the FIS using the specified options.

outFIS = anfis([x y],opt);

Compare the ANFIS output with the training data.

plot(x,y,x,evalfis(outFIS,x))
legend('Training Data','ANFIS Output')
Obtain ANFIS Training Error

Load training data. This data has a single input and a single output.

```matlab
load fuzex2trnData.dat
```

Specify the training options.

```matlab
opt = anfisOptions('InitialFIS',4,'EpochNumber',40);
opt.DisplayANFISInformation = 0;
opt.DisplayErrorValues = 0;
```
Train the FIS, and return the training error.

\[ \text{[fis,trainError]} = \text{anfis(fuzex2trnData,opt)}; \]

\text{trainError} contains the root mean squared error for the training data at each training epoch. The training error for fis is the minimum value in trainError.

\[
\text{fisRMSE} = \min(\text{trainError})
\]

\[
\text{fisRMSE} = 0.2572
\]

**Obtain ANFIS Step Size Profile**

Create single-input, single-output training data.

\[
x = (0:0.1:10)';
\]

\[
y = \sin(2*x)./\exp(x/5);
\]

Configure the ANFIS training options. Set the initial FIS, and suppress the training progress display.

\[
\text{opt} = \text{anfisOptions('InitialFIS',4,'EpochNumber',60)};
\]

\[
\text{opt.DisplayANFISInformation} = 0;
\]

\[
\text{opt.DisplayErrorValues} = 0;
\]

\[
\text{opt.DisplayStepSize} = 0;
\]

\[
\text{opt.DisplayFinalResults} = 0;
\]

A larger step size increase rate can make the training converge faster. However, increasing the step size increase rate too much can lead to poor convergence. For this example, try doubling the step size increase rate.

\[
\text{opt.StepSizeIncreaseRate} = 2^{*}\text{opt.StepSizeIncreaseRate};
\]

Train the FIS, and return the step size array.

\[
\text{[fis,~,stepSize]} = \text{anfis([x y],opt)};
\]

Plot the step size profile. An optimal step size profile should increase initially, reach a maximum, and then decrease for the rest of the training.
Validate ANFIS Training

Load training data.
load fuzex1trnData.dat

Load validation data.
load fuzex1chkData.dat
Specify the following training options:

- 4 input membership functions
- 30 training epochs
- Suppress training progress display

```matlab
opt = anfisOptions('InitialFIS',4,'EpochNumber',30);
opt.DisplayANFISInformation = 0;
opt.DisplayErrorValues = 0;
opt.DisplayStepSize = 0;
opt.DisplayFinalResults = 0;
```

Add the validation data to the training options.

```matlab
opt.ValidationData = fuzex1chkData;
```

Train the FIS, and return the validation results.

```matlab
[fis,trainError,stepSize,chkFIS,chkError] = anfis(fuzex1trnData,opt);
```

The training error, `trainError`, and validation error, `chkError`, arrays each contain one error value per training epoch. Plot the training error and the validation error.

```matlab
x = [1:30];
plot(x,trainError,'.b',x,chkError,'*r')
```
The minimum validation error occurs at epoch 17. The increase in validation error after this point indicates overfitting of the model parameters to the training data. Therefore, the tuned FIS at epoch 17, chkFIS, exhibits the best generalization performance.

**Input Arguments**

`trainingData` — Training data

array

Training data, specified as an array. For a fuzzy system with $N$ inputs, specify `trainingData` as an array with $N+1$ columns. The first $N$ columns contain input data,
and the final column contains output data. Each row of `trainingData` contains one data point.

Generally, training data should fully represent the features of the data the FIS is intended to model.

**options — Training options**

`anfisOptions` option set

Training options, specified as an `anfisOptions` option set. Using `options`, you can specify:

- An initial FIS structure to tune, `options.InitialFIS`.
- Validation data for preventing overfitting to training data, `options.ValidationData`.
- Training algorithm options, such as the maximum number of training epochs, `options.EpochNumber`, or the training error goal, `options.ErrorGoal`.
- Whether to display training progress information, such as the training error values for each training epoch, `options.DisplayErrorValues`.

**Output Arguments**

**fis — Trained fuzzy inference system**

`mamfis` object | `sugfis` object

Trained fuzzy inference system with membership function parameters tuned using the training data, returned as a `mamfis` or `sugfis` object. This fuzzy system corresponds to the epoch for which the training error is smallest. If two epochs have the same minimum training error, the FIS from the earlier epoch is returned.

**trainError — Root mean square training error**

`array`

Root mean square training error for each training epoch, returned as an array. The minimum value in `trainError` is the training error for fuzzy system `fis`.

**stepSize — Training step size**

`array`
Training step size for each epoch, returned as an array. The anfis training algorithm tunes the FIS parameters using gradient descent optimization methods. The training step size is the magnitude of the gradient transitions in the parameter space.

Ideally, the step size increases at the start of training, reaches a maximum, and then decreases for the remainder of the training. To achieve this step size profile, adjust the initial step size (options.InitialStepSize), step size increase rate (options.StepSizeIncreaseRate), and step size decrease rate options.StepSizeDecreaseRate.

**chkFIS — Tuned FIS for which the validation error is minimum**

mamfis object | sugfis object

Tuned FIS for which the validation error is minimum, returned as a mamfis or sugfis object. If two epochs have the same minimum validation error, the FIS from the earlier epoch is returned.

chkFIS is returned only when you specify validation data using options.ValidationData.

**chkError — Root mean square validation error**

array

Root mean square training error, returned as an array with length equal to the number of training epochs. The minimum value in chkError is the training error for fuzzy system chkFIS.

chkError is returned only when you specify validation data using options.ValidationData.

### Alternative Functionality

**tunefis Function**

Starting in R2019a, you can tune a fuzzy system using tunefis. This function provides several other options for tuning algorithms, specified by the tunefisOptions object.

To use ANFIS, specify the tuning algorithm as "anfis" in tunefisOptions. Then, use the options object as an input argument for tunefis. For example:
Create the initial fuzzy inference system, and define the tunable parameter settings.

```matlab
x = (0:0.1:10)';
y = sin(2*x)./exp(x/5);
options = genfisOptions('GridPartition');
options.NumMembershipFunctions = 5;
fisin = genfis(x,y,options);
[in,out,rule] = getTunableSettings(fisin);
```

Tune the membership function parameters with "anfis".

```matlab
opt = tunefisOptions("Method","anfis");
fisout = tunefis(fisin,[in;out],x,y,opt);
```

**Compatibility Considerations**

**Support for representing fuzzy inference systems as structures will be removed**

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

**References**


**See Also**

**Apps**
Neuro-Fuzzy Designer

**Functions**
anfisOptions | genfis | tunefis

**Topics**
“Neuro-Adaptive Learning and ANFIS” on page 3-92
“Predict Chaotic Time-Series using ANFIS” on page 3-125
“Modeling Inverse Kinematics in a Robotic Arm” on page 3-135

**Introduced before R2006a**
anfisOptions
Option set for anfis command

Syntax

opt = anfisOptions
opt = anfisOptions(Name,Value)

Description

opt = anfisOptions creates a default option set for tuning a Sugeno fuzzy inference system using anfis. Use dot notation to modify this option set for your specific application. Any options that you do not modify retain their default values.

opt = anfisOptions(Name,Value) creates an option set with options specified by one or more Name,Value pair arguments.

Examples

Create Option Set for ANFIS Training

Create a default option set.

opt = anfisOptions;

Specify training options using dot notation. For example, specify the following options:

• Initial FIS with 4 membership functions for each input variable
• Maximum number of training epochs equal to 30.

opt.InitialFIS = 4;
opt.EpochNumber = 30;

You can also specify options when creating the option set using one or more Name,Value pair arguments.
```matlab
opt2 = anfisOptions('InitialFIS',4,'EpochNumber',30);
```

# Input Arguments

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'EpochNumber',50` sets the maximum number of training epochs to 50.

### InitialFIS — Initial FIS structure

2 (default) | positive integer greater than 1 | vector of positive integers | FIS structure

Initial FIS structure to tune, specified as the comma-separated pair consisting of 'InitialFIS' and one of the following:

- Positive integer greater than 1 specifying the number of membership functions for all input variables. `anfis` generates an initial FIS structure with the specified number of membership functions using `genfis` with grid partitioning.
- Vector of positive integers with length equal to the number of input variables specifying the number of membership functions for each input variable. `anfis` generates an initial FIS structure with the specified numbers of membership functions using `genfis` with grid partitioning.
- FIS structure generated using `genfis` command with grid partitioning or subtractive clustering. The specified system must have the following properties:
  - Single output, obtained using weighted average defuzzification.
  - First or zeroth order Sugeno-type system; that is, all output membership functions must be the same type and be either 'linear' or 'constant'.
  - No rule sharing. Different rules cannot use the same output membership function; that is, the number of output membership functions must equal the number of rules.
  - Unity weight for each rule.
- No custom membership functions or defuzzification methods.

**EpochNumber — Maximum number of training epochs**

10 (default) | positive integer

Maximum number of training epochs, specified as the comma-separated pair consisting of 'EpochNumber' and a positive integer. The training process stops when it reaches the maximum number of training epochs.

**ErrorGoal — Training error goal**

0 (default) | scalar

Training error goal, specified as the comma-separated pair consisting of 'ErrorGoal' and a scalar. The training process stops when the training error is less than or equal to ErrorGoal.

**InitialStepSize — Initial training step size**

0.01 (default) | positive scalar

Initial training step size, specified as the comma-separated pair consisting of 'InitialStepSize' and a positive scalar.

The anfis training algorithm tunes the FIS parameters using gradient descent optimization methods. The training step size is the magnitude of each gradient transition in the parameter space. Typically, you can increase the rate of convergence of the training algorithm by increasing the step size. During optimization, anfis automatically updates the step size using StepSizeIncreaseRate and StepSizeDecreaseRate.

Generally, the step-size profile during training is a curve that increases initially, reaches some maximum, and then decreases for the remainder of the training. To achieve this ideal step-size profile, adjust the initial step-size and the increase and decrease rates (opt.StepSizeDecreaseRate, opt.StepSizeIncreaseRate).

**StepSizeDecreaseRate — Step-size decrease rate**

0.9 (default) | positive scalar less than 1

Step-size decrease rate, specified as the comma-separated pair consisting of 'StepSizeDecreaseRate' and a positive scalar less than 1. If the training error undergoes two consecutive combinations of an increase followed by a decrease, then anfis scales the step size by the decrease rate.

**StepSizeIncreaseRate — Step-size increase rate**

1.1 (default) | scalar greater than 1
Step-size increase rate, specified as the comma-separated pair consisting of 'StepSizeIncreaseRate' and a scalar greater than 1. If the training error decreases for four consecutive epochs, then anfis scales the step size by the increase rate.

**DisplayANFISInformation — Flag for showing ANFIS information**

1 (default) | 0

Flag for showing ANFIS information at the start of the training process, specified as the comma-separated pair consisting of 'DisplayANFISInformation' and one of the following:

- 1 — Display the following information about the ANFIS system and training data:
  - Number of nodes in the ANFIS system
  - Number of linear parameters to tune
  - Number of nonlinear parameters to tune
  - Total number of parameters to tune
  - Number of training data pairs
  - Number of checking data pairs
  - Number of fuzzy rules
- 0 — Do not display the information.

**DisplayErrorValues — Flag for showing training error values**

1 (default) | 0

Flag for showing training error values after each training epoch, specified as the comma-separated pair consisting of 'DisplayErrorValues' and one of the following:

- 1 — Display the training error.
- 0 — Do not display the training error.

**DisplayStepSize — Flag for showing step size**

1 (default) | 0

Flag for showing step size whenever the step size changes, specified as the comma-separated pair consisting of 'DisplayStepSize' and one of the following:

- 1 — Display the step size.
- 0 — Do not display the step size.
DisplayFinalResults — Flag for displaying final results
1 (default) | 0

Flag for displaying final results after training, specified as the comma-separated pair consisting of 'DisplayFinalResults' and one of the following:

- 1 — Display the results.
- 0 — Do not display the results.

ValidationData — Validation data
[] (default) | array

Validation data for preventing overfitting to the training data, specified as the comma-separated pair consisting of 'ValidationData' and an array. For a fuzzy system with N inputs, specify ValidationData as an array with N+1 columns. The first N columns contain input data and the final column contains output data. Each row of ValidationData contains one data point.

At each training epoch, the training algorithm validates the FIS using the validation data.

Generally, validation data should fully represent the features of the data the FIS is intended to model, while also being sufficiently different from the training data to test training generalization.

OptimizationMethod — Optimization method
1 (default) | 0

Optimization method used in membership function parameter training, specified as the comma-separated pair consisting of 'OptimizationMethod' and one of the following:

- 1 — Use a hybrid method, which uses a combination of backpropagation to compute input membership function parameters, and least squares estimation to compute output membership function parameters.
- 0 — Use backpropagation gradient descent to compute all parameters.

Output Arguments

opt — Training options for anfis command
anfisOptions option set

Training options for anfis command, returned as an anfisOptions option set.
See Also
anfis | genfis

Introduced in R2017a
**convertfis**

Convert previous versions of fuzzy inference data in current format

**Syntax**

```matlab
fisNew = convertfis(fisOld)
```

**Description**

In R2018b, the format of fuzzy inference systems changed from a structure format to an object format. To convert fuzzy systems in an old format to the new format, use `convertfis`.

`fisNew = convertfis(fisOld)` converts the old-format fuzzy inference system `fisOld` into the current object format.

**Examples**

**Convert Old-Format Fuzzy Inference System**

Load a fuzzy inference system created using an old format. For example, load a FIS structure from a MAT-file.

```matlab
load fisStructure
```

View the fields of the structure.

`fisStructure`

```matlab
fisStructure = struct with fields:
    name: 'tipper'
    type: 'mamdani'
    andMethod: 'min'
    orMethod: 'max'
    defuzzMethod: 'centroid'
```
Convert the structure to a `mamfis` object and view the object properties.

```matlab
fisObject = convertfis(fisStructure)
```

```matlab
fisObject =
    mamfis with properties:
        Name: "tipper"
        AndMethod: "min"
        OrMethod: "max"
        ImplicationMethod: "min"
        AggregationMethod: "max"
        DefuzzificationMethod: "centroid"
            Inputs: [1x2 fisvar]
            Outputs: [1x1 fisvar]
            Rules: [1x3 fisrule]
        DisableStructuralChecks: 0
```

See 'getTunableSettings' method for parameter optimization.

### Input Arguments

`fisOld` — Old-format fuzzy inference system

structure | matrix

Old-format fuzzy inference system, specified as a structure or a matrix.

### Output Arguments

`fisNew` — New-format fuzzy inference system

mamfis object | sugfis object

New-format fuzzy inference system, returned as a `mamfis` object or a `sugfis` object.
See Also

mamfis | sugfis

Introduced in R2018b
**convertToStruct**

Convert fuzzy inference system object into a structure

**Syntax**

```matlab
fisStructure = convertToStruct(fisObject)
```

**Description**

`fisStructure = convertToStruct(fisObject)` converts a fuzzy inference system object into a structure.

**Examples**

**Convert FIS Object into Structure**

Load a fuzzy inference system.

```matlab
fisObject = readfis('tipper')
```

```matlab
fisObject =
       mamfis with properties:
               Name: 'tipper'
             AndMethod: 'min'
              OrMethod: 'max'
     ImplicationMethod: 'min'
     AggregationMethod: 'max'
DefuzzificationMethod: 'centroid'
        Inputs: [1x2 fisvar]
        Outputs: [1x1 fisvar]
          Rules: [1x3 fisrule]
DisableStructuralChecks: 0
```
See 'getTunableSettings' method for parameter optimization.

Convert the fuzzy inference system object into a structure.

```matlab
fisStructure = convertToStruct(fisObject)
```

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

**Input Arguments**

- **fisObject** — Fuzzy inference system object
  mamfis object | sugfis object

  Fuzzy inference system object, specified as a mamfis or sugfis object.

**Output Arguments**

- **fisStructure** — Fuzzy inference system structure
  structure

  Fuzzy inference system structure, returned as a structure. The fields of the structure correspond to the properties of the FIS object. For object properties that are themselves objects, the corresponding structure field is a structure.

**See Also**

mamfis | sugfis
Topics
“Build Fuzzy Systems at the Command Line” on page 2-41

Introduced in R2018b
**convertToSugeno**

Convert Mamdani fuzzy inference system into Sugeno fuzzy inference system

**Syntax**

sugenoFIS = convertToSugeno(mamdaniFIS)

**Description**

`sugenoFIS = convertToSugeno(mamdaniFIS)` converts the Mamdani fuzzy inference system `mamdaniFIS` into a Sugeno fuzzy inference system `sugenoFIS`.

**Examples**

**Transform Mamdani FIS into Sugeno FIS**

Load a Mamdani fuzzy inference system.

```matlab
mam_fismat = readfis('mam22.fis');
```

Convert this system to a Sugeno fuzzy inference system.

```matlab
sug_fismat = convertToSugeno(mam_fismat);
```

Plot the output surfaces for both fuzzy systems.

```matlab
subplot(2,2,1)
gensurf(mam_fismat)
title('Mamdani system (Output 1)')
subplot(2,2,2)
gensurf(sug_fismat)
title('Sugeno system (Output 1)')
subplot(2,2,3)
gensurf(mam_fismat,gensurfOptions('OutputIndex',2))
title('Mamdani system (Output 2)')
The output surfaces for both systems are similar.

**Input Arguments**

mamdaniFIS — Mamdani fuzzy inference system
mamfis object | mamfistype2 object

Mamdani fuzzy inference system, specified as a mamfis or mamfistype2 object.
Output Arguments

sugenoFIS — Sugeno fuzzy inference system
sugfis object | sugfistype2 object

Sugeno fuzzy inference system, returned as one of the following:

• sugfis object when mamdaniFIS is a mamfis object
• sugfistype2 object when mamdaniFIS is a mamfistype2 object

sugenoFIS:

• Has constant output membership functions, whose values correspond to the centroids of the output membership functions in mamdaniFIS
• Uses the weighted-average defuzzification method
• Uses the product implication method
• Uses the sum aggregation method

The remaining properties of sugenoFIS, including the input membership functions and rule definitions remain unchanged from mamdaniFIS.

Tips

• If you have a functioning Mamdani fuzzy inference system, consider using convertToSugeno to convert to a more computationally efficient Sugeno structure to improve performance.

See Also

Functions
mamfis | mamfistype2 | sugfis | sugfistype2

Apps
Fuzzy Logic Designer

Topics
“Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2
Introduced in R2018b
**convertToType1**

Convert type-2 fuzzy inference system into type-1 fuzzy inference system

**Syntax**

\[ \text{fisT1} = \text{convertToType1}(\text{fisT2}) \]

**Description**

\( \text{fisT1} = \text{convertToType1}(\text{fisT2}) \) converts the type-2 fuzzy inference system \( \text{fisT2} \) into a type-1 fuzzy inference system \( \text{fisT1} \).

**Examples**

**Convert Type-2 FIS to Type-1 FIS**

Create a type-2 fuzzy inference system. For this example, Create a type-2 Mamdani FIS with two inputs, one output.

\[ \text{fisT2} = \text{mamfistype2}('\text{NumInputs}',2,'\text{NumOutputs}',1); \]

View the membership function for the first input variable.

\[ \text{plotmf} \left( \text{fisT2},'\text{input}',1 \right) \]
Convert fisT2 into a type-1 fuzzy inference system.

fisT1 = convertToType1(fisT2);

View the converted membership functions for the first input variable.

plotmf(fisT1,"input",1)
Input Arguments

`fisT2 — Type-2 fuzzy inference system`
mamfistype2 object | sugfistype2 object

Type-2 fuzzy inference system, specified as a mamfistype2 or sugfistype2 object.
Output Arguments

fisT1 — Type-1 fuzzy inference system
mamfis object | sugfis object

Type-1 fuzzy inference system, returned as one of the following:

- mamfis object when fisT2 is a mamfistype2 object
- sugfis object when fisT2 is a sugfistype2 object

The properties of fisT1 match the corresponding properties of fisT2, except that each type-2 membership function is converted to a type-1 membership function. The parameters of each type-1 membership function in fisT1 match the upper membership function parameters of the corresponding type-2 membership function in fisT2.

See Also

convertToSugeno | convertToType2

Topics

“Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2
“Type-2 Fuzzy Inference Systems” on page 2-7

Introduced in R2019b
**convertToType2**

Convert type-1 fuzzy inference system into type-2 fuzzy inference system

**Syntax**

\[
\text{fisT2} = \text{convertToType2(fisT1)}
\]

**Description**

\( \text{fisT2} = \text{convertToType2(fisT1)} \) converts the type-1 fuzzy inference system \( \text{fisT1} \) into a type-2 fuzzy inference system \( \text{fisT2} \).

**Examples**

**Convert Type-1 FIS to Type-2 FIS**

Create a type-1 fuzzy inference system. For this example, load the `tipper.fis` file.

\[
\text{fisT1} = \text{readfis('tipper')};
\]

View the membership function for the first input variable.

\[
\text{plotmf(fisT1,'input',1)}
\]
Convert `fisT1` into a type-2 fuzzy inference system.

```matlab
fisT2 = convertToType2(fisT1);
```

View the converted membership functions for the first input variable.

```matlab
plotmf(fisT2, "input", 1)
```
Create Type-2 Fuzzy Inference System from Data

To create a type-2 FIS from input/output data, you must first create a type-1 FIS using \texttt{genfis}.

Load training data and generate a FIS using subtractive clustering.

\begin{verbatim}
load clustdemo.dat
inputData = clustdemo(:,1:2);
outputData = clustdemo(:,3);
opt = genfisOptions('SubtractiveClustering',...)
\end{verbatim}
Convert the generated FIS to a type-2 FIS.

fisT2 = convertToType2(fisT1);

Since the initial type-1 FIS is a Sugeno system, only the input MFs are converted to type-2 MFs.

Input Arguments

fisT1 — Type-1 fuzzy inference system
mamfis object | sugfis object

Type-1 fuzzy inference system, specified as a mamfis or sugfis object.

Output Arguments

fisT2 — Type-2 fuzzy inference system
mamfistype2 object | sugfistype2 object

Type-2 fuzzy inference system, returned as one of the following:

- mamfistype2 object when fisT1 is a mamfis object
- sugfistype2 object when fisT1 is a sugfis object

The properties of fisT2 match the corresponding properties of fisT1, except that each type-1 membership function (except for Sugeno output membership functions) is converted to a type-2 membership function. The upper membership function parameters
of each type-2 membership function in fisT2 match the membership function parameters of the corresponding type-1 membership function in fisT1.

fisT2 has default LowerScale and LowerLag values and uses the default "karnikmendel" type reduction method.

See Also
convertToSugeno | convertToType1

Topics
“Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2
“Type-2 Fuzzy Inference Systems” on page 2-7

Introduced in R2019b
defuzz

Defuzzify membership function

Syntax

output = defuzz(x,mf,method)

Description

output = defuzz(x,mf,method) returns the defuzzified output value for membership function mf at the variable values in x using the specified defuzzification method.

Examples

Obtain Defuzzified Value

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

out = -3.2857

Input Arguments

x — Variable values
    vector
Variable values,

mf — Membership function values
    vector
Membership function values, specified as a vector with the same length as \( x \). Each element of \( mf \) contains a fuzzy membership value for the corresponding variable value in \( x \).

**method — Defuzzification method**

'centroid' | 'bisector' | 'mom' | 'lom' | 'som' | character vector | string

Defuzzification method, specified as one of the following:

- 'centroid' — Centroid of the area under the output fuzzy set
- 'bisector' — Bisector of the area under the output fuzzy set
- 'mom' — Mean of the values for which the output fuzzy set is maximum
- 'lom' — Largest value for which the output fuzzy set is maximum
- 'som' — Smallest value for which the output fuzzy set is maximum
- Character vector or string that contains the name of a custom function in the current working folder or on the MATLAB path

For more information on:

- The built-in defuzzification methods, see “Defuzzification Methods” on page 1-40.
- Custom defuzzification methods, see “Build Fuzzy Systems Using Custom Functions” on page 2-52

**Output Arguments**

**output — Defuzzified output value**

scalar

Defuzzified output value, returned as a scalar.

**See Also**

Fuzzy Logic Designer

**Topics**

“Foundations of Fuzzy Logic” on page 1-10
“Fuzzy Inference Process” on page 1-28
“Defuzzification Methods” on page 1-40

**Introduced before R2006a**
**dsigmf**

Difference between two sigmoidal membership functions

**Syntax**

\[ y = \text{dsigmf}(x, \text{params}) \]

**Description**

This function computes fuzzy membership values using the difference between two sigmoidal membership functions. You can also compute this membership function using a \text{fismf} object. For more information, see “\text{fismf} Object” on page 8-82.

This membership function is related to the \text{sigmf} and \text{psigmf} membership functions.

\[ y = \text{dsigmf}(x, \text{params}) \] returns fuzzy membership values computed using the difference between two sigmoidal membership functions. Each sigmoidal function is given by:

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

To specify the \( a \) and \( c \) parameters for each sigmoidal function, use \text{params}.

Membership values are computed for each input value in \( x \).

**Examples**

**Obtain Difference of Two Sigmoidal Functions**

\[
x = 0:0.1:10;
y = \text{dsigmf}(x, [5 2 5 7]);
\text{plot}(x,y)
\text{xlabel('dsigmf, P = [5 2 5 7]')}
\]
**Input Arguments**

\( x \) — Input values  
 scalar | vector  

Input values for which to compute membership values, specified as a scalar or vector.

\( \text{params} \) — Membership function parameters  
 vector of length four
Membership function parameters, specified as the vector \([a_1 \ c_1 \ a_2 \ c_2]\). Here, \(a_1\) and \(c_1\) are the parameters of the first sigmoidal function, and \(a_2\) and \(c_2\) are the parameters of the second sigmoidal function.

For each sigmoidal function, to open the function to the left or right, specify a negative or positive value for \(a\), respectively. The magnitude of \(a\) defines the width of the transition area, and parameter \(c\) defines the center of the transition area.

To define a unimodal membership function with a maximum value of 1, specify the same signs for \(a_1\) and \(a_2\), and select \(c\) values far enough apart to allow for both transition areas to reach 1.

**Output Arguments**

\(y\) — Membership value  
scalar | vector

Membership value returned as a scalar or a vector. The dimensions of \(y\) match the dimensions of \(x\). Each element of \(y\) is the membership value computed for the corresponding element of \(x\).

**Alternative Functionality**

**fismf Object**

You can create and evaluate a fismf object that implements the dsigmf membership function.

\[
\text{mf} = \text{fismf}('\text{dsigmf}',\text{P});  \\
\text{Y} = \text{evalmf(mf,}X)\text{);}
\]

Here, \(X\), \(P\), and \(Y\) correspond to the \(x\), \(\text{params}\), and \(y\) arguments of \(\text{dsigmf}\), respectively.
Extended Capabilities

**C/C++ Code Generation**
Generate C and C++ code using MATLAB® Coder™.

**See Also**
gauss2mf | gaussmf | gbellmf | pimf | psigmf | sigmf | smf | trapmf | trimf | zmf

**Topics**
“Membership Functions” on page 1-14

**Introduced before R2006a**
evalfis

Evaluate fuzzy inference system

Syntax

\[
\text{output} = \text{evalfis}(\text{fis}, \text{input}) \\
\text{output} = \text{evalfis}(\text{fis}, \text{input}, \text{options}) \\
[\text{output}, \text{fuzzifiedIn}, \text{ruleOut}, \text{aggregatedOut}, \text{ruleFiring}] = \text{evalfis}(\_\_\_)
\]

Description

\[
\text{output} = \text{evalfis}(\text{fis}, \text{input}) \text{ evaluates the fuzzy inference system } \text{fis} \text{ for the input values in } \text{input} \text{ and returns the resulting output values in } \text{output}.
\]

\[
\text{output} = \text{evalfis}(\text{fis}, \text{input}, \text{options}) \text{ evaluates the fuzzy inference system using specified evaluation options.}
\]

\[
[\text{output}, \text{fuzzifiedIn}, \text{ruleOut}, \text{aggregatedOut}, \text{ruleFiring}] = \text{evalfis}(\_\_\_) \text{ returns intermediate results from the fuzzy inference process. This syntax is not supported when } \text{fis} \text{ is a fistree object.}
\]

Examples

Evaluate Fuzzy Inference System

Load FIS.

\[
fis = \text{readfis('tipper')};
\]

Evaluate the FIS when the first input is 2 and the second input is 1.

\[
\text{output} = \text{evalfis}(\text{fis}, [2 1])
\]

\[
\text{output} = 7.0169
\]
Evaluate FIS for Multiple Input Combinations

Load FIS.

```matlab
fis = readfis('tipper');
```

Specify the input combinations to evaluate using an array with one row per input combination.

```matlab
input = [2 1;
         4 5;
         7 8];
```

Evaluate the FIS for the specified input combinations.

```matlab
output = evalfis(fis,input)
```

```
output = 3×1
       7.0169
       14.4585
      20.3414
```

Each row of `output` is the defuzzified output value for the corresponding row of `input`.

Specify Number of Output Samples for FIS Evaluation

Load FIS.

```matlab
fis = readfis('tipper');
```

Create an `evalfisOptions` option set, specifying the number of samples in the output fuzzy sets.

```matlab
options = evalfisOptions('NumSamplePoints',50);
```

Evaluate the FIS using this option set.

```matlab
output = evalfis(fis,[2 1],options);
```
Evaluate Tree of Fuzzy Inference Systems

Create a pair of Mamdani fuzzy inference systems.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = mamfis('Name','fis2','NumInputs',2,'NumOutputs',1);
```

Define the connection between the two.

```matlab
con = ["fis1/output1" "fis2/input1"];
```

Create a tree of fuzzy inference systems.

```matlab
tree = fistree([fis1 fis2],con);
```

Create an `evalfisOptions` option set, specifying the number of samples in the output fuzzy sets.

```matlab
options = evalfisOptions('NumSamplePoints',50);
```

Evaluate the `fistree` object using a specified input combination and this option set.

```matlab
y = evalfis(tree,[0.5 0.2 0.7],options)
y = 0.1553
```

Obtain Intermediate Fuzzy Inference Results

Load FIS.

```matlab
fis = readfis('tipper');
```

Evaluate the FIS, and return the intermediate inference results.

```matlab
[output,fuzzifiedIn,ruleOut,aggregatedOut,ruleFiring] = evalfis(fis,[2 1]);
```

You can examine the intermediate results to understand or visualize the fuzzy inference process. For example, view the aggregated output fuzzy set, which is the fuzzy set that `evalfis` defuzzifies to find the output value. Also, plot the defuzzified output value.
outputRange = linspace(fis.output.range(1),fis.output.range(2),length(aggregatedOut))'
plot(outputRange,aggregatedOut,[output output],[0 1])
xlabel('Tip')
ylabel('Output Membership')
legend('Aggregated output fuzzy set','Defuzzified output')

The length of aggregatedOutput corresponds to the number of sample points used to discretize output fuzzy sets.

**Evaluate Type-2 Fuzzy Inference System**

Create a type-2 Mamdani fuzzy inference system.
fis = mamfistype2('NumInputs',2,'NumOutputs',1);

Evaluate the FIS when the first input is 0.4 and the second input is 0.72.
output = evalfis(fis,[0.4 0.72])
output = 0.1509

The output of a type-2 FIS is a crisp value.

When you obtain intermediate fuzzy inference results for a type-2 FIS, you obtain intermediate results generated using both upper and lower MF values. For example, obtain the intermediate fuzzified input values.

[output,fuzzifiedInput] = evalfis(fis,[0.5 0.75]);

View the fuzzified input values.
fuzzifiedInput

fuzzifiedInput = 9×4

0 0 0 0
1.0000 0 1.0000 0
0 0 0 0
0 0.4000 0 0.2500
1.0000 0.4000 1.0000 0.2500
0 0.4000 0 0.2500
0 0.4000 0 0.2500
1.0000 0.4000 1.0000 0.2500
0 0.4000 0 0.2500

The first two columns contain the fuzzified values of the first and second inputs based on the upper MF for each input. The second two columns contain the fuzzified values for based on the lower MF for each input.

**Input Arguments**

**fis — Fuzzy inference system**
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object | fistree object
Fuzzy inference system to be evaluated, specified as one of the following:

- `mamfis` object — Mamdani fuzzy inference system
- `sugfis` object — Sugeno fuzzy inference system
- `mamfistype2` object — Type-2 Mamdani fuzzy inference system
- `sugfistype2` object — Type-2 Sugeno fuzzy inference system
- `fistree` object - Tree of interconnected fuzzy inference systems
- Homogeneous structure created using `getFISCodeGenerationData`. For an example, see “Generate Code for Fuzzy System Using MATLAB Coder” on page 6-11.

**input — Input values**

* `M-by-N_U` array

Input values, specified as an `M`-by-`N_U` array, where `N_U` is the number of input variables in `fis` and `M` is the number of input combinations to evaluate.

`evalfis` supports double-precision or single-precision input values.

**options — Evaluation options**

* `evalfisOptions` object

Evaluation options, specified as an `evalfisOptions` object.

**Output Arguments**

**output — Output values**

* `array`

Output values, returned as an `M`-by-`N_Y` array, where `N_Y` is the number of output variables in `fis`. `evalfis` evaluates `fis` for each row of `input` and returns the resulting defuzzified outputs in the corresponding row of `output`.

**fuzzfiedIn — Fuzzified input values**

* `array`

Fuzzified input values, returned as an array.
When \texttt{fis} is a type-1 fuzzy inference system, \texttt{fuzzifiedIn} is an $N_R$-by-$N_U$ array, where $N_R$ is the number of rules in \texttt{fis}. Element $(i,j)$ of \texttt{fuzzifiedIn} is the value of the input membership function for the $j$th input in the $i$th rule.

When \texttt{fis} is a type-2 fuzzy inference system, \texttt{fuzzifiedIn} is an $N_R$-by-$(2*N_U)$ array. The first $N_U$ columns contain the fuzzified values of the upper membership function for each rule, and the last $N_U$ columns contain the fuzzified values from the lower membership functions.

If \texttt{input} specifies multiple input combinations, then \texttt{fuzzifiedIn} corresponds to the combination in the last row of \texttt{input}.

For more information on fuzzifying input values, see “Fuzzify Inputs” on page 1-29.

This output argument is not supported when \texttt{fis} is a \texttt{fistree} object.

\textbf{ruleOut — Rule outputs}

array

Rule outputs, returned as an array. To obtain the output for each rule, \texttt{evalfis} applies the firing strength from the rule antecedent to the output membership function using the implication method specified in \texttt{fis}.

When \texttt{fis} is a type-1 Mamdani system, \texttt{ruleOut} is an $N_S$-by-$(N_R*N_Y)$ array, where $N_R$ is the number of rules, $N_Y$ is the number of outputs, and $N_S$ is the number of sample points used for evaluating output variable ranges. Each column of \texttt{ruleOut} contains the output fuzzy set for one rule. The first $N_R$ columns contain the rule outputs for the first output variable, the next $N_R$ columns correspond to the second output variable, and so on.

When \texttt{fis} is a type-2 Mamdani system, \texttt{ruleOut} is an $N_S$-by-$(2*N_R*N_Y)$ array. The first $N_R*N_Y$ columns contain the rule outputs generated using upper membership functions, and the last $N_R*N_Y$ columns contain the rule outputs generated using lower membership functions.

When \texttt{fis} is a type-1 Sugeno system, each rule output is a scalar value. In this case, \texttt{ruleOut} is an $N_R$-by-$N_Y$ array. Element $(j,k)$ of \texttt{ruleOut} is the value of the $k$th output variable for the $j$th rule.

When \texttt{fis} is a type-2 Sugeno system, \texttt{ruleOut} is an $N_R$-by-$(3*N_Y)$ array. The first $N_Y$ columns contain the rule output levels. The next $N_Y$ columns contain the corresponding rule firing strengths generated using upper membership functions. The last $N_Y$ columns contain the rule firing strengths generated using lower membership functions. For
example, in a three-output system, columns 4 and 7 contain the firing strengths for the output levels in column 1.

If input specifies multiple input combinations, then ruleOut corresponds to the combination in the last row of input.

For more information on fuzzy implication, see “Apply Implication Method” on page 1-31.

This output argument is not supported when fis is a fistree object.

**aggregatedOut — Aggregated output**

array | row vector

Aggregated output for each output variable, returned as an array.

$N_S$-by-$N_Y$ array or a row vector of length $N_Y$. For each output variable, evalfis combines the corresponding outputs from all the rules using the aggregation method specified in fis.

For a type-1 Mamdani system, the aggregate result for each output variable is a fuzzy set. In this case, aggregatedOut is as an $N_S$-by-$N_Y$ array, where $N_Y$ is the number of outputs and $N_S$ is the number of sample points used for evaluating output variable ranges. Each column of aggregatedOut contains the aggregate fuzzy set for one output variable.

For a type-2 Mamdani system, the aggregate result for each output variable is a fuzzy set. In this case, aggregatedOut is as an $N_S$-by-$(2*N_Y)$ array. The first $N_Y$ columns contain the aggregated outputs generated using upper membership functions, and the last $N_Y$ columns contain the aggregated outputs generated using lower membership functions.

When fis is a type-1 Sugeno system, the aggregate result for each output variable is a scalar value. In this case, aggregatedOut is a row vector of length $N_Y$, where element $k$ is the sum of the rule outputs for the $k$th output variable.

When fis is a type-2 Sugeno system, aggregatedOut is an $N_R$-by-$(3*N_Y)$ array. aggregatedOut contains the same data as ruleOut with the columns sorted based on the output levels. For example, in a three-output system, when the output levels in column 1 are sorted, the corresponding firing strengths in columns 4 and 7 are adjusted accordingly.

If input specifies multiple input combinations, then aggregatedOut corresponds to the combination in the last row of input.

For more information on fuzzy aggregation, see “Aggregate All Outputs” on page 1-31.
This output argument is not supported when `fis` is a `fistree` object.

**ruleFiring — Rule firing strengths**

column vector | array

Rule firing strength, returned as a column vector or array. To obtain the firing strength for each rule, `evalfis` evaluates the rule antecedents; that is, it applies fuzzy operator to the values of the fuzzified inputs.

For a type-1 fuzzy system, `ruleFiring` is a column vector of length $N_R$, where $N_R$ is the number of rules, and element $i$ is the firing strength of the $i$th rule.

For a type-2 fuzzy system, `ruleFiring` is an $N_R$-by-2 array. The first column contains the rule firing strengths generated using upper membership functions, and the second column contains the rule firing strengths generated using lower membership functions.

If `input` specifies multiple input combinations, then `ruleFiring` corresponds to the combination in the last row of `input`.

For more information on applying the fuzzy operator, see “Apply Fuzzy Operator” on page 1-30.

This output argument is not supported when `fis` is a `fistree` object.

**Alternative Functionality**

**App**

You can evaluate type-1 fuzzy inference systems using the **Rule Viewer** in the **Fuzzy Logic Designer** app.

**Simulink Block**

You can evaluate fuzzy inference systems using the Fuzzy Logic Controller block. For more information on mapping the arguments of `evalfis` to the Fuzzy Logic Controller block, see “Simulate Fuzzy Inference Systems in Simulink” on page 5-2.
Compatibility Considerations

**evalfis input argument order has changed**

*Behavior changed in R2018b*

The order of input arguments for `evalfis` has changed, which requires updates to your code.

**Update Code**

Previously, to evaluate a fuzzy inference system, `fis`, you specified the input variable values, `input`, as the first input argument. For example:

```matlab
output = evalfis(input,fis);
output = evalfis(input,fis,options);
```

Update your code to specify the fuzzy inference system as the first input argument. For example:

```matlab
output = evalfis(fis,input);
output = evalfis(fis,input,options);
```

**To specify the number of sample points for output fuzzy sets, you now use an evalfisOptions object**

*Behavior changed in R2018a*

To specify the number of sample points for output fuzzy sets, you now use an `evalfisOptions` object, which requires updates to your code.

**Update Code**

Previously, to specify the number of sample points, `numPts`, to use when evaluating output fuzzy sets of fuzzy inference system `fis`, you used an input argument. For example:

```matlab
output = evalfis(input,fis,numPts);
```

Update your code to specify the number of sample points using an `evalfisOptions` object. For example:

```matlab
opt = evalfisOptions('NumSamplePoints',numPts);
output = evalfis(input,fis,opt);
```
evalfis diagnostic message behavior has changed

Behavior changed in R2018a

The diagnostic message behavior of the evalfis function has changed. Previously, the evalfis function had the following behaviors for diagnostic conditions.

<table>
<thead>
<tr>
<th>Diagnostic Condition</th>
<th>Previous Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input values outside of their specified variable ranges</td>
<td>MATLAB warning</td>
</tr>
<tr>
<td>No rules fired for a given output at the current input values</td>
<td>MATLAB Command Window message</td>
</tr>
<tr>
<td>Empty output fuzzy sets</td>
<td>MATLAB Command Window message</td>
</tr>
</tbody>
</table>

Starting in R2018a, these diagnostic conditions are reported as MATLAB warnings by default. You can change this behavior by specifying the corresponding options in an evalfisOptions object.

**Update Code**

To disable the default warning messages, update your code to use an evalfisOptions object, and specify the diagnostic message options. For example, disable the empty output fuzzy set message.

```matlab
opt = evalfisOptions('EmptyOutputFuzzySetMessage','none');
output = evalfis(input,fis,opt);
```

**Intermediate fuzzy inference outputs for Sugeno systems are now analogous to outputs for Mamdani systems**

Behavior changed in R2018a

When evaluating a Sugeno system using the following syntax, the intermediate fuzzy inference results are now analogous to the intermediate results for Mamdani systems.

```matlab
[output,fuzzifiedInputs,ruleOutputs,aggregatedOutput] = evalfis(input,fis);
```

For a Sugeno system:

- `ruleOutputs` now returns an array that contains the scalar output value for each rule; that is, the product of the rule firing strength and the rule output level.
• **aggregatedOutput** now returns the sum of all the rule output values for each output variable.

Previously, for a Sugeno fuzzy system:

• **ruleOutputs** returned an array that contained the output level for each rule.
• **aggregatedOutput** returned an array that contained the firing strength for each rule.

Starting in R2018a, if your code returns intermediate fuzzy inference results when evaluating a Sugeno system using **evalfis**, modify your code to use the new **ruleOutputs** and **aggregatedOutput** results.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

• All **evalfis** syntaxes are supported for code generation. However, **mamfis**, **sugfis**, and **fistree** objects are not supported. To use **evalfis** for code generation, you must convert your FIS objects into homogeneous structures using **getFISCodeGenerationData**.
• Unlike the Fuzzy Logic Controller block, **evalfis** does not support fixed-point data for simulation or code generation.
• When evaluating a fuzzy inference system in Simulink, it is recommended to not use **evalfis** or **evalfisOptions** within a MATLAB Function block. Instead, evaluate your fuzzy inference system using a Fuzzy Logic Controller block.

## See Also

**Functions**

**evalfisOptions** | **fistree** | **mamfis** | **sugfis**
Topics
“Fuzzy Inference Process” on page 1-28
“Build Fuzzy Systems at the Command Line” on page 2-41

Introduced before R2006a
evalmf

Evaluate fuzzy membership function

Syntax

\[ y = \text{evalmf}(\text{mfT1},x) \]
\[ [y_{\text{Upper}},y_{\text{Lower}}] = \text{evalmf}(\text{mfT2},x) \]

Description

\( y = \text{evalmf}(\text{mfT1},x) \) evaluates one or more type-1 membership functions based on the input values in \( x \), returning the membership function values.

\[ [y_{\text{Upper}},y_{\text{Lower}}] = \text{evalmf}(\text{mfT2},x) \] evaluates one or more type-2 membership function based on the input values in \( x \), returning both the upper and lower membership function values.

Examples

Evaluate Membership Function

Evaluate a generalized bell-shaped membership function across a range of input values from 0 through 10.

\( x = 0:0.1:10; \)
\( \text{mf} = \text{fismf}('\text{gbellmf}',[2 4 6]); \)
\( y = \text{evalmf}(	ext{mf},x); \)

Plot the evaluation.

\( \text{plot}(x,y) \)
\( \text{xlabel('gbellmf, } P = [2 4 6]' \)
Evaluate Multiple Membership Functions

Create a vector of three Gaussian membership functions.

\[
\text{mf} = \left[ \text{fismf("gaussmf",[0.9 2.5],\'Name\','low\'}\right); \\
\text{fismf("gaussmf",[0.9 5],\'Name\','medium\'}); \\
\text{fismf("gaussmf",[0.9 7.55],\'Name\','high\'})];
\]

Specify the input range over which to evaluate the membership functions.

\[
x = (-2:0.1:12)';
\]
Evaluate the membership functions.

\[ y = \text{evalmf}(\text{mf},x); \]

Plot the evaluation results.

```matlab
plot(x,y)
xlabel('Input (x)')
ylabel('Membership value (y)')
legend('low','medium','high')
```
Evaluate Type-2 Membership Function

Create a triangular type-2 membership function.

mf = fismftype2("trimf",[5 7 9],"LowerLag",0.3,"LowerScale",0.8);

Evaluate the membership function across a range of input values from 0 through 10.

x = 0:0.1:10;
[yUpper,yLower] = evalmf(mf,x);

Plot the evaluated upper and lower MFs.

plot(x,yUpper,x,yLower)
legend('Upper MF','Lower MF','Location','northwest')
xlabel('Input')
ylabel('Membership value')
**Input Arguments**

**mfT1 — Type-1 membership function**

`fismf` object | vector of `fismf` objects

Type-1 membership function, specified as a `fismf` object or a vector of such objects.

**x — Input value**

scalar | vector | 2-D matrix

Input value, specified as a scalar, vector, or 2-D matrix. If `mf` is a:
• Single fismf object, then you can specify x as a scalar, vector, or matrix
• Vector of fismf objects, then you can specify x as a scalar or vector

mfT2 — Type-2 membership function
fismftype2 object | array of fismftype2 objects

Type-2 membership function, specified as a fismftype2 object or a vector of such objects.

Output Arguments

y — Membership values for a type-1 membership function
scalar | vector | 2-D matrix

Membership value for a type-1 membership function, returned as a scalar, vector, or 2-D matrix. If mfT1 is a:

• Single fismf object, then y is a scalar, vector, or matrix with the same dimensions as x. Each element of y is the evaluated membership value for the corresponding element of x.
• Vector of fismf objects, then y is an M-by-N matrix, where M and N are the lengths of mfT1 and x, respectively. y(i,j) is the evaluated value of membership function mfT1(i) for input value x(j).

yUpper — Upper MF membership values for a type-2 membership function
scalar | vector | 2-D matrix

Upper MF membership value for a type-2 membership function, returned as a scalar, vector, or 2-D matrix. If mfT2 is a:

• Single fismftype2 object, then y is a scalar, vector, or matrix with the same dimensions as x. Each element of y is the evaluated membership value for the corresponding element of x.
• Vector of fismftype2 objects, then y is an M-by-N matrix, where M and N are the lengths of mfT2 and x, respectively. y(i,j) is the evaluated value of membership function mfT2(i) for input value x(j).

yLower — Lower MF membership values for a type-2 membership function
scalar | vector | 2-D matrix
Lower MF membership value for a type-2 membership function, returned as a scalar, vector, or 2-D matrix. If \( \text{mfT2} \) is a:

- Single \( \text{fismftype2} \) object, then \( y \) is a scalar, vector, or matrix with the same dimensions as \( x \). Each element of \( y \) is the evaluated membership value for the corresponding element of \( x \).
- Vector of \( \text{fismftype2} \) objects, then \( y \) is an \( M \)-by-\( N \) matrix, where \( M \) and \( N \) are the lengths of \( \text{mfT2} \) and \( x \), respectively. \( y(i,j) \) is the evaluated value of membership function \( \text{mfT2}(i) \) for input value \( x(j) \).

**Compatibility Considerations**

**evalmf now takes a fismf object as an input argument**

*Behavior changed in R2018b*

`evalmf` now takes a `fismf` object as an input argument rather than the type and parameters of the membership function. Also, you can now evaluate multiple membership functions by passing an array of `fismf` objects to `evalmf`. There are differences between these approaches that require updates to your code.

**Update Code**

Previously, you evaluated a membership function for given input values, \( x \), by specifying the type of membership function, \( \text{type} \), and the membership functions parameters, \( \text{params} \).

\[
y = \text{evalmf}(x, \text{params}, \text{type});
\]

Update your code to first create a `fismf` object, \( \text{mf} \). Then, pass this object to `evalmf`.

\[
\text{mf} = \text{fismf}(\text{type}, \text{params});
y = \text{evalmf}(\text{mf}, x);
\]

Also, previously, to evaluate multiple membership functions you called `evalmf` once for each membership function.

\[
y1 = \text{evalmf}(x, \text{params1}, \text{type1});
y2 = \text{evalmf}(x, \text{params2}, \text{type2});
y3 = \text{evalmf}(x, \text{params3}, \text{type3});
\]
Now, you can evaluate multiple membership functions by passing an array of fismf objects to evalmf.

```matlab
mf1 = fismf(type1,params1);
mf2 = fismf(type2,params2);
mf3 = fismf(type3,params3);
y = evalmf([mf1 mf2 mf3],x);
```

Here, \(y = [y_1\ y_2\ y_3]'\);

**See Also**

fismf | fismftype2

**Topics**

“Foundations of Fuzzy Logic” on page 1-10

**Introduced before R2006a**
**fcm**

Fuzzy c-means clustering

**Syntax**

\[
[\text{centers,}U] = \text{fcm}(\text{data},N_c)
\]

\[
[\text{centers,}U] = \text{fcm}(\text{data},N_c,\text{options})
\]

\[
[\text{centers,}U,\text{objFunc}] = \text{fcm}(\_\_\_)
\]

**Description**

\([\text{centers,}U] = \text{fcm}(\text{data},N_c)\) performs fuzzy c-means clustering on the given data and returns \(N_c\) cluster centers.

\([\text{centers,}U] = \text{fcm}(\text{data},N_c,\text{options})\) specifies additional clustering options.

\([\text{centers,}U,\text{objFunc}] = \text{fcm}(\_\_\_)\) also returns the objective function values at each optimization iteration for all of the previous syntaxes.

**Examples**

**Cluster Data Using Fuzzy C-Means Clustering**

Load data.

```matlab
load fcmdata.dat
```

Find 2 clusters using fuzzy c-means clustering.

```matlab
[centers,U] = fcm(fcmdata,2);
Iteration count = 1, obj. fcn = 8.970479
Iteration count = 2, obj. fcn = 7.197402
Iteration count = 3, obj. fcn = 6.325579
```
Iteration count = 4, obj. fcn = 4.586142
Iteration count = 5, obj. fcn = 3.893114
Iteration count = 6, obj. fcn = 3.810804
Iteration count = 7, obj. fcn = 3.799801
Iteration count = 8, obj. fcn = 3.797862
Iteration count = 9, obj. fcn = 3.797508
Iteration count = 10, obj. fcn = 3.797444
Iteration count = 11, obj. fcn = 3.797432
Iteration count = 12, obj. fcn = 3.797430

Classify each data point into the cluster with the largest membership value.

```matlab
maxU = max(U);
index1 = find(U(1,:) == maxU);
index2 = find(U(2,:) == maxU);
```

Plot the clustered data and cluster centers.

```matlab
plot(fcmdata(index1,1),fcmdata(index1,2),'ob')
hold on
plot(fcmdata(index2,1),fcmdata(index2,2),'or')
plot(centers(1,1),centers(1,2),'xb','MarkerSize',15,'LineWidth',3)
plot(centers(2,1),centers(2,2),'xr','MarkerSize',15,'LineWidth',3)
hold off
```
Specify Fuzzy Overlap Between Clusters

Create a random data set.

```matlab
data = rand(100,2);
```

To increase the amount of fuzzy overlap between the clusters, specify a large fuzzy partition matrix exponent.

```matlab
options = [3.0 NaN NaN 0];
```

Cluster the data.
Configure Clustering Termination Conditions

Load the clustering data.

load clusterdemo.dat

Set the clustering termination conditions such that the optimization stops when either of the following occurs:

- The number of iterations reaches a maximum of 25.
- The objective function improves by less than 0.001 between two consecutive iterations.

options = [NaN 25 0.001 0];

The first option is NaN, which sets the fuzzy partition matrix exponent to its default value of 2. Setting the fourth option to 0 suppresses the objective function display.

Cluster the data.

[centers,U, objFun] = fcm(clusterdemo,3,options);

To determine which termination condition stopped the clustering, view the objective function vector.

objFun

objFun = 13×1

54.7257
42.9867
42.8554
42.1857
39.0857
31.6814
28.5736
27.1806
20.7359
15.7147
The optimization stopped because the objective function improved by less than 0.001 between the final two iterations.

## Input Arguments

**data — Data set to be clustered** 

*matrix*

Data set to be clustered, specified as a matrix with $N_d$ rows, where $N_d$ is the number of data points. The number of columns in `data` is equal to the data dimensionality.

**Nc — Number of clusters** 

*integer greater than 1*

Number of clusters to create, specified as an integer greater than 1.

**options — Clustering options** 

*vector*

Clustering options, specified as a vector with the following elements:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| options(1) | Exponent for the fuzzy partition matrix, $U$, specified as a scalar greater than 1.0. This option controls the amount of fuzzy overlap between clusters, with larger values indicating a greater degree of overlap.  
If your data set is wide with a lot of overlap between potential clusters, then the calculated cluster centers might be very close to each other. In this case, each data point has approximately the same degree of membership in all clusters. To improve your clustering results, decrease this value, which limits the amount of fuzzy overlap during clustering.  
For an example of fuzzy overlap adjustment, see “Adjust Fuzzy Overlap in Fuzzy C-Means Clustering” on page 4-8. | 2.0     |
Option | Description | Default
--- | --- | ---
option s(2) | Maximum number of iterations, specified as a positive integer. | 100
option s(3) | Minimum improvement in objective function between two consecutive iterations, specified as a positive scalar. | 1e-5
option s(4) | Information display flag indicating whether to display the objective function value after each iteration, specified as one of the following: • true — Display objective function. • false — Do not display objective function. | true

If any element of options is NaN, the default value for that option is used.

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 specified minimum.

Output Arguments

centers — Cluster centers
matrix
Final cluster centers, returned as a matrix with Nc rows containing the coordinates of each cluster center. The number of columns in centers is equal to the dimensionality of the data being clustered.

U — Fuzzy partition matrix
matrix
Fuzzy partition matrix, returned as a matrix with Nc rows and Nd columns. Element U(i,j) indicates the degree of membership of the jth data point in the ith cluster. For a given data point, the sum of the membership values for all clusters is one.

objFunc — Objective function values
vector
Objective function values for each iteration, returned as a vector.
Tips

- To generate a fuzzy inference system using FCM clustering, use the `genfis` command. For example, suppose you cluster your data using the following syntax:

  ```matlab
  [centers,U] = fcm(data,Nc,options);
  ```

  where the first M columns of `data` correspond to input variables, and the remaining columns correspond to output variables.

  You can generate a fuzzy system using the same training data and FCM clustering configuration. To do so:

  1. Configure clustering options.

     ```matlab
     opt = genfisOptions('FCMClustering');
     opt.NumClusters = Nc;
     opt.Exponent = options(1);
     opt.MaxNumIteration = options(2);
     opt.MinImprovement = options(3);
     opt.Verbose = options(4);
     ```

  2. Extract the input and output variable data.

     ```matlab
     inputData = data(:,1:M);
     outputData = data(:,M+1:end);
     ```

  3. Generate the FIS structure.

     ```matlab
     fis = genfis(inputData,outputData,opt);
     ```

     The fuzzy system, `fis`, contains one fuzzy rule for each cluster, and each input and output variable has one membership function per cluster. For more information, see `genfis` and `genfisOptions`.

Algorithms

Fuzzy c-means (FCM) is a clustering method that allows each data point to belong to multiple clusters with varying degrees of membership.

FCM is based on the minimization of the following objective function

\[
J_m = \sum_{i=1}^{D} \sum_{j=1}^{N} \mu_{ij}^m \|x_i - c_j\|^2,
\]
where

- $D$ is the number of data points.
- $N$ is the number of clusters.
- $m$ is fuzzy partition matrix exponent for controlling the degree of fuzzy overlap, with $m > 1$. Fuzzy overlap refers to how fuzzy the boundaries between clusters are, that is the number of data points that have significant membership in more than one cluster.
- $x_i$ is the $i$th data point.
- $c_j$ is the center of the $j$th cluster.
- $\mu_{ij}$ is the degree of membership of $x_i$ in the $j$th cluster. For a given data point, $x_i$, the sum of the membership values for all clusters is one.

fcm performs the following steps during clustering:

1. Randomly initialize the cluster membership values, $\mu_{ij}$.
2. Calculate the cluster centers:

$$c_j = \frac{\sum_{i=1}^{D} \mu_{ij}^m x_i}{\sum_{i=1}^{D} \mu_{ij}^m}.$$ 

3. Update $\mu_{ij}$ according to the following:

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{N} \left( \frac{\|x_i - c_j\|^{2m-1}}{\|x_i - c_k\|^{2m-1}} \right)}.$$ 

4. Calculate the objective function, $J_m$.

5. Repeat steps 2-4 until $J_m$ improves by less than a specified minimum threshold or until after a specified maximum number of iterations.

References

See Also
findcluster | genfis

Topics
“Fuzzy Clustering” on page 4-2
“Cluster Quasi-Random Data Using Fuzzy C-Means Clustering” on page 4-4
“Adjust Fuzzy Overlap in Fuzzy C-Means Clustering” on page 4-8

Introduced before R2006a
findcluster

Open clustering tool

Syntax

findcluster
finscluster(fileName)

Description

findcluster opens a UI to implement either fuzzy c-means or fuzzy subtractive clustering. For more information on:

- Clustering methods, see “Fuzzy Clustering” on page 4-2
- Using the Clustering tool, see “Data Clustering Using Clustering Tool” on page 4-48

finscluster(fileName) opens the UI, loads the data set in the file specified by fileName, and plots the first two dimensions of the data.

Examples

Open Clustering Tool and Load Data Set
findcluster('clusterdemo.dat')

Input Arguments

fileName — Data file name
string | character vector

Data file name, specified as a string or character vector.
The data set file must have the extension .dat. Each line of the data set file contains one data point. For example, if you have 5-dimensional data with 100 data points, the file contains 100 lines, and each line contains five values.

**Tips**

- Using the Clustering tool, you can obtain only the computed cluster centers. To obtain additional information for:
  - Fuzzy c-means clustering, such as the fuzzy partition matrix, cluster the data using `fcm`.
  - Subtractive clustering, such as the range of influence in each data dimension, cluster the data using `subclust`.
- To use the same clustering data with either `fcm` or `subclust`, first load the data file into the MATLAB workspace. For example, at the MATLAB command line, type:

  ```matlab
  load clusterdemo.dat
  ```

**See Also**

`fcm` | `subclust`

**Topics**

“Fuzzy Clustering” on page 4-2
“Data Clustering Using Clustering Tool” on page 4-48

**Introduced before R2006a**
fuzarith

Perform fuzzy arithmetic

Syntax

C = fuzarith(X,A,B,operator)

Description

C = fuzarith(X,A,B,operator) returns the fuzzy set C, which is the result of applying the specified fuzzy operator to the fuzzy sets A and B. The operation is performed across the universe of discourse X

Examples

Perform Fuzzy Arithmetic

Specify Gaussian and trapezoidal membership functions.

N = 501;
minX = -20;
maxX = 20;
x = linspace(minX,maxX,N);
A = trapmf(x,[-10 -2 1 3]);
B = gaussmf(x,[2 5]);

Evaluate the sum, difference, product, and quotient of A and B.

Csum = fuzarith(x,A,B,'sum');
Csub = fuzarith(x,A,B,'sub');
Cprod = fuzarith(x,A,B,'prod');
Cdiv = fuzarith(x,A,B,'div');

Plot the addition and subtraction results.
Plot the multiplication and division results.

```
figure
subplot(2,1,1)
plot(x,A,'--',x,B,':',x,Csum,'c')
title('Fuzzy Addition, A+B')
legend('A','B','A+B')
subplot(2,1,2)
plot(x,A,'--',x,B,':',x,Csub,'c')
title('Fuzzy Subtraction, A-B')
legend('A','B','A-B')
```
Input Arguments

X — Universe of discourse

vector
Universe of discourse, specified as a vector.

**A — Input fuzzy set**

*vector*

Input fuzzy set, specified as a vector with the same length as X. Each element of A is the value of the fuzzy set for the corresponding value of X.

A must be a convex fuzzy set. For more information, see “Algorithms” on page 8-120.

**B — Input fuzzy set**

*vector*

Input fuzzy set, specified as a vector with the same length as X. Each element of B is the value of the fuzzy set for the corresponding value of X.

B must be a convex fuzzy set. For more information, see “Algorithms” on page 8-120.

**operator — Fuzzy arithmetic operator**

'\texttt{sum}' | '\texttt{sub}' | '\texttt{prod}' | '\texttt{div}'

Arithmetic operator, specified as one of the following:

- 'sum' — Fuzzy addition
- 'sub' — Fuzzy subtraction
- 'prod' — Fuzzy multiplication
- 'div' — Fuzzy division

For more information on fuzzy arithmetic operations, see “Algorithms” on page 8-120.

**Note** Fuzzy addition can generate the message "divide by zero". However, this warning does not affect the accuracy of fuzarith.

**Output Arguments**

**C — Output fuzzy set**

*column vector*

Output fuzzy set, returned as a column vector with length equal to the length of X.
## Algorithms

To perform fuzzy arithmetic operations, the fuzzy operands (input fuzzy sets A and B) must be convex fuzzy sets. A fuzzy set is convex if, for each pair of points \( x_1 \) and \( x_2 \) in the universe of discourse \( X \) and \( \lambda \in [0,1] \),

\[
\mu(\lambda x_1 + (1 - \lambda)x_2) \geq \min(\mu(x_1), \mu(x_2))
\]

An \( \alpha \)-cut of a fuzzy set is the region in the universe of discourse for which the fuzzy set has a specific membership value, \( \alpha \). For a convex fuzzy set, every \( \alpha \)-cut defines a continuous region in the universe of discourse.

\texttt{fuzarith} uses the continuous regions defined by the \( \alpha \)-cuts of fuzzy sets A and B to compute the corresponding \( \alpha \)-cut of the output fuzzy set C. To do so, \texttt{fuzarith} uses interval arithmetic.

The following table shows how to compute the left and right boundaries of the output interval. Here:

- \([A_L, A_R]\) is the interval defined by the \( \alpha \)-cut of fuzzy set A.
- \([B_L, B_R]\) is the interval defined by the \( \alpha \)-cut of fuzzy set B.
- \([C_L, C_R]\) is the interval defined by the \( \alpha \)-cut of fuzzy set C.

<table>
<thead>
<tr>
<th>Interval Arithmetic Operator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition: ( C = A + B )</td>
<td>( C_L = A_L + B_L )  ( C_R = A_R + B_R )</td>
</tr>
<tr>
<td>Subtraction: ( C = A - B )</td>
<td>( C_L = A_L - B_R )  ( C_R = A_R - B_L )</td>
</tr>
<tr>
<td>Multiplication: ( C = A \ast B )</td>
<td>( C_L = \min(A_L \cdot B_L, A_L \cdot B_R, A_R \cdot B_L, A_R \cdot B_R) ) ( C_R = \max(A_L \cdot B_L, A_L \cdot B_R, A_R \cdot B_L, A_R \cdot B_R) )</td>
</tr>
<tr>
<td>Division: ( C = A / B )</td>
<td>( C_L = \min\left(\frac{A_L}{B_L}, \frac{A_L}{B_R}, \frac{A_R}{B_L}, \frac{A_R}{B_R}\right) ) ( C_R = \max\left(\frac{A_L}{B_L}, \frac{A_L}{B_R}, \frac{A_R}{B_L}, \frac{A_R}{B_R}\right) )</td>
</tr>
</tbody>
</table>
See Also

Topics
“What Is Fuzzy Logic?” on page 1-3
“Foundations of Fuzzy Logic” on page 1-10

Introduced before R2006a
gauss2mf

Gaussian combination membership function

Syntax

\[ y = \text{gauss2mf}(x, \text{params}) \]

Description

This function computes fuzzy membership values using a combination of two Gaussian membership functions. You can also compute this membership function using a \text{fismf} object. For more information, see “fismf Object” on page 8-124.

\[ y = \text{gauss2mf}(x, \text{params}) \] returns fuzzy membership values computed using a combination of two Gaussian membership functions computed. Each Gaussian function defines the shape of one side of the membership function and is given by:

\[ f(x; \sigma, c) = e^{-\frac{(x-c)^2}{2\sigma^2}} \]

To specify the standard deviation, \( \sigma \), and mean, \( c \), for each Gaussian function, use \text{params}.

Membership values are computed for each input value in \( x \).

Examples

Gaussian Combination Membership Functions

\[ x = [0:0.1:10]' \; \]
\[ y1 = \text{gauss2mf}(x,[2 \ 4 \ 1 \ 8]); \]
\[ y2 = \text{gauss2mf}(x,[2 \ 5 \ 1 \ 7]); \]
\[ y3 = \text{gauss2mf}(x,[2 \ 6 \ 1 \ 6]); \]
\[ y4 = \text{gauss2mf}(x,[2 \ 7 \ 1 \ 5]); \]
\[ y_5 = \text{gauss2mf}(x, \{2 \ 8 \ 1 \ 4\}); \]
\[ \text{plot}(x, \{y_1 \ y_2 \ y_3 \ y_4 \ y_5\}) \]

**Input Arguments**

\( x \) — Input values

scalar | vector

Input values for which to compute membership values, specified as a scalar or vector.

\( \text{params} \) — Membership function parameters

vector of length four
Membership function parameters, specified as the vector $[\sigma_1 \ c_1 \ \sigma_2 \ c_2]$. Here:

- $\sigma_1$ and $c_1$ are the standard deviation and mean of the left Gaussian function, respectively.
- $\sigma_2$ and $c_2$ are the standard deviation and mean of the right Gaussian function, respectively.

When $c_1 \leq c_2$, the \texttt{gauss2mf} function reaches a maximum value of 1 over the range $[c_1, c_2]$.

Otherwise, when $c_1 > c_2$, the maximum value is less than one.

**Output Arguments**

\texttt{y — Membership value}

\texttt{scalar | vector}

Membership value returned as a scalar or a vector. The dimensions of \texttt{y} match the dimensions of \texttt{x}. Each element of \texttt{y} is the membership value computed for the corresponding element of \texttt{x}.

**Alternative Functionality**

\texttt{fismf Object}

You can create and evaluate a \texttt{fismf} object that implements the \texttt{gauss2mf} membership function.

\begin{verbatim}
mf = fismf("gauss2mf",P);
Y = evalmf(mf,X);
\end{verbatim}

Here, \texttt{X}, \texttt{P}, and \texttt{Y} correspond to the \texttt{x}, \texttt{params}, and \texttt{y} arguments of \texttt{gauss2mf}, respectively.
Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
dsigmf | gaussmf | gbellmf | pimf | psigmf | sigmf | smf | trapmf | trimf | zmf

Topics
“Membership Functions” on page 1-14

Introduced before R2006a
gaussmf

Gaussian membership function

Syntax

\[ y = \text{gaussmf}(x, \text{params}) \]

Description

This function computes fuzzy membership values using a Gaussian membership function. You can also compute this membership function using a \textit{fismf} object. For more information, see “fismf Object” on page 8-128.

A Gaussian membership function is not the same as a Gaussian probability distribution. For example, a Gaussian membership function always has a maximum value of 1. For more information on Gaussian probability distributions, see “Normal Distribution” (Statistics and Machine Learning Toolbox).

\[ y = \text{gaussmf}(x, \text{params}) \] returns fuzzy membership values computed using the following Gaussian membership function:

\[
 f(x; \sigma, c) = e^{-\frac{(x - c)^2}{2\sigma^2}}
\]

To specify the standard deviation, \( \sigma \), and mean, \( c \), for the Gaussian function, use \text{params}.

Membership values are computed for each input value in \( x \).

Examples

Gaussian Membership Function

\[ x = 0:0.1:10; \]
\[ y = \text{gaussmf}(x,[2 5]); \]
plot(x,y)
xlabel('gaussmf, P=[2 5]')

**Input Arguments**

**x** — Input values

scalar | vector

Input values for which to compute membership values, specified as a scalar or vector.

**params** — Membership function parameters

vector of length two
Membership function parameters, specified as the vector $[\sigma \ c]$, where $\sigma$ is the standard deviation and $c$ is the mean.

**Output Arguments**

$y$ — Membership value
scalar | vector

Membership value returned as a scalar or a vector. The dimensions of $y$ match the dimensions of $x$. Each element of $y$ is the membership value computed for the corresponding element of $x$.

**Alternative Functionality**

**fismf Object**

You can create and evaluate a fismf object that implements the gaussmf membership function.

```matlab
mf = fismf("gaussmf",P);
Y = evalmf(mf,X);
```

Here, $X$, $P$, and $Y$ correspond to the $x$, $params$, and $y$ arguments of gaussmf, respectively.

**Extended Capabilities**

**C/C++ Code Generation**

Generate C and C++ code using MATLAB® Coder™.

**See Also**

dsigmf | gauss2mf | gbellmf | pimf | psigmf | sigmf | smf | trapmf | trimf | zmf
Topics
“Membership Functions” on page 1-14

Introduced before R2006a
gbellmf

Generalized bell-shaped membership function

Syntax

\[ y = \text{gbellmf}(x, \text{params}) \]

Description

This function computes fuzzy membership values using a generalized bell-shaped membership function. You can also compute this membership function using a fismf object. For more information, see “fismf Object” on page 8-132.

\[ y = \text{gbellmf}(x, \text{params}) \]

returns fuzzy membership values computed using the following generalized bell-shaped membership function:

\[ f(x; a, b, c) = \frac{1}{1 + \left| \frac{x-c}{a} \right|^{2b}} \]

To configure the membership function, specify parameters, \( a \), \( b \), and \( c \) using \( \text{params} \).

Membership values are computed for each input value in \( x \).

Examples

Generalized Bell-Shaped Membership Function

\[
x = 0:0.1:10;
y = \text{gbellmf}(x,[2 4 6]);
\text{plot}(x,y)
xlabel('gbellmf, P=[2 4 6]')
\]
Input Arguments

**x — Input values**
scalar | vector

Input values for which to compute membership values, specified as a scalar or vector.

**params — Membership function parameters**
vector of length two

Membership function parameters, specified as the vector \([a \ b \ c]\).
Here:

- $a$ defines the width of the membership function, where a larger value creates a wider membership function.
- $b$ defines the shape of the curve on either side of the central plateau, where a larger value creates a more steep transition.
- $c$ defines the center of the membership function.

**Output Arguments**

$y$ — Membership value
type: scalar | vector

Membership value returned as a scalar or a vector. The dimensions of $y$ match the dimensions of $x$. Each element of $y$ is the membership value computed for the corresponding element of $x$.

**Alternative Functionality**

**fismf Object**

You can create and evaluate a fismf object that implements the gbellmf membership function.

```
mf = fismf("gbellmf",P);
Y = evalmf(mf,X);
```

Here, $X$, $P$, and $Y$ correspond to the $x$, $params$, and $y$ arguments of gbellmf, respectively.

**Extended Capabilities**

**C/C++ Code Generation**

Generate C and C++ code using MATLAB® Coder™.
See Also
dsigmf | gauss2mf | gaussmf | pimf | psigmf | sigmf | smf | trapmf | trimf | zmf

Topics
“Membership Functions” on page 1-14

Introduced before R2006a
genfis

Generate fuzzy inference system object from data

Syntax

\[
\text{fis} = \text{genfis}(\text{inputData}, \text{outputData}) \\
\text{fis} = \text{genfis}(\text{inputData}, \text{outputData}, \text{options})
\]

Description

\( \text{fis} = \text{genfis}(\text{inputData}, \text{outputData}) \) returns a single-output Sugeno fuzzy inference system (FIS) using a grid partition of the given input and output data.

\( \text{fis} = \text{genfis}(\text{inputData}, \text{outputData}, \text{options}) \) returns an FIS generated using the specified input/output data and options. You can generate fuzzy systems using grid partitioning, subtractive clustering, or fuzzy c-means (FCM) clustering.

Examples

Generate Fuzzy Inference System Using Default Options

Define training data.

\[
\text{inputData} = [\text{rand}(10,1) \ 10*\text{rand}(10,1)-5]; \\
\text{outputData} = \text{rand}(10,1);
\]

Generate a fuzzy inference system.

\[
\text{fis} = \text{genfis}(\text{inputData}, \text{outputData});
\]

The generated system, \( \text{fis} \), is created using grid partitioning with default options.
**Generate FIS Using Grid Partitioning**

Define training data.

```matlab
inputData = [rand(10,1) 10*rand(10,1)-5];
outputData = rand(10,1);
```

Create a default `genfisOptions` option set for grid partitioning.

```matlab
opt = genfisOptions('GridPartition');
```

Specify the following input membership functions for the generated FIS:

- 3 Gaussian membership functions for the first input variable
- 5 triangular membership functions for the second input variable

```matlab
opt.NumMembershipFunctions = [3 5];
opt.InputMembershipFunctionType = ['gaussmf' 'trimf'];
```

Generate the FIS.

```matlab
fis = genfis(inputData,outputData,opt);
```

Plot the input membership functions. Each input variable has the specified number and type of input membership functions, evenly distributed over their input range.

```matlab
[x,mf] = plotmf(fis,'input',1);
subplot(2,1,1)
plot(x,mf)
xlabel('input 1 (gaussmf)')
[x,mf] = plotmf(fis,'input',2);
subplot(2,1,2)
plot(x,mf)
xlabel('input 2 (trimf)')
```
Generate FIS Using Subtractive Clustering

Obtain input and output training data.

```matlab
load clusterdemo.dat
inputData = clusterdemo(:,1:2);
outputData = clusterdemo(:,3);
```

Create a `genfisOptions` option set and specify the range of influence for each data dimension. Specify `0.5` and `0.25` as the range of influence for the first and second input variables. Specify `0.3` as the range of influence for the output data.
opt = genfisOptions('SubtractiveClustering',...
     'ClusterInfluenceRange',[0.5 0.25 0.3]);

Generate the FIS.

fis = genfis(inputData,outputData,opt);

The generated FIS contains one rule for each cluster.

showrule(fis)
ans = 3x83 char array
   '1. If (in1 is in1cluster1) and (in2 is in2cluster1) then (out1 is out1cluster1) (1)
   '2. If (in1 is in1cluster2) and (in2 is in2cluster2) then (out1 is out1cluster2) (1)
   '3. If (in1 is in1cluster3) and (in2 is in2cluster3) then (out1 is out1cluster3) (1)

**Generate FIS Using FCM Clustering**

Obtain the input and output data.

load *clusterdemo.dat*
inputData = clusterdemo(:,1:2);
outputData = clusterdemo(:,3);

Create a `genfisOptions` option set for FCM Clustering, specifying a Mamdani FIS type.

opt = genfisOptions('FCMClustering','FISType','mamdani');

Specify the number of clusters.

opt.NumClusters = 3;

Suppress the display of iteration information to the Command Window.

opt.Verbose = 0;

Generate the FIS.

fis = genfis(inputData,outputData,opt);

The generated FIS contains one rule for each cluster.
showrule(fis)
ans = 3x83 char array
  '1. If (in1 is in1cluster1) and (in2 is in2cluster1) then (out1 is out1cluster1) (1)
  '2. If (in1 is in1cluster2) and (in2 is in2cluster2) then (out1 is out1cluster2) (1)
  '3. If (in1 is in1cluster3) and (in2 is in2cluster3) then (out1 is out1cluster3) (1)

Plot the input and output membership functions.

[x,mf] = plotmf(fis,'input',1);
subplot(3,1,1)
plot(x,mf)
xlabel('Membership Functions for Input 1')
[x,mf] = plotmf(fis,'input',2);
subplot(3,1,2)
plot(x,mf)
xlabel('Membership Functions for Input 2')
[x,mf] = plotmf(fis,'output',1);
subplot(3,1,3)
plot(x,mf)
xlabel('Membership Functions for Output')
Create Type-2 Fuzzy Inference System from Data

To create a type-2 FIS from input/output data, you must first create a type-1 FIS using `genfis`.

Load training data and generate a FIS using subtractive clustering.

```matlab
load clusterdemo.dat
inputData = clusterdemo(:,1:2);
outputData = clusterdemo(:,3);
opt = genfisOptions('SubtractiveClustering',...
'ClusterInfluenceRange',[0.5 0.25 0.3]);
fisT1 = genfis(inputData,outputData,opt);
fisT1.Outputs

ans =
  fisvar with properties:
    Name: "out1"
    Range: [-0.1274 1.1458]
    MembershipFunctions: [1x3 fismf]

Convert the generated FIS to a type-2 FIS.

fisT2 = convertToType2(fisT1);

Since the initial type-1 FIS is a Sugeno system, only the input MFs are converted to type-2 MFs.

**Input Arguments**

**inputData — Input data**
array

Input data, specified as an N-column array, where N is the number of FIS inputs.

**outputData — Output data**
array

Output data, specified as an M-column array, where M is the number of FIS outputs.

When using grid partitioning, outputData must have one column. If you specify more than one column for grid partitioning, genfis uses the first column as the output data.

**options — FIS generation options**

genfisOptions option set
FIS generation options, specified as a `genfisOptions` option set. If you do not specify options, `genfis` uses a default grid partitioning option set.

You can generate fuzzy systems using one of the following methods, which you specify when you create the option set:

- Grid partitioning — Generate input membership functions by uniformly partitioning the input variable ranges, and create a single-output Sugeno fuzzy system. The fuzzy rule base contains one rule for each input membership function combination.
  
  ```
  options = genfisOptions('GridPartition');
  ```

- Subtractive clustering — Generate a Sugeno fuzzy system using membership functions and rules derived from data clusters found using subtractive clustering of input and output data. For more information on subtractive clustering, see `subclust`.
  
  ```
  options = genfisOptions('SubtractiveClustering');
  ```

- FCM Clustering — Generate a fuzzy system using membership function and rules derived from data clusters found using FCM clustering of input and output data. For more information on FCM clustering, see `fcm`.
  
  ```
  options = genfisOptions('FCMClustering');
  ```

### Output Arguments

- **fis** — Fuzzy inference system
  
  ```
  mamfis object | sugfis object
  ```

  Fuzzy inference system, returned as a `mamfis` or `sugfis` object. The properties of fis depend on the type of clustering used and the corresponding options.
### Clustering Type

<table>
<thead>
<tr>
<th>Clustering Type</th>
<th>Fuzzy System Type</th>
<th>Input Membership Functions</th>
<th>Fuzzy Rules</th>
<th>Output Membership Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Partitioning</td>
<td>Sugenno</td>
<td>Each input variable has evenly distributed input membership function. Specify the number of membership functions using <code>options.NumMembershipFunctions</code>. Specify the membership function type using <code>options.InputMembershipFunctionType</code>.</td>
<td>One rule for each input membership function combination. The consequent of each rule corresponds to a different output membership function.</td>
<td>One output membership function for each fuzzy rule. Specify the membership function type using <code>options.OutputMembershipFunctionType</code>.</td>
</tr>
<tr>
<td>Subtractive Clustering</td>
<td>Sugenno</td>
<td>Each input variable has one 'gaussmf' input membership function for each fuzzy cluster.</td>
<td>One rule for each fuzzy cluster</td>
<td>Each output variable has one 'linear' output membership function for each fuzzy cluster.</td>
</tr>
<tr>
<td>FCM Clustering</td>
<td>Mamdani</td>
<td>Each input variable has one 'gaussmf' input membership function for each fuzzy cluster.</td>
<td>One rule for each fuzzy cluster</td>
<td>Each output variable has one output membership function for each fuzzy cluster. The membership function type is 'gaussmf' for Mamdani systems and 'linear' for Sugeno systems.</td>
</tr>
</tbody>
</table>

If `fis` is a single-output Sugeno system, you can tune the membership function parameters using the `anfis` function.

Generating a type-2 FIS is not supported by `genfis`. Instead, generating a type-1 FIS and convert it using the `convertToType2` function.
Compatibility Considerations

Support for representing fuzzy inference systems as structures will be removed

Warns starting in R2019b

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

See Also

`anfis` | `fcm` | `genfisOptions` | `subclust`

Introduced in R2017a
**genfis1**

(To be removed) Generate Fuzzy Inference System structure from data using grid partition

**Note** genfis1 will be removed in a future release. Use genfis instead. For more information, see “Compatibility Considerations”.

**Syntax**

```matlab
fismat = genfis1(data)
```

```matlab
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) generates a single-output Sugeno-type fuzzy inference system using a grid partition on the data.

genfis1(data,numMFs,inmftype,outmftype) generates an FIS structure from a training data set, data, with the number and type of input membership functions and the type of output membership functions explicitly specified.

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 specify numMFs as a single number.
- **inmftype** is a character array in which each row specifies the membership function type associated with each input. This can be a character vector if the type of membership functions associated with each input is the same.
outmftype is a character vector 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.

The following table summarizes the default inference methods.

<table>
<thead>
<tr>
<th>Inference Method</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>prod</td>
</tr>
<tr>
<td>OR</td>
<td>max</td>
</tr>
<tr>
<td>Implication</td>
<td>prod</td>
</tr>
<tr>
<td>Aggregation</td>
<td>max</td>
</tr>
<tr>
<td>Defuzzification</td>
<td>wtaver</td>
</tr>
</tbody>
</table>

**Examples**

**Generate FIS Using Grid Partitioning**

Generate a FIS using grid partitioning.

```matlab
data = [rand(10,1) 10*rand(10,1)-5 rand(10,1)];
numMFs = [3 7];
mfType = char('pimf','trimf');
fismat = genfis1(data,numMFs,mfType);
```

To see the contents of fismat, use showfis(fismat).

Plot the FIS input membership functions.

```matlab
[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)')

**Compatibility Considerations**

**genfis1 will be removed**

*Not recommended starting in R2017a*

genfis1 will be removed in a future release. Use genfis instead. There are differences between these functions that require updates to your code.

**Update Code**

To generate a fuzzy system using grid partitioning, first create a default genfisOptions set.

```matlab
opt = genfisOptions('GridPartition');
```

You can modify the options using dot notation. Any options you do not modify remain at their default values.

Then, update your code to use genfis. For example, if your code has the following form:

```matlab
fis = genfis1(data,numMFs,inmftype,outmftype);
```

Use the following code instead:

```matlab
opt = genfisOptions('GridPartition');
opt.NumMembershipFunctions = numMFs;
opt.InputMembershipFunctionType = inmftype;
opt.OutputMembershipFunctionType = outmftype;
inputData = data(:,end-1);
outputData = data(:,end);
fis = genfis(inputData,outputData,opt);
```

**See Also**

anfis | genfis | genfis2 | genfis3
Introduced before R2006a
genfis2

(To be removed) Generate Fuzzy Inference System structure from data using subtractive clustering

**Note** genfis2 will be removed in a future release. Use genfis instead. For more information, see “Compatibility Considerations”.

**Syntax**

\[
\text{fismat} = \text{genfis2}(Xin,Xout,radii) \\
\text{fismat} = \text{genfis2}(Xin,Xout,radii,xBounds) \\
\text{fismat} = \text{genfis2}(Xin,Xout,radii,xBounds,options) \\
\text{fismat} = \text{genfis2}(Xin,Xout,radii,xBounds,options,user\_centers)
\]

**Description**

genfis2 generates a Sugeno-type FIS structure using subtractive clustering and requires separate sets of input and output data as input arguments. When there is only one output, genfis2 may be used to generate an initial FIS for anfis training. 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 an 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., \( \text{Xin} \) has two columns and \( \text{Xout} \) has one column), \( \text{radii} = [0.5 \ 0.4 \ 0.3] \) specifies that the ranges of influence in the first, second, and third data dimensions (i.e., the first column of \( \text{Xin} \), the second column of \( \text{Xin} \), and the column of \( \text{Xout} \)) are 0.5, 0.4, and 0.3 times the width of the data space, respectively. If \( \text{radii} \) 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.

- \( \text{xBounds} \) is a 2-by-\( N \) optional matrix that specifies how to map the data in \( \text{Xin} \) and \( \text{Xout} \) into a unit hyperbox, where \( N \) is the data (row) dimension. The first row of \( \text{xBounds} \) 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, \( \text{xBounds} = [-10 \ 0 \ -1; 10 \ 50 \ 1] \) 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 \( \text{xBounds} \) is an empty matrix or not provided, then \( \text{xBounds} \) defaults to the minimum and maximum data values found in each data dimension.

- \( \text{options} \) is an optional vector for specifying algorithm parameters to override the default values. These parameters are explained in the help text for \text{subclust}. Default values are in place when this argument is not specified.

- \( \text{user_centers} \) is an optional matrix for specifying custom cluster centers. \( \text{user_centers} \) has a size of \( J \)-by-\( N \) where \( J \) is the number of clusters and \( N \) is the total number of inputs and outputs.

The input membership function type is 'gaussmf', and the output membership function type is 'linear'.

The following table summarizes the default inference methods.

<table>
<thead>
<tr>
<th>Inference Method</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>prod</td>
</tr>
<tr>
<td>OR</td>
<td>probor</td>
</tr>
<tr>
<td>Implication</td>
<td>prod</td>
</tr>
<tr>
<td>Aggregation</td>
<td>max</td>
</tr>
<tr>
<td>Defuzzification</td>
<td>wtaver</td>
</tr>
</tbody>
</table>
Examples

Specify One Cluster Center Range of Influence For All Data Dimensions

Generate an FIS using subtractive clustering, and specify the cluster center range of influence.

\[
Xin = [7\times rand(50,1) \ 20\times rand(50,1) - 10];
Xout = 5\times rand(50,1);
fismat = \text{genfis2}(Xin,Xout,0.5);
\]

fismat uses a range of influence of 0.5 for all data dimensions.

To see the contents of fismat, use \text{showfis}(fismat).

Plot the input membership functions.

\[
[x, mf] = \text{plotmf}(\text{fismat}, \text{'input'}, 1);
\]

To see the contents of fismat, use \text{showfis}(fismat).

Specify Cluster Center Range of Influence For Each Data Dimension

Suppose the input data has two columns, and the output data has one column. Specify 0.5 and 0.25 as the range of influence for the first and second input data columns. Specify 0.3 as the range of influence for the output data.

\[
Xin = [7\times rand(50,1) \ 20\times rand(50,1) - 10];
Xout = 5\times rand(50,1);
fismat = \text{genfis2}(Xin,Xout,[0.5 \ 0.25 \ 0.3]);
\]

Specify Data Hyperbox Scaling Range

Suppose the input data has two columns, and the output data has one column. Specify the scaling range for the inputs and outputs to normalize the data into the [0 1] range. The
ranges for the first and second input data columns and the output data are: [-10 +10], [-5 +5], and [0 20].

\[
\text{Xin} = [7*\text{rand}(50,1) \ 20*\text{rand}(50,1)-10];
\text{Xout} = 5*\text{rand}(50,1);
\text{fismat} = \text{genfis2}(\text{Xin},\text{Xout},0.5,[\text{-10} \ -5 \ 0;\text{10} \ 5 \ 20]);
\]

Here, the third input argument, 0.5, specifies the range of influence for all data dimensions. The fourth input argument specifies the scaling range for the input and output data.

**Compatibility Considerations**

**genfis2 will be removed**

*Not recommended starting in R2017a*

genfis2 will be removed in a future release. Use genfis instead. There are differences between these functions that require updates to your code.

**Update Code**

To generate a fuzzy system using grid partitioning, first create a default genfisOptions set.

\[
\text{opt} = \text{genfisOptions}('\text{SubtractiveClustering}');
\]

You can modify the options using dot notation. Any options you do not modify remain at their default values.

Then, update your code to use genfis. For example, if your code has the following form:

\[
\text{fis} = \text{genfis2(inputData,outputData,radii,xBounds,options,userCenters});
\]

Use the following code instead:

\[
\text{opt} = \text{genfisOptions}('\text{SubtractiveClustering}');
\text{opt.ClusterInfluenceRange} = \text{radii};
\text{opt.DataScale} = \text{xBounds};
\text{opt.SquashFactor} = \text{options}(1);
\text{opt.AcceptRatio} = \text{options}(2);
\text{opt.RejectRatio} = \text{options}(3);
\]
opt.Verbose = options(4);
opt.CustomClusterCenters = userCenters;
fis = genfis(inputData, outputData, opt);

See Also
anfis | genfis | genfis1 | genfis3 | subclust

Introduced before R2006a
**genfis3**

(To be removed) Generate Fuzzy Inference System structure from data using FCM clustering

**Note** genfis3 will be removed in a future release. Use genfis instead. For more information, see “Compatibility Considerations”.

**Syntax**

```matlab
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**

`genfis3` generates an FIS using fuzzy c-means (FCM) clustering by extracting a set of rules that models the data behavior. The function requires separate sets of input and output data as input arguments. When there is only one output, you can use `genfis3` to generate an initial FIS for `anfis` training. 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 an FIS structure of the specified type, where `type` is either 'mamdani' or 'sugeno'.

- `fismat = genfis3(Xin,Xout,type,cluster_n)` generates an 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'. When cluster_n is 'auto', the function uses the subclust algorithm with a radii of 0.5 and the minimum and maximum values of Xin and Xout as xBounds to find the number of clusters. See subclust for more information.

fismat = genfis3(Xin,Xout,type,cluster_n,fcmoptions) generates an FIS structure of the specified type and number of clusters and uses the specified fcmoptions for the FCM algorithm. If you omit fcmoptions, the function uses the default FCM values. See fcm for information about these parameters.

The input membership function type is 'gaussmf'. By default, the output membership function type is 'linear'. However, if you specify type as 'mamdani', then the output membership function type is 'gaussmf'.

The following table summarizes the default inference methods.

<table>
<thead>
<tr>
<th>Inference Method</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>prod</td>
</tr>
<tr>
<td>OR</td>
<td>probor</td>
</tr>
<tr>
<td>Implication</td>
<td>prod</td>
</tr>
<tr>
<td>Aggregation</td>
<td>sum</td>
</tr>
<tr>
<td>Defuzzification</td>
<td>wtaver</td>
</tr>
</tbody>
</table>

Examples

Generate Sugeno-Type FIS and Specify Number of Clusters

Obtain the input and output data.

Xin = [7*rand(50,1) 20*rand(50,1)-10];
Xout = 5*rand(50,1);

Generate a Sugeno-type FIS with 3 clusters.

opt = NaN(4,1);
opt(4) = 0;
fismat = genfis3(Xin,Xout,'sugeno',3,0);
The fourth input argument specifies the number of clusters. The fifth input argument, \texttt{opt}, specifies the options for the FCM algorithm. The NaN entries of \texttt{opt} specify default option values. \texttt{opt(4)} turns off the display of iteration information at the command line.

To see the contents of \texttt{fismat}, use \texttt{showfis(fismat)}.

Plot the input membership functions.

\begin{verbatim}
[x, mf] = plotmf(fismat, 'input', 1);
subplot(2,1,1), plot(x, mf)
xlabel('Membership Functions for Input 1')
[x, mf] = plotmf(fismat, 'input', 2);
subplot(2,1,2), plot(x, mf)
xlabel('Membership Functions for Input 2')
\end{verbatim}

\section*{Compatibility Considerations}

\textbf{genfis3 will be removed}

\textit{Not recommended starting in R2017a}

\texttt{genfis3} will be removed in a future release. Use \texttt{genfis} instead. There are differences between these functions that require updates to your code.

\section*{Update Code}

To generate a fuzzy system using grid partitioning, first create a default \texttt{genfisOptions} set.

\begin{verbatim}
opt = genfisOptions('FCMClustering');
\end{verbatim}

You can modify the options using dot notation. Any options you do not modify remain at their default values.

Then, update your code to use \texttt{genfis}. For example, if your code has the following form:

\begin{verbatim}
fis = genfis3(inputData, outputData, type, cluster_n, fcmoptions);
\end{verbatim}

Use the following code instead:

\begin{verbatim}
opt = genfisOptions('FCMClustering');
opt.FISType = type;
opt.NumClusters = cluster_n;
\end{verbatim}
opt.Exponent = fcmoptions(1);
opt.MaxNumIteration = fcmoptions(2);
opt.MinImprovement = fcmoptions(3);
opt.Verbose = fcmoptions(4);
fs = genfis(inputData,outputData,opt);

See Also
anfis | fcm | genfis | genfis1 | genfis2

Introduced before R2006a
genfisOptions

Option set for genfis command

Syntax

opt = genfisOptions(clusteringType)
opt = genfisOptions(clusteringType,Name,Value)

Description

opt = genfisOptions(clusteringType) creates a default option set for generating a fuzzy inference system structure using genfis. The option set, opt, contains different options that depend on the specified clustering algorithm, clusteringType. Use dot notation to modify this option set for your specific application. Options that you do not modify retain their default values.

opt = genfisOptions(clusteringType,Name,Value) creates an option set with options specified by one or more Name,Value pair arguments.

Examples

Specify Options for FIS Generation

Create a default option set for the grid partitioning generation method.

opt = genfisOptions('GridPartition');

Modify the options using dot notation. For example, specify 3 membership functions for the first input and 4 membership functions for the second input.

opt.NumMembershipFunctions = [3 4];

You can also specify options when creating the option set. For example, create an option set for FCM clustering using 4 clusters.
opt2 = genfisOptions('FCMClustering','NumClusters',4);

### Input Arguments

**clusteringType — Clustering method**

'GridPartition' | 'SubtractiveClustering' | 'FCMClustering'

Clustering method for defining membership functions and fuzzy rules, specified as one of the following:

- 'GridPartition' — Generate input membership functions by uniformly partitioning the input variable ranges, and create a single-output Sugeno fuzzy system. The fuzzy rule base contains one rule for each input membership function combination.
- 'SubtractiveClustering' — Generate a Sugeno fuzzy system using membership functions and rules derived from data clusters found using subtractive clustering of input and output data. For more information on subtractive clustering, see `subclust`.
- 'FCMClustering' — Generate a fuzzy system using membership function and rules derived from data clusters found using FCM clustering of input and output data. For more information on FCM clustering, see `fcm`.

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'InputMembershipFunctionType','trimf'` sets triangular input membership functions for the grid partitioning algorithm.

### Grid Partitioning Options

**NumMembershipFunctions — Number of input membership functions**

2 (default) | integer greater than 1 | vector of integers greater than 1

Number of input membership functions for each input variable, specified as the comma-separated pair consisting of 'NumMembershipFunctions' and one of the following:

- Integer greater than 1 — Specify the same number of membership functions for all inputs.
• Vector of integer greater than 1 with length equal to the number of inputs — Specify a different number of membership functions for each input.

**InputMembershipFunctionType** — Input membership function type

<table>
<thead>
<tr>
<th>Membership function type</th>
<th>Description</th>
<th>For more information</th>
</tr>
</thead>
<tbody>
<tr>
<td>'gbellmf'</td>
<td>Generalized bell-shaped membership function</td>
<td>gbellmf</td>
</tr>
<tr>
<td>'gaussmf'</td>
<td>Gaussian membership function</td>
<td>gaussmf</td>
</tr>
<tr>
<td>'gauss2mf'</td>
<td>Gaussian combination membership function</td>
<td>gauss2mf</td>
</tr>
<tr>
<td>'trimf'</td>
<td>Triangular membership function</td>
<td>trimf</td>
</tr>
<tr>
<td>'trapmf'</td>
<td>Trapezoidal membership function</td>
<td>trapmf</td>
</tr>
<tr>
<td>'sigmf'</td>
<td>Sigmoidal membership function</td>
<td>sigmf</td>
</tr>
<tr>
<td>'dsigmf'</td>
<td>Difference between two sigmoidal membership functions</td>
<td>dsigmf</td>
</tr>
<tr>
<td>'psigmf'</td>
<td>Product of two sigmoidal membership functions</td>
<td>psigmf</td>
</tr>
<tr>
<td>'zmf'</td>
<td>Z-shaped membership function</td>
<td>zmf</td>
</tr>
<tr>
<td>'pimf'</td>
<td>Pi-shaped membership function</td>
<td>pimf</td>
</tr>
<tr>
<td>'smf'</td>
<td>S-shaped membership function</td>
<td>smf</td>
</tr>
<tr>
<td>Membership function type</td>
<td>Description</td>
<td>For more information</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Character vector or string</td>
<td>Name of a custom membership function in the current working folder or on the MATLAB path</td>
<td>“Build Fuzzy Systems Using Custom Functions” on page 2-52</td>
</tr>
</tbody>
</table>

- Character array or string array — Specify a different membership function type for each input. For example, specify different membership functions for a three-input system:

  ```matlab
  ['gbellmf','gaussmf','trimf']
  ```

**OutputMembershipFunctionType — Output membership function type**

- `'linear'` (default) | `'constant'`

Output membership function type for a single-output Sugeno system, specified as the comma-separated pair consisting of `'OutputMembershipFunctionType'` and one of the following:

- `'linear'` — The output of each rule is a linear function of the input variables, scaled by the antecedent result value.
- `'constant'` — The output of each rule is a constant, scaled by the antecedent result value.

**Subtractive Clustering Options**

**ClusterInfluenceRange — Range of influence of the cluster center**

- `0.5` (default) | scalar value in the range `[0, 1]` | vector

Range of influence of the cluster center for each input and output assuming the data falls within a unit hyperbox, specified as the comma-separated pair consisting of `'ClusterInfluenceRange'` one of the following:

- Scalar value in the range `[0, 1]` — Use the same influence range for all inputs and outputs.
- Vector — Use different influence ranges for each input and output.

Specifying a smaller range of influence usually creates more and smaller data clusters, producing more fuzzy rules.

**DataScale — Data scale factors**

- `'auto'` (default) | 2-by-N array
Data scale factors for normalizing input and output data into a unit hyperbox, specified as the comma-separated pair consisting of 'DataScale' and a 2-by-\(N\) array, where \(N\) is the total number of inputs and outputs. Each column of DataScale specifies the minimum value in the first row and the maximum value in the second row for the corresponding input or output data set.

When DataScale is 'auto', the genfis command uses the actual minimum and maximum values in the data to be clustered.

**SquashFactor — Squash factor**

1.25 (default) | positive scalar

Squash factor for scaling the range of influence of cluster centers, specified as the comma-separated pair consisting of 'SquashFactor' and a positive scalar. A smaller squash factor reduces the potential for outlying points to be considered as part of a cluster, which usually creates more and smaller data clusters.

**AcceptRatio — Acceptance ratio**

0.5 (default) | scalar value in the range \([0, 1]\)

Acceptance ratio, defined as a fraction of the potential of the first cluster center, above which another data point is accepted as a cluster center, specified as the comma-separated pair consisting of 'AcceptRatio' and a scalar value in the range \([0, 1]\). The acceptance ratio must be greater than the rejection ratio.

**RejectRatio — Rejection ratio**

0.15 (default) | scalar value in the range \([0, 1]\)

Rejection ratio, defined as a fraction of the potential of the first cluster center, below which another data point is rejected as a cluster center, specified as the comma-separated pair consisting of 'RejectRatio' and a scalar value in the range \([0, 1]\). The rejection ratio must be less than acceptance ratio.

**Verbose — Information display flag**

false (default) | true

Information display flag indicating whether to display progress information during clustering, specified as the comma-separated pair consisting of 'Verbose' and one of the following:

- false — Do not display progress information.
- **true** — Display progress information.

**CustomClusterCenters** — Custom cluster centers

```
[] (default) | C-by-N array
```

Custom cluster centers, specified the comma-separated pair consisting of 'CustomClusterCenters' and as a C-by-N array, where C is the number of clusters and N is the total number of inputs and outputs.

**FCM Clustering Options**

**FISType** — Fuzzy inference system type

```
'sugeno' (default) | 'mamdani'
```

Fuzzy inference system type, specified as the comma-separated pair consisting of 'FISType' and one of the following:

- 'sugeno' — Sugeno-type fuzzy system
- 'mamdani' — Mamdani-type fuzzy system

For more information on the types of fuzzy inference systems, see “Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2.

**NumClusters** — Number of clusters

```
'auto' | integer greater than 1
```

Number of clusters to create, specified as the comma-separated pair consisting of 'NumClusters' and 'auto' or an integer greater than 1. When NumClusters is 'auto', the `genfis` command estimates the number of clusters using subtractive clustering with a cluster influence range of 0.5.

NumClusters determines the number of rules and membership functions in the generated FIS.

**Exponent** — Exponent for the fuzzy partition matrix

```
2.0 (default) | scalar greater than 1.0
```

Exponent for the fuzzy partition matrix, specified as the comma-separated pair consisting of 'Exponent' and a scalar greater than 1.0. This option controls the amount of fuzzy overlap between clusters, with larger values indicating a greater degree of overlap.

If your data set is wide with significant overlap between potential clusters, then the calculated cluster centers can be very close to each other. In this case, each data point
has approximately the same degree of membership in all clusters. To improve your clustering results, decrease this value, which limits the amount of fuzzy overlap during clustering.

For an example of fuzzy overlap adjustment, see “Adjust Fuzzy Overlap in Fuzzy C-Means Clustering” on page 4-8.

MaxNumIteration — Maximum number of iterations

100 (default) | positive integer

Maximum number of iterations, specified as the comma-separated pair consisting of 'MaxNumIteration' and a positive integer.

MinImprovement — Minimum improvement in objective function

1e-5 (default) | positive scalar

Minimum improvement in objective function between two consecutive iterations, specified as the comma-separated pair consisting of 'MinImprovement' and a positive scalar.

Verbose — Information display flag

true (default) | false

Information display flag indicating whether to display the objective function value after each iteration, specified as the comma-separated pair consisting of 'Verbose' and one of the following:

• true — Display objective function.
• false — Do not display objective function.

Output Arguments

opt — Option set for genfis command

genfisOptions option set

Option set for genfis command, returned as a genfisOptions option set. The options in the option set depend on the specified clusteringType.
See Also
fcm | genfis | subclust

Introduced in R2017a
**gensurf**

Generate fuzzy inference system output surface

**Syntax**

```matlab
gensurf(fis)
gensurf(fis,options)
[X,Y,Z] = gensurf(___)
```

**Description**

gensurf(fis) generates the output surface for the fuzzy inference system, fis, plotting the first output variable against the first two input variables. For fuzzy systems with more than two inputs, the remaining input variables use the midpoints of their respective ranges as reference values.

gensurf(fis,options) generates the output surface using the specified options. To generate a surface using different inputs or outputs, or to specify nondefault plotting options, use this syntax.

[X,Y,Z] = gensurf(____) returns the variables that define the output surface for any of the previous syntaxes and suppresses the surface plot.

**Examples**

**Generate FIS Output Surface**

Load a fuzzy inference system.

```matlab
fis = readfis('tipper');
```

This fuzzy system has two inputs and one output.

Generate the output surface for the system.
Load a fuzzy inference system with two inputs and two outputs.

```matlab
defis = readfis('mam22.fis');
```

Create a surface generation option set, specifying the second output as the output to plot. By default, this output is plotted against the first two input variables.

```matlab
opt = gensurfOptions('OutputIndex',2);
```

**Generate FIS Output Surface for Second Output**

Load a fuzzy inference system with two inputs and two outputs.

```matlab
defis = readfis('mam22.fis');
```

Create a surface generation option set, specifying the second output as the output to plot. By default, this output is plotted against the first two input variables.

```matlab
opt = gensurfOptions('OutputIndex',2);
```
Plot the surface, using the specified option set.

gensurf(fis,opt)

Specify Reference Inputs for Surface Plot

Load a fuzzy inference system with four inputs and one output.

fis = readfis('slbb.fis');

Create a default gensurfOptions option set.
opt = gensurfOptions;

Specify plotting options to:

- Plot the output against the second and third input variable.
- Use 20 grid points for both inputs.
- Fix the first and fourth inputs at -0.5 and 0.1 respectively. Set the reference values for the second and third inputs to NaN.

opt.InputIndex = [2 3];
opt.NumGridPoints = 20;
opt.ReferenceInputs = [-0.5 NaN NaN 0.1];

Plot the output surface.

gensurf(fis,opt)
Return Surface Values and Suppress Plot

Load a fuzzy inference system.

```matlab
fis = readfis('tipper');
```

Generate the output surface, returning the surface data.

```matlab
[X,Y,Z] = gensurf(fis);
```

The output values, Z, are the FIS output evaluated at the corresponding X and Y grid points.
Input Arguments

fis — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

• mamfis object — Mamdani fuzzy inference system
• sugfis object — Sugeno fuzzy inference system
• mamfistype2 object — Type-2 Mamdani fuzzy inference system
• sugfistype2 object — Type-2 Sugeno fuzzy inference system

options — Surface generation options
gensurfOptions option set

Surface generation options, specified as a gensurfOptions option set.

Output Arguments

X — Grid values for first input variable
array | column vector

Grid values for first input variable, returned as one of the following:

• M-by-N array, where N and M are the number of grid points for the first and second inputs, respectively; that is options.NumGridPoints = [N M]. Each column of X contains one grid point value, repeated for every row.
• P-element column vector, where P is the number of grid points specified for a single input variable; that is options.NumGridPoints = P. Each element of contains one grid point value. This case applies when fis has only one input variable.

Y — Grid values for second input variable
array | []

Grid values for second input variable, returned as one of the following:
- \( M \)-by-\( N \) array, where \( N \) and \( M \) are the number of grid points for the first and second inputs respectively; that is \( \text{options.NumGridPoints} = [N \ M] \). Each row of \( Y \) contains one grid point value, repeated for every column.

- \([]\) when you specify only one input variable; that is, if you specify \( \text{options.InputIndex} \) as an integer.

### Z — Surface output values

*array | vector*

Surface output values for the output variable of \( \text{fis} \) specified by \( \text{options.OutputIndex} \), returned as one of the following:

- \( M \)-by-\( N \) array, where \( N \) and \( M \) are the number of grid points for the first and second inputs respectively; that is \( \text{options.NumGridPoints} = [N \ M] \). Each element of \( Z \) is the value of the FIS output, evaluated at the corresponding \( X \) and \( Y \) input values. For example, for a two-input system:

\[
Z(i,j) = \text{evalfis}(\text{fis},[X(i,j) \ Y(i,j)]);
\]

- \( P \)-element column vector, where \( P \) is the number of grid points specified for a single input variable; that is \( \text{options.NumGridPoints} = P \). Each element of \( Z \) is the value of the FIS output evaluated at the corresponding \( X \) input value.

When computing the value of \( Z \), \( \text{gensurf} \) sets the values of any inputs not specified by \( \text{options.InputIndex} \) to their corresponding reference values, as specified in \( \text{options.ReferenceInputs} \).

### Compatibility Considerations

#### Support for representing fuzzy inference systems as structures will be removed

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use \( \text{mamfis} \) and \( \text{sugfis} \) objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.
Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

**See Also**
- `evalfis`
- `gensurfOptions`
- `surfview`

**Introduced before R2006a**
**gensurfOptions**

Option set for gensurf function

**Syntax**

```matlab
opt = gensurfOptions
opt = gensurfOptions(Name,Value)
```

**Description**

`opt = gensurfOptions` creates a default option set for generating a fuzzy inference system output surface using `gensurf`. Use dot notation to modify this option set for your specific application. Any options that you do not modify retain their default values.

`opt = gensurfOptions(Name,Value)` creates an option set with options specified by one or more `Name,Value` pair arguments.

**Examples**

**Specify Options for Generating Output Surface**

Create a default `gensurfOptions` option set.

```matlab
opt = gensurfOptions;
```

Specify options using dot notation. For example, for a two-input, three-output fuzzy system, specify options to:

- Plot the surface for the second output against the values of the first and third inputs.
- Specify a reference value of `0.25` for the second input variable.

```matlab
opt.OutputIndex = 2;
opt.InputIndex = [1 3];
opt.ReferenceInputs = [NaN 0.25 NaN];
```
Any values you do not specify remain at their default values.

You can also specify one or more options when creating the option set. For example, create an option set, specifying 25 grid points for both plotted input variables:

```matlab
opt2 = gensurfOptions('NumGridPoints',25);
```

### Input Arguments

#### Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1,..., NameN, ValueN.

Example: 'InputIndex',[2 3] plots the output against the second and third input variables using a 3-D surface plot.

**InputIndex — Indices of input variables**

'auto' (default) | positive integer less than or equal to the number of inputs | two-element vector of positive integers

Indices of input variables to plot the output against, specified as the comma-separated pair consisting of 'InputIndex' and one of the following:

- Positive integer less than or equal to the number of inputs — Plot the output against a single input using a 2-D plot.
- Two-element vector of positive integers — Plot the output against two input variables using a 3-D surface plot.

When InputIndex is 'auto', gensurf uses the first two input variables by default.

**OutputIndex — Index of output variable**

'auto' (default) | positive integer less than or equal to the number of outputs

Index of output variable to plot, specified as the comma-separated pair consisting of 'OutputIndex' and a positive integer less than or equal to the number of outputs.

When OutputIndex is 'auto', gensurf uses the first output variable by default.
NumGridPoints — Number of grid points to plot
15 (default) | integer greater than 1 | two-element vector of integers greater than 1

Number of grid points to plot, specified as the comma-separated pair consisting of 'NumGridPoints' and one of the following:

• Integer greater than 1 — Specify the number of grid points when using a single input variable, or the same number of grid points for both inputs when using two inputs variables.
• Two-element vector of integers greater than 1 — Specify a different number of grid points for each input variable.

If you specify InputIndex as an integer and NumGridPoints as a vector, then gensurf uses the first element of NumGridPoints as the number of grid points for the specified input variable.

To plot a smoother surface, increase the number of grid points.

ReferenceInputs — Reference values for input variables
'auto' (default) | vector

Reference values for input variables not shown in the surface plot, specified as the comma-separated pair consisting of 'ReferenceInputs' and a vector with length equal to the number of FIS inputs. Specify NaN for the inputs specified in InputIndex.

When ReferenceInputs is 'auto', gensurf uses the midpoint of the range of each unused variable as a reference value.

NumSamplePoints — Number of sample points
101 (default) | integer greater than 1

Number of sample points to use when evaluating membership functions over the output variable range, specified as the comma-separated pair consisting of 'NumSamplePoints' and an integer greater than 1. For more information on membership function evaluation, see evalfis.

Note NumSamplePoints is not used by Sugeno-type systems.
Output Arguments

opt — Option set for gensurf command
gensurfOptions option set

Option set for gensurf command, returned as a gensurfOptions option set.

See Also

evalfis | gensurf

Introduced in R2017a
getfis

(To be removed) Get fuzzy system properties

**Note** `getfis` will be removed in a future release. Access fuzzy inference system properties using dot notation instead. For more information, see “Compatibility Considerations”.

**Syntax**

```matlab
getfis(sys)
```

```matlab
fisInfo = getfis(sys)
fisInfo = getfis(sys,fisProperty)
```

```matlab
varInfo = getfis(sys,varType,varIndex)
varInfo = getfis(sys,varType,varIndex,varProperty)
```

```matlab
mfInfo = getfis(sys,varType,varIndex,'mf',mfIndex)
mfInfo = getfis(sys,varType,varIndex,'mf',mfIndex,mfProperty)
```

**Description**

`getfis(sys)` prints the properties of the specified fuzzy inference system, `sys`, to the Command Window.

`fisInfo = getfis(sys)` returns the properties of the specified fuzzy inference system.

`fisInfo = getfis(sys,fisProperty)` returns the value of the specified property of the fuzzy inference system.

`varInfo = getfis(sys,varType,varIndex)` returns the properties of the specified input or output variable of a fuzzy inference system.

`varInfo = getfis(sys,varType,varIndex,varProperty)` returns the value of the specified variable property.
mfInfo = getfis(sys,varType,varIndex,'mf',mfIndex) returns the properties of the specified membership function of an input or output variable.

mfInfo = getfis(sys,varType,varIndex,'mf',mfIndex,mfProperty) returns the value of the specified membership function property.

Examples

Display Properties of Fuzzy Inference System

Load a fuzzy inference system.

sys = readfis('tipper');

Display the system properties.

gtfis(sys)

    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

Obtain Fuzzy Inference System Properties

Load fuzzy system.

sys = readfis('tipper');
Obtain the system properties.

\[
\text{prop} = \text{getfis(sys)};
\]

To obtain the value of a given property, specify the property name. For example, obtain the type of the fuzzy system.

\[
\text{type} = \text{getfis(sys,'type')};
\]

**Obtain Variable Properties**

Load fuzzy system.

\[
\text{sys} = \text{readfis('tipper')};
\]

Obtain the properties of the first input variable.

\[
\text{prop} = \text{getfis(sys,'input',1)};
\]

To obtain the value of a given property, specify the property name. For example, obtain the range of the output variable.

\[
\text{range} = \text{getfis(sys,'output',1,'range')};
\]

**Obtain Membership Function Properties**

Load fuzzy system.

\[
\text{sys} = \text{readfis('tipper')};
\]

For the second input variable, obtain the properties of its first membership function.

\[
\text{prop} = \text{getfis(sys,'input',2,'mf',1)};
\]

To obtain the value of a given property, specify the property name. For example, obtain the parameters of the second membership function of the output variable.
params = getfis(sys,'output',1,'mf',2,'params');

**Input Arguments**

**sys — Fuzzy inference system**
FIS structure

Fuzzy inference system, specified as an FIS structure.

**fisProperty — Fuzzy inference system property**

`'name'` | `'type'` | `'numInputs'` | `'numOutputs'` | ...  
Fuzzy inference system property, specified as one of the following:

- `'name'` — FIS name
- `'type'` — FIS type
- `'numInputs'` — Number of inputs
- `'numOutputs'` — Number of outputs
- `'numRules'` — Number of fuzzy rules.
- `'andMethod'` — And method
- `'orMethod'` — Or method
- `'defuzzMethod'` — Defuzzification method
- `'impMethod'` — Implication method
- `'aggMethod'` — Aggregation method
- `'ruleList'` — List of fuzzy rules

**varType — Variable type**

`'input'` | `'output'`  
Variable type, specified as either `'input'` or `'output'`, for input and output variables, respectively.

**varIndex — Variable index**

positive integer

Variable index, specified as a positive integer.
**varProperty — Variable property**

'name' | 'range' | 'nummfs'

Variable property, specified as one of the following:

- 'name' — Variable name
- 'range' — Variable value range
- 'nummfs' — Number of membership functions

**mfIndex — Membership function index**

positive integer

Membership function index, specified as a positive integer.

**mfProperty — Membership function property**

'name' | 'type' | 'params'

Membership function property, specified as one of the following:

- 'name' — Membership function name
- 'type' — Membership function type
- 'params' — Membership function parameters

For more information on membership functions, see “Membership Functions” on page 1-14.

**Output Arguments**

**fisInfo — Fuzzy inference system information**

structure | character vector | nonnegative integer | array

Fuzzy inference system information, returned as a structure, character vector, nonnegative integer, or array, depending on the value of fisProperty.

If you do not specify fisProperty, then fisInfo is returned as a structure with the following fields.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>FIS name, returned as a character vector.</td>
</tr>
<tr>
<td>type</td>
<td>FIS type, returned as a character vector.</td>
</tr>
<tr>
<td>andMethod</td>
<td>AND fuzzy operator method, returned as a character vector.</td>
</tr>
<tr>
<td>orMethod</td>
<td>OR fuzzy operator method, returned as a character vector.</td>
</tr>
<tr>
<td>defuzzMethod</td>
<td>Defuzzification method, returned as a character vector.</td>
</tr>
<tr>
<td>impMethod</td>
<td>Implication method, returned as a character vector.</td>
</tr>
<tr>
<td>aggMethod</td>
<td>Aggregation method, returned as a character vector.</td>
</tr>
</tbody>
</table>

**input**

Input variable information, returned as a structure or structure array. Each input variable structure contains the following fields:

- name — Variable name
- range — Variable range
- mf — Membership function names

**output**

Output variable information, returned as a structure or structure array. Each output variable structure contains the following fields:

- name — Variable name
- range — Variable range
- mf — Membership function names

**rule**

Fuzzy rule list, returned as a structure or structure array. Each rule structure contains the following fields:

- antecedent — Input membership function indices
- consequent — Output membership function indices
- weight — Rule weight
- connection — Fuzzy operator: 1 (AND), 2 (OR)

Otherwise, the value of fisInfo depends on the value of fisProperty according to the following table.

<table>
<thead>
<tr>
<th>fisProperty</th>
<th>fisInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>FIS name, returned as a character vector.</td>
</tr>
<tr>
<td>fisProperty</td>
<td>fisInfo</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>'type'</td>
<td>FIS type, returned as one of the following:</td>
</tr>
<tr>
<td></td>
<td>• 'mamdani' — Mamdani-type fuzzy system</td>
</tr>
<tr>
<td></td>
<td>• 'sugeno' — Sugeno-type fuzzy system</td>
</tr>
<tr>
<td>'numinputs'</td>
<td>Number of input variables, returned as a nonnegative integer.</td>
</tr>
<tr>
<td>'numoutputs'</td>
<td>Number of output variables, returned as a nonnegative integer.</td>
</tr>
<tr>
<td>'numrules'</td>
<td>Number of fuzzy rules, returned as a nonnegative integer.</td>
</tr>
<tr>
<td>'andmethod'</td>
<td>AND fuzzy operator method, returned as one of the following:</td>
</tr>
<tr>
<td></td>
<td>• 'min' — Minimum of fuzzified input values</td>
</tr>
<tr>
<td></td>
<td>• 'prod' — Product of fuzzified input values</td>
</tr>
<tr>
<td></td>
<td>• Character vector — Name of a custom AND function in the current working folder or on the MATLAB path</td>
</tr>
<tr>
<td>'ormethod'</td>
<td>OR fuzzy operator method, returned as one of the following:</td>
</tr>
<tr>
<td></td>
<td>• 'max' — Maximum of fuzzified input values</td>
</tr>
<tr>
<td></td>
<td>• 'probor' — Probabilistic OR of fuzzified input values</td>
</tr>
<tr>
<td></td>
<td>• Character vector — Name of a custom OR function in the current working folder or on the MATLAB path</td>
</tr>
</tbody>
</table>
**defuzzmethod**
Defuzzification method for computing crisp output values, returned as one of the following for Mamdani systems:
- **'centroid'** — Centroid of the area under the output fuzzy set
- **'bisector'** — Bisector of the area under the output fuzzy set
- **'mom'** — Mean of the values for which the output fuzzy set is maximum
- **'lom'** — Largest value for which the output fuzzy set is maximum
- **'som'** — Smallest value for which the output fuzzy set is maximum

For Sugeno systems, specify the defuzzification method as one of the following:
- **'wtaver'** — Weighted average of all rule outputs
- **'wtsum'** — Weighted sum of all rule outputs

The defuzzification method can also be returned as a character vector that contains the name of a custom defuzzification function in the current working folder or on the MATLAB path.

**impmethod**
Implication method for computing consequent fuzzy set, returned as one of the following:
- **'min'** — Truncate the consequent membership function at the antecedent result value.
- **'prod'** — Scale the consequent membership function by the antecedent result value.
- Character vector — Name of a custom implication function in the current working folder or on the MATLAB path.
<table>
<thead>
<tr>
<th>fisProperty</th>
<th>fisInfo</th>
</tr>
</thead>
</table>
| 'aggmethod' | Aggregation method for combining rule consequents, returned as one of the following:  
  • 'max' — Maximum of consequent fuzzy sets  
  • 'sum' — Sum of consequent fuzzy sets  
  • 'probor' — Probabilistic OR of consequent fuzzy sets  
  • Character vector — Name of a custom aggregation function in the current working folder or on the MATLAB path. |
| 'rulelist'  | Fuzzy rule list, returned as an array. For each fuzzy rule, the rule list contains one row with the following columns:  
  • $N_u$ columns of input membership function indices, where $N_u$ is the number of inputs. If a given variable is not included in a rule, the corresponding column entry is 0. Negative values indicate a NOT operation.  
  • $N_y$ columns of output membership function indices, where $N_y$ is the number of outputs. If a given variable is not included in a rule, the corresponding column entry is 0. Negative values indicate a NOT operation.  
  • Rule weight  
  • Fuzzy operator: 1 (AND), 2 (OR) |

**varInfo — Variable information**  
structure | character vector | nonnegative integer | row vector of length 2  

Variable information, returned as a structure, nonnegative integer, character vector, or row vector, depending on the value of varProperty.

If you do not specify `varProperty`, then `varInfo` is returned as a structure with the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Variable name, returned as a character vector.</td>
</tr>
<tr>
<td>NumMFs</td>
<td>Number of membership functions, returned as a nonnegative integer.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mf1, mf2, ..., mfN</td>
<td>Membership function names, returned as character vectors. mfInfo contains one field for each membership function.</td>
</tr>
<tr>
<td>range</td>
<td>Variable range, returned as a row vector of length 2.</td>
</tr>
</tbody>
</table>

Otherwise, the value of `varInfo` depends on the value of `varProperty` according to the following table.

<table>
<thead>
<tr>
<th>varProperty</th>
<th>varInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>Variable name, returned as a character vector.</td>
</tr>
<tr>
<td>'nummfs'</td>
<td>Number of membership functions, returned as a nonnegative integer.</td>
</tr>
<tr>
<td>'range'</td>
<td>Variable range, returned as a row vector of length 2.</td>
</tr>
</tbody>
</table>

**mfInfo — Membership function information**

structure | character vector | row vector

Membership function information, returned as a structure, character vector, or row vector, depending on the value of `mfProperty`.

If you do not specify `mfProperty`, then `mfInfo` is returned as a structure with the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Membership function name, returned as a character vector.</td>
</tr>
<tr>
<td>Type</td>
<td>Membership function type, returned as a character vector.</td>
</tr>
<tr>
<td>params</td>
<td>Membership function parameters, returned as a row vector.</td>
</tr>
</tbody>
</table>

Otherwise, the value of `mfInfo` depends on the value of `mfProperty` according to the following table.

<table>
<thead>
<tr>
<th>mfProperty</th>
<th>mfInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>'name'</td>
<td>Membership function name, returned as a character vector.</td>
</tr>
<tr>
<td>'type'</td>
<td>Membership function type, returned as a character vector.</td>
</tr>
<tr>
<td>'params'</td>
<td>Membership function parameters, returned as a row vector.</td>
</tr>
</tbody>
</table>
For more information on membership function, see “Membership Functions” on page 1-14.

## Compatibility Considerations

**getfis will be removed**  
*Not recommended starting in R2018b*

`getfis` will be removed in a future release. Access fuzzy inference system properties using dot notation instead. There are differences between these approaches that require updates to your code.

### Update Code

This table shows some typical usages of `getfis` for accessing fuzzy inference system properties and how to update your code to use dot notation instead.

<table>
<thead>
<tr>
<th>If your code has this form:</th>
<th>Use this code instead:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get(fis,'andmethod')</code></td>
<td><code>fis.AndMethod</code></td>
</tr>
<tr>
<td><code>getfis(fis,'input',1)</code></td>
<td><code>fis.Inputs(1)</code></td>
</tr>
<tr>
<td><code>getfis(fis,'input',1,'name')</code></td>
<td><code>fis.Inputs(1).Name</code></td>
</tr>
<tr>
<td><code>getfis(fis,'input',2,'mf',1)</code></td>
<td><code>fis.Inputs(2).MembershipFunctions(1)</code></td>
</tr>
<tr>
<td><code>getfis(fis,'input',2,'mf',1, ... params)</code></td>
<td><code>fis.Inputs(2).MembershipFunctions(1).Parameters</code></td>
</tr>
</tbody>
</table>

Previously, fuzzy inference systems were represented as structures. Now, fuzzy inference systems are represented as objects. Fuzzy inference system object properties have different names than the corresponding structure fields. For more information on fuzzy inference system objects, see `mamfis` and `sugfis`.

### See Also

`setfis` | `showfis`

*Introduced before R2006a*
getTunableValues

Obtain values of tunable parameters from fuzzy inference system

Syntax

`paramvals = getTunableValues(fis,paramset)`

Description

`paramvals = getTunableValues(fis,paramset)` returns tunable parameter values of the fuzzy inference system `fis`. To specify the parameter values to return, use `paramset`.

Examples

Obtain Values of Tunable Parameters from FIS

Create a fuzzy inference system, and define the tunable parameter settings of inputs, outputs, and rules.

```matlab
fis = mamfis('NumInputs',2,'NumOutputs',1);
[in,out,rule] = getTunableSettings(fis);
```

Obtain tunable parameter values of the inputs, outputs, and rules of the fuzzy inference system.

```matlab
paramVals = getTunableValues(fis,[in;out;rule]);
```
Input Arguments

**fis** — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object | fistree object

Fuzzy inference system, specified as a mamfis, sugfis, mamfistype2, sugfistype2, or fistree object.

**paramset** — Tunable parameter settings
array

Tunable parameter settings, specified as an array of input, output, and rule parameter settings in the input FIS. To obtain these parameter settings, use the getTunableSettings function with the input fis.

paramset can be the input, output, or rule parameter settings, or any combination of these settings.

Output Arguments

**paramvals** — Tunable parameter values
array

Tunable parameter values, returned as an array. The order of the values in paramvals matches the order of the parameters in paramset.

You can modify these parameter values, and then set them in your FIS using setTunableValues.

See Also

getTunableSettings | mamfis | mamfistype2 | setTunableValues | sugfis | sugfistype2 | tunefis

Introduced in R2019a
getFISCodeGenerationData
Create homogeneous fuzzy inference system structure

Syntax

fisOut = getFISCodeGenerationData(fisIn)
fisOut = getFISCodeGenerationData(fisIn,'FuzzySetType',setType)

Description
To generate code for evaluating a fuzzy inference system using MATLAB Coder, you must convert your fuzzy inference system object into a homogeneous structure using getFISCodeGenerationData.

fisOut = getFISCodeGenerationData(fisIn) converts a type-1 fuzzy inference system fisIn into a homogeneous structure fisOut. fisIn can be a FIS object or the name of a .fis file.

fisOut = getFISCodeGenerationData(fisIn,'FuzzySetType',setType) specifies the type of membership functions used in fisIn.

Examples

Convert FIS Object into Homogeneous Structure
Create a fuzzy inference system. For this example, load a fuzzy system from a file.

fisObject = readfis('tipper');

Convert the resulting mamfis object into a homogeneous structure.

fisStructure = getFISCodeGenerationData(fisObject);

In this structure, if a field is a structure array, all the elements of that array are the same size. For example, consider the elements of input variable array fisStructure.input.
The `name` fields are character vectors of the same length. Also, even though the second input variable has only two membership functions, the `mf` fields both contain three membership function structures. The original number of membership functions for a given input variable is stored in the `origNumMF` field.

**Load Fuzzy Inference System from File into Homogeneous Structure**

Load the fuzzy inference system saved in the file `tipper.fis` into a homogeneous structure.

```matlab
fis = getFISCodeGenerationData('tipper.fis');
```

**Convert Type-2 FIS Object into Homogeneous Structure**

Create a type-2 fuzzy inference system. For this example, create a default FIS with three inputs and two outputs.

```matlab
fisObject = mamfistype2('NumInputs',3,'NumOutputs',2);
```

Convert the resulting `mamfistype2` object into a homogeneous structure.
fisStructure = getFISCodeGenerationData(fisObject,'FuzzySetType','type2');

Input Arguments

fisIn — Input fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object | string | character vector

Input fuzzy inference system, specified as one of the following:

- mamfis, sugfis, mamfistype2, or sugfistype2 object.
  getFISCodeGenerationData supports fuzzy inference system objects for simulation only.
- String or character vector specifying a .fis file in the current working folder or on the MATLAB path. getFISCodeGenerationData supports fuzzy inference system file names for both simulation and code generation.

If fisIn is either a mamfistype2 or sugfistype2 object, then you must specify the setType as "type2".

When getFISCodeGenerationData loads a fuzzy system that uses custom functions, it writes additional files to the current folder to support code generation for the custom functions.

setType — Type of membership functions
"type1" (default) | "type2"

Type of membership functions used in fisIn, specified as one of the following:

- "type1" — Type-1 membership functions
- "type2" — Type-2 membership functions

Output Arguments

fisOut — Output fuzzy inference system
homogeneous structure
Output fuzzy inference system, returned as a homogeneous structure. In the homogeneous structure, if a field is a structure array, all the elements of that array are the same size. For example, in the input variable array `fisOut.input`:

- Names of all the variables are character vectors of the same length.
- Lengths of the membership function arrays for all variables are the same.

For any character vectors or structure arrays that are padded to increase their lengths, the original lengths of these elements are saved within the structure.

The `fisOut` structure is different than the structure created using `convertToStruct`.

**Extended Capabilities**

**C/C++ Code Generation**

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `getFISCodeGenerationData` supports fuzzy inference system objects for simulation only. To generate code for `getFISCodeGenerationData`, specify the input fuzzy inference system using a file name.
- It is good practice to not use `getFISCodeGenerationData` within a MATLAB Function block. This function is a utility function for generating code for evaluating a fuzzy inference system using MATLAB Coder.

**See Also**

evalfis | evalfisOptions | mamfis | mamfistype2 | sugfis | sugfistype2

*Introduced in R2018b*
getTunableSettings

Obtain tunable settings from fuzzy inference system

Syntax

\[
\text{in} = \text{getTunableSettings}(\text{fis}) \\
[\text{~,out}] = \text{getTunableSettings}(\text{fis}) \\
[\text{~,~,rule}] = \text{getTunableSettings}(\text{fis}) \\
[\text{in, out, rule}] = \text{getTunableSettings}(\text{fis}) \\
[\text{___}] = \text{getTunableSettings}(\text{fis},'\text{AsymmetricLag}',\text{true})
\]

Description

\(\text{in} = \text{getTunableSettings}(\text{fis})\) returns tunable settings of input variables of the fuzzy system \(\text{fis}\).

\(\text{[~,out]} = \text{getTunableSettings}(\text{fis})\) returns tunable settings of output variables of the fuzzy system \(\text{fis}\).

\(\text{[~,~,rule]} = \text{getTunableSettings}(\text{fis})\) returns tunable settings of rules of the fuzzy system \(\text{fis}\).

\(\text{[in, out, rule]} = \text{getTunableSettings}(\text{fis})\) returns tunable settings of inputs, outputs, and rules of the fuzzy system \(\text{fis}\).

\(\text{[___]} = \text{getTunableSettings}(\text{fis},'\text{AsymmetricLag}',\text{true})\) returns tunable settings which allow asymmetric lower membership function lag values. This syntax is supported only when \(\text{fis}\) is a type-2 fuzzy inference system.

Examples

Obtain Tunable Settings from FIS

Create two fuzzy inference systems, and define the connection between the two.
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = sugfis('Name','fis2','NumInputs',2,'NumOutputs',1);
con = ["fis1/output1" "fis2/input1"];

Create a tree of fuzzy inference systems.

```
tree = fistree([fis1 fis2],con);
```

Obtain the tunable settings of inputs, outputs, and rules of the fuzzy inference system.

```
[in,out,rule] = getTunableSettings(tree)
```

```
in=4x1 object
  4x1 VariableSettings array with properties:
    Type
    VariableName
    MembershipFunctions
    FISName

out=2x1 object
  2x1 VariableSettings array with properties:
    Type
    VariableName
    MembershipFunctions
    FISName

rule=18x1 object
  16x1 RuleSettings array with properties:
    Index
    Antecedent
    Consequent
    FISName
```

You can use dot notation to specify tunable settings.

For the first membership function of input 1:

- do not tune parameter 1,
• set the minimum ranges of the last two parameters to 0,
• and set the maximum ranges of the last two parameters to 1.

\[
\text{in(1).MembershipFunctions(1).Parameters.Free(1) = false;}
\]
\[
\text{in(1).MembershipFunctions(1).Parameters.Minimum(2:end) = 0;}
\]
\[
\text{in(1).MembershipFunctions(1).Parameters.Maximum(2:end) = 1;}
\]

For the first rule:

• set input 1 membership function index non-tunable,
• allow NOT logic for input 2 membership function index,
• and do not ignore output 1 membership function index.

\[
\text{rule(1).Antecedent.Free(1) = false;}
\]
\[
\text{rule(1).Antecedent.AllowNot(2) = true;}
\]
\[
\text{rule(1).Consequent.AllowEmpty(1) = false;}
\]

**Obtain Tunable Settings of Input and Output Variables from FIS**

Create two fuzzy inference systems, and define the connection between the two.

\[
\text{fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);}
\]
\[
\text{fis2 = sugfis('Name','fis2','NumInputs',2,'NumOutputs',1);}
\]
\[
\text{con = ['fis1/output1' 'fis2/input1'];}
\]

Create a tree of fuzzy inference systems.

\[
\text{tree = fistree([fis1 fis2],con);}
\]

Obtain the tunable settings of input and output variables of the fuzzy inference system.

\[
\text{[in,out] = getTunableSettings(tree)}
\]

\[
in=4x1 object
\]

4x1 VariableSettings array with properties:

- Type
- VariableName
- MembershipFunctions
- FISName
Obtain Tunable Settings of Input and Output Variables from Type-2 FIS

Create a type-2 fuzzy inference system.

```matlab
fis = mamfistype2('Name','fis1','NumInputs',2,'NumOutputs',1);
```

Obtain the tunable settings of the input and output variables of the fuzzy inference system.

```matlab
[in,out] = getTunableSettings(fis);
```

You can use dot notation to specify the tunable settings of the membership functions of the input and output variables.

For the first membership function of input 1, set the first and third upper membership function parameters as tunable.

```matlab
in(1).MembershipFunctions(1).UpperParameters.Free = [1 0 1];
```
For the first membership function of input 2, set the tunable range of the lower membership function scale to be between 0.7 and 0.9.

\[
\text{in(2).MembershipFunctions(1).LowerScale.Minimum = 0.7;}
\]
\[
\text{in(2).MembershipFunctions(1).LowerScale.Maximum = 0.9;}
\]

For the first membership function of output 1, set the tunable range of the lower membership function lag to be between 0.1 and 0.4.

\[
\text{in(2).MembershipFunctions(1).LowerLag.Minimum = 0.1;}
\]
\[
\text{in(2).MembershipFunctions(1).LowerLag.Maximum = 0.4;}
\]

**Specify Tunability of Parameter Settings**

Create a fuzzy inference system, and define the tunable parameter settings of inputs, outputs, and rules.

Create a FIS, and obtain its tunable settings.

\[
\text{fis = mamfis("NumInputs",2,"NumOutputs",2);} \\
\text{[in,out,rule] = getTunableSettings(fis);} \\
\]

You can specify all the input variables, output variables, or rules as tunable or nontunable. For example, set all the output variable settings as nontunable.

\[
\text{out = setTunable(out,0);} \\
\]

You can set the tunability of individual variables or rules. For example, set the first input variable as nontunable.

\[
\text{in(1) = setTunable(in(1),0);} \\
\]

You can set individual membership functions as nontunable. For example, set the first membership function of input 2 as nontunable.

\[
\text{in(2).MembershipFunctions(1) = setTunable(in(2).MembershipFunctions(1),0);} \\
\]

You can also specify the tunability of a subset of variables or rules. For example, set the first two rules as nontunable.

\[
\text{rule(1:2) = setTunable(rule(1:2),0);} \\
\]
**Input Arguments**

*fis* — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object | fistree object

Fuzzy inference system, specified as a mamfis, sugfis, mamfistype2, sugfistype2, or fistree object. The fuzzy system can be a fuzzy inference system or network of interconnected fuzzy inference systems.

**Output Arguments**

**in** — Tunable settings of input variables
array of VariableSettings objects

Tunable settings for input variables, returned as an array of VariableSettings objects. Each VariableSettings object contains tunability settings for the input variable indicated by its FISName and VariableName properties.

Specify the tunability settings of the membership functions for this variable, using its MembershipFunctions property.

**out** — Tunable settings of output variables
array of VariableSettings objects

Tunable settings for input variables, returned as an array of VariableSettings objects. Each VariableSettings object contains tunability settings for the output variable indicated by its FISName and VariableName properties.

Specify the tunability settings of the membership functions for this variable, using its MembershipFunctions property.

**rule** — Tunable settings of rules
array of RuleSettings objects

Tunable settings for rules, returned as an array of RuleSettings object. Each RuleSettings object contain tunability settings for a rule from the FIS indicated by its FISName property.
Specify the tunability settings of the antecedent and consequent for this variable, using its Antecedent and Consequent properties, respectively.

**See Also**
RuleSettings | VariableSettings | getTunableValues | setTunable | setTunableValues | tunefis

**Introduced in R2019a**
mam2sug

(To be removed) Transform Mamdani fuzzy inference system into Sugeno fuzzy inference system

**Note** mam2sug will be removed in a future release. Use convertToSugeno instead. For more information, see “Compatibility Considerations”.

**Syntax**

sugFIS = mam2sug(mamFIS)

**Description**

sugFIS = mam2sug(mamFIS) transforms a Mamdani fuzzy inference system into a Sugeno fuzzy inference system.

**Examples**

**Transform Mamdani FIS into Sugeno FIS**

Load a Mamdani fuzzy inference system.

mam_fismat = readfis('mam22.fis');

Convert this system to a Sugeno fuzzy inference system.

sug_fismat = mam2sug(mam_fismat);

Plot the output surfaces for both fuzzy systems.

subplot(2,2,1)
gensurf(mam_fismat)
title('Mamdani system (Output 1)')
subplot(2,2,2)
gensurf(sug_fismat)
title('Sugeno system (Output 1)')
subplot(2,2,3)
gensurf(mam_fismat,gensurfOptions('OutputIndex',2))
title('Mamdani system (Output 2)')
subplot(2,2,4)
gensurf(sug_fismat,gensurfOptions('OutputIndex',2))
title('Sugeno system (Output 2)')
The output surfaces for both systems are similar.

**Input Arguments**

mamFIS — Mamdani fuzzy inference system

structure

Mamdani fuzzy inference system, specified as a structure. Construct `mamFIS` at the command line or using the Fuzzy Logic Designer. For more information, see “Build Fuzzy Systems at the Command Line” on page 2-41 and “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17.

**Output Arguments**

sugFIS — Sugeno fuzzy inference system

structure

Sugeno fuzzy inference system, returned as a structure. `sugFIS`:

- Has constant output membership functions, whose values correspond to the centroids of the output membership functions in `mamFIS`
- Uses the weighted-average defuzzification method
- Uses the product implication method
- Uses the sum aggregation method

The remaining properties of `sugFIS`, including the input membership functions and rule definitions remain unchanged from `mamFIS`.

**Tips**

- If you have a functioning Mamdani fuzzy inference system, consider using `mam2sug` to convert to a more computationally efficient Sugeno structure to improve performance.
- If `sugFIS` has a single output variable and you have appropriate measured input/output training data, you can tune the membership function parameters of `sugFIS` using `anfis`. 
Compatibility Considerations

mam2sug will be removed
Not recommended starting in R2018b

mam2sug will be removed in a future release. Use convertToSugeno instead. To update your code, change the function name from mam2sug to convertToSugeno. The syntaxes are equivalent.

See Also
Fuzzy Logic Designer | convertToSugeno

Topics
“Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2
“Build Fuzzy Systems at the Command Line” on page 2-41
“Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17

Introduced before R2006a
mf2mf

(To be removed) Translate parameters between membership functions

**Note** *mf2mf* will be removed in a future release. Convert membership functions using dot notation on *fismf* objects instead. For more information, see “Compatibility Considerations”.

## 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.

**Note** 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** — Parameters of the membership function you are transforming from, specified as a row vector.
- **inType** — Type of membership function you are transforming from.
- **outType** — Type of membership function you are transforming to.

You can specify **inType** and **outType** as any of the following membership functions types:
<table>
<thead>
<tr>
<th>Membership function type</th>
<th>Description</th>
<th>For more information</th>
</tr>
</thead>
<tbody>
<tr>
<td>'gbellmf'</td>
<td>Generalized bell-shaped membership function</td>
<td>gbellmf</td>
</tr>
<tr>
<td>'gaussmf'</td>
<td>Gaussian membership function</td>
<td>gaussmf</td>
</tr>
<tr>
<td>'gauss2mf'</td>
<td>Gaussian combination membership function</td>
<td>gauss2mf</td>
</tr>
<tr>
<td>'trimf'</td>
<td>Triangular membership function</td>
<td>trimf</td>
</tr>
<tr>
<td>'trapmf'</td>
<td>Trapezoidal membership function</td>
<td>trapmf</td>
</tr>
<tr>
<td>'sigmf'</td>
<td>Sigmoidal membership function</td>
<td>sigmf</td>
</tr>
<tr>
<td>'dsigmf'</td>
<td>Difference between two sigmoidal membership functions</td>
<td>dsigmf</td>
</tr>
<tr>
<td>'psigmf'</td>
<td>Product of two sigmoidal membership functions</td>
<td>psigmf</td>
</tr>
<tr>
<td>'zmf'</td>
<td>Z-shaped membership function</td>
<td>zmf</td>
</tr>
<tr>
<td>'pimf'</td>
<td>Pi-shaped membership function</td>
<td>pimf</td>
</tr>
<tr>
<td>'smf'</td>
<td>S-shaped membership function</td>
<td>smf</td>
</tr>
</tbody>
</table>

**Examples**

**Translate Parameters Between Membership Functions**

```matlab
x = 0:0.1:5;
mf1 = [1 2 3];
mf2 = mf2mf(mf1,'gbellmf','trimf');
plot(x,gbellmf(x,mf1),x,trimf(x,mf2))
legend('Generalized bell-shaped','Triangle-shaped','Location','South')
ylim([-0.05 1.05])
```
Compatibility Considerations

**mf2mf will be removed**

*Not recommended starting in R2018b*

mf2mf will be removed in a future release. Convert membership functions using dot notation on `fismf` objects instead. There are differences between these approaches that require updates to your code.
Update Code

Previously, to change the type of a membership function in a fuzzy inference system, you converted the parameters using `mf2mf`.

```matlab
fis = readfis('tipper');
oldType = fis.input(1).mf(1).type;
oldParams = fis.input(1).mf(1).params;
fis.input(1).mf(1).type = newType;
fis.input(1).mf(1).params = mf2mf(oldParams,oldType,newType);
```

Now, when you change the type of membership function, the parameters are converted automatically.

```matlab
fis = readfis('tipper');
fis.Inputs(1).MembershipFunctions(1).Type = newType;
```

Previously, membership functions were represented as structures within a fuzzy inference system structure. Now, membership functions are represented as `fismf` objects within `mamfis` and `sugfis` objects. For more information on fuzzy inference system objects, see `mamfis` and `sugfis`.

See Also

dsigmf | evalmf | gauss2mf | gaussmf | gbellmf | pimf | psigmf | sigmf | smf | trapmf | trapmf | trimf | trimf | zmf

Topics

“Membership Functions” on page 1-14
“The Membership Function Editor” on page 2-25

Introduced before R2006a
mfedit
Open Membership Function Editor

Syntax

mfedit
mfedit(fis)
mfedit(fileName)

Description

Using the Membership Function Editor, you specify the range of each input and output variables. Then, for each variable, you define the number of membership functions, the type of each membership function, and the membership function parameters.

The Fuzzy Logic Designer app consists of several interactive interfaces for creating a fuzzy inference system (FIS), including the Membership Function Editor. For more information on interactively creating fuzzy systems, see “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17.

mfedit opens the Membership Function Editor with no fuzzy inference system loaded.
mfedit(fis) opens the Membership Function Editor and loads the fuzzy inference system fis.
mfedit(fileName) opens the Membership Function Editor and loads a fuzzy inference system from the file specified by fileName.

Examples

Open Membership Function Editor

Load or create a fuzzy inference system object. For this example, load the fuzzy system from a file.
fis = readfis('tipper');

Open the Membership Function Editor for this fuzzy system.

mfedit(fis)
Input Arguments

fis — Fuzzy inference system
mamfis object | sugfis object

Fuzzy inference system, specified as either a mamfis or sugfis object in the MATLAB workspace.

fileName — File name
string | character vector

File name specified as a string or character vector with or without the .fis extension. This file must be in the current working directory or on the MATLAB path.

Compatibility Considerations

Support for representing fuzzy inference systems as structures will be removed
 Warns starting in R2019b

Support for representing fuzzy inference systems as structures will be removed in a future release. Use mamfis and sugfis objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either mamfis or sugfis objects.

To convert existing fuzzy inference system structures to objects, use the convertfis function.

See Also

Apps
Fuzzy Logic Designer
Functions
addMF | plotmf | ruleedit | ruleview | surfview

Topics
“Membership Functions” on page 1-14
“The Membership Function Editor” on page 2-25

Introduced before R2006a
newfis

(To be removed) Create new fuzzy inference system

Note newfis will be removed in a future release. Use mamfis or sugfis instead. For more information, see “Compatibility Considerations”.

Syntax

fis = newfis(name)
fis = newfis(name,Name,Value)

Description

fis = newfis(name) returns a default Mamdani fuzzy inference system with the specified name.

fis = newfis(name,Name,Value) returns a fuzzy inference system with properties specified using one or more Name,Value pair arguments.

Examples

Create Fuzzy Inference System

Create a default Mamdani fuzzy inference system with the name, 'fis'.

sys = newfis('fis')

sys = struct with fields:
    name: 'fis'
    type: 'mamdani'
    andMethod: 'min'
    orMethod: 'max'
    defuzzMethod: 'centroid'
    impMethod: 'min'
Create Sugeno Fuzzy Inference System

Create a default Sugeno fuzzy inference system with the name, 'fis'.

```matlab
sys = newfis('fis','FISType','sugeno')
```

Specify Implication Methods for New Fuzzy Inference System

Create a Mamdani fuzzy inference system that uses 'bisector' defuzzification and 'prod' implication.

```matlab
sys = newfis('fis','DefuzzificationMethod','bisector', ... 'ImplicationMethod','prod');
```

Input Arguments

- **name** — Fuzzy inference system name
  - character vector | string

  Fuzzy inference system name, specified as a character vector or string.
**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'OrMethod','probor'` configures the fuzzy OR operator as a probabilistic OR function.

**FISType — Fuzzy inference system type**

`'mamdani'` (default) | `'sugeno'`

Fuzzy inference system type, specified as one of the following:

- `'mamdani'` — Mamdani-type fuzzy system
- `'sugeno'` — Sugeno-type fuzzy system

For more information on the types of fuzzy inference systems, see “Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2.

**AndMethod — AND fuzzy operator method**

`'min'` | `'prod'` | character vector | string

AND fuzzy operator method, specified as one of the following:

- `'min'` — Minimum of fuzzified input values. This method is the default when `FISType` is `'mamdani'`.
- `'prod'` — Product of fuzzified input values. This method is the default when `FISType` is `'sugeno'`.
- Character vector or string — Name of a custom AND function in the current working folder or on the MATLAB path. For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**OrMethod — OR fuzzy operator method**

`'max'` | `'probor'` | character vector | string

OR fuzzy operator method, specified as one of the following:
• 'max' — Maximum of fuzzified input values. This method is the default when \texttt{FISType} is 'mamdani'.
• 'probor' — Probabilistic OR of fuzzified input values. For more information, see \texttt{probor}. This method is the default when \texttt{FISType} is 'sugeno'.
• Character vector or string — Name of a custom OR function in the current working folder or on the MATLAB path. For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

\textbf{ImplicationMethod — Implication method}
\begin{verbatim}
'\texttt{min}' \mid '\texttt{prod}' \mid \texttt{character vector} \mid \texttt{string}
\end{verbatim}

Implication method for computing consequent fuzzy set, specified as one of the following:

• 'min' — Truncate the consequent membership function at the antecedent result value. This method is the default when \texttt{FISType} is 'mamdani'.
• 'prod' — Scale the consequent membership function by the antecedent result value. This method is the default when \texttt{FISType} is 'sugeno'.
• Character vector or string — Name of a custom implication function in the current working folder or on the MATLAB path. For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

\textbf{Note} No matter what implication method you specify, Sugeno systems always use 'prod' aggregation.

For more information on implication and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

\textbf{AggregationMethod — Aggregation method}
\begin{verbatim}
'\texttt{max}' \mid '\texttt{sum}' \mid \texttt{character vector} \mid \texttt{string}
\end{verbatim}

Aggregation method for combining rule consequents, specified as one of the following:

• 'max' — Maximum of consequent fuzzy sets. This method is the default when \texttt{FISType} is 'mamdani'.
• 'sum' — Sum of consequent fuzzy sets. This method is the default when \texttt{FISType} is 'sugeno'.
- ‘probor’ — Probabilistic OR of consequent fuzzy sets. For more information, see probor.
- Character vector or string — Name of a custom aggregation function in the current working folder or on the MATLAB path. For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

**Note** No matter what aggregation method you specify, Sugeno systems always use ‘sum’ aggregation.

For more information on aggregation and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**DefuzzificationMethod — Defuzzification method**

```
'centroid' | 'bisector' | 'mom' | 'lom' | 'som' | 'wtaver' | 'wtsum' | character vector | string
```

Defuzzification method for computing crisp output values.

If `FISType` is ‘mamdani’, specify the defuzzification method as one of the following:

- ‘centroid’ — Centroid of the area under the output fuzzy set. This method is the default for Mamdani systems.
- ‘bisector’ — Bisector of the area under the output fuzzy set
- ‘mom’ — Mean of the values for which the output fuzzy set is maximum
- ‘lom’ — Largest value for which the output fuzzy set is maximum
- ‘som’ — Smallest value for which the output fuzzy set is maximum

If `FISType` is ‘sugeno’, specify the defuzzification method as one of the following:

- ‘wtaver’ — Weighted average of all rule outputs. This method is the default for Sugeno systems.
- ‘wtsum’ — Weighted sum of all rule outputs

You can also specify the defuzzification method using a character vector or string that contains the name of a custom function in the current working folder or on the MATLAB path. For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.
For more information on defuzzification and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

Output Arguments

*`fis` — Fuzzy inference system*

FIS structure

Fuzzy inference system with the specified name, returned as an FIS structure. The fuzzy system is configured using the specified Name, Value pair arguments.

Fis has no input variables, output variables, or rules. To add variables or rules to fis, use `addvar` or `addRule`. You can also edit the fuzzy system using **Fuzzy Logic Designer**.

Compatibility Considerations

**newfis will be removed**

*Not recommended starting in R2018b*

newfis will be removed in a future release. Use `mamfis` or `sugfis` instead. There are differences between these functions that require updates to your code.

To create a Mamdani or Sugeno FIS, use `mamfis` or `sugfis`, respectively.

**Update Code**

This table shows some typical usages of `newfis` for creating fuzzy systems and how to update your code to use `mamfis` or `sugfis` instead.

<table>
<thead>
<tr>
<th>If your code has this form:</th>
<th>Use this code instead:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fis = newfis(name)</code></td>
<td><code>fis = mamfis('Name',name)</code></td>
</tr>
<tr>
<td><code>fis = newfis(name,'FISType','mamdani')</code></td>
<td><code>fis = mamfis('Name',name)</code></td>
</tr>
<tr>
<td><code>fis = newfis(name,'FISType','sugeno')</code></td>
<td><code>fis = sugfis('Name',name)</code></td>
</tr>
<tr>
<td>If your code has this form:</td>
<td>Use this code instead:</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>`fis = newfis(name,...</td>
<td>`fis = mamfis('Name',name,...</td>
</tr>
<tr>
<td>'FISType','mamdani',...</td>
<td>'AndMethod','prod')</td>
</tr>
<tr>
<td>'AndMethod','prod')</td>
<td></td>
</tr>
<tr>
<td>`fis = newfis(name,...</td>
<td>`fis = sugfis('Name',name,...</td>
</tr>
<tr>
<td>'FISType','sugeno',...</td>
<td>'OrMethod','probor')</td>
</tr>
<tr>
<td>'OrMethod','probor')</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

mamfis | readfis | sugfis | writeFIS

**Topics**

“Foundations of Fuzzy Logic” on page 1-10
“Fuzzy Inference Process” on page 1-28

**Introduced before R2006a**
parsrule

(To be removed) Parse fuzzy rules

Note parsrule will be removed in a future release. Use addRule instead. For more information, see “Compatibility Considerations”.

Syntax

outFIS = parsrule(inFIS,ruleList)
outFIS = parsrule(inFIS,ruleList,Name,Value)

Description

outFIS = parsrule(inFIS,ruleList) returns a fuzzy inference system, outFIS, that is equivalent to the input fuzzy system, inFIS. but with fuzzy rules replaced by the rules specified in ruleList.

outFIS = parsrule(inFIS,ruleList,Name,Value) parses the rules in ruleList using options specified by one or more Name,Value pair arguments.

Examples

Add Rules to Fuzzy Inference System

Load a fuzzy inference system (FIS).

fis = readfis('tipper');

Specify if-then rules using the default 'verbose' format.

rule1 = "If service is poor or food is rancid then tip is cheap";
rule2 = "If service is excellent and food is not rancid then tip is generous";
rules = [rule1 rule2];

Add the rules to the FIS.
fis2 = parsrule(fis, rules);

fis2 is equivalent to fis, except that the rule base is replaced with the specified rules.

### Add Rules Using Symbolic Expressions

Load a fuzzy inference system (FIS).

```matlab
gfis = readfis('tipper');
```

Specify the following rules using symbols:

- If service is poor or food is rancid then tip is cheap.
- If service is excellent and food is not rancid then tip is generous.

```matlab
rule1 = "service==poor | food==rancid => tip=cheap";
rul22 = "service==excellent & food~=rancid => tip=generous";
rules = [rule1 rule2];
```

Add the rules to the FIS using the 'symbolic' format.

```matlab
fis2 = parsrule(fis, rules, 'Format', 'symbolic');
```

### Add Rules Using Membership Function Indices

Load fuzzy inference system (FIS).

```matlab
fis = readfis('mam22.fis');
```

Specify the following rules using membership function indices:

- If angle is small and velocity is big, then force is negBig and force2 is posBig2.
- If angle is not small and velocity is small, then force is posSmall and force2 is negSmall2.

```matlab
rule1 = "1 2, 1 4 (1) : 1";
rul2 = "1 1, 3 2 (1) : 1";
rules = [rule1 rule2];
```

Add rules to FIS using the 'indexed' format.
fis2 = parsrule(fis, rules, 'Format', 'indexed');

**Add Rules Using French Language**

Load a fuzzy inference system (FIS).

fis = readfis('tipper');

Specify if-then rules using French keywords.

rule1 = "Si service est poor ou food est rancid alors tip est cheap";
rule2 = "Si service est excellent et food n''est pas rancid alors tip est generous";
rules = [rule1 rule2];

Add the rules to the FIS.

fis2 = parsrule(fis, rules, 'Language', 'francais');

**Add Single Rule to Fuzzy Inference System**

Load a fuzzy inference system (FIS).

a = readfis('tipper');

Add a rule to the FIS.

ruleTxt = 'If service is poor then tip is cheap';
a2 = parsrule(a, ruleTxt, 'verbose');

**Input Arguments**

- **inFIS — Fuzzy inference system**
  FIS structure

  Input fuzzy inference system, specified as an FIS structure. `parsrule` does not modify `inFIS`.

- **ruleList — Fuzzy rules**
  character array | string array | character vector | string
Fuzzy rules, specified as one of the following:

- Character array where each row corresponds to a rule. For example:
  ```matlab
  rule1 = 'If service is poor or food is rancid then tip is cheap';
  rule2 = 'If service is good then tip is average';
  rule3 = 'If service is excellent or food is delicious then tip is generous';
  ruleList = char(rule1, rule2, rule3);
  ```

- String array, where each element corresponds to a rule. For example:
  ```matlab
  ruleList = 
  "If service is poor or 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";
  ```

- Character vector or string to specify a single rule.

You can change the rule format and language using the `Format` and `Language` options.

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Format','symbolic'` sets the rule format to symbolic expressions.

#### Format — Rule format

`'verbose'` (default) | `'symbolic'` | `'indexed'`

Rule format, specified as the comma-separated pair consisting of `Format` and one of the following:

- `'verbose'` — Use linguistic expressions.
  
  `'If service is poor or food is rancid then tip is cheap 1'`

  Specify the rule weight at the end of the rule text. If you omit the weight, a default value of 1 is used.

  You can specify the rule language using the `Language` option.

- `'symbolic'` — Use language-neutral symbolic expressions.
  
  `'service==poor | food==rancid => tip=cheap 1'`

  Specify symbolic expressions using the following symbols.
### Rule Component Symbols

<table>
<thead>
<tr>
<th>Rule Component</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp;</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>IS (in antecedent)</td>
<td>==</td>
</tr>
<tr>
<td>IS (in consequent)</td>
<td>=</td>
</tr>
<tr>
<td>IS NOT</td>
<td>~=</td>
</tr>
<tr>
<td>Implication (then)</td>
<td>=&gt;</td>
</tr>
</tbody>
</table>

Specify the rule weight at the end of the rule text. If you omit the weight, a default value of 1 is used.

- **'indexed'** — Use input and output membership function (MF) indices.

Specify indexed rules in the following format:

'\langle input\ MFs\rangle, \langle output\ MFs\rangle, (\langle weight\rangle) : \langle logical\ operator\ -\ 1(AND),\ 2(OR)\rangle'

For example:

'1 1, 1 (1) : 2'

To indicate NOT operations for input and output membership functions, use negative indices. For example, to specify “not the second membership function,” use -2.

To indicate a don’t care condition for an input or output membership function, use 0.

#### Language — Rule language

- **'english'** (default) | **'francais'** | **'deutsch'**

Rule language for **'verbose'** format, specified as one of the following:

- **'english'** — Specify rules in English.
  'If service is poor or food is rancid then tip is cheap'
- **'francais'** — Specify rules in French.
  'Si service est poor ou food est rancid alors tip est cheap'
- **'deutsch'** — Specify rules in German.
  'Wenn service ist poor oder food ist rancid dann tip ist cheap'

The software parses the rules in `ruleList` using the following keywords.
<table>
<thead>
<tr>
<th>Rule Component</th>
<th>English</th>
<th>French</th>
<th>German</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start of antecedent</td>
<td><em>if</em></td>
<td><em>si</em></td>
<td><em>wenn</em></td>
</tr>
<tr>
<td>AND</td>
<td><em>and</em></td>
<td><em>et</em></td>
<td><em>und</em></td>
</tr>
<tr>
<td>OR</td>
<td><em>or</em></td>
<td><em>ou</em></td>
<td><em>oder</em></td>
</tr>
<tr>
<td>Start of consequent (implication)</td>
<td><em>then</em></td>
<td><em>alors</em></td>
<td><em>dann</em></td>
</tr>
<tr>
<td>IS</td>
<td><em>is</em></td>
<td><em>est</em></td>
<td><em>ist</em></td>
</tr>
<tr>
<td>IS NOT</td>
<td><em>is not</em></td>
<td><em>n''est_pas</em></td>
<td><em>ist nicht</em></td>
</tr>
</tbody>
</table>

**Output Arguments**

**outFIS — Output fuzzy inference system**

Fuzzy inference system, returned as an FIS structure. outFIS is the same as inFIS, except that the rule list contains only the rules specified in ruleList.

**Compatibility Considerations**

**parsrule will be removed**

*Not recommended starting in R2018b*

parsrule will be removed in a future release. Use addRule instead.

**Update Code**

If you previously added rules using linguistic or symbolic expressions with parsrule, you can specify rules using the same expressions with addRule. addRule automatically detects the format of the strings or character vectors in your rule list. Therefore, it is no longer necessary to specify the rule format. To add a rule list using addRule, use the following command:

```matlab
fis = addRule(fis,rules);
```

Previously, you could add rules using indexed expressions with parsrule.
rule1 = "1 2, 1 4 (1) : 1";
rule2 = "-1 1, 3 2 (1) : 1";
rules = [rule1 rule2];
fis = parsrule(fis,rules,'Format','indexed');

Now, specify these rules using arrays of indices.

rule1 = [1 2 1 4 1 1];
rule2 = [-1 1 3 2 1 1];
rules = [rule1; rule2];
fis = addRule(fis,rules);

If you previously specified rules using the 'Language' name-value pair argument with parsrule, this functionality has been removed and there is no replacement. Specify your rules using addRule a different rule format.

Previously, parsrule replaced the entire rule list in your fuzzy system. addRule appends your specified rules to the rule list.

**See Also**
addRule | ruleedit | showrule

**Introduced before R2006a**
**pimf**

Pi-shaped membership function

**Syntax**

\[ y = \text{pimf}(x, \text{params}) \]

**Description**

This function computes fuzzy membership values using a spline-based pi-shaped membership function. You can also compute this membership function using a `fismf` object. For more information, see "fismf Object" on page 8-229.

This membership function is related to the `smf` and `zmf` membership functions.

\[ y = \text{pimf}(x, \text{params}) \]

returns fuzzy membership values computed using a spline-based pi-shaped membership function. This membership function is the product of an `smf` function and a `zmf` function, and is given by:

\[
f(x; a, b, c, d) = \begin{cases} 
0, & x \leq a \\
2\left(\frac{x-a}{b-a}\right)^2, & a \leq x \leq \frac{a+b}{2} \\
1-2\left(\frac{x-b}{b-a}\right)^2, & \frac{a+b}{2} \leq x \leq b \\
1, & b \leq x \leq c \\
1-2\left(\frac{x-c}{d-c}\right)^2, & c \leq x \leq \frac{c+d}{2} \\
2\left(\frac{x-d}{d-c}\right)^2, & \frac{c+d}{2} \leq x \leq d \\
0, & x \geq d 
\end{cases}
\]

To specify the \( a, b, c, \) and \( d \) parameters, use `params`.

Membership values are computed for each input value in \( x \).
Examples

Pi-Shaped Membership Function

```matlab
x = 0:0.1:10;
y = pimf(x,[1 4 5 10]);
plot(x,y)
xlabel('pimf, P = [1 4 5 10]')
ylim([-0.05 1.05])
```
Input Arguments

\textbf{x} — Input values

\begin{verbatim}
scalar | vector
\end{verbatim}

Input values for which to compute membership values, specified as a scalar or vector.

\textbf{params} — Membership function parameters

\begin{verbatim}
vector of length two
\end{verbatim}

Membership function parameters, specified as the vector \([a \ b \ c \ d]\). Parameters \(a\) and \(d\) define the \textit{feet} of the membership function, and \(b\) and \(c\) define its \textit{shoulders}.

Output Arguments

\textbf{y} — Membership value

\begin{verbatim}
scalar | vector
\end{verbatim}

Membership value returned as a scalar or a vector. The dimensions of \(y\) match the dimensions of \(x\). Each element of \(y\) is the membership value computed for the corresponding element of \(x\).

Alternative Functionality

\textbf{fismf Object}

You can create and evaluate a \textit{fismf} object that implements the \textit{pimf} membership function.

\begin{verbatim}
mf = fismf("pimf",P);
Y = evalmf(mf,X);
\end{verbatim}

Here, \(X\), \(P\), and \(Y\) correspond to the \(x\), \(params\), and \(y\) arguments of \textit{pimf}, respectively.
Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

dsigmf | gauss2mf | gaussmf | gbellmf | psigmf | sigmf | smf | trapmf | trimf | zmf

Topics
"Membership Functions" on page 1-14

Introduced before R2006a
plotfis

Display fuzzy inference system

Syntax

plotfis(fis)

Description

plotfis(fis) displays a high-level diagram of a fuzzy inference system (FIS). The center of the display shows the name, type, and rule count for the FIS. The input variables with associated membership functions are displayed to the right, and the outputs with their associated membership functions are displayed on the left.

Examples

Display Fuzzy Inference System

Create a fuzzy inference system (FIS). For this example, read the FIS from the tipper.fis file.

fis = readfis('tipper');

Display the fuzzy system.

plotfis(fis)
Display Tree of Fuzzy Inference Systems

Create a `fistree` object from a pair of fuzzy inference systems.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = sugfis('Name','fis2','NumInputs',2,'NumOutputs',1);
con1 = ["fis1/output1" "fis2/input1"];
```
con2 = ["fis1/input1" "fis1/input2"];
tree = fistree([fis1 fis2],[con1; con2]);

Display the tree of fuzzy inference systems.

plotfis(tree)

FIS Names:
  fis1
  fis2

Connections:

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>fis1/output1</td>
<td>fis2/input1</td>
</tr>
<tr>
<td>fis1/input1</td>
<td>fis1/input2</td>
</tr>
</tbody>
</table>

Inputs:
  fis1/input1
  fis2/input2

Outputs:
  fis2/output1

For a fistree object, this function shows a description of the system in the Command Window instead of a figure.

**Input Arguments**

**fis** — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object | fistree object

Fuzzy inference system, specified as one of the following:

- **mamfis object** — Mamdani fuzzy inference system
- **sugfis object** — Sugeno fuzzy inference system
- **mamfistype2 object** — Type-2 Mamdani fuzzy inference system
- **sugfistype2 object** — Type-2 Sugeno fuzzy inference system
• fistree object — Tree of interconnected fuzzy inference systems

Compatibility Considerations

Support for representing fuzzy inference systems as structures will be removed
Warns starting in R2019b

Support for representing fuzzy inference systems as structures will be removed in a future release. Use mamfis and sugfis objects instead. There are differences between these representations that require updates to your code. These differences include:

• Object property names that differ from the corresponding structure fields.
• Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either mamfis or sugfis objects.

To convert existing fuzzy inference system structures to objects, use the convertfis function.

See Also
evalmf | fistree | mamfis | mamfistype2 | plotmf | readfis | sugfis | sugfistype2

Topics
“Build Fuzzy Systems at the Command Line” on page 2-41

Introduced before R2006a
**plotmf**

Plot membership functions for input or output variable

**Syntax**

```matlab
plotmf(fis,variableType,variableIndex)
plotmf(____,numPoints)
[xOut,mfOut] = plotmf(____)
[xOut,umfOut,lmfOut] = plotmf(____)
```

**Description**

`plotmf(fis,variableType,variableIndex)` plots the membership functions for an input or output variable in the fuzzy inference system `fis`.

`plotmf(____,numPoints)` specifies the number of data points to plot for each membership function.

```matlab
[xOut,mfOut] = plotmf(____)```

returns the universe of discourse (`xOut`) and membership function (`mfOut`) values without plotting them. Use this syntax when `fis` is a type-1 fuzzy inference system.

```matlab
[xOut,umfOut,lmfOut] = plotmf(____)`
```

returns the universe of discourse (`xOut`), upper membership function (`umfOut`), and lower membership function (`lmfOut`) values without plotting them. Use this syntax when `fis` is a type-2 fuzzy inference system.

**Examples**

**Plot Membership Functions for Input Variable**

Create a fuzzy inference system.
fis = readfis('tipper');

Plot the membership functions for the first input variable.

plotmf(fis, 'input', 1)

Specify Number of Points for Membership Function Plot

Create a fuzzy inference system.

fis = readfis('tipper');
Plot the membership functions for the first output variable using 101 data points for each membership function.

```matlab
plotmf(fis, 'output', 1, 101)
```

**Obtain Membership Function Plot Data**

Create a fuzzy inference system.

```matlab
fis = readfis('tipper');
```
Obtain the x-axis and y-axis data for the membership functions of the second input variable.

[xOut,yOut] = plotmf(fis,'input',2);

You can then, for example, plot a single membership function using this data.

plot(xOut(:,2),yOut(:,2))
xlabel('food')
ylabel('delicious membership')
Plot Membership Functions for Type-2 FIS

Create a type-2 fuzzy inference system.

```matlab
fis = mamfistype2('NumInputs',3,'NumOutputs',1);
```

Plot the membership functions for the second input variable.

```matlab
plotmf(fis,'input',1)
```

The type-2 membership functions have a footprint of uncertainty (FOU) between their upper and lower membership functions.

You can also obtain the plotting data without generating a plot.
[xOut,umfOut,lmfOut] = plotmf(fis,'input',1);

You can then plot individual membership functions or plot the data using your own custom formatting. For example, plot the upper and lower membership functions for only the second membership function of the first input variable.

plot(xOut(:,2),umfOut(:,2),'r',xOut(:,2),lmfOut(:,2),'b')
xlabel('input1')
ylabel('delicious membership')
**Input Arguments**

**fis** — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
- sugfistype2 object — Type-2 Sugeno fuzzy inference system

plotmf does not support plotting output membership functions of Sugeno systems.

**variableType** — Variable type
'input' | 'output'

Variable type, specified as one of the following:

- 'input' — Input variable
- 'output' — Output variable

**variableIndex** — Variable index
positive integer

Variable index, specified as a positive integer. If `variableType` is:

- 'input', then `variableIndex` must be less than or equal to the number of input variables in `fis`
- 'output', then `variableIndex` must be less than or equal to the number of output variables in `fis`

**numPoints** — Number of data points to plot
181 (default) | positive integer

Number of data points to plot, specified as a positive integer.
**Output Arguments**

**xOut** — Universe of discourse data

Array

Universe of discourse data, returned as a `numPoints`-by-$N_{MF}$ array, where $N_{MF}$ is the number of membership functions for the variable specified by `variableType` and `variableIndex`.

**mfOut** — Membership function data

Array

Membership function data for a type-1 membership function, returned as a `numPoints`-by-$N_{MF}$ array, where $N_{MF}$ is the number of membership functions for the variable specified by `variableType` and `variableIndex`.

**umfOut** — Upper membership function data

Array

Upper membership function data for a type-2 membership function, returned as a `numPoints`-by-$N_{MF}$ array, where $N_{MF}$ is the number of membership functions for the variable specified by `variableType` and `variableIndex`.

**lmfOut** — Lower membership function data

Array

Lower membership function data for a type-2 membership function, returned as a `numPoints`-by-$N_{MF}$ array, where $N_{MF}$ is the number of membership functions for the variable specified by `variableType` and `variableIndex`.

**Compatibility Considerations**

**Support for representing fuzzy inference systems as structures will be removed**

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:
• Object property names that differ from the corresponding structure fields.
• Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either *mamfis* or *sugfis* objects.

To convert existing fuzzy inference system structures to objects, use the *convertfis* function.

**See Also**

**Functions**

*evalmf* | *plotfis*

*Introduced before R2006a*
probor

Probabilistic OR

Syntax

\[ y = \text{probor}(x) \]

Description

\( y = \text{probor}(x) \) returns the probabilistic OR (also known as the \textit{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

Probabilistic OR

\[
\begin{align*}
x &= 0:0.1:10; \\
y1 &= \text{gaussmf}(x,[0.5 \ 4]); \\
y2 &= \text{gaussmf}(x,[2 \ 7]); \\
yy &= \text{probor}([y1;y2]); \\
\text{plot}(x,[y1;y2;yy]) \\
\text{legend}('y1','y2','yy') \\
\text{ylim}([-0.05 \ 1.05])
\end{align*}
\]
See Also

Introduced before R2006a
**psigmf**

Product of two sigmoidal membership functions

**Syntax**

\[ y = \text{psigmf}(x, \text{params}) \]

**Description**

This function computes fuzzy membership values using the product of two sigmoidal membership functions. You can also compute this membership function using a `fismf` object. For more information, see "fismf Object" on page 8-248.

This membership function is related to the `sigmf` and `dsigmf` membership functions.

\[ y = \text{psigmf}(x, \text{params}) \]

returns fuzzy membership values computed using the product of two sigmoidal membership functions. Each sigmoidal function is given by:

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

To specify the \( a \) and \( c \) parameters for each sigmoidal function, use `params`.

Membership values are computed for each input value in \( x \).

**Examples**

**Product of Two Sigmoidal Membership Functions**

\[ x = 0:0.1:10; \]
\[ y = \text{psigmf}(x, [2 3 -5 8]); \]
\[ \text{plot}(x,y) \]
\[ \text{xlabel('psigmf, P = [2 3 -5 8]')} \]
\[ \text{ylim([-0.05 1.05])} \]
**Input Arguments**

**x — Input values**

*scalar | vector*

Input values for which to compute membership values, specified as a scalar or vector.

**params — Membership function parameters**

*vector of length four*
Membership function parameters, specified as the vector \( [a_1 \; c_1 \; a_2 \; c_2] \). Here, \( a_1 \) and \( c_1 \) are the parameters of the first sigmoidal function, and \( a_2 \) and \( c_2 \) are the parameters of the second sigmoidal function.

For each sigmoidal function, to open the function to the left or right, specify a negative or positive value for \( a \), respectively. The magnitude of \( a \) defines the width of the transition area, and parameter \( c \) defines the center of the transition area.

To define a unimodal membership function with a maximum value of 1, specify opposite signs for \( a_1 \) and \( a_2 \), and select \( c \) values far enough apart to allow for both transition areas to reach 1.

**Output Arguments**

\( y \) — Membership value

scalar | vector

Membership value returned as a scalar or a vector. The dimensions of \( y \) match the dimensions of \( x \). Each element of \( y \) is the membership value computed for the corresponding element of \( x \).

**Alternative Functionality**

**fismf Object**

You can create and evaluate a fismf object that implements the psigmf membership function.

\[
\text{mf} = \text{fismf}("psigmf",P);
Y = \text{evalmf}(mf,X);
\]

Here, \( X, P, \) and \( Y \) correspond to the \( x, \) params, and \( y \) arguments of \text{psigmf}, respectively.
Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
dsigmf | gauss2mf | gaussmf | gbellmf | pimf | psigmf | sigmf | smf | trapmf | trimf | zmf

Topics
“Membership Functions” on page 1-14

Introduced before R2006a
readfis

Load fuzzy inference system from file

**Syntax**

```matlab
fis = readfis(fileName)
fis = readfis
```

**Description**

You can load a fuzzy inference system (FIS) from a .fis file using the `readfis` function. To save a FIS to a file, use the `writeFIS` function.

**Note** Do not manually edit the contents of a .fis file. Doing so can produce unexpected results when loading the file using `readfis`.

```matlab
fis = readfis(fileName) reads a FIS from the file specified by fileName.
```

```matlab
fis = readfis opens a dialog box for selecting and reading a .fis file.
```

**Examples**

**Load Fuzzy Inference System from File**

Load the fuzzy system stored in the file `tipper.fis`.

```matlab
fis = readfis('tipper')
```

```matlab
fis =
    mamfis with properties:
        Name:  "tipper"
        AndMethod: "min"
```
OrMethod: "max"
ImplicationMethod: "min"
AggregationMethod: "max"
DefuzzificationMethod: "centroid"
Inputs: [1x2 fisvar]
Outputs: [1x1 fisvar]
Rules: [1x3 fisrule]
DisableStructuralChecks: 0

See 'getTunableSettings' method for parameter optimization.

Input Arguments

**fileName — File name**
`string | character vector`

File name, specified as a string or character vector either with or without the `.fis` extension. This file must be in the current working directory or on the MATLAB path.

Output Arguments

**fis — Fuzzy inference system**
`mamfis object | sugfis object | mamfistype2 object | sugfistype2 object`

Fuzzy inference system, returned as one of the following:

- `mamfis object` — Mamdani fuzzy inference system
- `sugfis object` — Sugeno fuzzy inference system
- `mamfistype2 object` — Type-2 Mamdani fuzzy inference system
- `sugfistype2 object` — Type-2 Sugeno fuzzy inference system
Compatibility Considerations

Support for representing fuzzy inference systems as structures will be removed

Warns starting in R2019b

Support for representing fuzzy inference systems as structures will be removed in a future release. Use mamfis and sugfis objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either mamfis or sugfis objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

See Also

writeFIS

Topics

“Build Fuzzy Systems at the Command Line” on page 2-41

Introduced before R2006a
removeInput

Remove input variable from fuzzy inference system

Syntax

fisOut = removeInput(fisIn,inputName)

Description

fisOut = removeInput(fisIn,inputName) removes the input variable with the name inputName from fuzzy inference system fisIn and returns the resulting fuzzy system in fisOut.

Examples

Remove Input Variable from Fuzzy Inference System

Load fuzzy system.

fis = readfis("tipper");

View the input variables of fis.

fis.Inputs

ans =
1x2 fisvar array with properties:

Name
Range
MembershipFunctions

Details:

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>MembershipFunctions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
View the rules of fis.

fis.Rules

ans =
1x3 fisrule array with properties:

    Description
    Antecedent
    Consequent
    Weight
    Connection

Details:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;service==poor</td>
</tr>
<tr>
<td>2</td>
<td>&quot;service==good =&gt; tip=average (1)&quot;</td>
</tr>
</tbody>
</table>
| 3 | "service==excellent | food==delicious => tip=generous (1)"

Remove the service input variable.

fis = removeInput(fis,"service");

View the updated input variables.

fis.Inputs

ans =
  fisvar with properties:

    Name: "food"
    Range: [0 10]
    MembershipFunctions: [1x2 fismf]

View the updated rules.

fis.Rules
ans =  
1x2 fisrule array with properties:

    Description
    Antecedent
    Consequent
    Weight
    Connection

Details:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;food==rancid =&gt; tip=cheap (1)&quot;</td>
</tr>
</tbody>
</table>
| 2 | "food==delicious => tip=generous (1)"

service has been removed from the variables and rules of fis.

**Input Arguments**

- **fisIn — Fuzzy inference system**
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

  Fuzzy inference system, specified as one of the following:
  
  - mamfis object — Mamdani fuzzy inference system  
  - sugfis object — Sugeno fuzzy inference system  
  - mamfistype2 object — Type-2 Mamdani fuzzy inference system  
  - sugfistype2 object — Type-2 Sugeno fuzzy inference system

- **inputName — Input variable name**
  string | character vector

  Input variable name, specified as a string or character vector.
Output Arguments

**fisOut — Fuzzy inference system**

mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
- sugfistype2 object — Type-2 Sugeno fuzzy inference system

fisOut has the same properties as fisIn except:

- The input variable with the specified name is removed.
- The specified input variable is removed from any fuzzy rules. If a rule has only the specified input variable in its antecedent, then the entire rule is removed. If a rule has more than one input variable in its antecedent, then the specified input variable is removed from the antecedent.

**See Also**

addInput | fisvar

**Topics**

“Build Fuzzy Systems at the Command Line” on page 2-41

**Introduced in R2018b**
removeMF

Remove membership function from fuzzy variable

Syntax

fisOut = removeMF(fisIn,varName,mfName)
fisOut = removeMF(fisIn,varName,mfName,'VariableType',varType)
varOut = removeMF(varIn,varName,mfName)

Description

fisOut = removeMF(fisIn,varName,mfName) removes the membership function
mfName from the input or output variable varName in the fuzzy inference system fisIn
and returns the resulting fuzzy system in fisOut. To use this syntax, varName must be a
unique variable name within fisIn.

fisOut = removeMF(fisIn,varName,mfName,'VariableType',varType)
removes the membership function from either an input or output variable as specified by
varType. Use this syntax when your FIS has an input variable with the same name as an
output variable.

varOut = removeMF(varIn,varName,mfName) removes the membership function
mfName from the fuzzy variable varIn and returns the resulting fuzzy variable in
varOut.

Examples

Remove Membership Function from Fuzzy Inference System

Create a Mamdani fuzzy inference system with two inputs and one output. By default,
when you specify the number of inputs and outputs, mamfis adds three membership
functions to each variable.
fis = mamfis('NumInputs',3,'NumOutputs',1)

fis =
mamfis with properties:

  Name: "fis"
  AndMethod: "min"
  OrMethod: "max"
  ImplicationMethod: "min"
  AggregationMethod: "max"
  DefuzzificationMethod: "centroid"
  Inputs: [1x3 fisvar]
  Outputs: [1x1 fisvar]
  Rules: [1x27 fisrule]
  DisableStructuralChecks: 0

See 'getTunableSettings' method for parameter optimization.

Name the variables. For this example, give the second input variable and the output variable the same name.

fis.Inputs(1).Name = "speed";
fis.Inputs(2).Name = "throttle";
fis.Inputs(3).Name = "distance";
fis.Outputs(1).Name = "throttle";

View the membership functions for the first input variable.

plotmf(fis,"input",1)
Remove the second membership function, \textit{mf2}, from the first input variable.

\begin{verbatim}
   fis = removeMF(fis,"speed","mf2");
\end{verbatim}

View the membership functions again. The specified membership function has been removed.

\begin{verbatim}
   plotmf(fis,"input",1)
\end{verbatim}
If your system has an input variable with the same name as an output variable, you must specify the variable type when removing a membership function. For example, remove the \textit{mf3} membership function from the output variable.

\begin{verbatim}
fis = removeMF(fis,"throttle","mf3","VariableType","output");
\end{verbatim}

View the membership functions of the output variable.

\begin{verbatim}
plotmf(fis,"output",1)
\end{verbatim}
Remove Membership Function from Fuzzy Variable

Create a fuzzy variable with a specified range and add three membership functions

\[
\text{var} = \text{fisvar}([0\ 10]);
\]
\[
\text{var} = \text{addMF(\text{var}, "trimf", [0\ 2.5\ 5], "Name", "small");}
\]
\[
\text{var} = \text{addMF(\text{var}, "trimf", [2.5\ 5\ 7.5], "Name", "medium");}
\]
\[
\text{var} = \text{addMF(\text{var}, "trimf", [5\ 7.5\ 10], "Name", "large");}
\]

View the membership functions.

\[
\text{var.MembershipFunctions}
\]
ans =
    1x3 fismf array with properties:

    Type
    Parameters
    Name

Details:
    Name     Type     Parameters
         _______    _______    ___________________
     1    "small"     "trimf"      0    2.5      5
     2    "medium"    "trimf"    2.5      5    7.5
     3    "large"     "trimf"      5    7.5     10

Remove the medium membership function from the variable.

var = removeMF(var, "medium");

Verify that the membership was removed.

var.MembershipFunctions

ans =
    1x2 fismf array with properties:

    Type
    Parameters
    Name

Details:
    Name     Type     Parameters
         _______    _______    ___________________
     1    "small"     "trimf"      0    2.5      5
     2    "large"     "trimf"      5    7.5     10

**Input Arguments**

**fisIn — Fuzzy inference system**
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object
Fuzzy inference system, specified as one of the following:

- `mamfis` object — Mamdani fuzzy inference system
- `sugfis` object — Sugeno fuzzy inference system
- `mamfistype2` object — Type-2 Mamdani fuzzy inference system
- `sugfistype2` object — Type-2 Sugeno fuzzy inference system

**varName — Variable name**

string | character vector

Variable name, specified as a string or character vector. You can specify the name of either an input or output variable in your FIS.

**mfName — Membership function name**

string | character vector

Membership function name, specified as a string or character vector.

**varType — Variable type**

string | character vector

Variable type, specified as one of the following:

- "input" — Input variable
- "output" — Output variable

If your system has an input variable with the same name as an output variable, specify which variable to remove the membership function from using `varType`.

**varIn — Fuzzy variable**

fisvar object

Fuzzy variable, specified as a `fisvar` object.

### Output Arguments

**fisOut — Fuzzy inference system**

mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:
• **mamfis object** — Mamdani fuzzy inference system
• **sugfis object** — Sugeno fuzzy inference system
• **mamfistype2 object** — Type-2 Mamdani fuzzy inference system
• **sugfistype2 object** — Type-2 Sugeno fuzzy inference system

**fisOut** has the same properties as **fisIn** except:

• The membership function with the specified name is removed from the specified variable.
• The specified membership function is removed from any fuzzy rules. If a rule has only the specified membership function in its antecedent, then the entire rule is removed. If a rule has more than one membership function in its antecedent, then the specified membership function is removed from the antecedent.

**var0ut** — **Fuzzy variable**

**fisvar object**

Fuzzy variable, returned as a **fisvar object**. **varOut** has the same properties as **varIn** except the membership function with the specified name is removed.

**See Also**

**addMF**

**Topics**

“**Build Fuzzy Systems at the Command Line**” on page 2-41

**Introduced in R2018b**
removeOutput

Remove output variable from fuzzy inference system

Syntax

fisOut = removeOutput(fisIn,outputName)

Description

fisOut = removeOutput(fisIn,outputName) removes the output variable with the name outputName from fuzzy inference system fisIn and returns the resulting fuzzy system in fisOut.

Examples

Remove Output Variable from Fuzzy Inference System

Load fuzzy system.

fis = readfis("mam22");

View the output variables of fis.

fis.Outputs

ans =
1x2 fisvar array with properties:

    Name    Range    MembershipFunctions
    _______    _______    ___________________

Details:

        Name    Range    MembershipFunctions
        _______    _______    ____________________
View the rules of `fis`.

```matlab
fis.Rules
```

```
ans =
1x4 fisrule array with properties:

  Description
  Antecedent
  Consequent
  Weight
  Connection

Details:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;angle==small &amp; velocity==small =&gt; force=negBig, force2=posBig2 (1)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;angle==small &amp; velocity==big =&gt; force=negSmall, force2=posSmall2 (1)&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;angle==big &amp; velocity==small =&gt; force=posSmall, force2=negSmall2 (1)&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;angle==big &amp; velocity==big =&gt; force=posBig, force2=negBig2 (1)&quot;</td>
</tr>
</tbody>
</table>
```

Remove the `forceBig` output variable.

```matlab
fis = removeOutput(fis,"force2");
```

View the updated output variables.

```matlab
fis.Outputs
```

```
ans =
  fisvar with properties:

  Name: "force"
  Range: [-5 5]
  MembershipFunctions: [1x4 fismf]
```

View the updated rules.
fis.Rules

ans =
1x4 fisrule array with properties:

    Description  Antecedent  Consequent  Weight  Connection

Details:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;angle==small &amp; velocity==small =&gt; force=negBig (1)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;angle==small &amp; velocity==big =&gt; force=negSmall (1)&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;angle==big &amp; velocity==small =&gt; force=posSmall (1)&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;angle==big &amp; velocity==big =&gt; force=posBig (1)&quot;</td>
</tr>
</tbody>
</table>

force2 has been removed from the variables and rules of fis.

**Input Arguments**

**fisIn — Fuzzy inference system**
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
- sugfistype2 object — Type-2 Sugeno fuzzy inference system

**outputName — Output variable name**
string | character vector

Output variable name, specified as a string or character vector.
Output Arguments

fisOut — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

• mamfis object — Mamdani fuzzy inference system
• sugfis object — Sugeno fuzzy inference system
• mamfistype2 object — Type-2 Mamdani fuzzy inference system
• sugfistype2 object — Type-2 Sugeno fuzzy inference system

fisOut has the same properties as fisIn except:

• The input variable with the specified name is removed.
• The specified input variable is removed from any fuzzy rules. If a rule has only the specified input variable in its antecedent, then the entire rule is removed. If a rule has more than one input variable in its antecedent, then the specified input variable is removed from the antecedent.

See Also

addOutput | fisvar

Topics

“Build Fuzzy Systems at the Command Line” on page 2-41

Introduced in R2018b
rmmf

(To be removed) Remove membership function from fuzzy inference system

**Note** rmmf will be removed in a future release. Use `removeMF` instead. For more information, see “Compatibility Considerations”.

**Syntax**

```matlab
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`:

- Specify `varType` as either 'input' or 'output'.
- `varIndex` is an integer for the index of the variable. This index represents the order in which the variables are listed.
- `mfIndex` is an integer for the index of the membership function. This index represents the order in which the membership functions are listed.

**Examples**

**Remove Membership Function From Variable**

Create fuzzy inference system.

```matlab
fis = newfis('mysys');
```

Add an input variable with a single membership function to the system.
fis = addvar(fis,'input','temperature',[0 100]);
fis = addmf(fis,'input',1,'cold','trimf',[0 30 60]);

View the variable properties.
getfis(fis,'input',1)
ans = struct with fields:
    Name: 'temperature'
    NumMFs: 1
    mf1: 'cold'
    range: [0 100]

Remove the membership function. To do so, remove membership function 1 from input 1.
fis = rmmf(fis,'input',1,'mf',1);

View the variable properties.
getfis(fis,'input',1)
ans = struct with fields:
    Name: 'temperature'
    NumMFs: 0
    range: [0 100]

The variable now has no membership function.

**Compatibility Considerations**

**rmmf will be removed**

*Not recommended starting in R2018b*

rmmf will be removed in a future release. Use removeMF instead. There are differences between these functions that require updates to your code.

**Update Code**

The following table shows some typical usages of rmmf and how to update your code to use removeMF instead. Previously, you specified the index of the variable from which you
wanted to remove the membership function and the index of the membership function that you wanted to remove. Now, to remove a membership function, specify the variable name and the membership function name.

<table>
<thead>
<tr>
<th>If your code has this form:</th>
<th>Use this code instead:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fis = rmmf(fis,'input',1,'mf',1)</code></td>
<td><code>fis = removeMF(fis,&quot;service&quot;,&quot;poor&quot;)</code></td>
</tr>
<tr>
<td><code>fis = rmmf(fis,'output',1,'mf',1)</code></td>
<td><code>fis = removeMF(fis,&quot;tip&quot;,&quot;cheap&quot;)</code></td>
</tr>
</tbody>
</table>

**See Also**
- addMF
- addRule
- addvar
- plotmf
- removeMF
- rmvar

**Topics**
- “Membership Functions” on page 1-14
- “The Membership Function Editor” on page 2-25

**Introduced before R2006a**
rmvar

(To be removed) Remove variables from fuzzy inference system

**Note** rmvar will be removed in a future release. Use removeInput or removeOutput instead. For more information, see “Compatibility Considerations”.

**Syntax**

```matlab
fis = rmvar(fis,varType,varIndex)
[fis,errorMsg] = rmvar(fis,varType,varIndex)
```

**Description**

`fis = rmvar(fis,varType,varIndex)` removes the variable `varType`, of index `varIndex`, from the fuzzy inference system associated with the workspace FIS structure, `fis`:

- Specify `varType` as either 'input' or 'output'.
- `varIndex` is an integer for the index of the variable. This index represents the order in which the variables are listed.

`[fis,errorMsg] = rmvar(fis,varType,varIndex)` returns any error messages to the character vector, `errorMsg`.

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**
Remove Membership Function From Variable

Create fuzzy inference system.

```matlab
fis = newfis('mysys');
```

Add an input variable with a single membership function to the system.

```matlab
fis = addvar(fis,'input','temperature',[0 100]);
fis = addmf(fis,'input',1,'cold','trimf',[0 30 60]);
```

View the variable properties.

```matlab
getfis(fis,'input',1)
```

```matlab
ans = struct with fields:
    Name: 'temperature'
    NumMFs: 1
    mf1: 'cold'
    range: [0 100]
```

Remove the membership function. To do so, remove membership function 1 from input 1.

```matlab
fis = rmmf(fis,'input',1,'mf',1);
```

View the variable properties.

```matlab
getfis(fis,'input',1)
```

```matlab
ans = struct with fields:
    Name: 'temperature'
    NumMFs: 0
    range: [0 100]
```

The variable now has no membership function.

Compatibility Considerations

`rmvar` will be removed

*Not recommended starting in R2018b*
rmvar will be removed in a future release. Use removeInput or removeOutput instead. There are differences between these functions that require updates to your code.

To remove input or output variables from a fuzzy system, use removeInput or removeOutput, respectively.

**Update Code**

This table shows some typical usages of rmvar and how to update your code to use removeInput or removeOutput instead. Previously, you specified the index of the variable that you wanted to remove. Now, to remove a variable, specify the variable name.

<table>
<thead>
<tr>
<th>If your code has this form:</th>
<th>Use this code instead:</th>
</tr>
</thead>
<tbody>
<tr>
<td>fis = rmvar(fis,'input',1)</td>
<td>fis = removeInput(fis,&quot;service&quot;)</td>
</tr>
<tr>
<td>fis = rmvar(fis,'output',1)</td>
<td>fis = removeOutput(fis,&quot;tip&quot;)</td>
</tr>
</tbody>
</table>

Previously, you had to delete any rules from your fuzzy system that contained the variable you wanted to remove. removeInput and removeOutput automatically remove these variables from the rule set of your fuzzy system.

**See Also**

addMF | addRule | addvar | removeInput | removeOutput | rmmf

**Introduced before R2006a**
ruleedit

Open Rule Editor

Syntax

ruleedit(fis)
ruleedit(fileName)

Description

Use the Rule Editor to view or modify the rules of your fuzzy system. To define rules, you must specify the input and output variables of your FIS and their corresponding membership functions.

The Fuzzy Logic Designer app consists of several interactive interfaces for creating a fuzzy inference system (FIS), including the Rule Editor. For more information on interactively creating fuzzy systems, see “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17.

ruleedit(fis) opens the Rule Editor and loads the fuzzy inference system fis.

ruleedit(fileName) opens the Rule Editor and loads a fuzzy inference system from the file specified by fileName.

Examples

Open Rule Editor

Load or create a fuzzy inference system object. For this example, load the fuzzy system from a file.

fis = readfis('tipper');

Open the Rule Editor for this fuzzy system.
ruleedit(fis)

**Input Arguments**

fis — Fuzzy inference system
mamfis object | sugfis object
Fuzzy inference system, specified as either a `mamfis` or `sugfis` object in the MATLAB workspace.

**fileName — File name**  
string | character vector

File name specified as a string or character vector with or without the `.fis` extension. This file must be in the current working directory or on the MATLAB path.

## Compatibility Considerations

### Support for representing fuzzy inference systems as structures will be removed

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

## See Also

### Apps

- Fuzzy Logic Designer

### Functions

- `addRule`
- `mfedit`
- `ruleview`
- `showrule`
- `surfview`
Topics
“The Rule Editor” on page 2-33

Introduced before R2006a
**ruleview**

Open Rule Viewer

**Syntax**

```matlab
ruleview(fis)
ruleview(fileName)
```

**Description**

Use the Rule Viewer to view the inference process for your fuzzy system. You can adjust the input values and view the corresponding output of each fuzzy rule, the aggregated output fuzzy set, and the defuzzified output value. To view the inference process, you must specify the input and output variables of your FIS, their corresponding membership functions, and the fuzzy rules for your system.

The Fuzzy Logic Designer app consists of several interactive interfaces for creating a fuzzy inference system (FIS), including the Rule Viewer. For more information on interactively creating fuzzy systems, see “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17.

```matlab
ruleview(fis) opens the Rule Viewer and loads the fuzzy inference system fis.
ruleview(fileName) opens the Rule Viewer and loads a fuzzy inference system from the file specified by fileName.
```

**Examples**

**Open Rule Viewer**

Load or create a fuzzy inference system object. For this example, load the fuzzy system from a file.

```matlab
fis = readfis('tipper');
```
Open the Rule Viewer for this fuzzy system.

`ruleview(fis)`
Input Arguments

**fis — Fuzzy inference system**

mamfis object | sugfis object

Fuzzy inference system, specified as either a `mamfis` or `sugfis` object in the MATLAB workspace.

**fileName — File name**

string | character vector

File name specified as a string or character vector with or without the `.fis` extension. This file must be in the current working directory or on the MATLAB path.

Compatibility Considerations

**Support for representing fuzzy inference systems as structures will be removed**

 Warns starting in R2019b

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

See Also

**Apps**

Fuzzy Logic Designer
Functions
addRule | mfedit | ruleedit | showrule | surfview

Topics
“The Rule Viewer” on page 2-36

Introduced before R2006a
setfis

(To be removed) Set fuzzy system properties

**Note** setfis will be removed in a future release. Set fuzzy inference system properties using dot notation instead. For more information, see “Compatibility Considerations”.

**Syntax**

\[
\text{fis} = \text{setfis}(\text{fis}, \text{fisPropName}, \text{fisPropVal}) \\
\text{fis} = \text{setfis}(\text{fis}, \text{varType}, \text{varIndex}, \text{varPropName}, \text{varPropVal}) \\
\text{fis} = \text{setfis}(\text{fis}, \text{varType}, \text{varIndex}, \text{mf}, \text{mfIndex}, \text{mfPropName}, \text{mfPropVal})
\]

**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:

- **fis** — FIS structure in the MATLAB workspace.
- **varType** — Variable type, specified as either 'input' or 'output'.
- **varIndex** — Variable index, specified as a positive integer.
- **mfIndex** — Membership function index, specified as a positive integer.
- **fisPropName** — FIS property you want to set, specified as one of the following:
  - 'name'
  - 'type'
  - 'andmethod'
  - 'ormethod'
  - 'impmethod'
• 'aggmethod'
• 'defuzzmethod'
• fisPropVal — New value of the FIS property you want to set, specified as a character vector or string.
• varPropName — Variable property you want to set, specified as either 'name' or 'range'.
• varPropVal — New value of the variable property you want to set, specified as a character vector or string (for 'name'), or a two-element row vector (for 'range').
• mfPropName — Membership function property you want to set, specified as either 'name', 'type', or 'params'.
• mfPropVal — New value of the membership function property you want to set, specified as a character vector or string (for 'name' or 'type'), or a numerical row vector (for 'params').

Examples

Set Fuzzy Inference System Properties

Load a fuzzy inference system.

fis = readfis('tipper');

Set the defuzzification method to the bisector method.

fis = setfis(fis,'defuzzmethod','bisector');

View the defuzzification method of the updated FIS.

gtfis(fis,'defuzzmethod')

ans =
'bisector'

Set Variable Properties in FIS

Load fuzzy inference system.
fis = readfis('tipper');

Set the name of the first input variable to 'help'.

fis = setfis(fis,'input',1,'name','help');

View the name of the variable in the updated system.

getfis(fis,'input',1,'name')

ans =
    'help'

**Set Membership Function Properties in FIS**

Load a fuzzy inference system.

fis = readfis('tipper');

Change the type of the second membership function of the first input variable to a triangular membership function.

fis = setfis(fis,'input',1,'mf',2,'type','trimf');

When changing the type of a membership function, you must also set the parameters accordingly. To convert the original Gaussian membership function parameters to triangular membership function parameters, use the `mf2mf` command.

gaussParams = getfis(fis,'input',1,'mf',2,'params');
triParams = mf2mf(gaussParams,'gaussmf','trimf');

Set the membership function parameters to the converted values.

fis = setfis(fis,'input',1,'mf',2,'params',triParams);

View the updated membership function properties.

getfis(fis,'input',1,'mf',2)

ans = struct with fields:
    Name: 'good'
    Type: 'trimf'
Compatibility Considerations

setfis will be removed
Not recommended starting in R2018b

setfis will be removed in a future release. Set fuzzy inference system properties using dot notation instead. There are differences between these approaches that require updates to your code.

Update Code

This table shows some typical usages of setfis for setting fuzzy inference system properties and how to update your code to use dot notation instead.

<table>
<thead>
<tr>
<th>If your code has this form:</th>
<th>Use this code instead:</th>
</tr>
</thead>
<tbody>
<tr>
<td>fis = setfis(fis,'andmethod','prod')</td>
<td>fis.AndMethod = 'prod'</td>
</tr>
<tr>
<td>fis = setfis(fis,'input',1,... 'name','service')</td>
<td>fis.Inputs(1).Name = &quot;service&quot;</td>
</tr>
<tr>
<td>fis = setfis(fis,'input',2,... 'mf',1,... params,[5 10 15])</td>
<td>fis.Inputs(2).MembershipFunctions(1).Parameters = [5 10 15]</td>
</tr>
</tbody>
</table>

Previously, fuzzy inference systems were represented as structures. Now, fuzzy inference systems are represented as objects. Fuzzy inference system object properties have different names than the corresponding structure fields. For more information on fuzzy inference system objects, see mamfis and sugfis.

See Also
getfis

Introduced before R2006a
**setTunable**

**Package:** fuzzy.tuning

Set specified parameter settings as tunable or nontunable

**Syntax**

```matlab
paramsOut = setTunable(paramsIn,tunableFlag)
```

**Description**

`paramsOut = setTunable(paramsIn,tunableFlag)` sets the `paramsIn` parameters as tunable or nontunable using `tunableFlag`. The modified tunable parameter settings are returned in `paramsOut`.

**Examples**

**Specify Tunability of Parameter Settings**

Create a fuzzy inference system, and define the tunable parameter settings of inputs, outputs, and rules.

Create a FIS, and obtain its tunable settings.

```matlab
fis = mamfis("NumInputs",2,"NumOutputs",2);
[in,out,rule] = getTunableSettings(fis);
```

You can specify all the input variables, output variables, or rules as tunable or nontunable. For example, set all the output variable settings as nontunable.

```matlab
out = setTunable(out,0);
```

You can set the tunability of individual variables or rules. For example, set the first input variable as nontunable.
in(1) = setTunable(in(1),0);

You can set individual membership functions as nontunable. For example, set the first membership function of input 2 as nontunable.

in(2).MembershipFunctions(1) = setTunable(in(2).MembershipFunctions(1),0);

You can also specify the tunability of a subset of variables or rules. For example, set the first two rules as nontunable.

rule(1:2) = setTunable(rule(1:2),0);

**Input Arguments**

**paramsIn — Tunable parameter settings**

array | VariableSettings object | RuleSettings object | MembershipFunctionSettings object | MembershipFunctionSettingsType2 object

Tunable parameter settings, specified as one of the following:

- VariableSettings object or an array of such objects
- RuleSettingsObject object or an array of such objects
- MembershipFunctionSettings object or an array of such objects
- MembershipFunctionSettingsType2 object or an array of such objects

array of input, output, and rule parameter settings of a fuzzy system. To obtain these parameter settings, use `getTunableSettings` with the input FIS. `paramsetIn` can be the input parameter, the output parameter, the rule parameter, or some combination of these parameters as an array. The contents of the array depend on which parameters you would like to set.

**tunableFlag — Parameter tunability**

true or 1 | false or 0

Parameter tunability for the parameters specified in `paramsIn`, specified as a logical 1 (tunable) or 0 (nontunable).
Output Arguments

`paramsOut — Modified tunable parameter settings`  
VariableSettings object | RuleSettings object | MembershipFunctionSettings object | MembershipFunctionSettingsType2 object | vector

Modified tunable parameter settings, returned as one of the following:

- VariableSettings object or an array of such objects
- RuleSettingsObject object or an array of such objects
- MembershipFunctionSettings object or an array of such objects
- MembershipFunctionSettingsType2 object or an array of such objects

`paramsOut` is the same as `paramsetIn`, except with all tunable parameters set to the value specified in `tunableFlag`.

See Also

getTunableSettings | mamfis | sugfis | tunefis

Introduced in R2019a
**setTunableValues**

Specify tunable parameter values of a fuzzy inference system

**Syntax**

```matlab
fisOut = setTunableValues(fisIn,paramset,paramvals)
___  = setTunableValues( ___ , 'IgnoreInvalidParameters', ignoreInvalid)
```

**Description**

`fisOut = setTunableValues(fisIn,paramset,paramvals)` sets the tunable parameter values of fuzzy inference system `fisIn` and returns the resulting fuzzy system in `fisOut`. To specify the parameters to set, use `paramset`. Specify the new parameter values using `paramvals`.

```matlab
___  = setTunableValues( ___ , 'IgnoreInvalidParameters', ignoreInvalid)
```

sets a flag for ignoring invalid parameters values.

**Examples**

**Specify Tunable Parameter Values of a FIS**

Create a fuzzy inference system and define the tunable parameter settings of inputs, outputs, and rules.

```matlab
fis = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
[in,out,rule] = getTunableSettings(fis);
```

Obtain tunable parameter values of the inputs, outputs, and rules of the fuzzy inference system.

```matlab
paramVals = getTunableValues(fis,[in;out;rule]);
```

Redefine some of the values and update the tunable parameter values of the FIS.
paramVals(1:3) = [0 0 1];
fis = setTunableValues(fis,[in;out;rule],paramVals);

### Input Arguments

**fisIn — Fuzzy inference system**
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object | fistree object

Fuzzy inference system, specified as a mamfis, sugfis, mamfistype2, sugfistype2, or fistree object.

**paramset — Tunable parameter settings**
array

Tunable parameter settings, specified as an array of input, output, and rule parameter settings in the input FIS. To obtain these parameter settings, use the `getTunableSettings` function with the input `fis`.

*paramset* can be the input, output, or rule parameter settings, or any combination of these settings.

**paramvals — Tunable parameter values**
array

Tunable parameter values, specified as an array. The order of the values in `paramvals` matches the order of the parameters in `paramset`. To obtain the array of parameter values for a FIS, use `getTunableValues`.

**ignoreInvalid — Flag to ignore invalid parameters**
array

Flag to ignore invalid parameters, specified as either true or false. If true, invalid `paramvals` are replaced with the existing parameter values of a fuzzy system.
Output Arguments

fisOut — Modified fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object | fistree object

Modified fuzzy inference system, returned as a mamfis, sugfis, mamfistype2, or sugfistype2, or fistree object.

fisOut is the same as fisIn except that the parameters specified by paramset have the values specified by paramvals.

See Also
getTunableSettings | getTunableValues | mamfis | mamfistype2 | sugfis | sugfistype2 | tunefis

Introduced in R2019a
showfis

(To be removed) Display annotated Fuzzy Inference System

**Note** showfis will be removed in a future release. View the properties of your FIS directly instead. For more information, see “Compatibility Considerations”.

**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
Compatibility Considerations

**showfis will be removed**

*Not recommended starting in R2018b*

showfis will be removed in a future release. View the properties of your FIS directly instead.

Previously, you could view the properties of your fuzzy system, myFIS, using the showfis function.

```matlab
showfis(myFIS)
```

Now, you can view the properties directly instead.

```matlab
myFIS
```

To view additional FIS properties, use dot notation. For example, view information about the membership functions of the first input variable.

```matlab
myFIS.Inputs(1).MembershipFunctions
```

For more information on fuzzy inference systems and their properties, see *mamfis* and *sugfis*.

**See Also**

getfis

*Introduced before R2006a*
showrule

Display fuzzy inference system rules

Syntax

showrule(fis)
showrule(fis,Name,Value)

Description

showrule(fis) displays the rules in the fuzzy inference system fis.

showrule(fis,Name,Value) displays rules using options specified by one or more Name,Value pair arguments.

Examples

Display All Rules for a Fuzzy Inference System

Load fuzzy inference system.

fis = readfis('tipper');

Display rules using linguistic expressions.

showrule(fis)

ans = 3x78 char array
   '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)'        

Display rules using symbolic expressions.
Load fuzzy inference system.

```matlab
fis = readfis('tipper');
```

Display the first and third rules.

```matlab
showrule(fis,'RuleIndex',[1 3])
```

ans = 2x78 char array

'1. If (service is poor) or (food is rancid) then (tip is cheap) (1) '  
'3. If (service is excellent) or (food is delicious) then (tip is generous) (1)'

Display fuzzy rules in German language

Load fuzzy inference system.

```matlab
fis = readfis('tipper');
```

Display the rules in German using the 'deutsch' language.
showrule(fis,'Language','deutsch')

ans = 3x85 char array
    '1. Wenn (service ist poor) oder (food ist rancid) dann (tip ist cheap) (1)'
    '2. Wenn (service ist good) dann (tip ist average) (1)'
    '3. Wenn (service ist excellent) oder (food ist delicious) dann (tip ist generous)'

**Input Arguments**

**fis — Fuzzy inference system**

mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
- sugfistype2 object — Type-2 Sugeno fuzzy inference system

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'Format','symbolic' sets the rule display format to use language-neutral symbolic expressions.

**RuleIndex — Rules to display**

positive integer | vector of positive integers

Rules to display, specified as the comma-separated pair consisting of 'RuleIndex' and one of the following:

- Positive integer — Index of a single rule to display
- Vector of positive integers — Indices of multiple rules to display
The default vector includes the indices for all the rules in fis.

Format — Rule format
'verbose' (default) | 'symbolic' | 'indexed'

Rule format, specified as the comma-separated pair consisting of 'Format' and one of the following:
- 'verbose' — Use linguistic expressions.
  'If (service is poor) or (food is rancid) then (tip is cheap) (1)'
  The rule weight is displayed in parentheses at the end of the rule.
You can specify the rule language using the Language option.
- 'symbolic' — Use language-neutral symbolic expressions.
  '(service==poor) | (food==rancid) => (tip=cheap) (1)'
  The symbolic rules use the following symbols.

<table>
<thead>
<tr>
<th>Rule Component</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp;</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>IS (in antecedent)</td>
<td>==</td>
</tr>
<tr>
<td>IS (in consequent)</td>
<td>=</td>
</tr>
<tr>
<td>IS NOT</td>
<td>=~</td>
</tr>
<tr>
<td>Implication (then)</td>
<td>=&gt;</td>
</tr>
</tbody>
</table>

The rule weight is displayed in parentheses at the end of the rule.
- 'indexed' — Use input and output membership function (MF) indices and integer representation of fuzzy operators.

The indexed rules display in the following format:
'<input MFs>, <output MFs>, (<weight>) : <logical operator - 1 (AND), 2 (OR)>'

For example:
'1 1, 1 (1) : 2'
To indicate NOT operations for input and output membership functions, the software uses negative indices. For example, to indicate "not the second membership function," the software uses -2.

To indicate a don't care condition for an input or output membership function, the software uses 0.

**Language — Rule language**

'english' (default) | 'francais' | 'deutsch'

Rule language for 'verbose' format, specified as the comma-separated pair consisting of 'Language' and one of the following:

- 'english' — Display rules in English.
  - 'If (service is poor) or (food is rancid) then (tip is cheap) (1)'

- 'francais' — Display rules in French.
  - 'Si (service est poor) ou (food est rancid) alors (tip est cheap) (1)'

- 'deutsch' — Display rules in German.
  - 'Wenn (service ist poor) oder (food ist rancid) dann (tip ist cheap) (1)'

The software displays the FIS rules using the following keywords.

<table>
<thead>
<tr>
<th>Rule Component</th>
<th>English</th>
<th>French</th>
<th>German</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start of antecedent</td>
<td>if</td>
<td>si</td>
<td>wenn</td>
</tr>
<tr>
<td>AND</td>
<td>and</td>
<td>et</td>
<td>und</td>
</tr>
<tr>
<td>OR</td>
<td>or</td>
<td>ou</td>
<td>oder</td>
</tr>
<tr>
<td>Start of consequent</td>
<td>then</td>
<td>alors</td>
<td>dann</td>
</tr>
<tr>
<td>(implication)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td>is</td>
<td>est</td>
<td>ist</td>
</tr>
<tr>
<td>IS NOT</td>
<td>is not</td>
<td>n''est pas</td>
<td>ist nicht</td>
</tr>
</tbody>
</table>
Compatibility Considerations

Support for representing fuzzy inference systems as structures will be removed

Warns starting in R2019b

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

See Also

`addRule` | `ruleedit`

Introduced before R2006a
sigmf

Sigmoidal membership function

Syntax

\[ y = \text{sigmf}(x, \text{params}) \]

Description

This function computes fuzzy membership values using the difference between two sigmoidal membership functions. You can also compute this membership function using a fismf object. For more information, see “fismf Object” on page 8-304.

This membership function is related to the dsigmf and psigmf membership functions.

\[ y = \text{sigmf}(x, \text{params}) \] returns fuzzy membership values computed using the sigmoidal membership function given by:

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

To specify the \( a \) and \( c \) parameters, use params.

Membership values are computed for each input value in \( x \).

Examples

**Sigmoidal Membership Function**

```matlab
x = 0:0.1:10;
y = sigmf(x,[2 4]);
plot(x,y)
xlabel('sigmf, P = [2 4]')
ylabel(['Sigmoid, P = [2 4]'])
ylim([-0.05 1.05])
```
**Input Arguments**

- **x** — Input values  
  scalar | vector

  Input values for which to compute membership values, specified as a scalar or vector.

- **params** — Membership function parameters  
  vector of length four

  Membership function parameters, specified as the vector \([a \ c]\). To open the membership function to the left or right, specify a negative or positive value for \(a\), respectively. The
magnitude of \( a \) controls the width of the transition area, and \( c \) defines the center of the transition area.

**Output Arguments**

\( y \) — Membership value

scalar | vector

Membership value returned as a scalar or a vector. The dimensions of \( y \) match the dimensions of \( x \). Each element of \( y \) is the membership value computed for the corresponding element of \( x \).

**Alternative Functionality**

**fismf Object**

You can create and evaluate a fismf object that implements the sigmf membership function.

\[
\begin{align*}
\text{mf} &= \text{fismf}("\text{sigmf}",\text{P}); \\
\text{Y} &= \text{evalmf}(\text{mf},\text{X});
\end{align*}
\]

Here, \( X, P, \) and \( Y \) correspond to the \( x, \text{params}, \) and \( y \) arguments of \text{sigmf}, respectively.

**Extended Capabilities**

**C/C++ Code Generation**

Generate C and C++ code using MATLAB® Coder™.

**See Also**

\text{dsigmf|gauss2mf|gaussmf|gbellmf|pimf|psigmf|smf|trapmf|trimf|zmf}
Topics
“Membership Functions” on page 1-14

Introduced before R2006a
smf

S-shaped membership function

Syntax

\[ y = \text{smf}(x, \text{params}) \]

Description

This function computes fuzzy membership values using a spline-based S-shaped membership function. You can also compute this membership function using a \texttt{fismf} object. For more information, see “fismf Object” on page 8-308.

This membership function is related to the \texttt{zmf} and \texttt{pimf} membership functions.

\[ y = \text{smf}(x, \text{params}) \]

returns fuzzy membership values computed using the spline-based S-shaped membership function given by:

\[
\begin{align*}
  f(x; a, b) = & \begin{cases} 
    0, & x \leq a \\
    2 \left( \frac{x-a}{b-a} \right)^2, & a \leq x \leq \frac{a+b}{2} \\
    1 - 2 \left( \frac{x-b}{b-a} \right)^2, & \frac{a+b}{2} \leq x \leq b \\
    1, & x \geq b 
  \end{cases}
\end{align*}
\]

To specify the \( a \) and \( b \) parameters, use \texttt{params}.

Membership values are computed for each input value in \( x \).

Examples
S-Shaped Membership Function

```matlab
x = 0:0.1:10;
y = smf(x,[1 8]);
plot(x,y)
xlabel('smf, P = [1 8]')
ylim([-0.05 1.05])
```

Input Arguments

- **x** — Input values
  - scalar | vector
Input values for which to compute membership values, specified as a scalar or vector.

**params — Membership function parameters**
vector of length two

Membership function parameters, specified as the vector \([a \ b]\). Parameter \(a\) defines the *foot* of the membership function, and \(b\) defines its *shoulder*.

**Output Arguments**

**y — Membership value**
scalar | vector

Membership value returned as a scalar or a vector. The dimensions of \(y\) match the dimensions of \(x\). Each element of \(y\) is the membership value computed for the corresponding element of \(x\).

**Alternative Functionality**

**fismf Object**

You can create and evaluate a \(\text{fismf}\) object that implements the \(\text{smf}\) membership function.

\[
\text{mf} = \text{fismf}(\text{"smf"}, P);
\]

\[
Y = \text{evalmf(mf, X)};
\]

Here, \(X\), \(P\), and \(Y\) correspond to the \(x\), \(\text{params}\), and \(y\) arguments of \(\text{smf}\), respectively.

**Extended Capabilities**

**C/C++ Code Generation**
Generate C and C++ code using MATLAB® Coder™.
See Also
dsigmf | gauss2mf | gaussmf | gbellmf | pimf | psigmf | sigmf | trapmf | trimf | zmf

Topics
“Membership Functions” on page 1-14

Introduced before R2006a
subclust

Find cluster centers using subtractive clustering

Syntax

centers = subclust(data,clusterInfluenceRange)
centers = subclust(data,clusterInfluenceRange,Name,Value)
[centers,sigma] = subclust(____)

Description

centers = subclust(data,clusterInfluenceRange) clusters input data using subtractive clustering with the specified cluster influence range, and returns the computed cluster centers. The subtractive clustering algorithm on page 8-315 estimates the number of clusters in the input data.

centers = subclust(data,clusterInfluenceRange,Name,Value) clusters data using algorithm options specified by one or more Name,Value pair arguments.

[centers,sigma] = subclust(____) returns the sigma values specifying the range of influence of a cluster center in each of the data dimensions.

Examples

Find Cluster Centers Using Subtractive Clustering

Load data set.
load clusterdemo.dat

Find cluster centers using the same range of influence for all dimensions.
C = subclust(clusterdemo,0.6);  
Each row of C contains one cluster center.
C

C = 3×3

\[
\begin{bmatrix}
0.5779 & 0.2355 & 0.5133 \\
0.7797 & 0.8191 & 0.1801 \\
0.1959 & 0.6228 & 0.8363
\end{bmatrix}
\]

**Specify Bounds for Subtractive Clustering**

Load data set.

```matlab
load clusterdemo.dat
```

Define minimum and maximum normalization bounds for each data dimension. Use the same bounds for each dimension.

```
dataScale = [-0.2 -0.2 -0.2; 
              1.2  1.2  1.2];
```

Find cluster centers.

```
C = subclust(clusterdemo,0.5,'DataScale',dataScale);
```

**Specify Options for Subtractive Clustering**

Load data set.

```matlab
load clusterdemo.dat
```

Specify the following clustering options:

- Squash factor of 2.0 - Only find clusters that are far from each other.
- Accept ratio 0.8 - Only accept data points with a strong potential for being cluster centers.
- Reject ratio of 0.7 - Reject data points if they do not have a strong potential for being cluster centers.
Verbosity flag of 0 - Do not print progress information to the command window.

options = [2.0 0.8 0.7 0];

Find cluster centers, using a different range of influence for each dimension and the specified options.

C = subclust(clusterdemo,[0.5 0.25 0.3],'Options',options);

Obtain Cluster Influence Range for Each Data Dimension

Load data set.

load clusterdemo.dat

Cluster data, returning cluster sigma values, S.

[C,S] = subclust(clusterdemo,0.5);

Cluster sigma values indicate the range of influence of the computed cluster centers in each data dimension.

Input Arguments

data — Data set to be clustered
M-by-N array

Data to be clustered, specified as an M-by-N array, where M is the number of data points and N is the number of data dimensions.

clusterInfluenceRange — Range of influence of the cluster center
scalar value in the range [0, 1] | vector

Range of influence of the cluster center for each input and output assuming the data falls within a unit hyperbox, specified as the comma-separated pair consisting of 'ClusterInfluenceRange' one of the following:

• Scalar value in the range [0 1] — Use the same influence range for all inputs and outputs.
- Vector — Use different influence ranges for each input and output.

Specifying a smaller range of influence usually creates more and smaller data clusters, producing more fuzzy rules.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: 'DataScale','auto' sets the normalizing factors for the input and output signals using the minimum and maximum values in the data set to be clustered.

**DataScale — Data scale factors**

'auto' (default) | 2-by-N array

Data scale factors for normalizing input and output data into a unit hyperbox, specified as the comma-separated pair consisting of 'DataScale' and a 2-by-N array, where N is the total number of inputs and outputs. Each column of DataScale specifies the minimum value in the first row and the maximum value in the second row for the corresponding input or output data set.

When DataScale is 'auto', the genfis command uses the actual minimum and maximum values in the data to be clustered.

**Options — Clustering options**

vector

Clustering options, specified as the comma-separated pair consisting of 'Options' and a vector with the following elements:

**Options(1) — Squash factor**

1.25 (default) | positive scalar

Squash factor for scaling the range of influence of cluster centers, specified as a positive scalar. A smaller squash factor reduces the potential for outlying points to be considered as part of a cluster, which usually creates more and smaller data clusters.

**Options(2) — Acceptance ratio**

0.5 (default) | scalar value in the range [0, 1]
Acceptance ratio, defined as a fraction of the potential of the first cluster center, above which another data point is accepted as a cluster center, specified as a scalar value in the range [0, 1]. The acceptance ratio must be greater than the rejection ratio.

**Options(3) — Rejection ratio**
0.15 (default) | scalar value in the range [0, 1]

Rejection ratio, defined as a fraction of the potential of the first cluster center, below which another data point is rejected as a cluster center, specified as a scalar value in the range [0, 1]. The rejection ratio must be less than acceptance ratio.

**Options(4) — Information display flag**
false (default) | true

Information display flag indicating whether to display progress information during clustering, specified as one of the following:

- false — Do not display progress information.
- true — Display progress information.

**Output Arguments**

centers — Cluster centers
J-by-N array

Cluster centers, returned as a J-by-N array, where J is the number of clusters and N is the number of data dimensions.

**sigma — Range of influence of cluster centers**
N-element row vector

Range of influence of cluster centers for each data dimension, returned as an N-element row vector. All cluster centers have the same set of sigma values.

**Tips**

- To generate a fuzzy inference system using subtractive clustering, use the genfis command. For example, suppose you cluster your data using the following syntax:

  \[ C = \text{subclust}(\text{data}, \text{clusterInfluenceRange}, '\text{DataScale}', \text{dataScale}, '\text{Options}', \text{options}); \]
where the first \( M \) columns of \texttt{data} correspond to input variables, and the remaining columns correspond to output variables.

You can generate a fuzzy system using the same training data and subtractive clustering configuration. To do so:

1. Configure clustering options.

   ```
   opt = genfisOptions('SubtractiveClustering');
   opt.ClusterInfluenceRange = clusterInfluenceRange;
   opt.DataScale = dataScale;
   opt.SquashFactor = options(1);
   opt.AcceptRatio = options(2);
   opt.RejectRatio = options(3);
   opt.Verbose = options(4);
   ```

2. Extract input and output variable data.

   ```
   inputData = data(:,1:M);
   outputData = data(:,M+1:end);
   ```

3. Generate FIS structure.

   ```
   fis = genfis(inputData,outputData,opt);
   ```

The fuzzy system, \texttt{fis}, contains one fuzzy rule for each cluster, and each input and output variable has one membership function per cluster. You can generate only Sugeno fuzzy systems using subtractive clustering. For more information, see \texttt{genfis} and \texttt{genfisOptions}.

## Algorithms

Subtractive clustering assumes that each data point is a potential cluster center. The algorithm does the following:

1. Calculate the likelihood that each data point would define a cluster center, based on the density of surrounding data points.

2. Choose the data point with the highest potential to be the first cluster center.

3. Remove all data points near the first cluster center. The vicinity is determined using \texttt{clusterInfluenceRange}.

4. Choose the remaining point with the highest potential as the next cluster center.
Repeat steps 3 and 4 until all the data is within the influence range of a cluster center.

The subtractive clustering method is an extension of the mountain clustering method proposed in [2].

References


See Also
genfis

Topics
"Fuzzy Clustering" on page 4-2
"Model Suburban Commuting Using Subtractive Clustering" on page 4-22

Introduced before R2006a
surfview

Open Surface Viewer

Syntax

ruleview(fis)
ruleview(fileName)

Description

Use the Surface Viewer to view the output surface for your fuzzy system. To view the output surface, you must specify the input and output variables of your FIS, their corresponding membership functions, and the fuzzy rules for your system.

The Fuzzy Logic Designer app consists of several interactive interfaces for creating a fuzzy inference system (FIS), including the Surface Viewer. For more information on interactively creating fuzzy systems, see “Build Fuzzy Systems Using Fuzzy Logic Designer” on page 2-17.

ruleview(fis) opens the Surface Viewer and loads the fuzzy inference system fis.

ruleview(fileName) opens the Surface Viewer and loads a fuzzy inference system from the file specified by fileName.

Examples

Open Surface Viewer

Load or create a fuzzy inference system object. For this example, load the fuzzy system from a file.

fis = readfis('tipper');

Open the Surface Viewer for this fuzzy system.
Input Arguments

*fis* — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- *mamfis object* — Mamdani fuzzy inference system
• sugfis object — Sugeno fuzzy inference system
• mamfistype2 object — Type-2 Mamdani fuzzy inference system
• sugfistype2 object — Type-2 Sugeno fuzzy inference system

**Note** The Surface Viewer is the only interface of the **Fuzzy Logic Designer** app that supports type-2 fuzzy inference systems.

**fileName** — File name
string | character vector

File name specified as a string or character vector with or without the `.fis` extension. This file must be in the current working directory or on the MATLAB path.

**Tips**

- For systems with more than two input variables, you can view the output surface for any combination of two inputs. You must specify constant reference values for any other input signals using the **Ref. Input** value.
- By default, the surface plot updates automatically when you change the input or output variable selections or the number of grid points. To disable automatic plot updates, in the **Options** menu, clear the **Always evaluate** option. When this option is disabled, to update the plot, click **Evaluate**.
- To create a smoother plot, increase the **Plot points** value.
- To view the surface from different angles, click and drag on the plot area.

**Compatibility Considerations**

**Support for representing fuzzy inference systems as structures will be removed**
*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:
- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either `mamfis` or `sugfis` objects.

To convert existing fuzzy inference system structures to objects, use the `convertfis` function.

**See Also**

**Apps**
- Fuzzy Logic Designer

**Functions**
gensurf | mfedit | ruleedit | ruleview

**Topics**
- “The Surface Viewer” on page 2-38

**Introduced before R2006a**
trapmf

Trapezoidal membership function

Syntax

\[ y = \text{trapmf}(x,\text{params}) \]

Description

This function computes fuzzy membership values using a trapezoidal membership function. You can also compute this membership function using a \text{fismf} object. For more information, see “fismf Object” on page 8-323.

This membership function is related to the \text{trimf} membership function.

\[ y = \text{trapmf}(x,\text{params}) \]

returns fuzzy membership values computed using the following trapezoidal membership function:

\[
f(x; a, b, c, d) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right)
\]

To specify the parameters, \(a, b, c,\) and \(d,\) use \text{params}.

Membership values are computed for each input value in \(x.\)

Examples

Trapezoid-Shaped Membership Function

\[
x = 0:0.1:10;
y = \text{trapmf}(x,[1 5 7 8]);
\text{plot}(x,y)
xlabel('\text{\texttt{trapmf}}, P = [1 5 7 8]')
ylim([-0.05 1.05])
\]
Input Arguments

**x** — Input values  
scalar | vector

Input values for which to compute membership values, specified as a scalar or vector.

**params** — Membership function parameters  
vector of length two

Membership function parameters, specified as the vector \([a \ b \ c \ d]\). Parameters \(b\) and \(c\) define the shoulders of the membership function, and \(a\) and \(d\) define its feet.
The shape of the membership function depends on the relative values of $b$ and $c$:

- When $c$ is greater than $b$, the resulting membership function is trapezoidal.
- When $b$ is equal to $c$, the resulting membership function is equivalent to a triangular membership function with parameters $[a \ b \ d]$.
- When $c$ is less than $b$, the resulting membership function is triangular with a maximum value less than 1.

### Output Arguments

- **$y$ — Membership value**
  
  scalar | vector

  Membership value returned as a scalar or a vector. The dimensions of $y$ match the dimensions of $x$. Each element of $y$ is the membership value computed for the corresponding element of $x$.

### Alternative Functionality

**fismf Object**

You can create and evaluate a fismf object that implements the `trapmf` membership function.

```matlab
mf = fismf("trapmf",P);
Y = evalmf(mf,X);
```

Here, $X$, $P$, and $Y$ correspond to the $x$, $params$, and $y$ arguments of `trapmf`, respectively.

### Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using MATLAB® Coder™.
See Also
dsigmf | gauss2mf | gaussmf | gbellmf | pimf | psigmf | sigmf | smf | trimf | zmf

Topics
“Membership Functions” on page 1-14

Introduced before R2006a
trimf

Triangular membership function

Syntax

\[ y = \text{trimf}(x, \text{params}) \]

Description

This function computes fuzzy membership values using a triangular membership function. You can also compute this membership function using a \texttt{fismf} object. For more information, see “fismf Object” on page 8-327.

This membership function is related to the \texttt{trapmf} membership function.

\[ y = \text{trimf}(x, \text{params}) \] returns fuzzy membership values computed using the following triangular membership function:

\[
f(x; a, b, c) = \begin{cases} 
0, & x \leq a \\
\frac{x - a}{b - a}, & a \leq x \leq b \\
\frac{c - x}{c - b}, & b \leq x \leq c \\
0, & c \leq x
\end{cases}
\]

or, more compactly:

\[
f(x; a, b, c) = \max(\min\left(\frac{x - a}{b - a}, \frac{c - x}{c - b}\right), 0)
\]

To specify the parameters, \(a\), \(b\), and \(c\), use \texttt{params}.

Membership values are computed for each input value in \(x\).
Examples

Triangle-Shaped Membership Function

\[
x = 0:0.1:10;
y = \text{trimf}(x,[3 6 8]);
\text{plot}(x,y)
\text{xlabel('trimf, P = [3 6 8]')}
\text{ylim([-0.05 1.05])}
\]
Input Arguments

\( x \) — Input values

scalar | vector

Input values for which to compute membership values, specified as a scalar or vector.

\( \text{params} \) — Membership function parameters

vector of length three

Membership function parameters, specified as the vector \([a \ b \ c]\). Parameters \(a\) and \(c\) define the \textit{feet} of the membership function, and \(b\) defines its \textit{peak}.

Output Arguments

\( y \) — Membership value

scalar | vector

Membership value returned as a scalar or a vector. The dimensions of \(y\) match the dimensions of \(x\). Each element of \(y\) is the membership value computed for the corresponding element of \(x\).

Alternative Functionality

\textbf{fismf Object}

You can create and evaluate a \texttt{fismf} object that implements the \texttt{trimf} membership function.

\[
\text{mf} = \text{fismf}('\text{trimf}', \text{P});
\]

\[
\text{Y} = \text{evalmf}(	ext{mf}, \text{X});
\]

Here, \(X\), \(P\), and \(Y\) correspond to the \(x\), \(\text{params}\), and \(y\) arguments of \texttt{trimf}, respectively.
Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
dsigmf | gauss2mf | gaussmf | gbellmf | pimf | psigmf | sigmf | smf | trapmf | zmf

Topics
“Membership Functions” on page 1-14

Introduced before R2006a
tunefis

Tune fuzzy inference system or tree of fuzzy inference systems

Syntax

fisout = tunefis(fisin,paramset,in,out)
fisout = tunefis(fisin,paramset,custcostfcn)
fisout = tunefis(____,options)
[fisout,optoutputs] = tunefis(____)

Description

fisout = tunefis(fisin,paramset,in,out) tunes the fuzzy inference system fisin using the tunable parameter settings specified in paramset and the training data specified by in and out.

fisout = tunefis(fisin,paramset,custcostfcn) tunes the fuzzy inference system using a function handle to a custom cost function, custcostfcn.

fisout = tunefis(____,options) tunes the fuzzy inference system with additional options from the object options created using tunefisOptions.

[fisout,optoutputs] = tunefis(____) tunes the fuzzy inference system and returns additional information about the tuning algorithm in optoutputs.

Examples

Tune a Fuzzy Inference System

Create the initial fuzzy inference system using genfis.

x = (0:0.1:10)';
y = sin(2*x)./exp(x/5);
options = genfisOptions('GridPartition');
options.NumMembershipFunctions = 5;
fisin = genfis(x,y,options);

Obtain the tunable settings of inputs, outputs, and rules of the fuzzy inference system.
[in,out,rule] = getTunableSettings(fisin);

Tune the membership function parameters with "anfis".
fisout = tunefis(fisin,[in;out],x,y,tunefisOptions("Method","anfis"));

ANFIS info:
   Number of nodes: 24
   Number of linear parameters: 10
   Number of nonlinear parameters: 15
   Total number of parameters: 25
   Number of training data pairs: 101
   Number of checking data pairs: 0
   Number of fuzzy rules: 5

Start training ANFIS ...

1 0.0694086
2 0.0680259
3 0.0666663
4 0.0653198
5 0.0639961

Step size increases to 0.011000 after epoch 5.
6 0.0626917
7 0.0612787
8 0.0598881
9 0.0585193

Step size increases to 0.012100 after epoch 9.
10 0.0571712

Designated epoch number reached --> ANFIS training completed at epoch 10.

Minimal training RMSE = 0.057171

**Tune Specific Parameter Setting of Fuzzy Inference System**

Create the initial fuzzy inference system using genfis.
x = (0:0.1:10)';
y = sin(2*x)./exp(x/5);
options = genfisOptions('GridPartition');
options.NumMembershipFunctions = 5;
fisin = genfis(x,y,options);

Obtain the tunable settings of inputs, outputs, and rules of the fuzzy inference system.

[in,out,rule] = getTunableSettings(fisin);

Tune the rule parameter only. In this example, the pattern search method is used.

fisout = tunefis(fisin,rule,x,y,tunefisOptions('Method','patternsearch'));

Tune a Fuzzy Inference System with Custom Parameter Settings

Create the initial fuzzy inference system using genfis.
x = (0:0.1:10)';
y = sin(2*x)./exp(x/5);
options = genfisOptions('GridPartition');
options.NumMembershipFunctions = 5;
fisin = genfis(x,y,options);

Obtain the tunable settings of inputs, outputs, and rules of the fuzzy inference system.

[in,out,rule] = getTunableSettings(fisin);

You can tune with custom parameter settings using setTunable or dot notation.

Do not tune input 1.

in(1) = setTunable(in(1),false);

For output 1:

• do not tune membership functions 1 and 2,
• do not tune membership function 3,
• set the minimum parameter range of membership function 4 to -2,
• and set the maximum parameter range of membership function 5 to 2.

out(1).MembershipFunctions(1:2) = setTunable(out(1).MembershipFunctions(1:2),false);
out(1).MembershipFunctions(3).Parameters.Free = false;
out(1).MembershipFunctions(4).Parameters.Minimum = -2;
out(1).MembershipFunctions(5).Parameters.Maximum = 2;

For the rule settings,

• do not tune rules 1 and 2,
• set the antecedent of rule 3 to non-tunable,
• allow NOT logic in the antecedent of rule 4,
• and do not ignore any outputs in rule 3.

rule(1:2) = setTunable(rule(1:2),false);
rule(3).Antecedent.Free = false;
rule(4).Antecedent.AllowNot = true;
rule(3).Consequent.AllowEmpty = false;

Set the maximum number of iterations to 20 and tune the fuzzy inference system.
opt = tunefisOptions("Method","particleswarm");
opt.MethodOptions.MaxIterations = 20;
fisout = tunefis(fisin,[in;out;rule],x,y,opt);

<table>
<thead>
<tr>
<th>Iteration</th>
<th>f-count</th>
<th>Best f(x)</th>
<th>Mean f(x)</th>
<th>Stall Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90</td>
<td>0.3265</td>
<td>1.857</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>180</td>
<td>0.3265</td>
<td>4.172</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>270</td>
<td>0.3265</td>
<td>3.065</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>360</td>
<td>0.3265</td>
<td>3.839</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>450</td>
<td>0.3265</td>
<td>3.386</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>540</td>
<td>0.3265</td>
<td>3.249</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>630</td>
<td>0.3265</td>
<td>3.311</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>720</td>
<td>0.3265</td>
<td>2.901</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>810</td>
<td>0.3265</td>
<td>2.868</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>900</td>
<td>0.3181</td>
<td>2.71</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>990</td>
<td>0.3181</td>
<td>2.068</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1080</td>
<td>0.3181</td>
<td>2.692</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>1170</td>
<td>0.3165</td>
<td>2.146</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1260</td>
<td>0.3165</td>
<td>1.869</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1350</td>
<td>0.3165</td>
<td>2.364</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>1440</td>
<td>0.3165</td>
<td>2.07</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1530</td>
<td>0.3164</td>
<td>1.678</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>1620</td>
<td>0.2978</td>
<td>1.592</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>1710</td>
<td>0.2977</td>
<td>1.847</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>1800</td>
<td>0.2954</td>
<td>1.666</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>1890</td>
<td>0.2947</td>
<td>1.608</td>
<td>0</td>
</tr>
</tbody>
</table>

Optimization ended: number of iterations exceeded OPTIONS.MaxIterations.

Input Arguments

fisin — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object | fistree object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
• sugfistype2 object — Type-2 Sugeno fuzzy inference system
• fistree object — Tree of interconnected fuzzy inference systems

paramset — Tunable parameter settings
array

Tunable parameter settings, specified as an array of input, output, and rule parameter settings in the input FIS. To obtain these parameter settings, use the getTunableSettings function with the input fisin.

paramset can be the input, output, or rule parameter settings, or any combination of these settings.

in — Input training data
matrix

Input training data, specified as an $m$-by-$n$ matrix, where $m$ is the total number of input datasets and $n$ is the number of inputs. The number of input and output datasets must be the same.

out — Output training data
matrix

Output training data, specified as an $m$-by-$q$ matrix, where $m$ is the total number of output datasets and $q$ is the number of outputs. The number of input and output datasets must be the same.

options — FIS tuning options
tunefisOptions option set

FIS tuning options, specified as a tunefisOptions object. You can specify the tuning algorithm method and other options for the tuning process.

custcostfcn — custom cost functions
function handle

Custom cost function, specified as a function handle. The custom cost function evaluates fisout to calculate its cost with respect to an evaluation criterion, such as input/output data. custcostfcn must accept at least one input argument for fisout and returns a cost value. You can provide an anonymous function handle to attach additional data for cost calculation, as described in this example:
function fitness = custcost(cost,trainingData)
    ...
end
custcostfcn = @(fis)custcost(fis,trainingData);

Output Arguments

fisout — Fuzzy inference system
mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:
- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
- sugfistype2 object — Type-2 Sugeno fuzzy inference system
- sugfistype2 object — Tree of interconnected fuzzy inference systems

fisout is the same type of FIS as fisin.

optoutputs — Tuning algorithm summary
structure

Tuning algorithm summary, specified as a structure containing the following fields:
- tuningOutputs
- errorMessage

tuningOutputs is a structure with information about the tuning algorithm method specified by tunefisOptions in the options input argument. The specific outputs differ for each tuning algorithm. See the specific tuning method for details about its outputs:
- 'ga' — genetic algorithm
- 'particleswarm' — particle swarm
- 'patternsearch' — pattern search
- 'simulannealbnd' — simulated annealing algorithm
• 'anfis' — adaptive neuro-fuzzy

errorMessage is the message generated when updating fisin with new parameter values.

See Also
getTunableSettings | tunefisOptions

Introduced in R2019a
**update**

Update fuzzy rule using fuzzy inference system

**Syntax**

ruleOut = update(ruleIn,fis)

**Description**

ruleOut = update(ruleIn,fis) updates the fuzzy rule ruleIn using the information in fuzzy inference system fis and returns the resulting fuzzy rule in ruleOut.

**Examples**

**Create Fuzzy Rule Using Text Description**

Create a fuzzy rule using a verbose text description.

```
rule = fisrule("if service is poor and food is delicious then tip is average (1)";)
```

Alternatively, you can specify the same rule using a symbolic text description.

```
rule = fisrule("service==poor & food==delicious => tip=average")
```

rule =
  fisrule with properties:
    Description: "service==poor & food==delicious => tip=average (1)"
    Antecedent: []
    Consequent: []
      Weight: 1
    Connection: 1

Before using rule with a fuzzy system, update the rule Antecedent and Consequent properties using the update function.
fis = readfis("tipper");
rule = update(rule,fis)

rule =
  fisrule with properties:
    Description: "service==poor & food==delicious => tip=average (1)"
    Antecedent: [1 2]
    Consequent: 2
    Weight: 1
    Connection: 1

Create Fuzzy Rule Using Numeric Description

Create a fuzzy rule using a numeric description. Specify that the rule has two input variables.

rule = fisrule([1 2 2 0.5 1],2)

rule =
  fisrule with properties:
    Description: "input1==mf1 & input2==mf2 => output1=mf2 (0.5)"
    Antecedent: [1 2]
    Consequent: 2
    Weight: 0.5000
    Connection: 1

Before using rule with a fuzzy system, update the rule Description property using the update function.

fis = readfis("tipper");
rule = update(rule,fis)

rule =
  fisrule with properties:
    Description: "service==poor & food==delicious => tip=average (0.5)"
    Antecedent: [1 2]
    Consequent: 2
**Input Arguments**

**ruleIn — Fuzzy rule**

fisrule object | array of fisrule objects

Fuzzy rule, specified as a fisrule object or an array of fisrule objects. If ruleIn was created using a:

- Text description, its Antecedent and Consequent properties are updated using the input and output membership function indices in fis that correspond to the membership function names in the Description property of ruleIn
- Numeric description, its Description property is updated using the input and output membership function names in fis that correspond to the membership function indices in the Antecedent and Consequent properties of ruleIn

If you specify ruleIn as an array of fisrule objects, then all of the rules are updated accordingly.

**fis — Fuzzy inference system**

mamfis object | sugfis object | mamfistype2 object | sugfistype2 object

Fuzzy inference system, specified as one of the following:

- mamfis object — Mamdani fuzzy inference system
- sugfis object — Sugeno fuzzy inference system
- mamfistype2 object — Type-2 Mamdani fuzzy inference system
- sugfistype2 object — Type-2 Sugeno fuzzy inference system

**Output Arguments**

**ruleOut — Fuzzy rule**

fisrule object | array of fisrule objects

Fuzzy rule, returned as a fisrule object or an array of fisrule objects.
See Also
fisrule

Topics
“Build Fuzzy Systems at the Command Line” on page 2-41

Introduced in R2018b
**writeFIS**

Save fuzzy inference system to file

**Syntax**

```matlab
writeFIS(fis,fileName)
writeFIS(fis)
writeFIS(fis,fileName,"dialog")
```

**Description**

You can save a fuzzy inference system (FIS) in a .fis file using the `writeFIS` function. To load the saved file, use the `readfis` function.

**Note** Do not manually edit the contents of a .fis file. Doing so can produce unexpected results when loading the file using `readfis`.

`writeFIS(fis,fileName)` saves the fuzzy inference system `fis` to the current working folder using file name `fileName`.

`writeFIS(fis)` opens a dialog box for saving a FIS. In this dialog box, specify the name and location of the .fis file.

`writeFIS(fis,fileName,"dialog")` opens a dialog box for saving a FIS, setting the name of the file in the dialog box to `fileName`. In the dialog box, specify the location for the file.

**Examples**
Save Fuzzy Inference System to File

Create a fuzzy inference system, and add an input variable with membership functions.

```matlab
fis = mamfis('Name','tipper');
fis = addInput(fis,[0 10], 'Name','service');
fis = addMF(fis,'service','gaussmf',[1.5 0], 'Name','poor');
fis = addMF(fis,'service','gaussmf',[1.5 5], 'Name','good');
fis = addMF(fis,'service','gaussmf',[1.5 10], 'Name','excellent');
```

Save the fuzzy system in the current working folder in the file `myFile.fis`.

```matlab
writeFIS(fis,'myFile');
```

Input Arguments

- **fis** — Fuzzy inference system
  - `mamfis` object |
  - `sugfis` object |
  - `mamfistype2` object |
  - `sugfistype2` object

  Fuzzy inference system, specified as one of the following:
  - `mamfis` object — Mamdani fuzzy inference system
  - `sugfis` object — Sugeno fuzzy inference system
  - `mamfistype2` object — Type-2 Mamdani fuzzy inference system
  - `sugfistype2` object — Type-2 Sugeno fuzzy inference system

- **fileName** — File name
  - `string` | `character vector`

  File name, specified as a string or character vector. If you do not specify the `.fis` extension in the file name, `writeFIS` adds the extension.

Compatibility Considerations

`writefis` is now `writeFIS`

Behavior changed in R2018b
writefis is now writeFIS. To update your code, change the function name from writefis to writeFIS. The syntaxes are equivalent.

**Support for representing fuzzy inference systems as structures will be removed**

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use mamfis and sugfis objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either mamfis or sugfis objects.

To convert existing fuzzy inference system structures to objects, use the convertfis function.

**See Also**

readfis

**Topics**

“Build Fuzzy Systems at the Command Line” on page 2-41

**Introduced in R2018b**
**zmf**

Z-shaped membership function

**Syntax**

\[ y = zm(x, \text{params}) \]

**Description**

This function computes fuzzy membership values using a spline-based Z-shaped membership function. You can also compute this membership function using a `fismf` object. For more information, see “fismf Object” on page 8-346.

This membership function is related to the `smf` and `pimf` membership functions.

\[ y = zm(x, \text{params}) \]

returns fuzzy membership values computed using the spline-based Z-shaped membership function given by:

\[
   f(x; a, b) = \begin{cases} 
   1, & x \leq a \\
   1 - 2 \left(\frac{x - a}{b - a}\right)^2, & a \leq x \leq \frac{a + b}{2} \\
   2 \left(\frac{x - b}{b - a}\right)^2, & \frac{a + b}{2} \leq x \leq b \\
   0, & x \geq b
   \end{cases}
\]

To specify the \( a \) and \( b \) parameters, use `params`.

Membership values are computed for each input value in \( x \).

**Examples**
Z-Shaped Membership Function

```matlab
x = 0:0.1:10;
y = zmf(x,[3 7]);
plot(x,y)
xlabel('zmf, P = [3 7]')
ylim([-0.05 1.05])
```

**Input Arguments**

- **x** — Input values
  
  scalar | vector
Input values for which to compute membership values, specified as a scalar or vector.

**params — Membership function parameters**
vector of length two

Membership function parameters, specified as the vector \([a \ b]\). Parameter \(a\) defines the *shoulder* of the membership function, and \(b\) defines its *foot*.

**Output Arguments**

**y — Membership value**
scalar | vector

Membership value returned as a scalar or a vector. The dimensions of \(y\) match the dimensions of \(x\). Each element of \(y\) is the membership value computed for the corresponding element of \(x\).

**Alternative Functionality**

**fismf Object**

You can create and evaluate a `fismf` object that implements the `zmf` membership function.

```matlab
mf = fismf("zmf",P);
Y = evalmf(mf,X);
```

Here, \(X\), \(P\), and \(Y\) correspond to the \(x\), \(params\), and \(y\) arguments of `zmf`, respectively.

**Extended Capabilities**

**C/C++ Code Generation**
Generate C and C++ code using MATLAB® Coder™.
See Also
dsigmf | gauss2mf | gaussmf | gbellmf | pimf | psigmf | sigmf | smf | trapmf | trimf

Topics
“Membership Functions” on page 1-14

Introduced before R2006a
ClauseParameters

Parameter settings for rule clauses

Description

A `ClauseParameters` object contains tunable settings for either the antecedent or consequent of a fuzzy rule.

Creation

Create a `ClauseParameters` object using the `getTunableSettings` function. The third output of `getTunableSettings` contains `RuleSettings` objects. The Antecedent and Consequent properties of each `RuleSettings` object are `ClauseParameter` objects for specifying the tunable settings of the corresponding rule.

Properties

Free — Clause parameter values available for tuning

A logical 1 or 0, or an array of logical values. To apply different settings to each clause parameter, specify an array of logical values. To apply the same setting to all clause parameter values, specify a scalar logical value.

When the `ClauseParameters` object represents a rule antecedent, the clause parameter values are the membership functions corresponding to each input variable.

When the `ClauseParameters` object represents a rule consequent, the clause parameter values are the membership functions corresponding to each output variable.

AllowNot — Flag indicating whether to allow NOT logic in rule clauses

A logical 1 or 0, or an array of logical values.
Flag indicating whether to allow NOT logic in rule clauses, specified as a logical 1 or 0, or an array of logical values. To apply different settings to each clause parameter, specify an array of logical values. To apply the same setting to all clause parameter values, specify a scalar logical value.

**AllowEmpty — Flag indicating whether to allow ignoring inputs and outputs in rule clauses**

1 | 0 | array of logical values

Flag indicating whether to allow ignoring inputs and outputs in rule clauses, specified as a logical 1 or 0, or an array of logical values. To apply different settings to each clause parameter, specify an array of logical values. To apply the same setting to all clause parameter values, specify a scalar logical value.

### Examples

**Obtain Tunable Settings of Rules from FIS**

Create two fuzzy inference systems, and define the connection between the two.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = sugfis('Name','fis2','NumInputs',2,'NumOutputs',1);
con =ificaciones(["fis1/output1" "fis2/input1"]);
```

Create a tree of fuzzy inference systems.

```matlab
tree = fistree([fis1 fis2],con);
```

Obtain the tunable settings of rules of the fuzzy inference system.

```matlab
 [~,~,rule] = getTunableSettings(tree)
```

```
rule=18×1 object
16x1 RuleSettings array with properties:
    Index
    Antecedent
    Consequent
    FISName
```

9-3
You can use dot notation to specify the tunable settings of rules.

For the first rule, do not tune input 1 membership function index and do not ignore output 1 membership function index.

```matlab
rule(1).Antecedent.Free(1) = false;
rule(1).Consequent.AllowEmpty(1) = false;
```

For the second rule, allow NOT logic for input 2 membership function index.

```matlab
rule(2).Antecedent.AllowNot(2) = true;
```

**See Also**
RuleSettings | VariableSettings | getTunableSettings

**Introduced in R2019a**
evalfisOptions

Option set for evalfis function

Description

Use an evalfisOptions object to specify options for the evalfis function.

Creation

Syntax

```matlab
opt = evalfisOptions
opt = evalfisOptions(Name,Value)
```

Description

`opt = evalfisOptions` creates an option set for the `evalfis` function with default options. To modify the properties of this option set, use dot notation.

`opt = evalfisOptions(Name,Value)` sets properties using name-value pairs. For example, `evalfisOptions('NumSamplePoints',51)` creates an option set and sets the number of output fuzzy set samples to 51. You can specify multiple name-value pairs. Enclose each property name in single quotes.

Properties

`NumSamplePoints` — Number of sample points in output fuzzy sets
101 (default) | integer greater than 1

Number of sample points in output fuzzy sets, specified as an integer greater than 1.

To reduce memory usage while evaluating a Mamdani FIS, specify fewer samples. Doing so sacrifices the accuracy of the defuzzified output value.
Reducing the number of samples can make the output area for defuzzification zero. In this case, the defuzzified output value is the midpoint of the output variable range.

**Note** evalfis ignores this property when evaluating a Sugeno FIS.

**OutOfRangeInputValueMessage** — Diagnostic message behavior when an input is out of range
"warning" (default) | "error" | "none"

Diagnostic message behavior when an input is out of range, specified as one of the following:
- "warning" — Report the diagnostic message as a warning.
- "error" — Report the diagnostic message as an error.
- "none" — Do not report the diagnostic message.

When an input value is out of range, corresponding rules in the fuzzy system can have unexpected firing strengths.

**NoRuleFiredMessage** — Diagnostic message behavior when no rules fire
"warning" (default) | "error" | "none"

Diagnostic message behavior when no rules fire, specified as one of the following:
- "warning" — Report the diagnostic message as a warning.
- "error" — Report the diagnostic message as an error.
- "none" — Do not report the diagnostic message.

When NoRuleFiredMessage is "warning" or "none" and no rules fire for a given output, the defuzzified output value is set to its mean range value.

**EmptyOutputFuzzySetMessage** — Diagnostic message behavior when an output fuzzy set is empty
"warning" (default) | "error" | "none"

Diagnostic message behavior when an output fuzzy set is empty, specified as one of the following:
- "warning" — Report the diagnostic message as a warning.
• "error" — Report the diagnostic message as an error.
• "none" — Do not report the diagnostic message.

When EmptyOutputFuzzySetMessage is "warning" or "none" and an output fuzzy set is empty, the defuzzified value for the corresponding output is set to its mean range value.

This diagnostic message applies only to Mamdani systems.

Object Functions

Object Functions

Examples

Create Option Set for Evaluating FIS

Create option set object, specifying the number of sample points for output fuzzy sets.

options = evalfisOptions('NumSamplePoints',51)

options = EvalFISOptions with properties:

    NumSamplePoints: 51
    OutOfRangeInputValueMessage: "warning"
    NoRuleFiredMessage: "warning"
    EmptyOutputFuzzySetMessage: "warning"

Alternatively, create a default option set, and configure properties using dot notation.

options = evalfisOptions;
options.NumSamplePoints = 51;
Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

• When used for code generation, an evalfisOptions object stores itsOutOfRangeInputValueMessage, NoRuleFiredMessage, andEmptyOutputFuzzySetMessage properties as character vectors rather than strings.
• When evaluating a fuzzy inference system in Simulink, it is recommended to not useevalfis or evalfisOptionswithin a MATLAB Function block. Instead, evaluate your fuzzy inference system using the Fuzzy Logic Controller block.

See Also

Functions
evalfis

Introduced in R2018a
fismf

Fuzzy membership function

Description

Use a fismf object to represent a type-1 fuzzy membership function. For each input and output variable in a fuzzy inference system (FIS), one or more membership functions define the possible linguistic sets for that variable. For more information on membership functions, see “Foundations of Fuzzy Logic” on page 1-10.

Creation

Syntax

mf = fismf
mf = fismf(type,parameters)
mf = fismf('Name',name)
mf = fismf(type,parameters,'Name',name)

Description

mf = fismf creates a fuzzy membership function (MF) with default type, parameters, and name. To change the membership function properties, use dot notation.

mf = fismf(type,parameters) sets the Type and Parameters properties.

mf = fismf('Name',name) sets the Name property.

mf = fismf(type,parameters,'Name',name) sets the Type, Parameters, and Name properties.
# Properties

**Name — Membership function name**
"mf" (default) | string | character vector

Membership function name, specified as a string or character vector.

**Type — Membership function type**
"trimf" (default) | string | character vector | function handle

Membership function type, specified as a string or character vector that contains the name of a function in the current working folder or on the MATLAB path. You can also specify a handle to such a function. When you specify Type, you must also specify Parameters.

This table describes the values that you can specify for Type.

<table>
<thead>
<tr>
<th>Membership Function Type</th>
<th>Description</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;gbellmf&quot;</td>
<td>Generalized bell-shaped membership function</td>
<td>gbellmf</td>
</tr>
<tr>
<td>&quot;gaussmf&quot;</td>
<td>Gaussian membership function</td>
<td>gaussmf</td>
</tr>
<tr>
<td>&quot;gauss2mf&quot;</td>
<td>Gaussian combination membership function</td>
<td>gauss2mf</td>
</tr>
<tr>
<td>&quot;trimf&quot;</td>
<td>Triangular membership function</td>
<td>trimf</td>
</tr>
<tr>
<td>&quot;trapmf&quot;</td>
<td>Trapezoidal membership function</td>
<td>trapmf</td>
</tr>
<tr>
<td>&quot;sigmf&quot;</td>
<td>Sigmoidal membership function</td>
<td>sigmf</td>
</tr>
<tr>
<td>&quot;dsigmf&quot;</td>
<td>Difference between two sigmoidal membership functions</td>
<td>dsigmf</td>
</tr>
<tr>
<td>&quot;psigmf&quot;</td>
<td>Product of two sigmoidal membership functions</td>
<td>psigmf</td>
</tr>
<tr>
<td>&quot;zmf&quot;</td>
<td>Z-shaped membership function</td>
<td>zmf</td>
</tr>
<tr>
<td>&quot;pimf&quot;</td>
<td>Pi-shaped membership function</td>
<td>pimf</td>
</tr>
<tr>
<td>&quot;smf&quot;</td>
<td>S-shaped membership function</td>
<td>smf</td>
</tr>
<tr>
<td>Membership Function Type</td>
<td>Description</td>
<td>For More Information</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;constant&quot;</td>
<td>Constant membership function for Sugeno output membership functions</td>
<td>“Sugeno Fuzzy Inference Systems” on page 2-3</td>
</tr>
<tr>
<td>&quot;linear&quot;</td>
<td>Linear membership function for Sugeno output membership functions</td>
<td></td>
</tr>
<tr>
<td>String or character vector</td>
<td>Name of a custom membership function in the current working folder or on the MATLAB path. Custom output membership functions are not supported for Sugeno systems.</td>
<td>“Build Fuzzy Systems Using Custom Functions” on page 2-52</td>
</tr>
<tr>
<td>Function handle</td>
<td>Handle to a custom membership function in the current working folder or on the MATLAB path. Custom output membership functions are not supported for Sugeno systems.</td>
<td></td>
</tr>
</tbody>
</table>

**Note** When you change Type using dot notation, the values in Parameters are automatically converted for the new membership function type.

**Parameters — Membership function parameters**

```
[0 0.5 1] (default) | vector
```

Membership function parameters, specified as a vector. The length of the parameter vector depends on the membership function type. When you specify Parameters, you must also specify Type.

**Object Functions**

evalmf Evaluate fuzzy membership function
Examples

Create Membership Function
Create fuzzy membership function with default settings.

```matlab
mf = fismf;
```
To modify the membership function settings, use dot notation. For example, specify a Gaussian membership function with a standard deviation of 2 and a mean of 10.

```matlab
mf.Type = "gaussmf";
mf.Parameters = [2 10];
```

Create Membership Function with Specified Parameters
Create a trapezoidal membership function with specified parameters.

```matlab
mf = fismf("trapmf",[10 15 20 25]);
```

Create Membership Function with Specified Name
Create a membership function with the name "large".

```matlab
mf = fismf("Name","large");
```

See Also
fisrule| fisvar | mamfis | sugfis

Topics
"Build Fuzzy Systems at the Command Line" on page 2-41

Introduced in R2018b
**fismftype2**

Interval type-2 fuzzy membership function

**Description**

Use a `fismftype2` object to represent an interval type-2 fuzzy membership function (MF), which introduce additional uncertainty into a fuzzy inference system.

An interval type-2 membership function is represented by an upper and a lower membership function. The values of the upper membership function are always greater than or equal to the corresponding lower membership function values. The area enclosed by these membership functions is the footprint of uncertainty (FOU). For example, the following plot shows three type-2 membership functions for a given input variable.
For more information on type-2 membership functions, see “Type-2 Fuzzy Inference Systems” on page 2-7.

**Creation**

**Syntax**

```matlab
mf = fismftype2
mf = fismftype2(type,upperParameters)
mf = fismftype2(__,Name,Value)
```
**Description**

\( mf = \text{fismftype2} \) creates a type-2 fuzzy membership function with default name, type, upper MF parameters, and lower MF configuration. To change the membership function properties, use dot notation.

\( mf = \text{fismftype2}(\text{type}, \text{upperParameters}) \) sets the Type and UpperParameters properties of the membership function.

\( mf = \text{fismftype2}(\_\_\_, \text{Name}, \text{Value}) \) sets the Name, LowerScale, or LowerLag properties of the membership function using one or more name-value pair arguments for any of the other syntaxes.

**Properties**

**Name — Membership function name**

"mf" (default) | string | character vector

Membership function name, specified as a string or character vector.

**Type — Membership function type**

"trimf" (default) | string | character vector | function handle

Membership function type for both the upper and lower membership function, specified as a string or character vector that contains the name of a function in the current working folder or on the MATLAB path. You can also specify a handle to such a function. When you specify Type, you must also specify UpperParameters.

This table describes the values that you can specify for Type.

<table>
<thead>
<tr>
<th>Membership Function Type</th>
<th>Description</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;gbellmf&quot;</td>
<td>Generalized bell-shaped membership function</td>
<td>gbellmf</td>
</tr>
<tr>
<td>&quot;gaussmf&quot;</td>
<td>Gaussian membership function</td>
<td>gaussmf</td>
</tr>
<tr>
<td>&quot;gauss2mf&quot;</td>
<td>Gaussian combination membership function</td>
<td>gauss2mf</td>
</tr>
<tr>
<td>&quot;trimf&quot;</td>
<td>Triangular membership function</td>
<td>trimf</td>
</tr>
<tr>
<td>Membership Function Type</td>
<td>Description</td>
<td>For More Information</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>&quot;trapmf&quot;</td>
<td>Trapezoidal membership function</td>
<td>trapmf</td>
</tr>
<tr>
<td>&quot;sigmf&quot;</td>
<td>Sigmoidal membership function</td>
<td>sigmf</td>
</tr>
<tr>
<td>&quot;dsigmf&quot;</td>
<td>Difference between two sigmoidal membership functions</td>
<td>dsigmf</td>
</tr>
<tr>
<td>&quot;psigmf&quot;</td>
<td>Product of two sigmoidal membership functions</td>
<td>psigmf</td>
</tr>
<tr>
<td>&quot;zmf&quot;</td>
<td>Z-shaped membership function</td>
<td>zmf</td>
</tr>
<tr>
<td>&quot;pimf&quot;</td>
<td>Pi-shaped membership function</td>
<td>pimf</td>
</tr>
<tr>
<td>&quot;smf&quot;</td>
<td>S-shaped membership function</td>
<td>smf</td>
</tr>
<tr>
<td>String or character vector</td>
<td>Name of a custom membership function in the current working folder or on the MATLAB path. Custom output membership functions are not supported for Sugeno systems.</td>
<td>“Build Fuzzy Systems Using Custom Functions” on page 2-52</td>
</tr>
</tbody>
</table>

**Note** When you change Type using dot notation, the values in Parameters are automatically converted for the new membership function type.

**UpperParameters — Upper membership function parameters**

[0 0.5 1] (default) | vector

Upper membership function parameters, specified as a vector. The length of the parameter vector depends on the membership function type. When you specify Parameters, you must also specify Type.
**LowerScale — Lower membership function scaling factor**

1 (default) | positive scalar less than or equal to 1

Lower membership function scaling factor, specified as a positive scalar less than or equal to 1. Use `LowerScale` to define the maximum value of the lower membership function.

Depending on the value of `LowerLag`, the actual maximum lower membership function value can be less than `LowerScale`.

**LowerLag — Lower membership function delay factor**

scalar value between 0 and 1 | vector of length 2

Lower membership function delay factor, specified as a scalar value or a vector of length two. You can specify lag values between 0 and 1, inclusive.

The following membership function types support only a scalar `LowerLag` value:

- Symmetric MFs — `gaussmf` and `gbellmf`
- One-sided MFs — `sigmf`, `smf`, and `zmf`

All other built-in membership functions support either a scalar or vector `LowerLag` value. For these membership functions, when you specify a:

- Scalar value, the same lag value is used for both the left and right side of the membership function.
- Vector value, you can define different lag values for the left and right sides of the membership function.

The lag value defines the point at which the lower membership function value starts increasing from zero based on the value of the upper membership function. For example, a lag value of 0.1 indicates that the lower membership function becomes positive when the upper membership function has a membership value of 0.1.

By default, the lag value is 0.2. However, for some membership function types and upper membership function parameters, the software is unable to set a lower lag value to 0.2. In such a case, the default lag value is set to a different valid value.

When the lag value is zero, the lower membership function starts increasing at the same point as the upper membership function.
Some membership function types restrict the maximum lag value. For example, LowerLag must be less than 1 for the gaussmf, gauss2mf, gbellmf, sigmf, dsigmf, and psigmf membership functions.

**Object Functions**
evalmf  Evaluate fuzzy membership function

**Examples**

**Create Type-2 Membership Function**

Create type-2 membership function with default settings.

```matlab
mf = fismftype2;
```

To modify the membership function settings, use dot notation. For example, specify a Gaussian upper membership function with a standard deviation of 2 and a mean of 10.

```matlab
mf.Type = "gaussmf";
mf.UpperParameters = [2 10];
```

Specify the maximum lower membership function value as 0.8.

```matlab
mf.LowerScale = 0.8;
```

Configure the lower membership function to start increasing when the upper membership function reaches 0.3.

```matlab
mf.LowerLag = 0.3;
```

**Create Type-2 Membership Function with Specified Upper MF Parameters**

Create a trapezoidal type-2 membership function with specified upper MF parameters.

```matlab
mf = fismftype2("trapmf",[3 4 6 7])
```

```matlab
mf =
   fismftype2 with properties:
```
By default, the lower membership function has a maximum value of 1 and starts increasing when the upper MF is 0.2.

**Configure Lower MF Parameters**

Create a triangular type-2 membership function, specifying a maximum lower MF value of 0.9 and a membership function lag of 0.1.

```matlab
mf = fismftype2("trimf",[1 2 3],'LowerScale',0.9,'LowerLag',0.1);
```

**See Also**

fismf | fisrule | fisvar | mamfistype2 | sugfistype2

**Topics**

“Build Fuzzy Systems at the Command Line” on page 2-41
“Type-2 Fuzzy Inference Systems” on page 2-7

**Introduced in R2019b**
fisrule

Fuzzy rule

Description

Use fisrule objects to represent fuzzy if-then rules that relate input membership function conditions to corresponding output membership functions. The if portion of a fuzzy rule is the antecedent, which specifies the membership function for each input variable. The then portion of a fuzzy rule is the consequent, which specifies the membership function for each output variable. For more information on membership functions and fuzzy rules, see “Foundations of Fuzzy Logic” on page 1-10.

Creation

To create fuzzy rule objects, use the fisrule function. Using this function, you can create a single fuzzy rule or a vector of multiple fuzzy rules.

Syntax

```matlab
rule = fisrule
rule = fisrule(ruleText)
rule = fisrule(ruleValues,numInputs)
```

Description

`rule = fisrule` creates a single fuzzy rule with the default description "input1==mf1 => output1=mf1".

`rule = fisrule(ruleText)` creates one or more fuzzy rules using the text descriptions in `ruleText`.

`rule = fisrule(ruleValues,numInputs)` creates one or more fuzzy rules using the numeric rule values in `ruleValues`. Specify the number of rule input variables using `numInputs`.
Input Arguments

**ruleText — Text rule description**

string | character vector | string array | character array

Text rule description, specified as one of the following:

- String or character vector specifying a single rule
  
  ```
  rule = "If service is poor or food is rancid then tip is cheap";
  ```

- String array, where each element corresponds to a rule. For example:

  ```
  ruleList = ["If service is poor or 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"];
  ```

- Character array where each row corresponds to a rule. For example:

  ```
  rule1 = 'If service is poor or food is rancid then tip is cheap';
  rule2 = 'If service is good then tip is average';
  rule3 = 'If service is excellent or food is delicious then tip is generous';
  ruleList = char(rule1,rule2,rule3);
  ```

For each rule, use one of the following rule text formats:

- **Verbose** — Linguistic expression in the following format, using the IF and THEN keywords:

  ```
  "IF <antecedent> THEN <consequent> (<weight>)"
  ```

  In `<antecedent>`, specify the membership function for each input variable using the IS or IS NOT keyword. Connect these conditions using the AND or OR keywords. If a rule does not use a given input variable, omit it from the antecedent.

  In `<consequent>`, specify the condition for each output variable using the IS or IS NOT keyword, and separate these conditions using commas. The IS NOT keyword is not supported for Sugeno outputs. If a rule does not use a given output variable, omit it from the consequent.

  Specify the weight using a positive numerical value.

  For example:

  ```
  "IF A IS a AND B IS NOT b THEN X IS x, Y IS NOT y (1)"
  ```

- **Symbolic** — Expression that uses the symbols in the following table instead of keywords. There is no symbol for the IF keyword.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>IS (in rule antecedent)</td>
</tr>
<tr>
<td>~=</td>
<td>IS NOT</td>
</tr>
<tr>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>=&gt;</td>
<td>THEN</td>
</tr>
<tr>
<td>=</td>
<td>IS (in rule consequent)</td>
</tr>
</tbody>
</table>

For example, the following symbolic rule is equivalent to the previous verbose rule.

"A==a & B~=b => X=x, Y~=y (1)"

When you specify a rule using a text description, fisrule sets the Description, Weight, and Connection properties of the rule based on the description.

**ruleValues — Numeric rule description**

row vector | numeric array

Numeric rule description, specified as one of the following:

- Row vector to specify a single fuzzy rule
- Array, where each row of ruleValues specifies one rule

For each row, the numeric rule description has \( M+N+2 \) columns, where \( M \) is the number of input variables and \( N \) is the number of output variables. Each column contains the following information:

- The first \( M \) columns specify input membership function indices and correspond to the Antecedent property of the rule. To indicate a NOT condition, specify a negative value. If a rule does not use a given input, set the corresponding index to 0. For each rule, at least one input membership function index must be nonzero.
- The next \( N \) columns specify output membership function indices and correspond to the Consequent property of the rule. To indicate a NOT condition for Mamdani systems, specify a negative value. NOT conditions are not supported for Sugeno outputs. If a rule does not use a given output, set the corresponding index to 0. For each rule, at least one output membership function index must be nonzero.
- Column \( M+N+1 \) specifies the rule weight and corresponds to the Weight property of the rule.
• The final column specifies the antecedent fuzzy operator and corresponds to the Connection property of the rule.

When you specify a rule using `ruleValues`, `fisrule` sets the Description property using default variable and membership function names.

**numInputs — Number of input variables**
positive integer

Number of input variables for the rule, specified as a positive integer. If you specify the rule description using `ruleValues`, you must also specify the number of input variables. `fisrule` parses the rule antecedent values into the membership function indices for the input and output variables using `numInputs`.

## Properties

**Description — Text rule description**
string | character vector

Text rule description, specified as a string or character vector. The rule description is stored as a symbolic expression no matter how you specify the rule. For example, if you specify the following verbose rule using `ruleText`:

"IF A IS a AND B IS NOT b THEN X IS x, Y IS NOT y (1)"

The stored rule is:

"A==a & B~=b => X=x, Y~=y (1)"

For more information on the verbose and symbolic rule formats, see the `ruleText` input argument.

When you specify a rule using `ruleValues`, `fisrule` sets the Description property using default variable and membership function names. Before using the rule in a fuzzy system, you must update the description to use the variable and membership function names from that fuzzy system using the `update` function.

**Antecedent — Rule antecedent**
numeric vector

Rule antecedent, specified as a numeric vector of length `M`, where `M` is the number of input variables. Each element of Antecedent contains one of the following values:
• Positive integer — The index of an input membership function, which represents an **IS** condition
• Negative integer — The negative of an input membership function, which represents an **IS NOT** condition
• 0 — A **don't care** condition, which means that the rule does not use the corresponding input variable

This value is set when you create a fuzzy rule using `ruleValues`. If you create a fuzzy rule using `ruleText`, before using the rule in a fuzzy system, you must populate the **Antecedent** property using the `update` function.

If you update the indices in the rule antecedent using dot notation, the **Description** property is not updated to reflect the changes. To update the rule description, use the `update` function.

**Consequent — Rule consequent**

`numeric vector`

Rule consequent, specified as a numeric vector of length $N$, where $N$ is the number of output variables. Each element of **Consequent** contains one of the following values:

• Positive integer — The index of an output membership function, which represents an **IS** condition
• Negative integer — The negative of an output membership function, which represents an **IS NOT** condition
• 0 — A **don't care** condition, which means that the rule does not use the corresponding output variable

This value is set when you create a fuzzy rule using `ruleValues`. If you create a fuzzy rule using `ruleText`, before using the rule in a fuzzy system, you must populate the **Consequent** property using the `update` function.

If you update the indices in the rule consequent using dot notation, the **Description** property is not updated to reflect the changes. To update the rule description, use the `update` function.

**Weight — Rule weight**

`1 (default) | positive numeric scalar`

Rule weight, specified as a positive numeric scalar in the range [0 1].
If you update the rule weight using dot notation, the weight value in the Description property text is also updated.

**Connection — Rule antecedent connection**

1 | 2

Rule antecedent connection, specified as one of the following:

- 1 — Evaluate rule antecedents using the AND operator.
- 2 — Evaluate rule antecedents using the OR operator.

If you update the rule connection using dot notation, the antecedent operators in the Description property text are also updated.

**Object Functions**

*update*  
Update fuzzy rule using fuzzy inference system

**Examples**

**Create Fuzzy Rule**

Create a default fuzzy rule.

```matlab
rule = fisrule
```

```matlab
rule =
    fisrule with properties:
    
    Description: "input1==mf1 => output1=mf1 (1)"
    Antecedent: 1
    Consequent: 1
    Weight: 1
    Connection: 1
```

To modify the rule properties, use dot notation. For example, specify a rule weight of 0.5.

```matlab
rule.Weight = 0.5;
```
Create Fuzzy Rule Using Text Description

Create a fuzzy rule using a verbose text description.

```matlab
rule = fisrule("if service is poor and food is delicious then tip is average (1)");
```

Alternatively, you can specify the same rule using a symbolic text description.

```matlab
rule = fisrule("service==poor & food==delicious => tip=average")
```

```matlab
rule =
    fisrule with properties:
    Description: "service==poor & food==delicious => tip=average (1)"
    Antecedent: []
    Consequent: []
        Weight: 1
    Connection: 1
```

Before using `rule` with a fuzzy system, update the rule Antecedent and Consequent properties using the `update` function.

```matlab
fis = readfis("tipper");
rule = update(rule,fis)
```

```matlab
rule =
    fisrule with properties:
    Description: "service==poor & food==delicious => tip=average (1)"
    Antecedent: [1 2]
    Consequent: 2
        Weight: 1
    Connection: 1
```

Create Fuzzy Rule Using Numeric Description

Create a fuzzy rule using a numeric description. Specify that the rule has two input variables.
rule = fisrule([1 2 2 0.5 1],2)

rule =
    fisrule with properties:

    Description: "input1==mf1 & input2==mf2 => output1=mf2 (0.5)"
    Antecedent: [1 2]
    Consequent: 2
    Weight: 0.5000
    Connection: 1

Before using rule with a fuzzy system, update the rule Description property using the update function.

fis = readfis("tipper");
rule = update(rule,fis)

rule =
    fisrule with properties:

    Description: "service==poor & food==delicious => tip=average (0.5)"
    Antecedent: [1 2]
    Consequent: 2
    Weight: 0.5000
    Connection: 1

Create Multiple Fuzzy Rules

Create a string array of text rule descriptions.

rules1 = ["if service is poor or food is rancid then tip is cheap (0.5)"
          "if service is excellent and food is not rancid then tip is generous (0.75)"
          ...
          "if service is perfect and food is not rancid then tip is enormous (0.95)"
          ];

Create an array of fuzzy rules using these descriptions.

fuzzyRules1 = fisrule(rules1)

fuzzyRules1 =
    1x2 fisrule array with properties:

    Description
Antecedent
Consequent
Weight
Connection

Details:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;service==poor</td>
</tr>
<tr>
<td>2</td>
<td>&quot;service==excellent &amp; food~==rancid =&gt; tip=generous (0.75)&quot;</td>
</tr>
</tbody>
</table>

Alternatively, you can specify multiple rules using an array of numeric rule descriptions.

```matlab
rules2 = [1 1 1 0.5 2;
          2 -1 3 0.75 1];
fuzzyRules2 = fisrule(rules2,2)
```

`fuzzyRules2 =
1x2 fisrule array with properties:

Description
Antecedent
Consequent
Weight
Connection

Details:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
</table>
| 1 | "input1==mf1 | input2==mf1 => output1=mf1 (0.5)"
| 2 | "input1==mf2 & input2~==mf1 => output1=mf3 (0.75)"

**See Also**

fismf | fisvar | mamfis | mamfistype2 | sugfis | sugfistype2

**Topics**

“Build Fuzzy Systems at the Command Line” on page 2-41
Introduced in R2018b
fistree

Network of connected fuzzy inference systems

Description

Use a fistree object to represent a tree of interconnected fuzzy inference systems.

Creation

Syntax

```matlab
fisTree = fistree(fis,connections)
fisTree = fistree(__,'DisableStructuralChecks',disableChecks)
```

Description

`fisTree = fistree(fis,connections)` creates a network of interconnected fuzzy inference system objects, setting its FIS and Connections properties.

`fisTree = fistree(__,'DisableStructuralChecks',disableChecks)` sets the DisableStructuralChecks property.

Properties

**FIS — Fuzzy inference systems**
array

This property is read-only.

Fuzzy inference systems, specified as an array FIS objects. You can specify any combination of `mamfis`, `sugfis`, `mamfistype2`, and `sugfistype2` objects. Each fuzzy inference system in the FIS array must have at least one input and one output for
fistree construction. To evaluate a fistree, each fuzzy inference system must have at least one rule.

**Connections — Connections between fuzzy inference systems**

string array

Connections between fuzzy inference systems, specified as a two-dimensional string array. Each row represents a connection between two FIS objects. Specify connections as follows:

1. Output-to-input connections, 
   
   ```
   "fromFISName/fromFISOutputName" "toFISName/toFISInputName"
   ```
   
   In this case, output of "fromFISName" is used as the input of "toFISName". "fromFISName" and "toFISName" must be different.

2. Input-to-input connections, 
   
   ```
   "fromFISName/fromFISInputName" "toFISName/toFISInputName"
   ```

   In this case, inputs of "fromFISName" and "toFISName" use the same input values for evaluation. "fromFISName" and "toFISName" can be same or different.

The following diagram describes different connection types.
Connection ["fis1/x" "fis2/c"] is specified between output "x" of "fis1" and input "c" of "fis2". Connection ["fis1/a" "fis1/b"] is specified between inputs "a" and "b" of "fis1". In this diagram, the fistree inputs are "fis1/a" and "fis2/d" and the output is "fis2/y".

Connections must satisfy the following conditions:

1. A fistree object must have at least one FIS input without any incoming connection and one FIS output without any outgoing connection.
2. A FIS input cannot have more than one incoming connection.
3. A FIS output can have more than one outgoing connection.
4. An input and output of the same FIS cannot be connected. In other words, you cannot create loops between connected FIS objects.
5. Symmetric connections cannot be specified between two inputs, ["fis1/a" "fis1/b"; "fis1/b" "fis1/a"] is not allowed. Either ["fis1/a" "fis1/b"] or ["fis1/b" "fis1/a"] can be specified.
6. Self-input loops are not allowed, ["fis1/a" "fis1/a"] cannot be specified.

Inputs — Inputs to the FIS tree

```
string array
```

Inputs to the FIS tree, specified as an array of strings. Inputs are automatically determined using the specified connections of the fistree object. FIS inputs with no incoming connections are included in Inputs. Update this property by updating the connections of the fistree object.

Outputs — Outputs of the FIS tree

```
string
```

Outputs of the FIS tree, specified as a string. Outputs are automatically determined using the specified connections of the fistree object. FIS outputs without any outgoing connections are included in Outputs. You can update this property after initial construction of the fistree object. Existing outputs can be removed or new outputs can be added. Outputs cannot be empty.

DisableStructuralChecks — Flag for disabling structural checks

```
false (default) | true
```

Flag for disabling structural checks, inputs, and outputs, specified as either false or true. Set DisableStructuralChecks to true to disable automatic updates of
connections, inputs, and outputs when a FIS is updated after construction of a \texttt{fistree} object. Disabling structural checks can produce an unexpected failure in the \texttt{evalfis} function.

**Object Functions**

- \texttt{evalfis} : Evaluate fuzzy inference system
- \texttt{getTunableSettings} : Obtain tunable settings from fuzzy inference system
- \texttt{getTunableValues} : Obtain values of tunable parameters from fuzzy inference system
- \texttt{setTunableValues} : Specify tunable parameter values of a fuzzy inference system

**Examples**

**Create a Tree of Connected Fuzzy Inference Systems**

Create a Mamdani fuzzy inference system and a Sugeno fuzzy inference system.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = sugfis('Name','fis2','NumInputs',2,'NumOutputs',1);
```

Define the desired connections between the two fuzzy inference systems.

```matlab
con1 = ["fis1/output1" "fis2/input1"];
con2 = ["fis1/input1" "fis1/input2"];
```

Create a tree of fuzzy inference systems.

```matlab
tree = fistree([fis1 fis2],[con1; con2])
```

```matlab
Tree = fistree with properties:

FIS: [1x2 FuzzyInferenceSystem]
Connections: [2x2 string]
Inputs: [2x1 string]
Outputs: "fis2/output1"
DisableStructuralChecks: 0
```

See 'getTunableSettings' method for parameter optimization.
Update Fuzzy Inference Systems in a FIS Tree

Create a FIS tree.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = mamfis('Name','fis2','NumInputs',2,'NumOutputs',1);

fisT = fistree([fis1 fis2],[]);
```

Display the FIS tree configuration.

```matlab
plotfis(fisT)
```

FIS Names:
- fis1
- fis2

Connections:
- []

Inputs:
- fis1/input1
- fis1/input2
- fis2/input1
- fis2/input2

Outputs:
- fis1/output1
- fis2/output1

Add FIS

Add fis3 to fisT.

```matlab
fis3 = mamfis('Name','fis3','NumInputs',2,'NumOutputs',1);
fisT.FIS(end+1) = fis3;
```

Add connections between fis1, fis2, and fis3.

```matlab
fisT.Connections = ["fis1/output1" "fis3/input1";"fis2/output1" "fis3/input2"];
```

Display the updated FIS tree configuration.

```matlab
plotfis(fisT)
```
FIS Names:
  fis1
  fis2
  fis3

Connections:
  From  To
  -----------  -----------
  fis1/output1  fis3/input1
  fis2/output1  fis3/input2

Inputs:
  fis1/input1
  fis1/input2
  fis2/input1
  fis2/input2

Outputs:
  fis3/output1

**Remove FIS**

Remove fis1 from fisT.

fisT.FIS(1) = [];

Display the updated FIS tree configuration.

plotfis(fisT)

FIS Names:
  fis2
  fis3

Connections:
  From  To
  -----------  -----------
  fis2/output1  fis3/input2

Inputs:
  fis2/input1
  fis2/input2
  fis3/input2
  fis3/input1
Use Same Value for Multiple Inputs of a FIS Tree

Create fis1, fis2, and fis3, each with two inputs and one input.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = mamfis('Name','fis2','NumInputs',2,'NumOutputs',1);
fis3 = mamfis('Name','fis3','NumInputs',2,'NumOutputs',1);
```

Create a connection between output 1 of fis1 and input 1 of fis3.

```matlab
con1 = ['fis1/output1' 'fis3/input1'];
```

Create a connection between output 1 of fis2 and input 2 of fis3.

```matlab
con2 = ['fis2/output1' 'fis3/input2'];
```

Create a connection between input 2 of fis1 and input 1 of fis2.

```matlab
con3 = ['fis1/input2' 'fis2/input1'];
```

Create the FIS tree.

```matlab
fuzzTree = fistree([fis1 fis2 fis3],[con1;con2;con3]);
```

Display the inputs of the FIS tree.

```matlab
fuzzTree Inputs
ans = 3x1 string array
    "fis1/input1"
    "fis1/input2"
    "fis2/input2"
```

Evaluate the fuzzy tree. Specify values for input 1 of fis1, input 2 of fis1, and input 2 of fis2. The value for input 2 of fis1 is also sent to input 1 of fis2.

```matlab
output = evalfis(fuzzTree,[0.8 0.25 0.7]);
```
**Update FIS Tree Outputs**

This example shows how to add or remove FIS tree outputs.

**Add Outputs**

Create fis1, fis2, and fis3, each with two inputs and one input.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = mamfis('Name','fis2','NumInputs',2,'NumOutputs',1);
fis3 = mamfis('Name','fis3','NumInputs',2,'NumOutputs',1);
```

Create a connection between output 1 of fis1 and input 1 of fis3.

```matlab
con1 = ['fis1/output1' 'fis3/input1'];
```

Create a connection between output 1 of fis2 and input 2 of fis3.

```matlab
con2 = ['fis2/output1' 'fis3/input2'];
```

Create the FIS tree.

```matlab
fuzzTree = fistree([fis1 fis2 fis3],[con1;con2]);
```

Display outputs of the FIS tree. By default, the only open FIS output (from fis3) is an output of the FIS tree.

```matlab
fuzzTree.Outputs
```

```matlab
ans =
    "fis3/output1"
```

Add the output of fis2 outputs to the tree output list.

```matlab
fuzzTree.Outputs(end+1) = "fis2/output1";
```

Display the updated output list of the FIS tree.

```matlab
fuzzTree.Outputs
```

```matlab
ans = 2x1 string array
    "fis3/output1"
    "fis2/output1"
```

Evaluate the FIS tree. The result contains the outputs from fis3 and fis2.
evalfis(fuzzTree,[0.5 0.2 0.8 0.45])
ans = 1x2
   0.1507    0.1579

Remove Outputs

Remove the first output from the list.

fuzzTree.Outputs(1) = [];

Display the updated output list of the FIS tree.

fuzzTree.Outputs
ans =
    "fis2/output1"

Evaluate the FIS tree again. The result now contains the output of only fis2.

evalfis(fuzzTree,[0.5 0.2 0.8 0.45])
ans = 0.1579

Create Incremental FIS Tree

This example shows construction of an incremental FIS tree. For more information on the types of fuzzy tree structures, see “Fuzzy Trees” on page 2-68.

Create fuzzy systems fis1, fis2, and fis3, each with two inputs and one input.

fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis1.Inputs(1).Name = "color";
fis1 Inputs(2).Name = "doors";
fis2 = mamfis('Name','fis2', 'NumInputs',2,'NumOutputs',1);
fis2 Inputs(2).Name = "power";
fis3 = mamfis('Name','fis3','NumInputs',2,'NumOutputs',1);
fis3 Inputs(2).Name = "autopilot";
fis3 Outputs(1).Name = "predition";

Create a connection between output 1 of fis1 and input 1 of fis2.
con1 = ["fis1/output1" "fis2/input1"];

Create a connection between output 1 of fis2 and input 1 of fis3.

con2 = ["fis2/output1" "fis3/input1"];

Create the FIS tree.

incTree = fistree([fis1 fis2 fis3],[con1;con2]);

Display the inputs of the FIS tree.

incTree.Inputs

ans = 4x1 string array
    "fis1/color"
    "fis1/doors"
    "fis2/power"
    "fis3/autopilot"

Display outputs of the FIS tree.

incTree.Outputs

ans =
    "fis3/predition"

**Create Cascaded FIS Tree**

This example shows construction of a cascaded FIS tree. For more information on the types of fuzzy tree structures, see “Fuzzy Trees” on page 2-68.

Create fuzzy systems fis1, fis2, fis3, and fis4, each with two inputs and one input.

fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis1.Inputs(1).Name = "dist_obs";
fis1.Inputs(2).Name = "angle_obs";
fis2 = mamfis('Name','fis2','NumInputs',2,'NumOutputs',1);
fis2.Inputs(1).Name = "dist_tar";
fis2.Inputs(2).Name = "angle_tar";
fis3 = mamfis('Name','fis3','NumInputs',2,'NumOutputs',1);
fis4 = mamfis('Name','fis4','NumInputs',2,'NumOutputs',1);
fis4.Inputs(2).Name = "preheading_robot";
fis4.Outputs(1).Name = "heading_robot";

Create a connection between output 1 of fis1 and input 1 of fis3.
con1 = ["fis1/output1" "fis3/input1"];

Create a connection between output 1 of fis2 and input 2 of fis3.
con2 = ["fis2/output1" "fis3/input2"];

Create a connection between output 1 of fis3 and input 1 of fis4.
con3 = ["fis3/output1" "fis4/input1"];

Create the FIS tree.
casTree = fistree([fis1 fis2 fis3 fis4],[con1;con2;con3]);

Display the inputs of the FIS tree.
casTree.Inputs
ans = 5x1 string array
   "fis1/dist_obs"
   "fis1/angle_obs"
   "fis2/dist_tar"
   "fis2/angle_tar"
   "fis4/preheading_robot"

Display the outputs of the FIS tree.
casTree.Outputs
ans =
   "fis4/heading_robot"

**Create Aggregated FIS Tree**

This example shows construction of an aggregated FIS tree. For more information on the types of fuzzy tree structures, see “Fuzzy Trees” on page 2-68.
Create fuzzy systems `fis1`, `fis2`, and `fis3`, each with two inputs and one output.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis1.Inputs(1).Name = "dist_obs";
fis1.Inputs(2).Name = "angle_obs";
fis2 = mamfis('Name','fis2','NumInputs',2,'NumOutputs',1);
fis2.Inputs(1).Name = "dist_tar";
fis2.Inputs(2).Name = "angle_tar";
fis3 = mamfis('Name','fis3','NumInputs',2,'NumOutputs',1);
fis3.Outputs(1).Name = "heading_robot";
```

Create a connection between output 1 of `fis1` and input 1 of `fis3`.

```matlab
con1 = "fis1/output1" "fis3/input1";
```

Create a connection between output 1 of `fis2` and input 2 of `fis3`.

```matlab
con2 = "fis2/output1" "fis3/input2";
```

Create the FIS tree.

```matlab
aggTree = fistree([fis1 fis2 fis3],[con1;con2]);
```

Display the inputs of the FIS tree.

```matlab
aggTree.Inputs
ans = 4x1 string array
"fis1/dist_obs"
"fis1/angle_obs"
"fis2/dist_tar"
"fis2/angle_tar"
```

Display the outputs of the FIS tree.

```matlab
aggTree.Outputs
ans =
"fis3/heading_robot"
```
Create and Evaluate Parallel FIS Tree

This example shows construction of a parallel FIS tree. For more information on the types of fuzzy tree structures, see “Fuzzy Trees” on page 2-68.

Create fuzzy systems fis1, fis2, and fis3, each with two inputs and one input.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = mamfis('Name','fis2','NumInputs',2,'NumOutputs',1);
```

Create the FIS tree such that all of the FIS objects are in parallel; that is, there are no interconnections and all the FIS outputs are FIS tree outputs.

```matlab
parTree = fistree([fis1 fis2],[]);
```

Display the inputs of the FIS tree.

```matlab
parTree.Inputs
```

```matlab
ans = 4x1 string array
    "fis1/input1"
    "fis1/input2"
    "fis2/input1"
    "fis2/input2"
```

Display the outputs of the FIS tree.

```matlab
parTree.Outputs
```

```matlab
ans = 2x1 string array
    "fis1/output1"
    "fis2/output1"
```

Evaluate the FIS tree.

```matlab
output = evalfis(parTree,[0.1 0.3 0.8 0.4]);
```

Generate the final output by summing the FIS tree outputs.

```matlab
finalOutput = sum(output);
```
See Also
mamfis | mamfistype2 | sugfis | sugfistype2 | tunefis

Topics
“Fuzzy Trees” on page 2-68
“Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2

Introduced in R2019a
fisvar

Fuzzy variable

Description

Use fisvar objects to represent the input and output variables in a fuzzy inference system (FIS). For more information on creating fuzzy inference systems, see mamfis, sugfis, mamfistype2, and sugfistype2.

Creation

Syntax

var = fisvar
var = fisvar(range)
var = fisvar('Name',name)
var = fisvar(range,'Name',name)

Description

var = fisvar creates a fuzzy variable with a default name, default range, and no membership functions. To change the variable properties, use dot notation.

var = fisvar(range) sets the Range property.

var = fisvar('Name',name) sets the Name property.

var = fisvar(range,'Name',name) sets both the Range and Name properties.

Properties

Name — Variable name

"var" (default) | string | character vector
Variable name, specified as a string or character vector.

**Range — Variable range**

[0 1] (default) | two-element vector

Variable range, specified as a two-element element vector where the first element is less than the second element. The first element specifies the lower bound of the range, and the second element specifies the upper bound of the range.

**MembershipFunctions — Membership functions**

[] (default) | vector of fismf objects | vector of fismftype2 objects

Membership functions, specified as a vector of fismf or fismftype2 objects. To add membership functions to a fuzzy variable:

- Use the `addMF` function.
- Create a vector of fismf objects, and assign it to `MembershipFunctions`.
- Create a vector of fismftype2 objects, and assign it to `MembershipFunctions`.

You can modify the properties of the membership functions using dot notation.

**Object Functions**

- `addMF` Add membership function to fuzzy variable
- `removeMF` Remove membership function from fuzzy variable

**Examples**

**Create Fuzzy Variable**

Create a fuzzy variable with default properties.

```matlab
var = fisvar;
```

To modify the properties of a `fisvar` object, use dot notation. For example, specify the range of the fuzzy variable to be from -5 to 5.

```matlab
var.Range = [-5 5];
```
Create Fuzzy Variable with Specified Range

Create a fuzzy variable with an input range from -10 to 10.

```matlab
var = fisvar([-10 10]);
```

Create Fuzzy Variable with Specified Name

Create a fuzzy variable with the name "speed".

```matlab
var = fisvar("Name","speed");
```

Add Membership Function to Fuzzy Variable

Create a fuzzy variable with a specified range.

```matlab
var = fisvar([0 1]);
```

Add a membership function to the variable, specifying a trapezoidal membership function, and set the membership function parameters.

```matlab
var = addMF(var,"trapmf",[-0.5 0 0.2 0.4]);
```

You can also specify the name of your membership when you add it to a fuzzy variable. For example, add a membership function called "large".

```matlab
var = addMF(var,"trapmf",[0.6 0.8 1 1.5],"Name","large");
```

View the membership functions.

```matlab
var.MembershipFunctions
ans =
    1x2 fismf array with properties:
         Type    Parameters    Name
```
Alternatively, you can add a default membership function to a fuzzy variable and set its parameters using dot notation.

```matlab
var = fisvar([0 1]);
var = addMF(var);
var.MembershipFunctions(1).Type = "trapmf";
var.MembershipFunctions(1).Parameters = [-0.5 0 0.2 0.4];
```

### Add Type-2 Membership Function to Fuzzy Variable

Create a fuzzy variable with a specified range. By default, this variable has no membership functions.

```matlab
var = fisvar([0 9]);
```

To add a type-2 membership function to a variable with no existing membership functions, specify either a LowerLag or LowerScale value for the membership function. For example specify a lower scale value.

```matlab
var = addMF(var,"trimf",[0 3 6],"LowerScale",1);
```

Once a variable contains a type-2 membership function, you can add additional type-2 membership functions without specifying one of these parameters.

```matlab
var = addMF(var,"trimf",[3 6 9]);
```

View the membership functions.

```matlab
var.MembershipFunctions
ans =
    1x2 fismftype2 array with properties:
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Upper Parameters</th>
<th>Lower Scale</th>
<th>Lower Lag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;mf1&quot;</td>
<td>&quot;trimf&quot;</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>&quot;mf2&quot;</td>
<td>&quot;trimf&quot;</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

**See Also**

fismf | fisftype2 | fisrule | mamfis | mamfistype2 | sugfis | sugfistype2

**Topics**

"Build Fuzzy Systems at the Command Line" on page 2-41

**Introduced in R2018b**
mamfis

Mamdani fuzzy inference system

Description

Use a mamfis object to represent a type-1 Mamdani fuzzy inference system (FIS).

As an alternative to a type-1 Mamdani system, you can create a:

- Type-1 Sugeno system using a sugfis object
- Type-2 Mamdani system using a mamfistype2 object
- Type-2 Sugeno system using a sugfistype2 object

For more information on the different types of fuzzy inference systems, see “Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2 and “Type-2 Fuzzy Inference Systems” on page 2-7.

Creation

To create a Mamdani FIS object, use one of the following methods:

- The mamfis function.
- If you have input and output training data (inputData and outputData, respectively), you can use the genfis function with the FCM clustering method.

```matlab
opt = genfisOptions('FCMClustering','FISType','mamdani');
fis = genfis(inputData,outputData,opt);
```
- If you have a .fis file for a Mamdani system, you can use the readfis function.

Syntax

```matlab
fis = mamfis
fis = mamfis(Name,Value)
```
**Description**

`fis = mamfis` creates a Mamdani FIS with default property values. To modify the properties of the fuzzy system, use dot notation.

`fis = mamfis(Name,Value)` specifies FIS configuration information or sets object properties using name-value pair arguments. You can specify multiple name-value pairs. Enclose names in quotes.

**Input Arguments**

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'NumInputs',2` configures the fuzzy system to have two input variables.

**NumInputs — Number of FIS inputs**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (default)</td>
<td>Nonnegative integer</td>
</tr>
</tbody>
</table>

Number of FIS inputs, specified as the comma-separated pair consisting of `'NumInputs'` and a nonnegative integer.

**NumInputMFs — Number of membership functions for each FIS input**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 (default)</td>
<td>Positive integer</td>
</tr>
</tbody>
</table>

Number of membership functions for each FIS input, specified as the comma-separated pair consisting of `'NumInputMFs'` and a positive integer.

**NumOutputs — Number of FIS outputs**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (default)</td>
<td>Nonnegative integer</td>
</tr>
</tbody>
</table>

Number of FIS outputs, specified as the comma-separated pair consisting of `'NumOutputs'` and a nonnegative integer.

**NumOutputMFs — Number of membership functions for each FIS output**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 (default)</td>
<td>Positive integer</td>
</tr>
</tbody>
</table>

Number of membership functions for each FIS output, specified as the comma-separated pair consisting of `'NumOutputMFs'` and a positive integer.
Number of membership functions for each FIS output, specified as the comma-separated pair consisting of 'NumOutputMFs' and a positive integer.

**MFType — Membership function type**

"trimf" (default) | "gaussmf"

Membership function type for both input and output variables, specified as the comma-separated pair consisting of "MFType" and either "trimf" (triangular MF) or "gaussmf" (Gaussian MF). For each input and output variable, the membership functions are uniformly distributed over the variable range with approximately 80% overlap in the MF supports.

**AddRules — Flag for automatically adding rules**

"allcombinations" (default) | "none"

Flag for automatically adding rules, specified as the comma-separated pair consisting of "AddRules" and one of the following:

- "allcombinations" — If both NumInputs and NumOutputs are greater than zero, create rules with antecedents that contain all input membership function combinations. Each rule consequent contains all the output variables and uses the first membership function of each output.
- "none" — Create a FIS without any rules.

**Properties**

**Name — FIS name**

"fis" (default) | string | character vector

FIS name, specified as a string or character vector.

**AndMethod — AND operator method**

"min" (default) | "prod" | string | character vector | function handle

AND operator method for combining fuzzified input values in a fuzzy rule antecedent, specified as one of the following:

- "min" — Minimum of fuzzified input values
- "prod" — Product of fuzzified input values
• String or character vector — Name of a custom AND function in the current working folder or on the MATLAB path
• Function handle — Custom AND function in the current working folder or on the MATLAB path

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**OrMethod — OR operator method**

"max" (default) | "probor" | string | character vector | function handle

OR operator method for combining fuzzified input values in a fuzzy rule antecedent, specified as one of the following:

• "max" — Maximum of fuzzified input values.
• "probor" — Probabilistic OR of fuzzified input values. For more information, see `probor`.
• String or character vector — Name of a custom OR function in the current working folder or on the MATLAB path.
• Function handle — Custom OR function in the current working folder or on the MATLAB path.

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**ImplicationMethod — Implication method**

"min" (default) | "prod" | string | character vector | function handle

Implication method for computing the consequent fuzzy set, specified as one of the following:

• "min" — Truncate the consequent membership function at the antecedent result value.
• "prod" — Scale the consequent membership function by the antecedent result value.
• String or character vector — Name of a custom implication function in the current working folder or on the MATLAB path.
• Function handle — Custom implication function in the current working folder or on the MATLAB path.

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on implication and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

AggregationMethod — Aggregation method
"max" (default) | "sum" | "probor" | string | character vector | function handle

Aggregation method for combining rule consequents, specified as one of the following:
• "max" — Maximum of consequent fuzzy sets
• "sum" — Sum of consequent fuzzy sets
• "probor" — Probabilistic OR of consequent fuzzy sets. For more information, see probor.
• String or character vector — Name of a custom aggregation function in the current working folder or on the MATLAB path
• Function handle — Custom aggregation function in the current working folder or on the MATLAB path

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on aggregation and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

DefuzzificationMethod — Defuzzification method
"centroid" (default) | "bisector" | "mom" | "lom" | "som" | string | character vector | function handle

Defuzzification method for computing crisp output values from the aggregated output fuzzy set, specified as one of the following:
• "centroid" — Centroid of the area under the output fuzzy set
• "bisector" — Bisector of the area under the output fuzzy set
- "mom" — Mean of the values for which the output fuzzy set is maximum
- "lom" — Largest value for which the output fuzzy set is maximum
- "som" — Smallest value for which the output fuzzy set is maximum
- String or character vector — Name of a custom defuzzification function in the current working folder or on the MATLAB path
- Function handle — Custom defuzzification function in the current working folder or on the MATLAB path

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on defuzzification and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**Inputs — FIS input variables**
vector of fisvar objects

FIS input variables, specified as a vector of fisvar objects. To add and remove input variables, use addInput and removeInput, respectively.

You can also create a vector of fisvar objects and assign it to Inputs using dot notation.

You can add membership functions to input variables using the addMF function.

**Outputs — FIS output variables**
vector of fisvar objects

FIS output variables, specified as a vector of fisvar objects. To add and remove output variables, use addOutput and removeOutput, respectively.

You can also create a vector of fisvar objects and assign it to Outputs using dot notation.

You can add membership functions to output variables using the addMF function.

**Rules — FIS rules**
vector of fisrule objects

FIS input variables, specified as a vector of fisrule objects. To add fuzzy rules, use the addRule function.
You can also create a vector of fisrule objects and assign it to Rules using dot notation.

To remove a rule, set the corresponding rule vector element to []. For example, to remove the tenth rule from the rule list, type:

```matlab
fis.Rules(10) = [];
```

**DisableStructuralChecks — Flag for disabling consistency checks**

false (default) | true

Flag for disabling consistency checks when property values change, specified as a logical value.

By default, when you change the value of a property of a mamfis object, the software verifies whether the new property value is consistent with the other object properties. These checks can affect performance, particularly when creating and updating fuzzy systems within loops.

To disable these checks, which results in faster FIS construction, set DisableStructuralChecks to true.

**Note** Disabling structural checks can result in an invalid mamfis object.

To reenable the consistency checks, first verify that the changes you made to the FIS are consistent and produce a valid mamfis object. Then, set DisableStructuralChecks to false. If the mamfis object is invalid, reenabling the consistency checks generates an error.

**Object Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addInput</td>
<td>Add input variable to fuzzy inference system</td>
</tr>
<tr>
<td>removeInput</td>
<td>Remove input variable from fuzzy inference system</td>
</tr>
<tr>
<td>addOutput</td>
<td>Add output variable to fuzzy inference system</td>
</tr>
<tr>
<td>removeOutput</td>
<td>Remove output variable from fuzzy inference system</td>
</tr>
<tr>
<td>addRule</td>
<td>Add rule to fuzzy inference system</td>
</tr>
<tr>
<td>addMF</td>
<td>Add membership function to fuzzy variable</td>
</tr>
<tr>
<td>removeMF</td>
<td>Remove membership function from fuzzy variable</td>
</tr>
<tr>
<td>evalfis</td>
<td>Evaluate fuzzy inference system</td>
</tr>
</tbody>
</table>
writeFIS  Save fuzzy inference system to file
convertToType2  Convert type-1 fuzzy inference system into type-2 fuzzy inference system

**Examples**

**Create Mamdani Fuzzy Inference System**

Create a Mamdani fuzzy inference system with default property values.

```matlab
fis = mamfis;
```

Modify the system properties using dot notation. For example, configure `fis` to use centroid defuzzification.

```matlab
fis.DefuzzificationMethod = "centroid";
```

Alternatively, you can specify one of more FIS properties when you create a fuzzy system. For example, create a Mamdani fuzzy system with specified AND and OR methods.

```matlab
fis = mamfis("AndMethod","prod","OrMethod","probor");
```

**Specify Number of Inputs and Outputs for Mamdani System**

Create a Mamdani fuzzy inference system with three inputs and one output.

```matlab
fis = mamfis("NumInputs",3,"NumOutputs",1)
fis =
mamfis with properties:
    Name: "fis"
    AndMethod: "min"
    OrMethod: "max"
    ImplicationMethod: "min"
    AggregationMethod: "max"
    DefuzzificationMethod: "centroid"
    Inputs: [1x3 fisvar]
    Outputs: [1x1 fisvar]
```
By default, the software creates a rule for each possible input combination.

**Alternative Functionality**

**App**

You can interactively create a Mamdani FIS using the Fuzzy Logic Designer app. You can then export the system to the MATLAB workspace.

**See Also**

fismf | fisrule | fisvar | sugfis

**Topics**

“Build Fuzzy Systems at the Command Line” on page 2-41

**Introduced in R2018b**
mamfistype2

Interval type-2 Mamdani fuzzy inference system

Description

Use a mamfistype2 object to represent an interval type-2 Mamdani fuzzy inference system (FIS).

As an alternative to a type-2 Mamdani system, you can create a:

• Type-2 Sugeno system using a sugfistype2 object
• Type-1 Mamdani system using a mamfis object
• Type-1 Sugeno system using a sugfis object

For more information on the different types of fuzzy inference systems, see “Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2 and “Type-2 Fuzzy Inference Systems” on page 2-7.

Creation

To create a type-2 Mamdani FIS object, use one of the following methods:

• The mamfistype2 function.
• If you have input and output training data (inputData and outputData, respectively), you can create a type-1 FIS using the genfis function with the FCM clustering method. You can then convert this FIS to a type-2 system using convertToType2.

```matlab
opt = genfisOptions('FCMClustering','FISType','mamdani');
fis1 = genfis(inputData,outputData,opt);
fis = convertToType2(fis1);
```
• If you have a .fis file for a type-2 Mamdani system, you can use the readfis function.
Syntax

\[
fis = \text{mamfistype2}
\]
\[
fis = \text{mamfistype2}(\text{Name},\text{Value})
\]

Description

\[
fis = \text{mamfistype2}
\] creates a type-2 Mamdani FIS with default property values. To modify the properties of the fuzzy system, use dot notation.

\[
fis = \text{mamfistype2}(\text{Name},\text{Value})
\] specifies FIS configuration information or sets object properties using name-value pair arguments. You can specify multiple name-value pairs. Enclose names in quotes.

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of \text{Name}, \text{Value} arguments. \text{Name} is the argument name and \text{Value} is the corresponding value. \text{Name} must appear inside quotes. You can specify several name and value pair arguments in any order as \text{Name1,Value1,...,NameN,ValueN}.

Example: 'NumInputs',2 configures the fuzzy system to have two input variables

**NumInputs — Number of FIS inputs**

\[
0 \text{ (default) } | \text{ nonnegative integer}
\]

Number of FIS inputs, specified as the comma-separated pair consisting of 'NumInputs' and a nonnegative integer.

**NumInputMFs — Number of membership functions for each FIS input**

\[
3 \text{ (default) } | \text{ positive integer}
\]

Number of membership functions for each FIS input, specified as the comma-separated pair consisting of 'NumInputMFs' and a positive integer.

**NumOutputs — Number of FIS outputs**

\[
0 \text{ (default) } | \text{ nonnegative integer}
\]

Number of FIS outputs, specified as the comma-separated pair consisting of 'NumOutputs' and a nonnegative integer.
Number of FIS outputs, specified as the comma-separated pair consisting of 'NumOutputs' and a nonnegative integer.

**NumOutputMFs — Number of membership functions for each FIS output**

3 (default) | positive integer

Number of membership functions for each FIS output, specified as the comma-separated pair consisting of 'NumOutputMFs' and a positive integer.

**MFType — Membership function type**

"trimf" (default) | "gaussmf"

Membership function type for both input and output variables, specified as the comma-separated pair consisting of "MFType" and either "trimf" (triangular MF) or "gaussmf" (Gaussian MF). For each input and output variable, the membership functions are uniformly distributed over the variable range with approximately 80% overlap in the MF supports.

**AddRules — Flag for automatically adding rules**

"allcombinations" (default) | "none"

Flag for automatically adding rules, specified as the comma-separated pair consisting of "AddRules" and one of the following:

- "allcombinations" — If both NumInputs and NumOutputs are greater than zero, create rules with antecedents that contain all input membership function combinations. Each rule consequent contains all the output variables and uses the first membership function of each output.
- "none" — Create a FIS without any rules.

**Properties**

**Name — FIS name**

"fis" (default) | string | character vector

FIS name, specified as a string or character vector.

**AndMethod — AND operator method**

"min" (default) | "prod" | string | character vector | function handle
AND operator method for combining fuzzified input values in a fuzzy rule antecedent, specified as one of the following:

- "min" — Minimum of fuzzified input values
- "prod" — Product of fuzzified input values
- String or character vector — Name of a custom AND function in the current working folder or on the MATLAB path
- Function handle — Custom AND function in the current working folder or on the MATLAB path

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**OrMethod — OR operator method**

"max" (default) | "probor" | string | character vector | function handle

OR operator method for combining fuzzified input values in a fuzzy rule antecedent, specified as one of the following:

- "max" — Maximum of fuzzified input values.
- "probor" — Probabilistic OR of fuzzified input values. For more information, see `probor`.
- String or character vector — Name of a custom OR function in the current working folder or on the MATLAB path.
- Function handle — Custom OR function in the current working folder or on the MATLAB path.

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**ImplicationMethod — Implication method**

"min" (default) | "prod" | string | character vector | function handle

Implication method for computing the consequent fuzzy set, specified as one of the following:
- "min" — Truncate the consequent membership function at the antecedent result value.
- "prod" — Scale the consequent membership function by the antecedent result value.
- String or character vector — Name of a custom implication function in the current working folder or on the MATLAB path.
- Function handle — Custom implication function in the current working folder or on the MATLAB path.

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on implication and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**AggregationMethod — Aggregation method**

"max" (default) | "sum" | "probor" | string | character vector | function handle

Aggregation method for combining rule consequents, specified as one of the following:

- "max" — Maximum of consequent fuzzy sets
- "sum" — Sum of consequent fuzzy sets
- "probor" — Probabilistic OR of consequent fuzzy sets. For more information, see *probor*.
- String or character vector — Name of a custom aggregation function in the current working folder or on the MATLAB path
- Function handle — Custom aggregation function in the current working folder or on the MATLAB path

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on aggregation and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**DefuzzificationMethod — Defuzzification method**

"centroid" (default)

Defuzzification method for computing crisp output values from the aggregated output fuzzy set. Type-2 Mamdani systems support only centroid defuzzification.
**Inputs — FIS input variables**

vector of fisvar objects

FIS input variables, specified as a vector of fisvar objects. To add and remove input variables, use addInput and removeInput, respectively.

You can also create a vector of fisvar objects and assign it to Inputs using dot notation.

You can add membership functions to input variables using the addMF function.

**Outputs — FIS output variables**

vector of fisvar objects

FIS output variables, specified as a vector of fisvar objects. To add and remove output variables, use addOutput and removeOutput, respectively.

You can also create a vector of fisvar objects and assign it to Outputs using dot notation.

You can add membership functions to output variables using the addMF function.

**Rules — FIS rules**

vector of fisrule objects

FIS input variables, specified as a vector of fisrule objects. To add fuzzy rules, use the addRule function.

You can also create a vector of fisrule objects and assign it to Rules using dot notation.

To remove a rule, set the corresponding rule vector element to []. For example, to remove the tenth rule from the rule list, type:

```matlab
fis.Rules(10) = [];```

**DisableStructuralChecks — Flag for disabling consistency checks**

false (default) | true

Flag for disabling consistency checks when property values change, specified as a logical value.

By default, when you change the value of a property of a mamfistype2 object, the software verifies whether the new property value is consistent with the other object
properties. These checks can affect performance, particularly when creating and updating fuzzy systems within loops.

To disable these checks, which results in faster FIS construction, set `DisableStructuralChecks` to `true`.

**Note** Disabling structural checks can result in an invalid `mamfistype2` object.

To reenable the consistency checks, first verify that the changes you made to the FIS are consistent and produce a valid `mamfistype2` object. Then, set `DisableStructuralChecks` to `false`. If the `mamfistype2` object is invalid, reenabling the consistency checks generates an error.

**TypeReductionMethod — Type-reduction method**

"karnikmendel" (default) | "ekm" | "iasc" | "eiasc" | string | function handle

Type-reduction method for converting a type-2 output fuzzy set to an interval type-1 fuzzy set, specified as one of the following:

- "karnikmendel" — Karnik-Mendel
- "ekm" — Enhanced Karnik-Mendel
- "iasc" — Iterative algorithm with stop condition
- "eiasc" — Enhanced iterative algorithm
- String — Name of a custom type-reduction function in the current working directory or on the MATLAB path.
- Function handle — Function handle to a custom type-reduction function in the current working folder or on the MATLAB path.

For more information on type reduction, see “Type-2 Fuzzy Inference Systems” on page 2-7.

**Object Functions**

- `addInput` Add input variable to fuzzy inference system
- `removeInput` Remove input variable from fuzzy inference system
- `addOutput` Add output variable to fuzzy inference system
- `removeOutput` Remove output variable from fuzzy inference system
addRule     Add rule to fuzzy inference system
addMF       Add membership function to fuzzy variable
removeMF    Remove membership function from fuzzy variable
evalfis     Evaluate fuzzy inference system
writeFIS    Save fuzzy inference system to file
convertToType1 Convert type-2 fuzzy inference system into type-1 fuzzy inference system

Examples

Create Type-2 Mamdani Fuzzy Inference System

Create a type-2 Mamdani fuzzy inference system with default property values.

\[
fis = \text{mamfistype2};
\]

Modify the system properties using dot notation. For example, set the type reduction method to use the enhanced Karnik-Mendel method.

\[
fis.\text{TypeReductionMethod} = "ekm";
\]

Alternatively, you can specify one of more FIS properties when you create a fuzzy system.

\[
fis = \text{mamfistype2}(\text{'TypeReductionMethod'},"ekm");
\]

Specify Number of Inputs and Outputs for Type-2 Mamdani System

Create a type-2 Mamdani fuzzy inference system with three inputs and one output.

\[
fis = \text{mamfis}(['\text{NumInputs}',3,'\text{NumOutputs}',1])
\]

\[
fis = 
\text{mamfis with properties:}
\]

- Name: "fis"
- AndMethod: "min"
- OrMethod: "max"
- ImplicationMethod: "min"
- AggregationMethod: "max"
DefuzzificationMethod: "centroid"
Inputs: [1x3 fisvar]
Outputs: [1x1 fisvar]
Rules: [1x27 fisrule]
DisableStructuralChecks: 0

See 'getTunableSettings' method for parameter optimization.

By default, the software creates a rule for each possible input combination.

See Also
fismftype2 | fisrule | fisvar

Topics
“Build Fuzzy Systems at the Command Line” on page 2-41

Introduced in R2019b
**MembershipFunctionSettings**

Tunable parameter settings for fuzzy membership functions

**Description**

A `MembershipFunctionSettings` object contains tunable parameter settings for a type-1 membership function. Using this object, you can specify the tunability settings for the parameters of the corresponding membership function.

For more information on the tunable settings of a type-2 membership function, see `MembershipFunctionSettingsType2`.

**Creation**

Create `MembershipFunctionSettings` objects using the `getTunableSettings` function with a `mamfis`, `sugfis`, or `fistree` object. The first and second outputs of `getTunableSettings` contain `VariableSettings` objects for input and output variables, respectively. If a `VariableSettings` object corresponds to a variable with type-1 membership functions, then its `MembershipFunctions` property contains `MembershipFunctionSettings` objects.

**Properties**

- **Parameters — Membership function parameter tunable settings**
  - NumericParameters object

  Membership function parameter tunable settings, specified as a `NumericParameters` object.

**Object Functions**

- `setTunable`  Set specified parameter settings as tunable or nontunable
Examples

Obtain Tunable Settings of Input and Output Variables from FIS

Create two fuzzy inference systems, and define the connection between the two.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = sugfis('Name','fis2','NumInputs',2,'NumOutputs',1);
con = ["fis1/output1" "fis2/input1"];```

Create a tree of fuzzy inference systems.

```matlab
tree = fistree([fis1 fis2],con);
```

Obtain the tunable settings of input and output variables of the fuzzy inference system.

```matlab
[in,out] = getTunableSettings(tree)
in=4×1 object
   4x1 VariableSettings array with properties:
       Type
       VariableName
       MembershipFunctions
       FISName

out=2×1 object
   2x1 VariableSettings array with properties:
       Type
       VariableName
       MembershipFunctions
       FISName
```

You can use dot notation to specify the tunable settings of input and output variables.

For the first membership function of input 1, set the first and third parameters to tunable.

```matlab
in(1).MembershipFunctions(1).Parameters.Free = [1 0 1];
```

For the first membership function of input 2, set the minimum parameter range to 0.

```matlab
in(2).MembershipFunctions(1).Parameters.Free = [0 0 0];
```
in(2).MembershipFunctions(1).Parameters.Minimum = 0;

For the first membership function of output 2, set the maximum parameter range to 1.
out(2).MembershipFunctions(1).Parameters.Maximum = 1;

**See Also**
NumericParameters | VariableSettings | getTunableSettings

**Introduced in R2019a**
**MembershipFunctionSettingsType2**

Tunable parameter settings for type-2 fuzzy membership functions

**Description**

A `MembershipFunctionSettingsType2` object contains tunable parameter settings for a type-2 membership function. Using this object, you can specify the tunability settings for the corresponding MF, including the upper MF parameters, the lower MF scale, and the lower MF lag.

For more information on the tunable settings of a type-1 membership function, see `MembershipFunctionSettings`.

**Creation**

Create `MembershipFunctionSettingsType2` objects using the `getTunableSettings` function with a `mamfistype2`, `sugfistype2`, or `fistree` object. The first and second outputs of `getTunableSettings` contain `VariableSettings` objects for input and output variables, respectively. If a `VariableSettings` object corresponds to a variable with type-2 membership functions, then its `MembershipFunctions` property contains `MembershipFunctionSettingsType2` objects.

**Properties**

**UpperParameters — Upper membership function parameter tunable settings**

`NumericParameters` object

Upper membership function parameter tunable settings, specified as a `NumericParameters` object.

**LowerScale — Lower membership function scale tunable settings**

`NumericParameters` object

Lower membership function scale tunable settings, specified as a `NumericParameters` object.
**LowerLag — Lower membership function lag tunable settings**
NumericParameters object

Lower membership function lag tunable settings, specified as a NumericParameters object.

**Object Functions**

setTunable Set specified parameter settings as tunable or nontunable

**Examples**

**Obtain Tunable Settings of Input and Output Variables from Type-2 FIS**

Create a type-2 fuzzy inference system.

```matlab
fis = mamfistype2('Name','fis1','NumInputs',2,'NumOutputs',1);
```

Obtain the tunable settings of the input and output variables of the fuzzy inference system.

```matlab
[in,out] = getTunableSettings(fis);
```

You can use dot notation to specify the tunable settings of the membership functions of the input and output variables.

For the first membership function of input 1, set the first and third upper membership function parameters as tunable.

```matlab
in(1).MembershipFunctions(1).UpperParameters.Free = [1 0 1];
```

For the first membership function of input 2, set the tunable range of the lower membership function scale to be between 0.7 and 0.9.

```matlab
in(2).MembershipFunctions(1).LowerScale.Minimum = 0.7;
in(2).MembershipFunctions(1).LowerScale.Maximum = 0.9;
```

For the first membership function of output 1, set the tunable range of the lower membership function lag to be between 0.1 and 0.4.
in(2).MembershipFunctions(1).LowerLag.Minimum = 0.1;
in(2).MembershipFunctions(1).LowerLag.Maximum = 0.4;

See Also
MembershipFunctionSettings | NumericParameters | VariableSettings | getTunableSettings

Introduced in R2019b
NumericParameters

Tunable numeric parameter settings of membership functions

Description

A NumericParameters object contains tunable settings for the numeric properties of a fuzzy membership function.

Creation

Create a NumericParameters object using the getTunableSettings function. The first and second outputs of getTunableSettings contain VariableSettings objects for input and output variables, respectively. The MembershipFunctions property of each VariableSettings object contains NumericParameters objects for specifying the tunable settings of the membership function properties.

Properties

Free — Parameter values available for tuning

vector of logical values | 1 | 0

Parameter values available for tuning, specified as one of the following:

- Vector of logical values when the NumericParameters contains tunable settings for the Parameters property of a type-1 membership function or the UpperParameters property of a type-2 membership function
- Logical 1 or 0 when the NumericParameters object contains tunable settings for either the LowerScale or LowerLag properties of a type-2 membership function

Minimum — Minimum parameter values

vector | scalar

Minimum parameter values, specified as one of the following:
• Vector when the `NumericParameters` contains tunable settings for the `Parameters` property of a type-1 membership function or the `UpperParameters` property of a type-2 membership function

• Scalar value when the `NumericParameters` object contains tunable settings for either the `LowerScale` or `LowerLag` properties of a type-2 membership function

**Maximum — Maximum parameter values**

`vector | scalar`

Maximum parameter values, specified as one of the following:

• Vector when the `NumericParameters` contains tunable settings for the `Parameters` property of a type-1 membership function or the `UpperParameters` property of a type-2 membership function

• Scalar value when the `NumericParameters` object contains tunable settings for either the `LowerScale` or `LowerLag` properties of a type-2 membership function.

**Examples**

**Obtain Tunable Settings of Input and Output Variables from FIS**

Create two fuzzy inference systems, and define the connection between the two.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = sugfis('Name','fis2','NumInputs',2,'NumOutputs',1);
con = ['fis1/output1' 'fis2/input1'];
```

Create a tree of fuzzy inference systems.

```matlab
tree = fistree([fis1 fis2],con);
```

Obtain the tunable settings of input and output variables of the fuzzy inference system.

```matlab
[in,out] = getTunableSettings(tree)
in=4x1 object
4x1 VariableSettings array with properties:
Type
VariableName```
MembershipFunctions
FISName

out=2×1 object
2x1 VariableSettings array with properties:
  Type
  VariableName
  MembershipFunctions
  FISName

You can use dot notation to specify the tunable settings of input and output variables.

For the first membership function of input 1, set the first and third parameters to tunable.
in(1).MembershipFunctions(1).Parameters.Free = [1 0 1];

For the first membership function of input 2, set the minimum parameter range to 0.
in(2).MembershipFunctions(1).Parameters.Minimum = 0;

For the first membership function of output 2, set the maximum parameter range to 1.
out(2).MembershipFunctions(1).Parameters.Maximum = 1;

**Obtain Tunable Settings of Input and Output Variables from Type-2 FIS**

Create a type-2 fuzzy inference system.
fis = mamfistype2('Name','fis1','NumInputs',2,'NumOutputs',1);

Obtain the tunable settings of the input and output variables of the fuzzy inference system.
[in,out] = getTunableSettings(fis);

You can use dot notation to specify the tunable settings of the membership functions of the input and output variables.

For the first membership function of input 1, set the first and third upper membership function parameters as tunable.
in(1).MembershipFunctions(1).UpperParameters.Free = [1 0 1];

For the first membership function of input 2, set the tunable range of the lower membership function scale to be between 0.7 and 0.9.

in(2).MembershipFunctions(1).LowerScale.Minimum = 0.7;
in(2).MembershipFunctions(1).LowerScale.Maximum = 0.9;

For the first membership function of output 1, set the tunable range of the lower membership function lag to be between 0.1 and 0.4.

in(2).MembershipFunctions(1).LowerLag.Minimum = 0.1;
in(2).MembershipFunctions(1).LowerLag.Maximum = 0.4;

See Also
MembershipFunctionSettings | VariableSettings | getTunableSettings

Introduced in R2019a
RuleSettings

Tunable parameter settings of fuzzy rules

Description

A RuleSettings object is created using the getTunableSettings function with a mamfis, sugfis, or fistree object. When the third output is specified, getTunableSettings returns tunable parameter settings of fuzzy rules. Specify the settings of the Antecedent and Consequent properties.

Creation

Create a RuleSettings object using getTunableSettings with three outputs.

Properties

FISName — Name of fuzzy inference system
string

This property is read-only.

Name of fuzzy inference system, specified as a string.

Index — Index of rule in fuzzy inference system
double

This property is read-only.

Index of rule in fuzzy inference system, specified as an integer.

Antecedent — Antecedent parameter settings of rule
ClauseParameters object
Antecedent parameter settings of rule, specified as a `ClauseParameters` object. Each antecedent parameter consists of the properties `AllowNot`, `AllowEmpty`, and `Free`. You can specify these properties.

**Consequent — Consequent parameter settings of rule**
ClauseParameters object

Consequent parameter settings of rule, specified as a `ClauseParameters` object. Each consequent parameter consists of the properties `AllowNot`, `AllowEmpty`, and `Free`. You can specify these properties.

## Object Functions

**setTunable**  
Set specified parameter settings as tunable or nontunable

## Examples

### Obtain Tunable Settings of Rules from FIS

Create two fuzzy inference systems, and define the connection between the two.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);
fis2 = sugfis('Name','fis2','NumInputs',2,'NumOutputs',1);
con = {'fis1/output1' 'fis2/input1'};

Create a tree of fuzzy inference systems.

tree = fistree([fis1 fis2],con);

Obtain the tunable settings of rules of the fuzzy inference system.

```matlab
[~,~,rule] = getTunableSettings(tree)
```

```matlab
rule=18x1 object
16x1 RuleSettings array with properties:
    Index
    Antecedent
    Consequent
    FISName
```
You can use dot notation to specify the tunable settings of rules.

For the first rule, do not tune input 1 membership function index and do not ignore output 1 membership function index.

```matlab
rule(1).Antecedent.Free(1) = false;
rule(1).Consequent.AllowEmpty(1) = false;
```

For the second rule, allow NOT logic for input 2 membership function index.

```matlab
rule(2).Antecedent.AllowNot(2) = true;
```

**See Also**

ClauseParameters | VariableSettings | getTunableSettings

**Introduced in R2019a**
sugfis

Sugeno fuzzy inference system

Description

Use a sugfis object to represent a type-1 Sugeno fuzzy inference system (FIS).

As an alternative to a type-1 Sugeno system, you can create a:

- Type-1 Mamdani system using a mamfis object
- Type-2 Sugeno system using a sugfistype2 object
- Type-2 Mamdani system using a mamfistype2 object

For more information on the different types of fuzzy inference systems, see “Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2 and “Type-2 Fuzzy Inference Systems” on page 2-7.

Creation

To create a Sugeno FIS object, use one of the following methods:

- The sugfis function.
- If you have input/output data, you can use the genfis function.
- If you have a .fis file for a Sugeno system, you can use the readfis function.
- Convert an existing Mamdani FIS to a Sugeno FIS using convertToSugeno.

Syntax

fis = sugfis
fis = sugfis(Name,Value)
**Description**

\[ \text{fis} = \text{sugfis} \] creates a Sugeno FIS with default property values. To modify the properties of the fuzzy system, use dot notation.

\[ \text{fis} = \text{sugfis}(\text{Name},\text{Value}) \] specifies FIS configuration information or sets object properties using name-value pair arguments. You can specify multiple name-value pairs. Enclose names in quotes.

**Input Arguments**

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'NumInputs',2` configures the fuzzy system to have two input variables

**NumInputs** — Number of FIS inputs

\[
\begin{align*}
&0 \text{ (default)} \mid \text{nonnegative integer} \\
&\text{Number of FIS inputs, specified as the comma-separated pair consisting of } '\text{NumInputs}' \text{ and a nonnegative integer.}
\end{align*}
\]

**NumInputMFs** — Number of membership functions for each FIS input

\[
\begin{align*}
&3 \text{ (default)} \mid \text{positive integer} \\
&\text{Number of membership functions for each FIS input, specified as the comma-separated pair consisting of } '\text{NumInputMFs}' \text{ and a positive integer.}
\end{align*}
\]

**NumOutputs** — Number of FIS outputs

\[
\begin{align*}
&0 \text{ (default)} \mid \text{nonnegative integer} \\
&\text{Number of FIS outputs, specified as the comma-separated pair consisting of } '\text{NumOutputs}' \text{ and a nonnegative integer.}
\end{align*}
\]

**NumOutputMFs** — Number of membership functions for each FIS output

\[
\begin{align*}
&3 \text{ (default)} \mid \text{positive integer} \\
&\text{Number of membership functions for each FIS output}
\end{align*}
\]
Number of membership functions for each FIS output, specified as the comma-separated pair consisting of 'NumOutputMFs' and a positive integer.

**MFType — Membership function type**

"trimf" (default) | "gaussmf"

Membership function type for input variables, specified as the comma-separated pair consisting of 'MFType' and either "trimf" (triangular MF) or "gaussmf" (Gaussian MF). For each input variable, the membership functions are uniformly distributed over the variable range with approximately 80% overlap in the MF supports.

Output membership functions are set to "constant" and uniformly distributed over the output variable ranges.

**AddRules — Flag for automatically adding rules**

"allcombinations" (default) | "none"

Flag for automatically adding rules, specified as the comma-separated pair consisting of "AddRules" and one of the following:

- "allcombinations" — If both NumInputs and NumOutputs are greater than zero, create rules with antecedents that contain all input membership function combinations. Each rule consequent contains all the output variables and uses the first membership function of each output.
- "none" — Create a FIS without any rules.

**Properties**

**Name — FIS name**

"fis" (default) | string | character vector

FIS name, specified as a string or character vector.

**AndMethod — AND operator method**

"prod" (default) | "min" | string | character vector | function handle

AND operator method for combining fuzzified input values in a fuzzy rule antecedent, specified as one of the following:

- "prod" — Product of fuzzified input values
• "min" — Minimum of fuzzified input values
• String or character vector — Name of a custom AND function in the current working folder or on the MATLAB path
• Function handle — Custom AND function in the current working folder or on the MATLAB path

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**OrMethod — OR operator method**

"probor" (default) | "max" | string | character vector | function handle

OR operator method for combining fuzzified input values in a fuzzy rule antecedent, specified as one of the following:

• "probor" — Probabilistic OR of fuzzified input values. For more information, see probor.
• "max" — Maximum of fuzzified input values.
• String or character vector — Name of a custom OR function in the current working folder or on the MATLAB path.
• Function handle — Custom OR function in the current working folder or on the MATLAB path.

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**ImplicationMethod — Implication method**

"prod" (default)

Implication method for computing consequent fuzzy set, specified as "prod". Sugeno systems always use the "prod" implication method, which scales the consequent membership function by the antecedent result value.

For more information on implication and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.
**AggregationMethod — Aggregation method**

"sum" (default)

Aggregation method for combining rule consequents, specified as "sum". Sugeno systems always use the "sum" aggregation method, which is the sum of the consequent fuzzy sets.

For more information on aggregation and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**DefuzzificationMethod — Defuzzification method**

"wtaver" (default) | "wtsum"

Defuzzification method for computing crisp output values from the aggregated output fuzzy set, specified as one of the following:

- "wtaver" — Weighted average of all rule outputs
- "wtsum" — Weighted sum of all rule outputs

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on defuzzification and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**Inputs — FIS input variables**

vector of fisvar objects

FIS input variables, specified as a vector of fisvar objects. To add and remove input variables, use addInput and removeInput, respectively. You can modify the properties of the input variables using dot notation.

You can also create a vector of fisvar objects and assign it to Inputs using dot notation.

You can add membership functions to input variables using the addMF function.

**Outputs — FIS output variables**

vector of fisvar objects

FIS output variables, specified as a vector of fisvar objects. To add and remove output variables, use addOutput and removeOutput, respectively.
You can also create a vector of fisvar objects and assign it to Outputs using dot notation.

You can add membership functions to output variables using the addMF function.

**Rules — FIS rules**
vector of fisrule objects

FIS input variables, specified as a vector of fisrule objects. To add fuzzy rules, use the addRule function.

You can also create a vector of fisrule objects and assign it to Rules using dot notation.

To remove a rule, set the corresponding rule vector element to []. For example, to remove the tenth rule from the rule list, type:

```plaintext
fis.Rules(10) = [];
```

**DisableStructuralChecks — Flag for disabling consistency checks**
false (default) | true

Flag for disabling consistency checks when property values change, specified as a logical value.

By default, when you change the value of a property of a sugfis object, the software verifies whether the new property value is consistent with the other object properties. These checks can affect performance, particularly when creating and updating fuzzy systems within loops.

To disable these checks, which results in faster FIS construction, set DisableStructuralChecks to true.

**Note** Disabling structural checks can result in an invalid sugfis object.

To reenable the consistency checks, first verify that the changes you made to the FIS are consistent and produce a valid sugfis object. Then, set DisableStructuralChecks to false. If the sugfis object is invalid, reenabling the consistency checks generates an error.
### Object Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addInput</td>
<td>Add input variable to fuzzy inference system</td>
</tr>
<tr>
<td>removeInput</td>
<td>Remove input variable from fuzzy inference system</td>
</tr>
<tr>
<td>addOutput</td>
<td>Add output variable to fuzzy inference system</td>
</tr>
<tr>
<td>removeOutput</td>
<td>Remove output variable from fuzzy inference system</td>
</tr>
<tr>
<td>addRule</td>
<td>Add rule to fuzzy inference system</td>
</tr>
<tr>
<td>addMF</td>
<td>Add membership function to fuzzy variable</td>
</tr>
<tr>
<td>removeMF</td>
<td>Remove membership function from fuzzy variable</td>
</tr>
<tr>
<td>evalfis</td>
<td>Evaluate fuzzy inference system</td>
</tr>
<tr>
<td>writeFIS</td>
<td>Save fuzzy inference system to file</td>
</tr>
<tr>
<td>convertToType2</td>
<td>Convert type-1 fuzzy inference system into type-2 fuzzy inference system</td>
</tr>
</tbody>
</table>

### Examples

#### Create Sugeno Fuzzy Inference System

Create a Sugeno fuzzy inference system with default property values.

```matlab
fis = sugfis;
```

Modify the system properties using dot notation. For example, configure `fis` to use weighted-sum defuzzification.

```matlab
fis.DefuzzicationMethod = "wtsum";
```

Alternatively, you can specify one of more FIS properties when you create a fuzzy system. For example, create a Sugeno fuzzy system with specified AND and OR methods.

```matlab
fis = sugfis("AndMethod","min","OrMethod","max");
```

#### Specify Number of Inputs and Outputs for Sugeno System

Create a Sugeno fuzzy inference system with three inputs and one output.

```matlab
fis = sugfis("NumInputs",3,"NumOutputs",1)
```
Create Sugeno FIS with Linear Output Membership Functions

Load a Sugeno FIS from a file.

```matlab
fis = readfis('sugeno1');
```

The output variable has two membership functions. View the properties of the first membership function.

```matlab
fis.Outputs(1).MembershipFunctions(1)
```

```matlab
ans =
    fismf with properties:
        Type: "linear"
                Parameters: [-1 -1]
                Name: "line1"
```

View the properties of the second membership function.

```matlab
fis.Outputs(1).MembershipFunctions(2)
```
ans =
    fismf with properties:
        Type: "linear"
        Parameters: [1 -1]
        Name: "line2"

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

fis.Rules
ans =
    1x2 fisrule array with properties:
        Description
        Antecedent
        Consequent
        Weight
        Connection

Details:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;input==low =&gt; output=line1 (1)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;input==high =&gt; output=line2 (1)&quot;</td>
</tr>
</tbody>
</table>

Plot the input membership functions of this system. The low membership function generally refers to input values less than zero, while high refers to values greater than zero.

plotmf(fis,'input',1)
Plot the output surface for this FIS.

gensurf(fis)
The overall fuzzy system output switches smoothly from the line called line1 to the line called line2.

**Alternative Functionality**

**App**

You can interactively create a Sugeno FIS using the **Fuzzy Logic Designer** or **Neuro-Fuzzy Designer** apps. You can then export the system to the MATLAB workspace.
See Also
fismf | fisrule | fisvar | mamfis

Topics
“Build Fuzzy Systems at the Command Line” on page 2-41

Introduced in R2018b
sugfistype2

Interval type-2 Sugeno fuzzy inference system

Description

Use a sugfistype2 object to represent an interval type-2 Sugeno fuzzy inference system (FIS).

As an alternative to a type-2 Sugeno system, you can create a:

• Type-2 Mamdani system using a mamfistype2 object
• Type-1 Sugeno system using a sugfis object
• Type-1 Mamdani system using a mamfis object

For more information on the different types of fuzzy inference systems, see “Mamdani and Sugeno Fuzzy Inference Systems” on page 2-2 and “Type-2 Fuzzy Inference Systems” on page 2-7.

Creation

To create a type-2 Sugeno FIS object, use one of the following methods:

• The sugfistype2 function.
• If you have input/output data, you can use the genfis function. You can then convert this FIS to a type-2 system using convertToType2.
• If you have a .fis file for a Sugeno system, you can use the readfis function.
• Convert an existing type-2 Mamdani FIS to a Sugeno FIS using convertToSugeno.

Syntax

fis = sugfistype2
fis = sugfistype2(Name,Value)
Description

\( \text{fis} = \text{sugfistype2} \) creates a type-2 Sugeno FIS with default property values. To modify the properties of the fuzzy system, use dot notation.

\( \text{fis} = \text{sugfistype2}(\text{Name},\text{Value}) \) specifies FIS configuration information or sets object properties using name-value pair arguments. You can specify multiple name-value pairs. Enclose names in quotes.

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of \text{Name},\text{Value} arguments. \text{Name} is the argument name and \text{Value} is the corresponding value. \text{Name} must appear inside quotes. You can specify several name and value pair arguments in any order as \text{Name1,Value1,...,NameN,ValueN}.

Example: ‘NumInputs’,2 configures the fuzzy system to have two input variables

\text{NumInputs — Number of FIS inputs}

\( 0 \) (default) | nonnegative integer

Number of FIS inputs, specified as the comma-separated pair consisting of ‘NumInputs’ and a nonnegative integer.

\text{NumInputMFs — Number of membership functions for each FIS input}

\( 3 \) (default) | positive integer

Number of membership functions for each FIS input, specified as the comma-separated pair consisting of ‘NumInputMFs’ and a positive integer.

\text{NumOutputs — Number of FIS outputs}

\( 0 \) (default) | nonnegative integer

Number of FIS outputs, specified as the comma-separated pair consisting of ‘NumOutputs’ and a nonnegative integer.

\text{NumOutputMFs — Number of membership functions for each FIS output}

\( 3 \) (default) | positive integer
Number of membership functions for each FIS output, specified as the comma-separated pair consisting of 'NumOutputMFs' and a positive integer.

MFType — Membership function type
"trimf" (default) | "gaussmf"

Membership function type for input variables, specified as the comma-separated pair consisting of 'MFType' and either "trimf" (triangular MF) or "gaussmf" (Gaussian MF). For each input variable, the membership functions are uniformly distributed over the variable range with approximately 80% overlap in the MF supports.

Output membership functions are set to "constant" and uniformly distributed over the output variable ranges.

AddRules — Flag for automatically adding rules
"allcombinations" (default) | "none"

Flag for automatically adding rules, specified as the comma-separated pair consisting of "AddRules" and one of the following:

- "allcombinations" — If both NumInputs and NumOutputs are greater than zero, create rules with antecedents that contain all input membership function combinations. Each rule consequent contains all the output variables and uses the first membership function of each output.
- "none" — Create a FIS without any rules.

Properties

Name — FIS name
"fis" (default) | string | character vector

FIS name, specified as a string or character vector.

AndMethod — AND operator method
"prod" (default) | "min" | string | character vector | function handle

AND operator method for combining fuzzified input values in a fuzzy rule antecedent, specified as one of the following:

- "prod" — Product of fuzzified input values
• "min" — Minimum of fuzzified input values
• String or character vector — Name of a custom AND function in the current working folder or on the MATLAB path
• Function handle — Custom AND function in the current working folder or on the MATLAB path

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**OrMethod — OR operator method**

"probor" (default) | "max" | string | character vector | function handle

OR operator method for combining fuzzified input values in a fuzzy rule antecedent, specified as one of the following:

• "probor" — Probabilistic OR of fuzzified input values. For more information, see probor.
• "max" — Maximum of fuzzified input values.
• String or character vector — Name of a custom OR function in the current working folder or on the MATLAB path.
• Function handle — Custom OR function in the current working folder or on the MATLAB path.

For more information on using custom functions, see “Build Fuzzy Systems Using Custom Functions” on page 2-52.

For more information on fuzzy operators and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**ImplicationMethod — Implication method**

"prod" (default)

Implication method for computing consequent fuzzy set, specified as "prod". Sugeno systems always use the "prod" implication method, which scales the consequent membership function by the antecedent result value.

For more information on implication and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.
**AggregationMethod — Aggregation method**

"sum" (default)

Aggregation method for combining rule consequents, specified as "sum". Sugeno systems always use the "sum" aggregation method, which is the sum of the consequent fuzzy sets.

For more information on aggregation and the fuzzy inference process, see “Fuzzy Inference Process” on page 1-28.

**DefuzzificationMethod — Defuzzification method**

"wtaver" (default)

Defuzzification method for computing crisp output values from the aggregated output fuzzy set. Type-2 Sugeno systems support only weighted-average defuzzification.

**Inputs — FIS input variables**

vector of fisvar objects

FIS input variables, specified as a vector of fisvar objects. To add and remove input variables, use `addInput` and `removeInput`, respectively. You can modify the properties of the input variables using dot notation.

You can also create a vector of fisvar objects and assign it to `Inputs` using dot notation.

You can add membership functions to input variables using the `addMF` function.

**Outputs — FIS output variables**

vector of fisvar objects

FIS output variables, specified as a vector of fisvar objects. To add and remove output variables, use `addOutput` and `removeOutput`, respectively.

You can also create a vector of fisvar objects and assign it to `Outputs` using dot notation.

You can add membership functions to output variables using the `addMF` function.

**Rules — FIS rules**

vector of fisrule objects

FIS input variables, specified as a vector of fisrule objects. To add fuzzy rules, use the `addRule` function.
You can also create a vector of fisrule objects and assign it to Rules using dot notation.

To remove a rule, set the corresponding rule vector element to []. For example, to remove the tenth rule from the rule list, type:

```matlab
fis.Rules(10) = [];
```

**DisableStructuralChecks — Flag for disabling consistency checks**

*false (default) | true*

Flag for disabling consistency checks when property values change, specified as a logical value.

By default, when you change the value of a property of a sugfistype2 object, the software verifies whether the new property value is consistent with the other object properties. These checks can affect performance, particularly when creating and updating fuzzy systems within loops.

To disable these checks, which results in faster FIS construction, set DisableStructuralChecks to true.

**Note** Disabling structural checks can result in an invalid sugfistype2 object.

To reenable the consistency checks, first verify that the changes you made to the FIS are consistent and produce a valid sugfistype2 object. Then, set DisableStructuralChecks to false. If the sugfistype2 object is invalid, reenabling the consistency checks generates an error.

**TypeReductionMethod — Type-reduction method**

*"karnikmendel" (default) | "ekm" | "iasc" | "eiasc" | string | function handle*

Type-reduction method for converting a type-2 output fuzzy set to an interval type-1 fuzzy set, specified as one of the following:

- "karnikmendel" — Karnik-Mendel
- "ekm" — Enhanced Karnik-Mendel
- "iasc" — Iterative algorithm with stop condition
- "eiasc" — Enhanced iterative algorithm
• String — Name of a custom type-reduction function in the current working directory or on the MATLAB path.

• Function handle — Function handle to a custom type-reduction function in the current working folder or on the MATLAB path.

For more information on type reduction, see “Type-2 Fuzzy Inference Systems” on page 2-7.

**Object Functions**

- `addInput` — Add input variable to fuzzy inference system
- `removeInput` — Remove input variable from fuzzy inference system
- `addOutput` — Add output variable to fuzzy inference system
- `removeOutput` — Remove output variable from fuzzy inference system
- `addRule` — Add rule to fuzzy inference system
- `addMF` — Add membership function to fuzzy variable
- `removeMF` — Remove membership function from fuzzy variable
- `evalfis` — Evaluate fuzzy inference system
- `writeFIS` — Save fuzzy inference system to file
- `convertToType1` — Convert type-2 fuzzy inference system into type-1 fuzzy inference system

**Examples**

**Create Type-2 Sugeno Fuzzy Inference System**

Create a type-2 Sugeno fuzzy inference system with default property values.

```
fis = sugfistype2;
```

Modify the system properties using dot notation. For example, set the type reduction method to use the enhanced Karnik-Mendel method.

```
fis.TypeReductionMethod = "ekm";
```

Alternatively, you can specify one of more FIS properties when you create a fuzzy system.

```
fis = sugfistype2('TypeReductionMethod','ekm');
```
Specify Number of Inputs and Outputs for Type-2 Sugeno System

Create a type-2 Sugeno fuzzy inference system with three inputs and one output. A type-2 Sugeno system uses type-2 membership functions only for its input variables.

```matlab
fis = sugfistype2("NumInputs",3,"NumOutputs",1)
```

```
fis =
    sugfistype2 with properties:
        Name: "fis"
        AndMethod: "prod"
        OrMethod: "probor"
        ImplicationMethod: "prod"
        AggregationMethod: "sum"
        DefuzzificationMethod: "wtaver"
        Inputs: [1x3 fisvar]
        Outputs: [1x1 fisvar]
        Rules: [1x27 fisrule]
        DisableStructuralChecks: 0
        TypeReductionMethod: "karnikmendel"

    See 'getTunableSettings' method for parameter optimization.
```

By default, the software creates a rule for each possible input combination.

See Also
fismftype2 | fisrule | fisvar

Topics
“Build Fuzzy Systems at the Command Line“ on page 2-41

Introduced in R2019b
**tunefisOptions**

Option set for `tunefis` function

**Description**

Use a `tunefisOptions` object to specify options for tuning fuzzy systems. Specify options such as the optimization method, optimization type, and distance metric for cost measurement. Then, use this object as an input for the `tunefis` function.

**Creation**

**Syntax**

```matlab
opt = tunefisOptions
opt = tunefisOptions(Name,Value)
```

**Description**

`opt = tunefisOptions` creates a default option set for tuning a fuzzy system using `tunefis`. Use dot notation to modify the properties of this option set for your specific application.

`opt = tunefisOptions(Name,Value)` creates an option set with the specified “Properties” on page 9-100 using one or more name-value pair arguments.

**Properties**

**Method — Tuning algorithm**

- `'ga'` (default)
- `'particleswarm'`
- `'patternsearch'`
- `'simulannealbnd'`
- `'anfis'`

Tuning algorithm, specified as one of the following:
• 'ga' — genetic algorithm
• 'particleswarm' — particle swarm
• 'patternsearch' — pattern search
• 'simulannealbnd' — simulated annealing algorithm
• 'anfis' — adaptive neuro-fuzzy

These tuning algorithms use solvers from the “Global Optimization Toolbox”, except for 'anfis'. The 'MethodOptions' property differs for each algorithm, and corresponds to the options input argument for the respective solver. If you specify MethodOptions without specifying algorithm name, then Method is determined based on MethodOptions.

The 'anfis' tuning method supports tuning only type-1 Sugeno fuzzy inference systems with one output variable.

**MethodOptions — Tuning algorithm options**

Options created using `optimoptions`

Tuning algorithm options, specified as an option object for the specified tuning algorithm Method. This property differs for each algorithm and is created using `optimoptions`. If MethodOptions is not specified, it is created according to Method. Use dot notation to modify options in MethodOptions.

**OptimizationType — Type of optimization**

'tuning' (default) | 'learning'

Type of optimization, specified as one of the following:

• 'tuning'
• 'learning'

When tuning is selected, the existing input, output, and rule parameters are optimized without learning new rules. When learning is selected, new rules are added based on NumMaxRules. The anfis algorithm supports only tuning.

**NumMaxRules — Maximum number of rules in FIS**

inf (default) | integer

Maximum number of rules in FIS after optimization, specified as an integer. The number of rules in a FIS (after optimization) may be less than NumMaxRules since duplicate rules
with same antecedent values are removed from the rule base. Default value is \( \text{inf} \), which indicates existing rules are used when \( \text{OptimizationType} \) is 'tuning' and maximum number of possible rules are used when \( \text{OptimizationType} \) is 'learning'. For a \( \text{fistree} \) object, \( \text{NumMaxRules} \) indicates that the number of rules in each FIS of a \( \text{fistree} \) is less than or equal to the specified value. \( \text{anfis} \) does not support rule parameter optimization.

**IgnoreInvalidParameters — Invalid parameter use**

true (default) | false

Invalid parameter use, specified as either true or false. If true, the \( \text{tunefis} \) function ignores invalid parameter values generated in the tuning process. Default value is true. \( \text{anfis} \) ignores this parameter value.

**DistanceMetric — Type of distance metric**

'rmse' (default) | 'norm1' | 'norm2'

Type of distance metric used for cost measurement of the optimized parameter values with respect to the training data, specified as 'rmse', 'norm1', or 'norm2'. \( \text{anfis} \) supports only 'rmse'.

**UseParallel — Parallel computation**

false (default) | true

Parallel computation, specified as either true or false. If true, \( \text{tunefis} \) function uses parallel computation in the optimization process. \( \text{anfis} \) does not support parallel optimization.

### Examples

**Specify Options for FIS Tuning**

Create a default option set using the particle swarm tuning algorithm.

```matlab
opt = tunefisOptions("Method","particleswarm")
```

```
opt =
    tunefisOptions with properties:
        Method: "particleswarm"
```
You can modify the options using dot notation. For example, set the maximum number of
iterations to 20.

```
opt.MethodOptions.MaxIterations = 20;
```

You can also specify other options when creating the option set. In this example, set the
OptimizationType to "learning" to learn new rules.

```
opt2 = tunefisOptions("Method","particleswarm","OptimizationType","learning")
```

```
opt2 =
    tunefisOptions with properties:
        Method: "particleswarm"
        MethodOptions: [1x1 optim.options.Particleswarm]
        OptimizationType: "learning"
        NumMaxRules: Inf
        IgnoreInvalidParameters: 1
        DistanceMetric: "rmse"
        UseParallel: 0
```

See Also
getTunableSettings | tunefis

Introduced in R2019a
VariableSettings

Tunable parameter settings of fuzzy variables

Description

A VariableSettings object contains tunable parameter settings for either an input or output variable of a fuzzy inference system. Using this object, you can specify the tunability settings for the membership functions of the corresponding variable.

Creation

Create a VariableSettings object using the getTunableSettings function. The first and second outputs of getTunableSettings contain VariableSettings objects for input and output variables, respectively.

Properties

FISName — Name of fuzzy inference system
string

This property is read-only.

Name of fuzzy inference system, specified as a string.

Type — Type of variable
"input" | "output"

This property is read-only.

Type of variable, specified as either "input" or "output" for input and output variables, respectively.

VariableName — Name of variable
string
This property is read-only.

Name of variable, specified as a string.

**MembershipFunctions — Membership function settings**
MembershipFunctionSettings object | MembershipFunctionSettingsType2 object

Membership function settings, specified as one of the following:

- `MembershipFunctionSettings` object when the corresponding variable contains type-1 membership functions
- `MembershipFunctionSettingsType2` object when the corresponding variable contains type-2 membership functions

**Object Functions**
`setTunable`  Set specified parameter settings as tunable or nontunable

**Examples**

**Obtain Tunable Settings of Input and Output Variables from FIS**

Create two fuzzy inference systems, and define the connection between the two.

```matlab
fis1 = mamfis('Name','fis1','NumInputs',2,'NumOutputs',1);  
fis2 = sugfis('Name','fis2','NumInputs',2,'NumOutputs',1);  
con = ['fis1/output1' 'fis2/input1'];
```

Create a tree of fuzzy inference systems.

```matlab
tree = fistree([fis1 fis2],con);
```

Obtain the tunable settings of input and output variables of the fuzzy inference system.

```matlab
[in,out] = getTunableSettings(tree)
in=4x1 object
  4x1 VariableSettings array with properties:
    Type
```
VariableName
MembershipFunctions
FISName

out=2x1 object
2x1 VariableSettings array with properties:

  Type
  VariableName
  MembershipFunctions
  FISName

You can use dot notation to specify the tunable settings of input and output variables.

For the first membership function of input 1, set the first and third parameters to tunable.
in(1).MembershipFunctions(1).Parameters.Free = [1 0 1];

For the first membership function of input 2, set the minimum parameter range to 0.
in(2).MembershipFunctions(1).Parameters.Minimum = 0;

For the first membership function of output 2, set the maximum parameter range to 1.
out(2).MembershipFunctions(1).Parameters.Maximum = 1;

See Also
MembershipFunctionSettings | MembershipFunctionSettingsType2 |
RuleSettings | getTunableSettings

Introduced in R2019a
Blocks — Alphabetical List
**Diff. Sigmoidal MF**

Difference of two sigmoids membership function in Simulink software

![Diff. Sigmoidal MF](image)

**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.

**Extended Capabilities**

**C/C++ Code Generation**
Generate C and C++ code using Simulink® Coder™.

**See Also**

dsigmf

**Introduced before R2006a**
Fuzzy Logic Controller

Evaluate fuzzy inference system

Library: Fuzzy Logic Toolbox

Description

The Fuzzy Logic Controller block implements a fuzzy inference system (FIS) in Simulink. You specify the FIS to evaluate using the FIS name parameter.

For more information on fuzzy inference, see “Fuzzy Inference Process” on page 1-28.

To display the fuzzy inference process in the Rule Viewer during simulation, use the Fuzzy Logic Controller with Ruleviewer block.

Ports

Input

in — Input signal

scalar | vector

For a single-input fuzzy inference system, the input is a scalar signal. For a multi-input fuzzy system, combine the inputs into a vector signal using blocks such as:

- Mux
- Vector Concatenate
- Bus Creator
Output

**out** — Defuzzified output signal

```
scalar | vector
```

For a single-output FIS, the output is a scalar signal. For a multi-output FIS, the output is a vector signal. To split system outputs into scalar signals, use the Demux block.

**fi** — Fuzzified input values

```
matrix
```

Fuzzified input values, obtained by evaluating the input membership functions of each rule at the current input values.

For a type-1 FIS, `fi` is an $N_R$-by-$N_U$ matrix signal, where $N_R$ is the number of FIS rules. Element $(i,j)$ of `fi` is the value of the input membership function for the $j$th input in the $i$th rule.

For a type-2 FIS, `fi` is an $N_R$-by-$(2*N_U)$ matrix signal. The first $N_U$ columns contain the fuzzified values of the upper membership function for each rule, and the last $N_U$ columns contain the fuzzified values from the lower membership functions.

For more information on fuzzifying input values, see “Fuzzify Inputs” on page 1-29.

**Dependencies**

To enable this port, select the **Fuzzified inputs (fi)** parameter.

**rfs** — Rule firing strengths

```
column vector
```

Rule firing strengths, obtained by evaluating the antecedent of each rule; that is, applying the fuzzy operator to the values of the fuzzified inputs.

For a type-1 FIS, `rfs` is a column vector signal of length $N_R$, where $N_R$ is the number of rules, and element $i$ is the firing strength of the $i$th rule.

For a type-2 FIS, `rfs` is an $N_R$-by-2 matrix signal. The first column contains the rule firing strengths generated using upper membership functions, and the second column contains the rule firing strengths generated using lower membership functions.

For more information on applying fuzzy operators, see “Apply Fuzzy Operator” on page 1-30.
Dependencies

To enable this port, select the **Rule firing strengths (rfs)** parameter.

**ro — Rule outputs**

matrix

Rule outputs, obtained by applying the rule firing strengths to the output membership functions using the implication method specified in the FIS.

For a type-1 Mamdani FIS, $ro$ is an $N_S$-by-$(N_R N_Y)$ matrix signal, where $N_R$ is the number of rules, $N_Y$ is the number of outputs, and $N_S$ is the number of sample points used for evaluating output variable ranges. Each column of $ro$ contains the output fuzzy set for one rule. The first $N_R$ columns contain the rule outputs for the first output variable, the next $N_R$ columns correspond to the second output variable, and so on.

For a type-2 Mamdani FIS, $ro$ is an $N_S$-by-$(2 N_R N_Y)$ matrix signal. The first $N_R N_Y$ columns contain the rule outputs generated using upper membership functions, and the last $N_R N_Y$ columns contain the rule outputs generated using lower membership functions.

For a type-1 Sugeno system, each rule output is a scalar value. In this case, $ro$ is an $N_R$-by-$N_Y$ matrix signal. Element $(j,k)$ of $ro$ is the value of the $k$th output variable for the $j$th rule.

For a type-2 Sugeno system, $ro$ is an $N_R$-by-$(3 N_Y)$ array. The first $N_Y$ columns contain the rule output levels. The next $N_Y$ columns contain the corresponding rule firing strengths generated using upper membership functions. The last $N_Y$ columns contain the rule firing strengths generated using lower membership functions. For example, in a three-output system, columns 4 and 7 contain the firing strengths for the output levels in column 1.

For more information on fuzzy implication, see “Apply Implication Method” on page 1-31.

Dependencies

- To enable this port, select the **Rule outputs (ro)** parameter.
- To specify $N_S$, use the **Number of samples for output discretization** parameter.

**ao — Aggregated output**

matrix | row vector

Aggregate output for each output variable, obtained by combining the corresponding outputs from all the rules using the aggregation method specified in the FIS.
For a type-1 Mamdani fuzzy inference system, the aggregate result for each output variable is a fuzzy set. In this case, \(a_0\) is as an \(N_S\)-by-\(N_Y\) matrix signal, where \(N_Y\) is the number of outputs and \(N_S\) is the number of sample points used for evaluating output variable ranges. Each column of \(a_0\) contains the aggregate fuzzy set for one output variable.

For a type-2 Mamdani FIS, the aggregate result for each output variable is a fuzzy set. In this case, \(a_0\) is as an \(N_S\)-by-(\(2*N_Y\)) matrix signal. The first \(N_Y\) columns contain the aggregated outputs generated using upper membership functions, and the last \(N_Y\) columns contain the aggregated outputs generated using lower membership functions.

For a type-1 Sugeno system, the aggregate result for each output variable is a scalar value. In this case, \(a_0\) is a row vector of length \(N_Y\), where element \(k\) is the sum of the rule outputs for the \(k\)th output variable.

For a type-2 Sugeno system, \(a_0\) is an \(N_R\)-by-(\(3*N_Y\)) array. \(\text{aggregatedOut}\) contains the same data as \(ro\) with the columns sorted based on the output levels. For example, in a three-output system, when the output levels in column 1 are sorted, the corresponding firing strengths in columns 4 and 7 are adjusted accordingly.

For more information on fuzzy aggregation, see “Aggregate All Outputs” on page 1-31.

Dependencies

- To enable this port, select the **Aggregated outputs (ao)** parameter.
- To specify \(N_S\), use the **Number of samples for output discretization** parameter.

### Parameters

#### General

**FIS name — Fuzzy inference system**

- mamfis object | sugfis object | mamfistype2 object | sugfistype2 object | file name

Fuzzy inference system to evaluate, specified as one of the following:

- mamfis or sugfis object — Specify the name of a type-1 FIS object in the MATLAB workspace.
- mamfistype2 or sugfistype2 object — Specify the name of a type-2 FIS object in the MATLAB workspace.
• File name — Specify the name of a .fis file in the current working folder or on the MATLAB path. Including the file extension in the file name is optional.

To save a type-1 fuzzy inference system to a .fis file:

• In Fuzzy Logic Designer or Neuro-Fuzzy Designer, select File > Export > To File.
• At the command line, use writeFIS.

To save a type-2 fuzzy inference system to a .fis file, use writeFIS.

**Number of samples for output discretization — Number of points in output fuzzy sets**

101 (default) | integer greater than 1

Number of samples for discretizing the range of output variables, specified as an integer greater than 1. This value corresponds to the number of points in the output fuzzy set for each rule.

To reduce memory usage while evaluating a Mamdani FIS, specify a lower number of samples. Doing so sacrifices the accuracy of the defuzzified output value. Specifying a low number of samples can make the output area for defuzzification zero. In this case, the defuzzified output value is the midpoint of the output variable range.

**Note** The block ignores this parameter when evaluating a Sugeno FIS.

**Data type — Signal data type**

double (default) | single | fixed-point | expression

Signal data type, specified as one of the following:

• double — Double-precision signals
• single — Single-precision signals
• fixdt(1,16,0) — Fixed-point signals with binary point scaling
• fixdt(1,16,2^0,0) — Fixed-point signals with slope and bias scaling
• Expression — Expression that evaluates to one of these data types
For fixed-point data types, you can configure the signedness, word length, and scaling parameters using the **Data Type Assistant**. For more information, see “Specifying a Fixed-Point Data Type” (Simulink).

**Fuzzified inputs (fi) — Enable fi output port**

*off (default) | on*

Enable output port for accessing intermediate fuzzified input data.

**Rule firing strengths (rfs) — Enable rfs output port**

*off (default) | on*

Enable output port for accessing intermediate rule firing strength data.

**Rule outputs (ro) — Enable ro output port**

*off (default) | on*

Enable output port for accessing intermediate rule output data.

**Aggregated outputs (ao) — Enable ao output port**

*off (default) | on*

Enable output port for accessing intermediate aggregate output data.

**Simulate using — Simulation mode**

*Interpreted execution (default) | Code generation*

Simulation mode, specified as one of the following:

- **Interpreted execution** — Simulate fuzzy systems using precompiled MEX files for single and double data types. Using this option reduces the initial compilation time of the model.

- **Code generation** — Simulate fuzzy system without precompiled MEX files. Use this option when simulating fuzzy systems for code generation applications.

For fixed-point data types, the Fuzzy Logic Controller block always simulates using **Code generation mode**.
Diagnostics

Out of range input value — Diagnostic message behavior when an input is out of range

warning (default) | error | none

Diagnostic message behavior when an input is out of range, specified as one of the following:

• warning — Report the diagnostic message as a warning.
• error — Report the diagnostic message as an error.
• none — Do not report the diagnostic message.

When an input value is out of range, corresponding rules in the fuzzy system can have unexpected firing strengths.

Dependencies

• Diagnostic messages are provided only when the Simulate using parameter is interpreted execution.

No rule fired — Diagnostic message behavior when no rules fire

warning (default) | error | none

Diagnostic message behavior when no rules fire for a given output variable, specified as one of the following:

• warning — Report the diagnostic message as a warning.
• error — Report the diagnostic message as an error.
• none — Do not report the diagnostic message.

When No rule fired is warning or none and no rules fire for a given output, the defuzzified output value is set to its mean range value.

Dependencies

• Diagnostic messages are provided only when the Simulate using parameter is interpreted execution.

Empty output fuzzy set — Diagnostic message behavior when an output fuzzy set is empty

warning (default) | error | none
Diagnostic message behavior when an output fuzzy set is empty, specified as one of the following:

- **warning** — Report the diagnostic message as a warning.
- **error** — Report the diagnostic message as an error.
- **none** — Do not report the diagnostic message.

When **Empty output fuzzy set** is **warning** or **none** and an output fuzzy set is empty, the defuzzified value for the corresponding output is set to its mean range value.

**Dependencies**

- This diagnostic message applies to Mamdani systems only.
- Diagnostic messages are provided only when the **Simulate using** parameter is **Interpreted execution**.

**Compatibility Considerations**

**Support for representing fuzzy inference systems as structures will be removed**

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use **mamfis** and **sugfis** objects instead. There are differences between these representations that require updates to your code. These differences include:

- Object property names that differ from the corresponding structure fields.
- Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either **mamfis** or **sugfis** objects.

To convert existing fuzzy inference system structures to objects, use the **convertfis** function.
Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

PLC Code Generation
Generate Structured Text code using Simulink® PLC Coder™.

Fixed-Point Conversion
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Blocks
Fuzzy Logic Controller with Ruleviewer

Apps
Fuzzy Logic Designer | Neuro-Fuzzy Designer

Functions
evalfis | genfis | mamfis | mamfistype2 | readfis | sugfis | sugfistype2 | writeFIS

Topics
“Fuzzy Inference Process” on page 1-28
“Simulate Fuzzy Inference Systems in Simulink” on page 5-2

Introduced before R2006a
Fuzzy Logic Controller with Ruleviewer

Evaluate fuzzy inference system and view rules

Library: Fuzzy Logic Toolbox

Description

The Fuzzy Logic Controller with Ruleviewer block implements a fuzzy inference system (FIS) in Simulink and displays the fuzzy inference process in the Rule Viewer during the simulation. You specify the FIS to evaluate using the \textbf{FIS matrix} parameter. To change the time between Rule Viewer updates, specify the \textbf{Refresh rate} in seconds.

For more information on fuzzy inference, see “Fuzzy Inference Process” on page 1-28.

The Fuzzy Logic Controller with Ruleviewer block does not support all the features supported by the Fuzzy Logic Controller block. The Fuzzy Logic Controller with Ruleviewer block:

- Only supports double-precision data.
- Uses 101 points for discretizing output variable ranges.
- Only supports \textit{Interpreted execution} simulation mode.
- Does not have additional output ports for accessing intermediate fuzzy inference results.

Ports

Input

\textit{Port\_1(In1)} — Input signal

\texttt{scalar | vector}

For a single-input fuzzy inference system, the input is a scalar. For a multi-input fuzzy system, combine the inputs into a vector signal using blocks such as:
• Mux
• Vector Concatenate
• Bus Creator

**Output**

`Port_1(Out1) — Defuzzified output signal`

`scalar | vector`

For a single-output fuzzy inference system, the output is a scalar. For a multi-output fuzzy system, the output is a vector. To split system outputs into scalar signals, use the Demux block.

**Parameters**

`FIS matrix — Fuzzy inference system`

`mamfisobject | sugfisobject`

Fuzzy inference system to evaluate, specified as a `mamfis` or `sugfis` object. Specify the name of a FIS object in the MATLAB workspace.

`Refresh rate — Time between rule viewer updates`

`scalar`

Time between rule viewer updates in seconds, specified as a scalar. During simulation, the Rule Viewer display updates at the specified rate to show the inference process for the latest input signal values.

**Compatibility Considerations**

**Support for representing fuzzy inference systems as structures will be removed**

*Warns starting in R2019b*

Support for representing fuzzy inference systems as structures will be removed in a future release. Use `mamfis` and `sugfis` objects instead. There are differences between these representations that require updates to your code. These differences include:
• Object property names that differ from the corresponding structure fields.
• Objects store text data as strings rather than as character vectors.

Also, all Fuzzy Logic Toolbox functions that accepted or returned fuzzy inference systems as structures now accept and return either mamfis or sugfis objects.

To convert existing fuzzy inference system structures to objects, use the convertfis function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

Usage notes and limitations:
• Generating code using the Fuzzy Logic Controller with Ruleviewer block produces the same code as using the Fuzzy Logic Controller block. However, the Fuzzy Logic Controller with Ruleviewer block does not support:
  • Generating code for single-point or fixed-point data.
  • Changing the number of samples for discretizing the output variable range.

See Also

Blocks
Fuzzy Logic Controller

Apps
Fuzzy Logic Designer | Neuro-Fuzzy Designer

Functions
evalfis | mamfis | readfis | sugfis

Topics
“Fuzzy Inference Process” on page 1-28
“Simulate Fuzzy Inference Systems in Simulink” on page 5-2

**Introduced before R2006a**
Gaussian MF

Gaussian membership function in Simulink software

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 \( \sigma \) is the standard deviation.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

gaussmf

Introduced before R2006a
Gaussian2 MF

Combination of two Gaussian membership functions in Simulink software

Description

The Gaussian2 MF block implements a membership function based on a combination of two Gaussian functions. The two Gaussian functions are given by:

\[ f_k(x) = \exp\left(-0.5\frac{(x - c_k)^2}{\sigma_k^2}\right) \]

where \( k=1,2 \). The parameters \( c_1 \) and \( \sigma_1 \) are the mean and standard deviation defining the left-most curve. The parameters \( c_2 \) and \( \sigma_2 \) are the mean and standard deviation defining the right-most curve.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

gauss2mf

Introduced before R2006a
Generalized Bell MF

Generalized bell membership function in Simulink software

![Generalized Bell MF](image)

**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.

**Extended Capabilities**

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

gbellmf

*Introduced before R2006a*
Pi-shaped MF

Pi-shaped membership function in Simulink software

Description

The Pi-shaped MF block implements a membership function in Simulink based on a spline-based curve, so named because of its $\Pi$ 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.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

pimf

Introduced before R2006a
Probabilistic OR

Probabilistic OR function in Simulink software

Description

The Probabilistic OR block outputs the probabilistic OR value for the vector signal input, based on

\[ y = 1 - \text{prod}(1 - x) \]

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks
Probabilistic Rule Agg

Functions
probor

Introduced before R2006a
Probabilistic Rule Agg

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

\[ y = 1 - \text{prod}(1 - [a; b]) \]

The two inputs, \( a \) and \( b \), are row vectors.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Blocks
Probabilistic OR

Functions
probor

Introduced before R2006a
Prod. Sigmoidal MF

Product of two sigmoid membership functions in Simulink software

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

\[ f_k(x) = \frac{1}{1 + \exp(-a_k(x - c_k))} \]

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. Parameters \( a_1 \) and \( a_2 \) should be positive and negative respectively.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

psigmf

Introduced before R2006a
S-shaped MF

S-shaped membership function in Simulink software

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.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

smf

Introduced before R2006a
Sigmoidal MF

Sigmoidal membership function in Simulink software

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.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

sigmf

Introduced before R2006a
Trapezoidal MF

Trapezoidal membership function in Simulink software

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

trapmf

Introduced before R2006a
**Triangular MF**

Triangular membership function in Simulink software

![Triangular MF](image)

**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.

**Extended Capabilities**

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**See Also**

*trimf*

*Introduced before R2006a*
Z-shaped MF

Z-shaped membership function in Simulink software

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.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also
zmf

Introduced before R2006a
Appendices
Bibliography


See Also

More About
• “What Is Fuzzy Logic?” on page 1-3
• “Foundations of Fuzzy Logic” on page 1-10
• “Fuzzy Inference Process” on page 1-28