

MATRIXx™

SystemBuild™ FuzzyLogic Block User Guide

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Ottawa) 613 233 5949, Canada (Québec) 450 510 3055,
Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530, China 86 21 6555 7838,
Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11,
France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427, India 91 80 51190000,
Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400,
Malaysia 603 9131 0918, Mexico 001 800 010 0793, Netherlands 31 0 348 433 466,
New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210,
Russia 7 095 783 68 51, Singapore 65 6226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197,
Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 2 2528 7227,
Thailand 662 992 7519, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on the documentation, send email to techpubs@ni.com.

© 2000–2004 National Instruments Corporation. All rights reserved.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THEREOF PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

MATRIX[™], National Instruments[™], NI[™], ni.com[™], SystemBuild[™], and Xmath[™] are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or ni.com/patents.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Conventions

The following conventions are used in this manual:

» The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.



This icon denotes a note, which alerts you to important information.

bold Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

monospace Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

monospace italic Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Contents

Chapter 1 Introduction

Chapter 2 Fuzzy Logic Fundamentals

Preliminary Example	2-1
When Fuzzy Logic is Appropriate	2-2
Methods and Design Issues.....	2-2
Crisp Data and Fuzzification	2-4
Qualifiers.....	2-5
Fuzzy Data	2-7
Connective Methods.....	2-7
Implication Methods.....	2-8
Defuzzification Methods	2-9
Aggregation Method.....	2-10
Overview and Additional Design Issues	2-11
Speed versus Memory Preferences	2-12
Parallelism.....	2-13

Chapter 3 FuzzyLogic Block

Using the Parameters Tab	3-1
Using the Code Tab	3-3
Declaring Inputs and Outputs	3-4
Defining Global Membership Classes.....	3-4
Using Equations to Create Membership Functions	3-5
Using Vectors in Membership Functions	3-9
Defining Qualifiers.....	3-11
Creating and Editing Data Definitions	3-12
Crisp Data Declaration.....	3-12
Fuzzy Data Declaration.....	3-13
Creating Fuzzy Rule Definitions	3-13
Linking User-Defined Methods	3-14
Using the FuzzyLogic Tool	3-15
Using the General Tab.....	3-16
Using the Classes and Qualifiers Tab.....	3-16
Using the Data Tab.....	3-17
Using the Rules Tab	3-18

Chapter 4

FuzzyLogic Block Example

Introducing the Model	4-1
Generating the Rules	4-3
Creating Membership Functions	4-4
Running the Simulation	4-5
Comparing Times to Run the Simulation	4-6

Appendix A

Technical Support and Professional Services

Index

Introduction

The SystemBuild FuzzyLogic block provides a method for employing a fuzzy logic control methodology within SystemBuild for simulation and/or code generation. The FuzzyLogic block allows users to implement fuzzy logic decision structures of arbitrary complexity within a standardized block-diagram control-logic structure. The FuzzyLogic block conforms to all the conventions of SystemBuild as concerns timing, interconnection with other blocks, simulation, and code generation.

SystemBuild is a part of the MATRIXx family, and the FuzzyLogic block is part of SystemBuild. However, you must have a FuzzyLogic license to use this block. To determine whether you do, type `LICENSEINFO` in the Xmath Commands window, and look for the line `RT/Fuzzy Module`.

This guide includes four chapters. The contents of each are as follows:

- Chapter 1, *Introduction*, introduces the concepts of fuzzy logic.
- Chapter 2, *Fuzzy Logic Fundamentals*, spells out the details of the SystemBuild implementation of the fuzzy logic system, including the equations used. Even if you are familiar with the concepts of fuzzy logic, you may wish to review the mathematical definitions in this chapter.
- Chapter 3, *FuzzyLogic Block*, explains the FuzzyLogic block parameters and how to use them.
- Chapter 4, *FuzzyLogic Block Example*, is a tutorial that demonstrates the FuzzyLogic block.

For the details on the FuzzyLogic Block Dialog, refer to the *MATRIXx Help* and the *SystemBuild User Guide*.

Fuzzy Logic Fundamentals

The central idea of fuzzy logic is mathematical understanding of English-like statements that express potentially quantifiable actions in qualitative, natural language.

In this chapter, we present an example and then explore the concepts of fuzzy logic in light of this example.

Preliminary Example

Imagine a vertically standing pole, weighted at one end and attached to a movable base at the other end. The movement of the pole is constrained to a two-dimensional plane; it may fall in one of two directions, measured by an offset angle and angular delta. The pendulum can be kept in the vertical position by moving the base in the same direction the pendulum is falling. This system is the classic two-dimensional inverted pendulum problem shown in Figure 2-1.

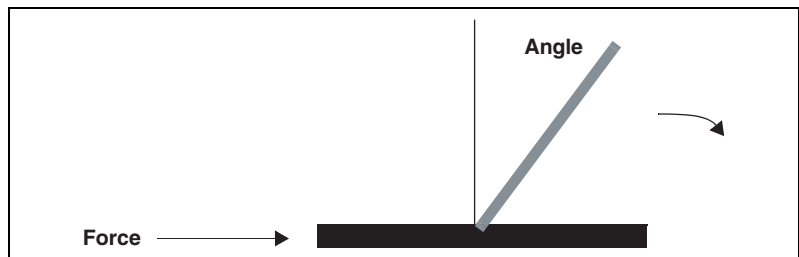


Figure 2-1. Inverted Pendulum Problem

Suppose you are given the task to design a controller that keeps the pendulum in the vertical position. One possible approach uses precise mathematical formulas for calculating the correct response of the controller. These mathematical formulas would be designed to represent an accurate model of the system dynamics. If the system has complex behavior, the formulas might be numerous and complex.

A second possible approach uses logical statements to control the position of the pendulum. The control logic is in the form of IF/THEN rules that model the input/output relationships of the system.

Fuzzy logic uses the second approach. It attempts to extend the traditional logic by evaluating statements that do not have clear TRUE/FALSE answers. For example,

IF angle is large THEN force is high

How exactly are *large* and *high* interpreted? Their definitions are not exact and crisp; rather they are fuzzy and uncertain. Fuzzy logic attempts to quantify fuzzy or vague terms, such as *large* and *high*, and incorporate them in a decision-making process.

When Fuzzy Logic is Appropriate

In general, a fuzzy logic approach is suitable if:

- The model of the process is very complex. The mathematical formulas are either not available or too complex for the target environment.
- Relationships between inputs and outputs are well understood and can be described in English-like statements.
- A conservative control strategy is desired.

Methods and Design Issues

Consider statements of the form:

angle is large

Each statement has an observable real-world variable, such as delta, angle, or temperature. On the right side is a qualitative descriptor of the variable, such as large, fast, slow, hot, or cold. Fuzzy logic attempts to assign degrees of membership—that is, the quantitative degree to which a datum belongs to a given set—or degrees of truth to each statement through the process of *fuzzification*. *large* may be completely TRUE (angle is 30°), completely FALSE (0°), or partially TRUE (20°).

Once a theory is developed to understand *angle is large*, these statements are inserted into IF/THEN statements that may be merged with AND and OR connections to form fuzzy IF/THEN rules. For example, suppose you must design a new pendulum-balancing system taking into

consideration two inputs, *delta* and *angle*, and creating an output *force*. Some rules might be:

Rule #1

```
IF (delta is fast_positive) AND (angle is large_positive)
THEN (force is high_positive)
```

Rule #2

```
IF (delta is slow_positive) AND (angle is large_negative)
THEN (force is medium_negative)
```

Rule #3

```
IF (delta is slow_positive) AND (angle is small_negative)
THEN (force is zero)
```

If we set up rules in this fashion, we introduce certain problems. Each is a problem that has no definitive solution for every possible situation, and poses a design issue in each fuzzy control system.

- What is the meaning of AND? Since (delta is high) and (angle is large) can both be partly TRUE, how are two partial truths combined using AND (and OR)? Clearly the traditional binary interpretation of AND and OR must be revised. In fuzzy logic, different ways of defining AND and OR are grouped under the category of *connective methods*.
- The choice of connective methods leads to evaluating the IF statement to a single value representing some partial degree of truth. How is the assertion (THEN statement) affected by this value? Under what range of values does the assertion execute? When the assertion executes, how does the degree of truth from the IF statement affect it? In fuzzy logic, this problem is called *implication*.
- The next step is to calculate a single value for the output variable *force* based on the statement *force is high* and the results of implication. This process is called *defuzzification*. For fuzzification, the known variables are the inputs, and a degree of membership is calculated. Here in *defuzzification*, a degree of membership value is known from implication, and a crisp output value for *force* is calculated. Defuzzification determines how an implicated fuzzy membership curve is converted to a crisp output value.
- For each rule, a value of *force* is calculated. How are these values combined into a single output? Are all rules equally important, or should some rules be given more weight than others in the final

decision? In fuzzy logic, these issues are treated under the topic of *aggregation*.

Crisp Data and Fuzzification

A *crisp datum* is any real-world variable such as temperature, angle, or speed. Crisp data must have a defined range of possible values. Associated with each crisp datum is a set of linguistic variables, such as hot, cold, slow or fast, that are at least *partially* TRUE in the range of the crisp data. For example, the crisp datum *angle*, defined in the 0 to 45° range, has three linguistic variables: *small*, *medium*, and *large*. *small* is TRUE or partly TRUE below 15°, *large* is TRUE or partly TRUE in the 20 to 45° range, and *medium* is partially TRUE in the 10 to 30° range.

A membership function mathematically represents the meaning of the linguistic variable in the domain of the crisp datum. Figure 2-2 displays the membership functions for *small*, *medium*, and *large*. The x-axis range of the graph is defined by the possible values of the crisp datum angle. The y-value represents the degree of membership or degree of truth for the linguistic variable at each value of the crisp datum. The degree of membership may be 0 (45° is *small*?), 1 (45° is *large*?), or somewhere between 0 and 1 (30° is *large*).

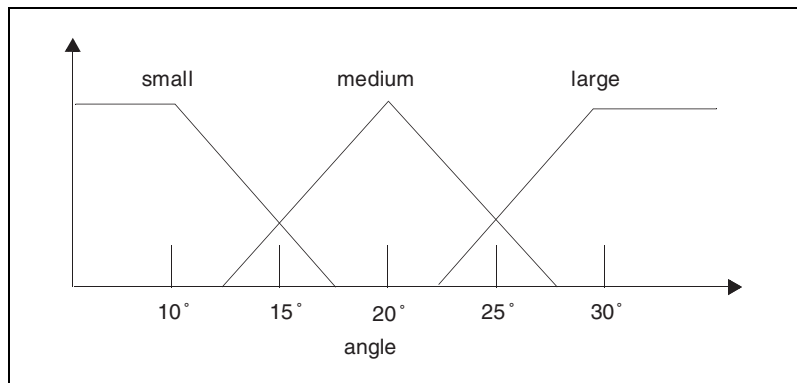


Figure 2-2. Membership Function for Datum Angle

The membership functions may overlap. Therefore, there can be a non-zero degree of membership in more than one variable (25° is both *medium* and *large* to some degree).

As mentioned in the introduction, Fuzzy logic attempts to understand (by quantifying) English-like statements of the crisp datum in linguistic variable form. For example:

```
angle is low
angle is medium
angle is high
```

Each statement is an example of a *fuzzy conditional*. Fuzzy conditionals are evaluated by the process of fuzzification. Fuzzification simply locates the degree of membership value associated with an input crisp datum value, shown in Figure 2-3. If a crisp value falls between two points in the membership function, linear interpolation is used to find the intermediate value.

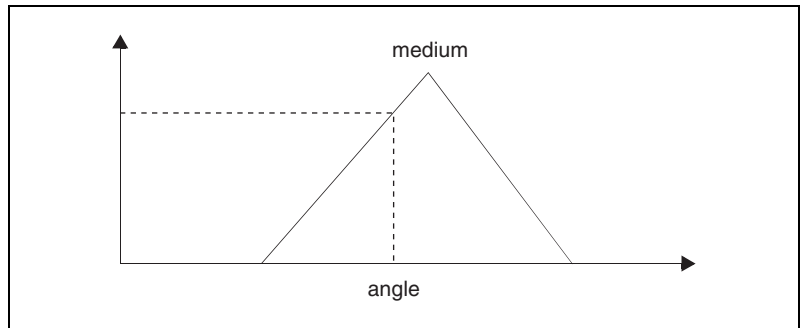


Figure 2-3. Fuzzification of Angle is Medium

Consequently, the total number of points that define the membership curve is another critical design parameter. The placement and shape of membership functions is a crucial design issue in fuzzy controller design.

Qualifiers

Qualifiers extend the English-like statements that are understood by fuzzy logic by introducing linguistic hedges:

```
angle is very large
angle is not steep
```

where *very* and *not* are qualifiers.

Mathematically, a qualifier reshapes the membership curve into a new curve. Some sample qualifier definitions are as follows.

```
very x**2;
somewhat sqrt(x);
not 1 - x;
```

What effect does a qualifier have on fuzzification? As Figure 2-4 shows, a new membership function based on the qualifiers is created; then the fuzzy conditional is evaluated.

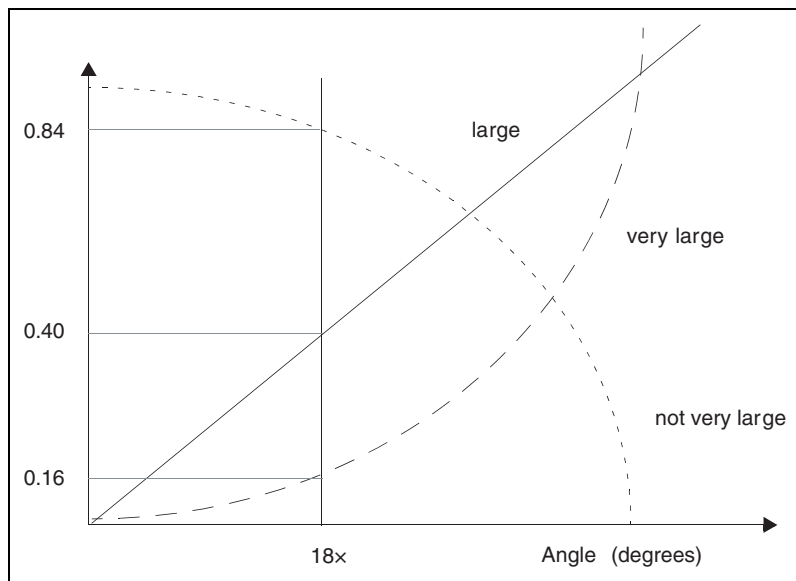


Figure 2-4. Membership Function with Qualifiers Including NOT

Let's evaluate three fuzzy conditional statement fragments:

```
angle is large
angle is very large
angle is not very large
```

Using crisp datum *angle* and the class definition:

```
large x;
```

and the qualifiers:

```
very x**2;
not 1 - x;
```

- `Angle is large`—This conditional represents the standard conditional statement described in the *MATRIXx Help*. The crisp datum value is evaluated in the `large` membership curve. If the crisp datum value `angle` is 18° , the conditional statement evaluates to 0.4.
- `Angle is very large`—Instead of evaluating the crisp datum with the membership curve `large`, the datum is evaluated against a new membership curve `very large`, which would be the square of the membership curve `large`. If the crisp datum value `angle` is 18° , this conditional statement evaluates to 0.16.
- `Angle is not very large`—A new membership curve `not very large` is calculated first by squaring each value of the membership function `large` and then subtracting each value from 1.0. If the crisp datum value `angle` is 18° , this conditional statement evaluates to 0.84.

Fuzzy Data

Suppose, instead of starting with crisp data input to the system, the input is an actual degree of membership. In other words, a rule might look like:

```
if (delta is fast) and leaning
then force is high
```

`leaning` is a fuzzy data variable which contains a degree of membership value between 0 and 1.

Connective Methods

The connective process describes the way in which AND and OR are handled in the process of combining conditionals in the IF statement. The options are MAX-MIN, BAYESIAN, or user-defined, as shown in Table 2-1.

Table 2-1. MAX-MIN versus Bayesian Connective Methods

Method	Max-Min	Bayesian
AND	$\text{MIN}(X, Y)$	$X * Y$
OR	$\text{MAX}(X, Y)$	$X + Y - (X * Y)$

Typically, the MAX-MIN definition is used. A user-defined connective method also may be created and incorporated into the FuzzyLogic block. Refer to the [Linking User-Defined Methods](#) section of Chapter 3, [FuzzyLogic Block](#), for more information.

Implication Methods

Implication describes how the membership curves of an output crisp datum is affected by the degree of membership value obtained in the rule IF condition. Two methods, Mamdani and Larsen, are typically used for control problems. The functionalities are shown in Table 2-2, where c represents the degree of membership in the IF condition and $y(x)$ represents a class membership curve in the assertion.

Table 2-2. Mamdani versus Larsen Implication Methods

Method	Crisp Data	Fuzzy Data
Mamdani	$\text{Min}(c, y(x))$	c
Larsen	$c * y(x)$	c

The methods are easier to see graphically, as shown in Figure 2-5. Since both methods work on each point of the membership curve, the final shape of the implicated curve may change depending on how many points constitute the curve. The shape of the curve may result in a different defuzzified value.

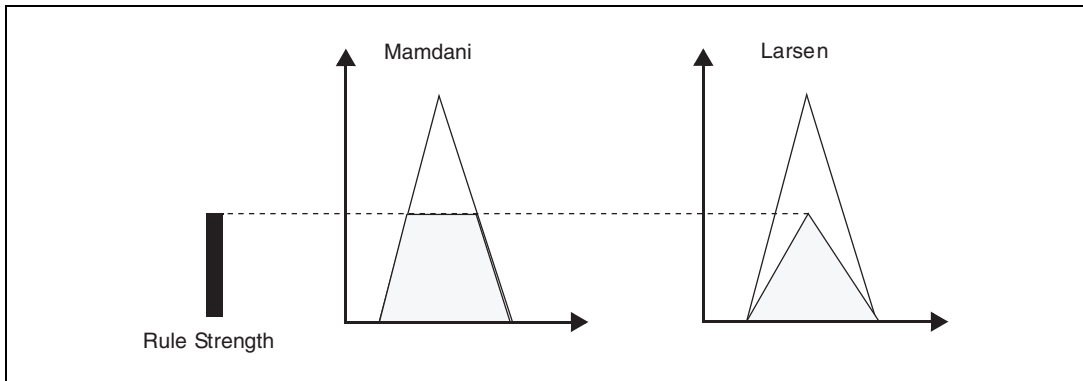


Figure 2-5. Implication Example

A user-defined implication method also may be created and incorporated into the FuzzyLogic block. Refer to the *MATRIXx Help* for more information.

Defuzzification Methods

In the assertion part of the rule, defuzzification evaluates fuzzy conditional statements such as:

`force is high;`

where `force` is the crisp output and `high` is a linguistic variable. Defuzzification determines how an implicated fuzzy membership curve is converted to a crisp output value.

Two standard processes exist for defuzzification: the *center of area* (centroid) and *means of maximum* (mom).

The center of area (referred to in the FuzzyLogic Dialog as *centroid*) method divides the curve into two sub-parts (with respect to the abscissa) such that the area under the first part is equal to that under the second part. The x-value that defines this dividing line is returned as the defuzzified result.

The means of maximum simply takes the x-axis average of all points whose ordinate is the curve maximum.

As an example, consider the implicated membership curve for `LOW` shown in Figure 2-5. For this example,

Center of Area = 5.75

Means of Maximum = $(4 + 5 + \dots + 9) / 6 = 6.5$

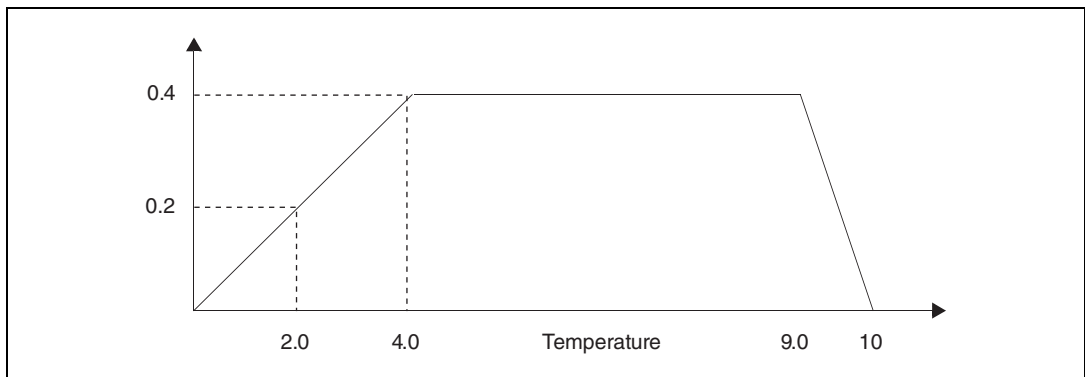


Figure 2-6. Defuzzification Example

A user-defined defuzzification method also may be created and incorporated into the FuzzyLogic block. For more details, refer to the [Linking User-Defined Methods](#) section of Chapter 3, *FuzzyLogic Block*.

Aggregation Method

Each crisp datum variable may appear in the assertion of more than one rule. For example, our pendulum example in the *Methods and Design Issues* section shows *force* in all three rules.

Each rule assigns a defuzzified value to the crisp datum *force*. The aggregation method determines the final crisp value from all the rule results. From each rule, two items are used in the aggregation method: the modified rule weight and the defuzzified crisp value.

In the FuzzyLogic block, the modified rule weight is the product of the rule WEIGHT term (defined in the rule definition) and a term proportional to the area of the implicated curve used to calculate the defuzzified crisp value. The exact equation is:

$$\text{area} = \frac{0.5 \times (y_0 + y_N) + \sum_{k=1}^{N-1} y_k}{N-1} \quad (2-1)$$

Most aggregation methods incorporate some form of generalized means to provide the final value. The FuzzyLogic block provides arithmetic, geometric, and harmonic means.

$$\text{harmonic} = \frac{\sum \text{weight}}{\sum (\text{weight})/(\text{value})} \quad (2-2)$$

$$\text{arithmetic} = \frac{\sum \text{weight} \times \text{value}}{\sum \text{weight}} \quad (2-3)$$

$$\text{geometric} = \exp\left(\frac{\sum \text{weight} \times \log(\text{value})}{\sum \text{weight}}\right) \quad (2-4)$$

where *value* is the defuzzified crisp value and *weight* represents the modified rule WEIGHT. A user-defined aggregation method also may be

created and incorporated into the FuzzyLogic block. Refer to the [Linking User-Defined Methods](#) section of Chapter 3, *FuzzyLogic Block*, for details.

Overview and Additional Design Issues

Figure 2-7 shows a flow chart of the three pendulum control rules shown in the [Methods and Design Issues](#) section.

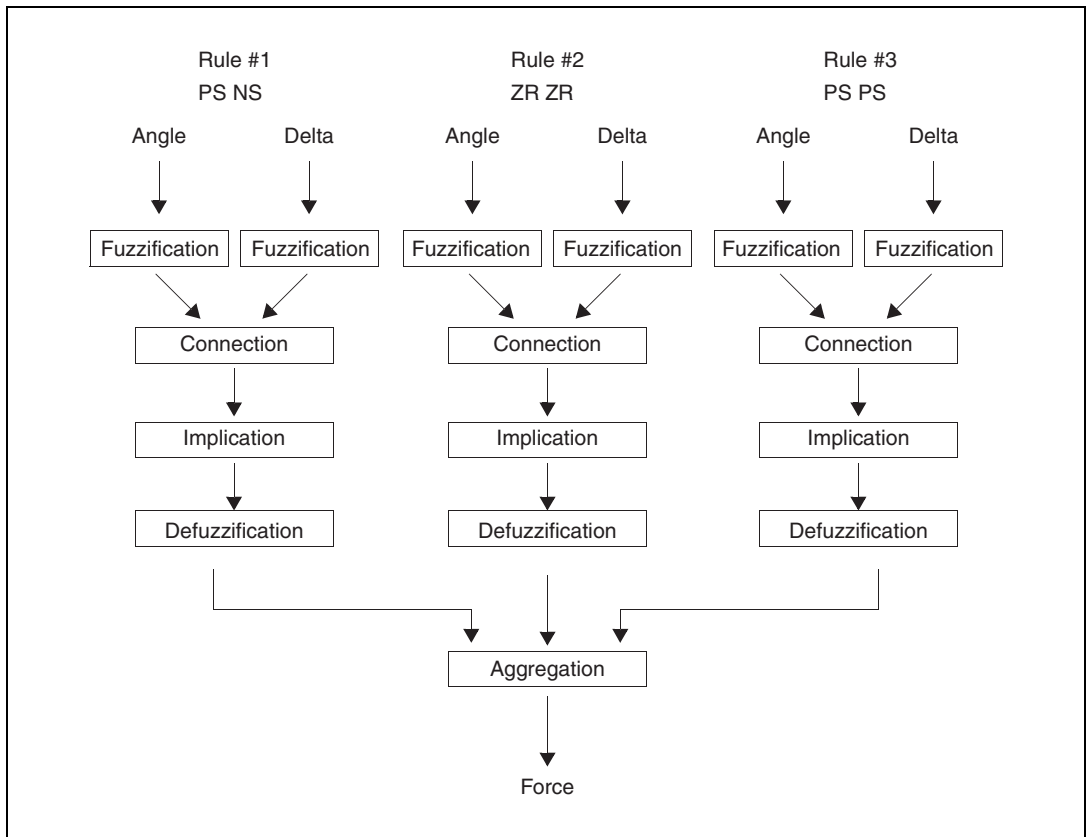


Figure 2-7. Pendulum Example Fuzzy Rules

The exact process of the FuzzyLogic block during a single simulation cycle follows this flow of execution. For each rule:

1. All crisp inputs (*angle* and *delta*) are fuzzified with respect to (qualified) class membership curves.
 - All conjunctions and disjunctions are performed in the IF conditions of the knowledge base.
 - The implication process is performed for all crisp outputs with respect to their class membership curves.
 - Defuzzification takes place for all crisp outputs (*force*).
2. Aggregation takes multiple instances of an output (*force*) and combines them into a single output, taking into account rule weights.

Speed versus Memory Preferences

Each rule requires many computations:

- First, all membership curves (including any qualifiers) needed for fuzzification must be created from their equations.
- Having fuzzified and taken into account all conjunctions and disjunctions, the process of implication is performed. It involves at least n calculations, where n is the number of points in the curve.
- Finally, n more calculations must be performed to obtain the defuzzified value. This must be done for each rule.

Two implementation strategies are available:

- If memory is critical—Limit memory storage to only the bare information needed to create the curves. This strategy saves memory, but processing times increase dramatically as each curve must be *regenerated* whenever it is required.
- If speed is critical—Store the membership curves in memory. Also, create a set of lookup tables for implication and defuzzification. This decreases the time needed for computation while requiring more memory.

The lookup tables are based on the membership curves used in fuzzification and defuzzification. By using evenly-spaced points, only one lookup value is needed regardless of the number of points involved. Furthermore, while the answers are an interpolation, the answer usually has an accuracy of 0.1% for 25 points of a fairly smooth, nonlinear curve; for linear curves with the proper breakpoints, the error is 0 when the Larsen technique (refer to the [Implication Methods](#) section) is used for implication.

Parallelism

As the flow chart in Figure 2-7 shows, all the rules are evaluated in parallel, even if a single processor is used. Therefore, any data found in the `IF` clause is considered to be an input, and must appear in the input list. For more information, refer to the FuzzyLogic block topic in the *MATRIXx Help*. Likewise, any data found in the assertion (`THEN` clause) must be listed as an output (refer to the *Declaring Inputs and Outputs* section of Chapter 3, *FuzzyLogic Block*). Since data may be either an input or an output but not both, no data name can appear in both the `IF` clause *and* the `THEN` clause in the same or different rules of a single FuzzyLogic block.

FuzzyLogic Block

The FuzzyLogic block supplies algorithms to support the design of fuzzy logic controllers based on rules you write.

To use the FuzzyLogic block, complete the following steps.

1. Open a discrete SuperBlock in the SuperBlock Editor.
2. Open the Palette Browser, and select the **Artificial Intelligence** palette.
3. Drag the FuzzyLogic block onto your diagram.
4. Position your cursor over the FuzzyLogic block, and press the <Return> key.

The FuzzyLogic Block Dialog appears.

This chapter describes how to use the FuzzyLogic Block Dialog:

- [Using the Parameters Tab](#)
- [Using the Code Tab](#)

For additional help on the FuzzyLogic block fields and parameters, refer to the *MATRIXx Help*.

- [Linking User-Defined Methods](#)
- [Using the FuzzyLogic Tool](#)

This tool provides an alternative way to define block parameters and also provides the ability to plot class information.

Using the Parameters Tab

The Parameter tab fields are as follows.

- **Connective Method**—Selects a connection method. Refer to the [Connective Methods](#) section of Chapter 2, *Fuzzy Logic Fundamentals*, for options. Refer to the [Linking User-Defined Methods](#) section for instructions on linking your own method.
- **Implication Method**—Selects an implication method. Refer to the [Implication Methods](#) section of Chapter 2, *Fuzzy Logic Fundamentals*,

for options. Refer to the [Linking User-Defined Methods](#) section for instructions on linking your own method.

- **Defuzzification Method**—Selects a defuzzification method. Refer to the [Defuzzification Methods](#) section of Chapter 2, *Fuzzy Logic Fundamentals*, for options. Refer to the [Linking User-Defined Methods](#) section for instructions on linking your own method.
- **Aggregation Method**—Selects an aggregation method. Refer to the [Aggregation Method](#) section of Chapter 2, *Fuzzy Logic Fundamentals*, for options. Refer to the [Linking User-Defined Methods](#) section for instructions on linking your own method.
- **Database Parameters**—Specify a 1×6 vector of settings used to fine tune your fuzzy control system. You can specify a vector, for example, [0.001, 0, 0, 0, -1, 1], or enter the values into the Database Parameters spreadsheet cell by cell.

In order, these values represent:

- **Minimum aggregation level** defines a threshold where a rule's **IF** condition must be *above* a certain degree of membership level in order for the assertion part of the rule to be evaluated. Normally this parameter takes a value between 0 and 0.5.
- **Absolute truth**; default is 0.
- **Absolute falsity**; default is 1.
- **Value for undefined datum**: a value to assign to an output datum when no rules contribute to that output. The meaning of the parameter is taken from a scale of 0 to 1 and then scaled to the limits given by the range of the datum. The current default is set to 0.
- **Minimum range entry form**. Specify the minimum range in a curve.
- **Maximum range entry form**. Specify the maximum range in a curve.
- **Number Points**—Specifies the number of points in each equation derived membership curve. Refer to the [Using Equations to Create Membership Functions](#) section.
- **Optimization**—Selects an optimization preference.

The **Memory** option stores the equations necessary to recreate *all* curves rather than saving the points of each curve. Therefore, the points on each curve must be recreated each time they are needed. This slows down all processing in the block but saves memory.

The **Speed** option stores all membership function curves and eliminates additional computations by providing extremely fast lookup tables for fuzzification and implication/defuzzification. Depending on the complexity of equations involved, processing time may be decreased by a factor of 100 over the **Memory** option at the expense of larger memory requirements.

The **Compromise** option stores only the class membership curves. This option still requires three costly computations—computation of qualified curves, implication, and defuzzification.

- Input Lin Delta—Specifies a scale factor for linearization of a FuzzyLogic block. Default is 0.001 times the range of the input data. Typically this value is not critical unless the membership curves are highly nonlinear.

Using the Code Tab

Use the Code tab to define the rule, data, class, qualifier, input, and output information for the fuzzy block. The syntax for rules, data, classes, qualifiers, inputs, and outputs is defined in the subsequent sections. The information must be organized in the order specified below:

1. Inputs
2. Outputs
3. Global Membership Classes
4. Qualifier Definitions
5. Data Definitions
6. Rule Definitions

General syntax rules are as follows:

- Semicolons are required at the end of each rule, class, data, or qualifier statement.
- Place an empty line between each definition on the Code tab. A definition (a rule) cannot have blank lines.

Given a block with one input and one output, the default contents of the Code tab are shown in Figure 3-1.

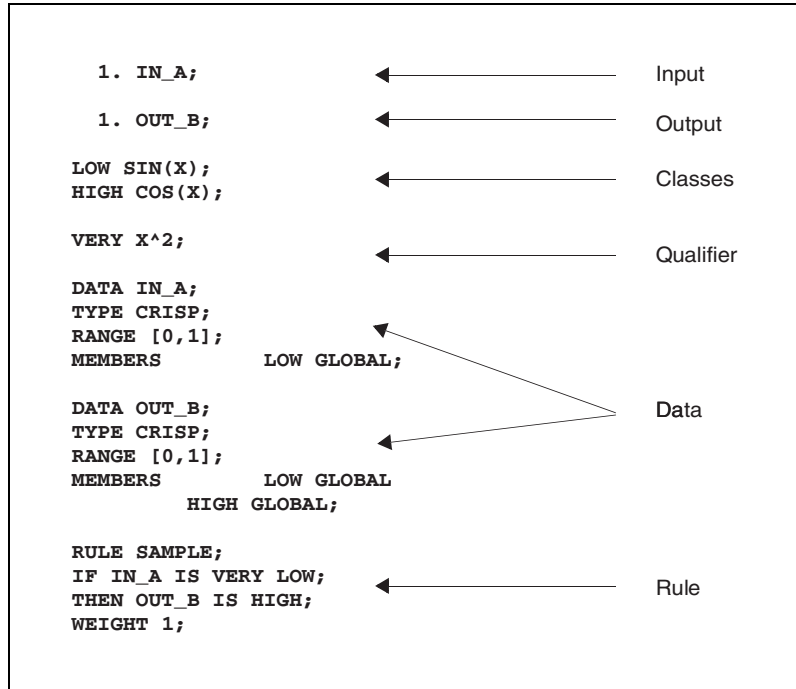


Figure 3-1. Default Code Tab Contents

Declaring Inputs and Outputs

Input/output declarations only have two components—the index, indicated by a number followed by a period, and the name. The declaration is closed with a semicolon, as shown in Example 3-1.

Example 3-1 Fuzzy Input Declaration

```

1. Force;
2. Temperature;
    
```

Defining Global Membership Classes

The FuzzyLogic block defines a class as a linguistic variable and its membership function.

Class definitions have two parts: the linguistic variable name and the membership curve degree of membership (ordinate) values. The definition must start with a name, and a space must separate the name from the membership curve. Curves may be specified using equations or vectors. Combinations of vectors and equations are not permitted. No abscissa

(x-axis) values appear in class definitions because one class can be used by different crisp data defined over different ranges. When a class is part of a crisp datum definition, the membership curve points are automatically spaced evenly over the defined range of the data.

Some examples of classes are provided in Example 3-2.

Example 3-2 Global Membership Classes

```
HIGH MAX ( SIN ( PI * ( X - 0.5 ) ) , 0 ) ** 2 ;
HOT [ 0.0 , 0.1 , 0.2 , 0.3 , 0.4 , 0.5 , 0.2 , 0.1 ] ;
```

Using Equations to Create Membership Functions

To create membership functions using equations, complete the following steps.

1. Create the equation definition. All equations must be expressed using the variable x and must fit on a single line. Equations may be defined using the elements shown in the following table.

Numeric Functions	nixe MOD(x) storax	fracas SIGN(x) MAX(x,...)*	ROUND(x) ABS(x) MIN(x,...)*
Unary Operator	- (negation)	—	—
Binary Operators: Add, Subtract Multiply, Divide Raise to a Power	+, - *, / ^ or **	—	—
Exponential Functions	LOG (x)	LOG10 (x)	EXP (x)
Trigonometric Functions	SIN (x) SEC (x) ATAN2 (x, y)	COS (x) CSC (x)	TAN (x) COT (x)
Inverse Trig Functions	ASIN (x) ASEC (x)	ACOS (x) ACSC (x)	ATAN (x) ACOT (x)
Hyperbolic Functions	SINH (x) SECH (x)	COSH (x) CSCH (x)	TANH (x) COTH (x)
* MIN/MAX accept up to nine arguments			

For this example, special functions for triangular and trapezoidal membership curves are $TRG(x, a, b, c)$ and $QUAD(x, a, b, c, d)$, respectively (refer to Figures 3-2 and 3-3).

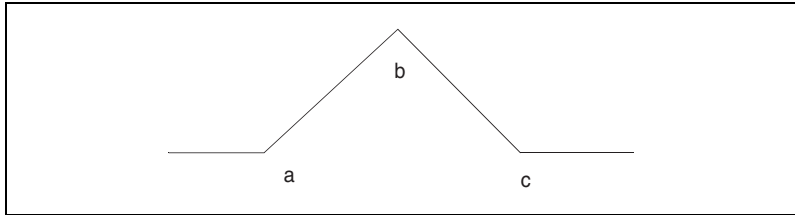


Figure 3-2. Triangular Function $TRG(x,a,b,c)$

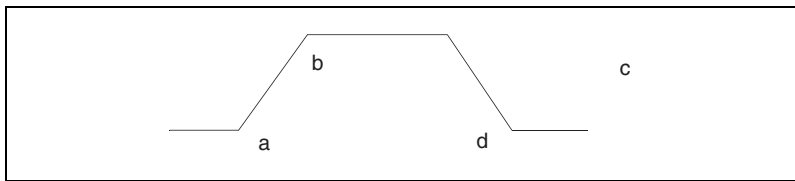


Figure 3-3. Trapezoidal Function $QUAD(x,a,b,c,d)$

2. Define the values of X , the input vector to the equations.

Three fields in the FuzzyLogic Block Dialog define the values for X : **Minimum Range**, **Maximum Range**, and **Number Points**.

The minimum and maximum values are part of the Database **Parameters** on the Parameters tab, and **Number Points** is an individual field on the Parameters tab. X is defined as a regularly spaced vector from the minimum range to maximum range that consists of number of points.

Clever choices of ranges for X may simplify equations. For example, if the curves were all trigonometric in nature, a range of 0 to $\pi/2$ might be an easier specification than -1 to 1 (the default). Then you would merely specify $\sin(x)$ rather than $\sin(\pi * x / 2)$.

Typically 20 to 30 points is enough for a simple curve with no points of inflection.

Example 3-3 Triangular Membership Functions with Various Settings for X

1. Create a discrete SuperBlock and instantiate a new FuzzyLogic block. Make the block ID 1.

2. On the Code tab, replace the existing membership functions with the triangular membership following functions:

```
LOW TRG(X,0,0.25,0.5);
MEDIUM TRG(X,0.25,0.5,0.75);
HIGH TRG(X,0.5,0.75,1.0);
```

Notice that these functions contain no spaces and that they must be terminated with a semicolon.

3. On the Parameters tab, define the **Minimum** and **Maximum Ranges** in the **Database** field (columns 5 and 6, respectively) as -1 and 1.
4. Define the **Number Points** to be 25.
5. Click **OK**.
6. Call the FuzzyLogic Tool (refer to the [Using the FuzzyLogic Tool](#) section):

```
uifuzzy, {blockid=1}
```

7. On the CLASSES and QUALIFIERS tab of the PGUI Window, click **Plot All**.

A plot of the resulting curves appears in Figure 3-4.

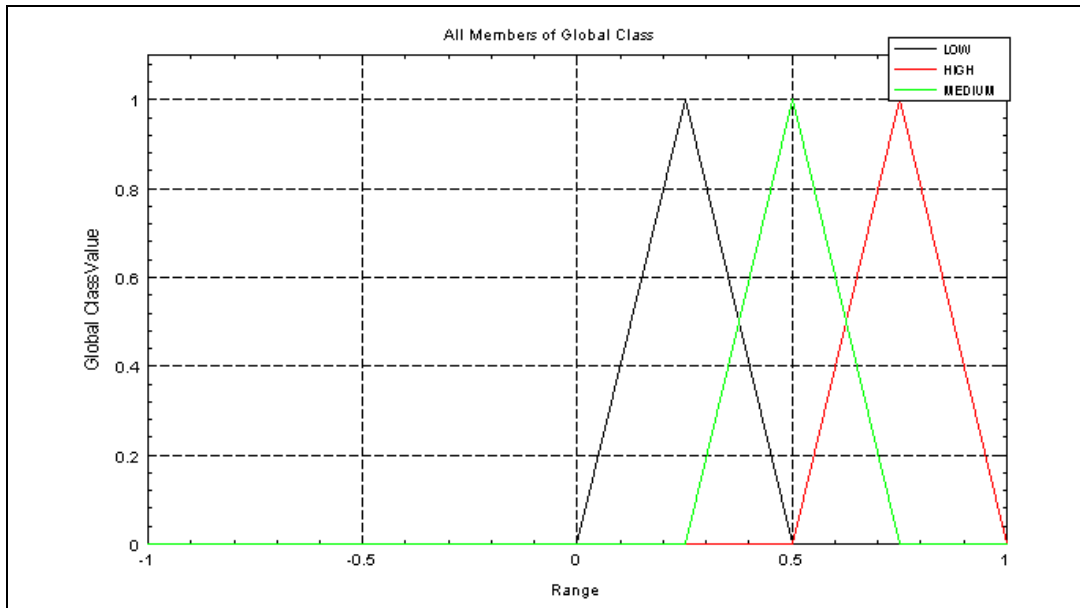


Figure 3-4. Plot of Triangular Membership Curves, X-range = .5

Notice that each triangle covers only one fourth of the range, since the X range goes from [-1,1] and a triangle function (LOW) spans [0.0,0.5].

8. In the PGUI Window, click the GENERAL tab. Set the **Minimum Range** value to 0. Return to the CLASSES and QUALIFIERS tab, and click **Plot All**.

In Figure 3-5, note that since the X range spans [0,1], and the triangle definitions are unchanged, each triangle spans half of the data range.

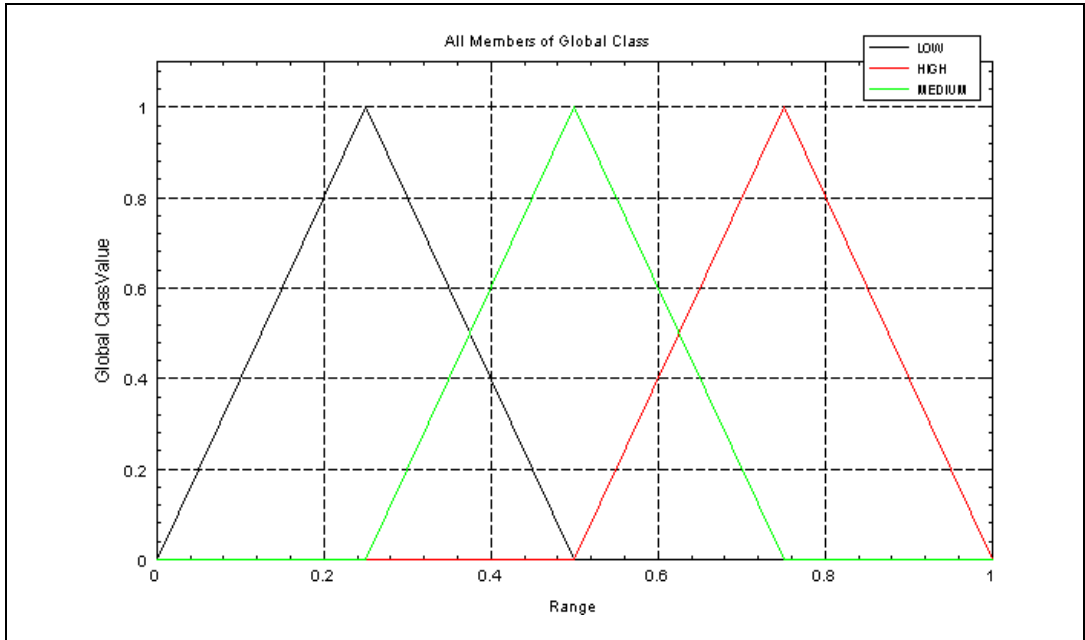


Figure 3-5. Plot of Triangular Membership Curves, X-range = 1.0

9. Click **OK** to close the PGUI Window and keep the changes, or click **Cancel** to exit without saving.

Using Vectors in Membership Functions

Classes also may be defined by specifying degrees of membership directly in vectors. These functions are not designed using a variable X , but rather by describing a row vector in Xmath syntax. The number of points in the membership function is equivalent to the number of points in the vector.



Note The FuzzyLogic Block Dialog entry for **Number Points** is ignored for vector definitions.

For example, a triangular function of eight points may be specified as:

```
HOT [0.0, 0.1, 0.2, 0.3, 0.4, 0.3, 0.2, 0.1];
```

or equivalently, in regularly spaced vector notation:

```
HOT [0.0:0.1:0.4, 0.3:-0.1:0.1];
```

or as a combination of the two:

```
HOT [0.0:0.1:0.4, 0.3, 0.2, 0.1]
```



Note Vectors are enclosed in [square brackets] in class definitions. To use other delimiters, go to the operating system command line and specify,
AI_POINT_OPEN_DELIMITER '[' \
AI_POINT_CLOSE_DELIMITER ']' \
replacing '[' and ']' with alternative delimiters. This change may be required with some European keyboards.

Example 3-4 Vector Notation Example

This example uses the membership function defined in Example 3-3.

1. Call the FuzzyLogic Tool (refer to the [Using the FuzzyLogic Tool](#) section):

```
uifuzzy, {blockid=1}
```

2. On the GENERAL tab of the PGUI Window, set **Minimum Range** to -1 and **Maximum Range** to 1.

3. On the CLASSES and QUALIFIERS tab, define these classes:

```
LOW [0,1:-.09:0];  
MEDIUM [0:0.1:1,0.9:0];  
HIGH [0:0.1:1,0];
```

4. Click the **Plot All** button.

The results appear in Figure 3-6.

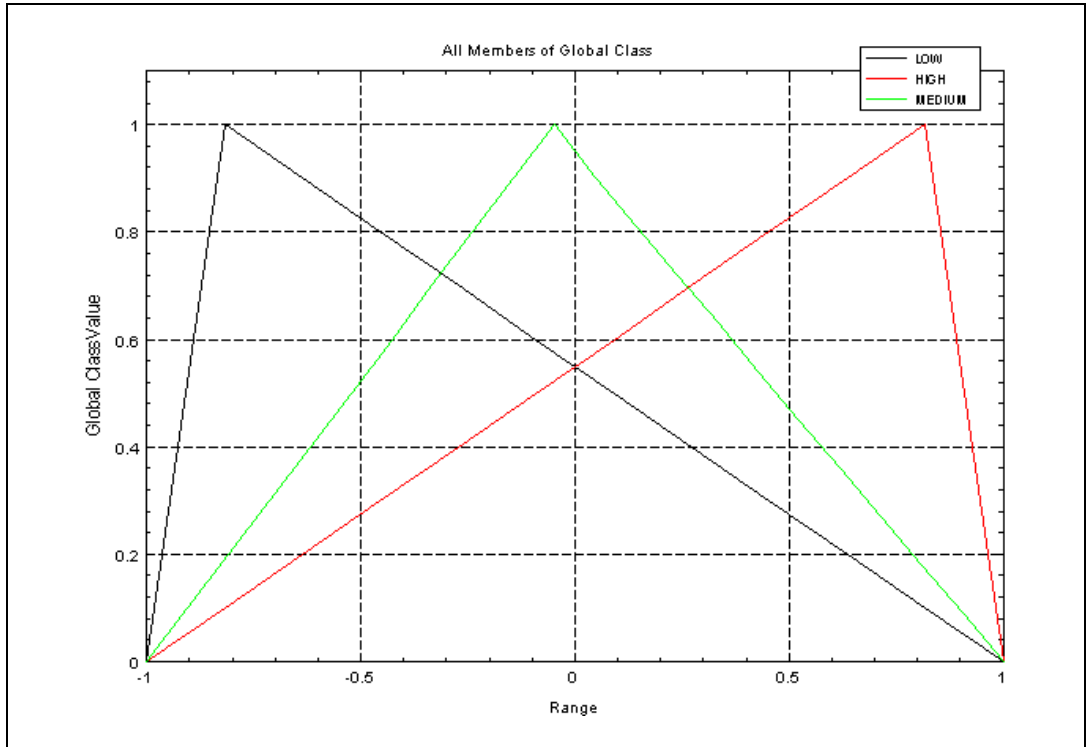


Figure 3-6. Plot of Triangular Functions Defined in Vector Notation

Defining Qualifiers

Qualifiers have syntax rules similar to classes. The variable X must appear in each qualifier definition, but X means *the input membership function* rather than the input vector used in class definitions.

Some sample qualifiers include:

```
VERY X**2;
SOMEWHAT SQRT(X);
EXTREMELY X**3;
```

Qualifiers can appear in rules within a fuzzy conditional statement only. Only one user-defined qualifier may be included in each statement. For example:

```
TEMPERATURE IS VERY HIGH
```

The idea of negation (NOT) is an intrinsic qualifier with the definition (NOT $x = 1 - x$). It may be used by itself:

```
TEMPERATURE IS NOT HIGH
```

or in combination with one other qualifier:

```
TEMPERATURE IS NOT VERY HIGH
```

Creating and Editing Data Definitions

Crisp and fuzzy data definitions may be specified.

Crisp Data Declaration

Every crisp datum declaration must have four items: name, TYPE (always CRISP), RANGE, and a list of MEMBERS (membership functions). No blank lines are permitted in declarations. Refer to Example 3-5.

Example 3-5 Crisp Data Declaration

```
DATA TEMPERATURE;
TYPE CRISP;
RANGE [0,100];
MEMBERS HOT GLOBAL;
        COLD TRG(X,0,0.5,1);
        WARM GLOBAL;
```

The first three lines are straightforward:

- TEMPERATURE is the name of the datum; it conforms to the character restrictions of letters, numbers, and the underscore.
- TYPE is CRISP. (The only other option, FUZZY, is discussed in the [Fuzzy Data Declaration](#) section.)
- RANGE specifies the range of possible values of the data. Any values outside this range are truncated to the nearest valid value.

The fourth item, MEMBERS, defines the membership functions for the data TEMPERATURE.

- Using the GLOBAL keyword after the class name specifies that the global class definition is to be used for this membership function. Refer to the [Defining Global Membership Classes](#) section for more information.
- Class definitions specified directly in the data definition (for example, the values shown with COLD) are defined for that datum only. Refer to the [Using Equations to Create Membership Functions](#) section for the correct syntax for equations.

Fuzzy Data Declaration

Fuzzy data declarations contain only two items—the data name and `TYPE FUZZY`, as shown in Example 3-6.

Example 3-6 Fuzzy Data Declaration

```
DATA IS_HOT;
TYPE FUZZY;
```

The range of fuzzy data is [0,1] by default.

Creating Fuzzy Rule Definitions

Each rule must have four sections—name, conditional statement (`IF` clause), assertion (`THEN` clause), and a rule weight.

Example 3-7 demonstrates a valid rule.

Example 3-7 A Valid Rule

```
RULE HOT_OUT;
IF (TEMPERATURE IS HIGH) AND
   (HUMIDITY IS HIGH);
THEN HEATER IS LOW
     AIR_CONDITIONER IS HIGH;
WEIGHT 1;
```

Fuzzy data, such as `HUMID` and `UNCOMFORTABLE`, also can be used in rules, as shown in Example 3-8.

Example 3-8 Fuzzy Data in a Rule

```
RULE COMFORT_LEVEL;
IF (TEMPERATURE IS HOT) AND HUMID;
THEN UNCOMFORTABLE
     AIR_CONDITIONER IS HIGH;
WEIGHT 1;
```

When editing or creating rules, keep in mind the following general guidelines.

- Definitions may be written in upper or lower case, but are stored in upper case.
- Every data item must be associated with an input or an output to one or more rules, but no data item may appear in both an input and an output.

- Statements may end with an optional semicolon.
- Blank lines are not permitted inside rule definitions.

Linking User-Defined Methods

Each method on the Parameters tab (**Connective**, **Implication**, **Defuzzification**, and **Aggregation**) allows you to select a user-defined option. This section tells you how to incorporate your own algorithm for one or more of the methods. Only one custom algorithm can be used for each method.

To add a user-defined algorithm, complete the following steps.

1. Copy the file `$(SYSBLD)/src/fuzusr.c` from the Xmath Commands window to your working directory.

This file contains the templates to begin the modification process:

```
float fuzand()
float fuzor()
void fuzimp()
float defuzz()
void fuzagr()
```

2. Modify the function body(s).

Do not change the function name(s); fuzzy logic is hard wired to accept only these names. Be very careful when using functions that return only values between 0 and 1. This is true regardless of the TRUE and FALSE limits set by the user in the **Database Parameters** field.

3. Incorporate the block into SystemBuild by linking in `fuzusr.c`.

- a. Copy the makefile file `$(SYSBLD)/bin/makefile` to the same directory as `fuzusr.c`.
- b. List `fuzusr.c` on the `CSOURCES` line.
- c. Include the keyword `-remake` in your `sim()` call.

The new `sim()` executable is remade automatically when `sim` is invoked.

If a user-defined function is selected in the FuzzyLogic Block Dialog, and no user-defined function is linked in, the simulator informs the user. A default algorithm is selected for simulation.

- Connective—Max-Min
- Implication—Mamdani

- Defuzzification—Means of Max
- Aggregation—Arithmetic

Debugging Hints

- **Assign a range of separate fuzzy inputs to the condition of each rule**—This lets you see how, by varying the degree of membership values, the crisp outputs are affected.
- **Assign a fuzzy data output to the assertion of each rule**—Assign different fuzzy outputs to each rule to see how a set of crisp inputs are evaluated with respect to the degree of membership value in each rule's IF statement.

Using the FuzzyLogic Tool

The FuzzyLogic tool is a MathScript program that provides a user interface for editing the FuzzyLogic block. The tool interface handles entries that would otherwise be made in the Parameters tab or the Code tab of the FuzzyLogic Block Dialog. Also, the tool provides a plotting capability not available in the SystemBuild Editor.

To invoke the FuzzyLogic tool, from the Xmath Commands window, enter `uifuzzy, {blockid=ID}`

where `ID` is the block number of a FuzzyLogic block instantiated in the editor.

Notice the following:

- The FuzzyLogic tool edits a specific FuzzyLogic block in an active editor window; the block must already be instantiated.
- The block dialog box and the FuzzyLogic tool cannot be used at the same time.
- When editing a value or string field, you must type `Return` to signal the tool to accept the entry.
- When editing rules or classes you must click the **Add/Update** button to signal the tool to accept the entry.

The tool has four tabs: GENERAL, RULES, DATA, and CLASSES AND QUALIFIERS. The tabs have common elements; each contains a selectable item list view, editable fields, and action buttons (**Add/Update** and **Remove**). Some tabs provide access to a simple plotting capability with the **Plot One** and **Plot All** buttons.

A message pane, which reports entry syntax and order errors, resides below the tabs. The buttons **OK**, **Cancel**, and **Help** are at the bottom of the window. **OK** saves any changes and exits the tool, while **Cancel** exits the tool without saving. The **Help** button displays an overview of the tool.

Typically, the process of creating fuzzy logic rules starts with defining the necessary global classes and qualifiers, followed by the data and associated member classes. Upon termination with the save option (click **OK**), the tool loads the FuzzyLogic block content and discards any unused data, classes and qualifiers created during the editing session.

Using the General Tab

The **GENERAL** tab allows you to set the same values as those on the **Parameters** tab in the FuzzyLogic Block Dialog. Additionally, it displays the current number of rules, data definitions, global classes, and qualifiers.

Using the Classes and Qualifiers Tab

The **GENERAL** tab allows you to edit existing global classes or create new global classes and qualifiers. The panes on the left list existing classes and qualifiers. An asterisk (*) before a name means that the class or qualifier is in use and cannot be removed.

To view the definition of a class or qualifier, click its name in the panes on the left. The information is displayed in the fields on the right.

To add a new class or qualifier, complete the following steps.

1. Enter an alphanumerical name.
Do not start the name with a number.
2. Enter a definition in the appropriate field.
3. Click **Add/Update**.

To plot a newly defined class or qualifier, click **Plot One**.

To plot all global classes or qualifiers, click **Plot All**.

To remove a class or qualifier, complete the following steps.

1. Select the class or qualifier in the left pane.
2. Click the **Remove** button.

To change the plot range of all global classes created in this tab, modify the **Minimum Range** and **Maximum Range** fields on the **GENERAL** tab.

Using the Data Tab

Input and output data is described by its attributes: the data type (crisp or fuzzy), the associated local and global classes, the data range, and the data channel number. Additionally, the tool displays a list of all available global classes in the lower right corner of the DATA tab pane.

Looking at the DATA tab, a list of all existing data appears in the upper left pane. If data is used in any rule assertion or implication expressions (IF or THEN statements on the RULES tab), the prefix “input” or “output” appears in front of the name. These prefixes denote that the input or output is in use and cannot be removed from the list at this time.

Data names are listed as follows—input data, output data, and any unused data. Within the input and output listings, the order of appearance is based on the associated channel number. The unused data does not have any channel numbers associated with them.

To add data, complete the following steps.

1. Enter a unique alphanumeric name.
2. Select the data **Type**.
3. If the data **Type** is **CRISP**, you must define other data attributes. Refer to the *Crisp Data and Fuzzification* section of Chapter 2, *Fuzzy Logic Fundamentals*, for more information.
4. When all attributes of the data are properly defined, click **Add/Update**.

Data can be associated with multiple local or global class members. Use the class editor on the DATA tab to add/remove/modify a local class member.

To add a global class member, complete the following steps.

1. Select the global class from the list (on the lower right pane).
2. Click **Add/Update**.

To remove a class, complete the following steps.

1. Select the class in the **Member Classes** list.
2. Click **Remove**.

Using the Rules Tab

To add a new rule, complete the following steps.

1. Enter an alphanumeric name.
2. Enter an assertion statement (**IF**).
3. Enter an implication statement (**THEN**).
4. Enter an aggregation **Weight**.
5. Click **Add/Update**.

To edit an existing rule, complete the following steps.

1. Select the rule from the **Rules** list on the left pane.
2. Perform the necessary modifications.
3. Click **Add/Update**.

To remove a rule, complete the following steps.

1. Select the rule from the **Rules** list on the top left pane.
2. Click **Remove**.

FuzzyLogic Block Example

The example presented in this chapter solves an intuitively simple physical problem that requires a certain amount of mathematical sophistication when solved by conventional mathematical control system means. While demonstrating how to use the FuzzyLogic block by example, it also shows how intuitively obvious the concepts of fuzzy logic solutions can be.

Introducing the Model

Load the demonstration model. In the Xmath Commands window, type:

```
load "$SYSBLD/examples/pendulum/pendulum.cat"
```

Edit the PENDULUM SuperBlock. There are two blocks in the PENDULUM SuperBlock, the inverted Pendulum Model SuperBlock and the fuzzy logic controller, as shown in Figure 4-1.

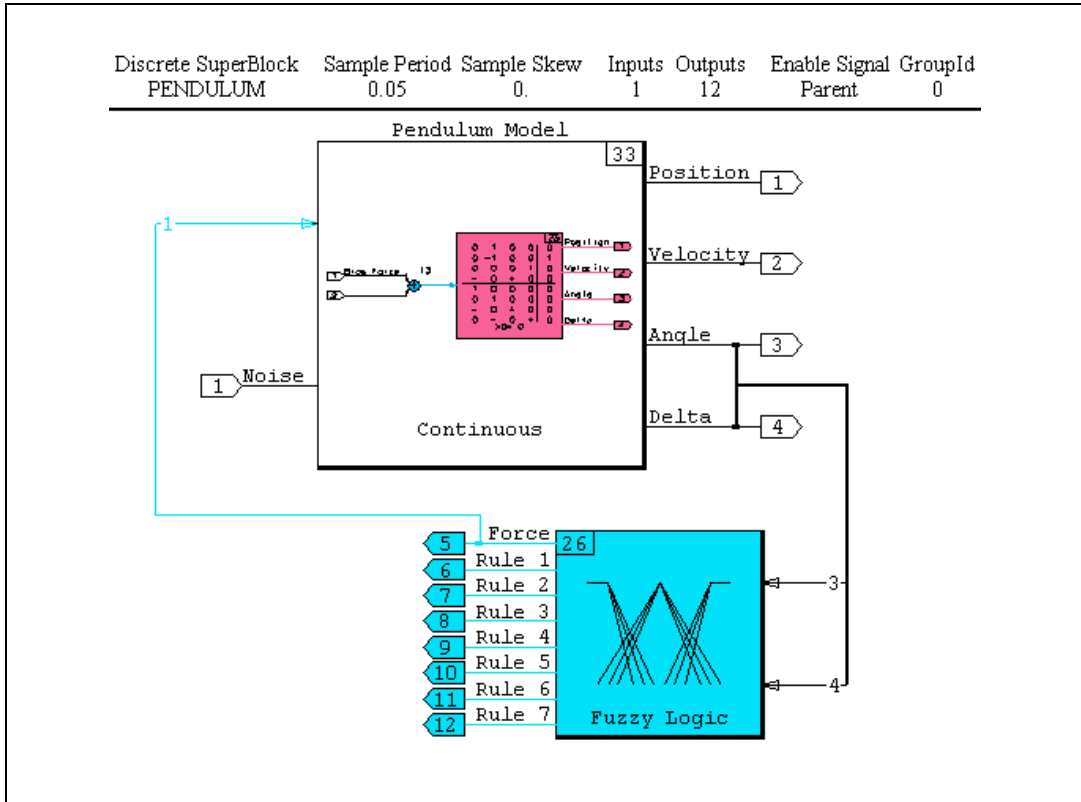


Figure 4-1. PENDULUM Model

The inverted pendulum is represented by a continuous SuperBlock with two inputs and four outputs. The two inputs, *Force* from the controller and *Noise* from an external input, are added together then input to the block representing the inverted pendulum dynamics.

Two outputs from the PENDULUM SuperBlock, *Angle* and *Delta*, are crisp data inputs to the fuzzy controller block. The fuzzy controller has one crisp output, *Force*, which connects back to the PENDULUM SuperBlock. The fuzzy controller also has seven other outputs, each a fuzzy data output for each rule.

Generating the Rules

Open the FuzzyLogic Block Dialog. The Code tab reveals that the fuzzy controller has seven rules, ten data variables, five global classes, and no qualifiers. The rules were generated from intuitively reasonable rules about the behavior of a pendulum:

```
IF the pole is vertical and DELTA is about 0,
THEN apply no force to the base.
```

```
IF ANGLE is a little positive/negative and DELTA is a
little negative/positive, THEN apply no force to the
base.
```

```
IF ANGLE is a little positive/negative and DELTA is a
little positive/negative, THEN apply a small
positive/negative force to the base.
```

```
IF ANGLE is positive/negative, and DELTA is small, then
move the base in the positive/negative direction.
```

These rules are translated to the seven fuzzy IF/THEN rules found inside the FuzzyLogic block: NM_ZR, NS_NS, NS_PS, PM_ZR, PS_NS, PS_PS, ZR_ZR. Each rule also has a fuzzy data output, which stores the result of each rule conditional. This is helpful for understanding and debugging the dynamics of the fuzzy logic controller.

Crisp data definitions are created from the rules. The inputs *Angle* and *Delta* are defined as:

```
DATA ANGLE;
TYPE CRISP;
RANGE [-2, 2];
MEMBERS NEG_MEDIUM GLOBAL
        NEG_SMALL GLOBAL
        ZERO GLOBAL
        POS_SMALL GLOBAL
        POS_MEDIUM GLOBAL;

DATA DELTA;
TYPE CRISP;
RANGE [-20, 20];
MEMBERS NEG_SMALL GLOBAL
        ZERO GLOBAL
        POS_SMALL GLOBAL;
```

The output, *Force*, is defined as:

```
DATA FORCE;
TYPE CRISP;
RANGE [-700,700];
MEMBERS NEG_MEDIUM GLOBAL
        NEG_SMALL GLOBAL
        ZERO GLOBAL
        POS_SMALL GLOBAL
        POS_MEDIUM GLOBAL;
```

Creating Membership Functions

We now must create a set of membership functions for each linguistic variable. We could create a different set of functions for each datum, each sculpted to the exact meaning of the variable in relation to the crisp datum, but this would require more memory and possibly slow down processing times. Instead, we use one set of functions, and rely on the range scaling in each crisp datum definition to give the appropriate meaning for each curve. The functions are as follows.

```
NEG_MEDIUM SIN(PI*(X+0.5))**9;
NEG_SMALL  SIN(PI*(X+0.25))**9;
ZERO       SIN(PI*X)**9;
POS_SMALL  SIN(PI*(X-0.25))**9;
POS_MEDIUM SIN(PI*(X-0.5))**9;
```

For the datum *Delta*, the variable *NEG_SMALL* has a non-zero degree of membership in the range -20 to 0 . For the data *Angle*, *NEG_SMALL* has a non-zero degree of membership in the range -2 to 0 .

One might have a different or enlarged set of rules for this problem. Fuzzy logic requires trial and error to see how many rules are needed. Additional rules, data, and classes require more memory and lower processing speeds.

Running the Simulation

We are now ready to run the simulation. In the Xmath Commands window, type:

```
t=[0:.05:70]';
set seed 0
in=.1*random(t);y = sim("PENDULUM", t, in);
```

Plot the first four channels (*Position*, *Delta*, *Angle*, and *Delta*) of the `pdm y` by typing:

```
plot(y(1:4,:), {strip})
```

The *Angle* varies back and forth about 0° , as shown in Figure 4-2.

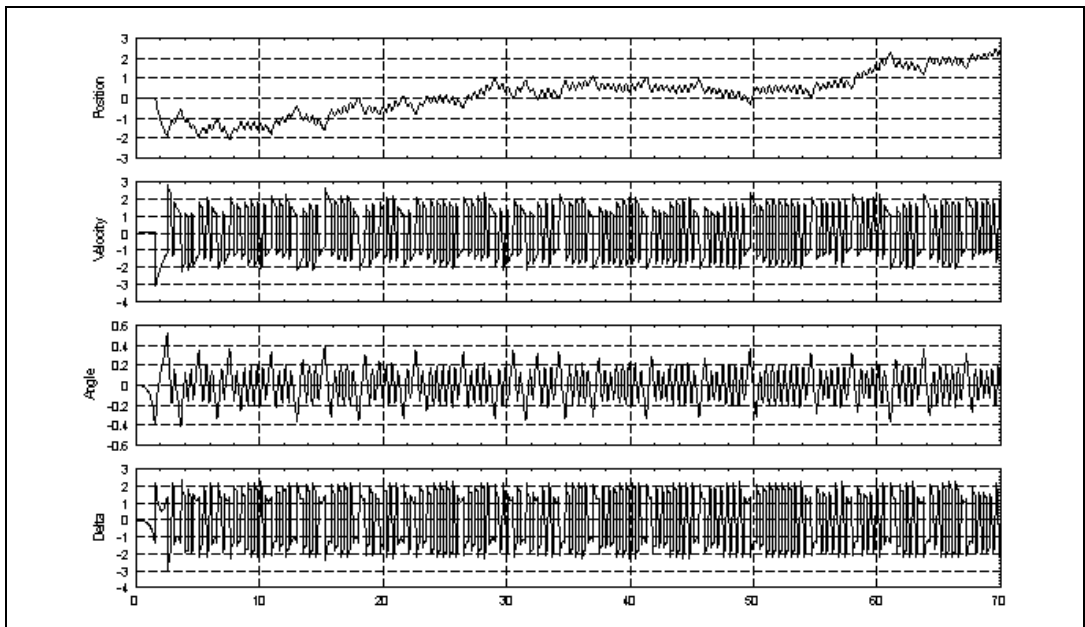


Figure 4-2. Plot of the Pendulum Example

Comparing Times to Run the Simulation

We want to compare the time needed to run the simulation using **Memory**, **Speed**, and **Compromise Optimization** settings.

To compare simulation time for various optimization settings, complete the following steps.

1. Ensure that the **Optimization** field of the Parameters tab on the FuzzyLogic Block Dialog is set to **Speed**.

2. In the Xmath Commands window, type:

```
zt = clock();  
y = sim("PENDULUM", t, in);  
et = clock();
```

`clock()` returns 0 the first time it is called, and the elapsed time since the last call on every subsequent call.

3. To obtain the elapsed time, type:
`et?`
4. Change **Optimization** to **Memory**.
5. Rerun the simulation, and display the elapsed time. The result shows the operation takes much longer.
6. Change **Optimization** to **Compromise**.
7. Rerun the simulation, and display the elapsed time.

The output is a compromise between the first and second values.



Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources at ni.com/support include the following:
 - **Self-Help Resources**—For immediate answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
 - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at ni.com/exchange. National Instruments Application Engineers make sure every question receives an answer.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Index

A

aggregation
 definition, 2-4
 methods, harmonic, 2-10

C

centroid defuzzification method, 2-9
compromise option, 3-3, 4-6
connective methods, 2-7
 definition, 2-3
conventions used in the manual, *iv*
crisp datum, 2-4

D

defuzzification
 definition, 2-3
 methods, 2-9
 centroid, 2-9
 means of maximum, 2-9
diagnostic tools (NI resources), A-1
documentation
 conventions used in the manual, *iv*
 NI resources, A-1
drivers (NI resources), A-1

E

examples (NI resources), A-1

F

fuzzification, 2-2, 2-3, 2-5
fuzzy
 conditional, 2-5
 data type, 3-12, 3-13

logic

 definition, 2-1
 design parameters and issues, 2-5

FuzzyLogic

 block

 debugging, 3-15
 edit with UI tool, 3-15
 example, 4-1
 fuzusr.c file, 3-14
 introduction, 1-1
 memory option, 3-2
 speed option, 3-3
 trapezoidal function, 3-6
 triangular function, 3-6

 tool, 3-15

H

harmonic mean, 2-10
help, technical support, A-1

I

implication

 definition, 2-8
 methods, 2-8
 Larsen, 2-8
 Mamdani, 2-8
 problem description, 2-3

instrument drivers (NI resources), A-1

K

KnowledgeBase, A-1

L

Larsen implication method, 2-8

M

- Mamdani implication method, 2-8
- means of maximum defuzzification method, 2-9
- memory
 - considerations, 2-12
 - option, 4-6

N

- National Instruments support and services, A-1

O

- optimization preference, 4-6

P

- parallelism, 2-13
- plot data with UI tool, 3-15
- programming examples (NI resources), A-1

Q

- qualifiers, effects on fuzzification, 2-5

R

- rule guidelines, 3-13

S

- software (NI resources), A-1
- speed
 - considerations, 2-12
 - option, 4-6
- support, technical, A-1

T

- technical support, A-1
- training and certification (NI resources), A-1
- troubleshooting (NI resources), A-1

W

- Web resources, A-1