MATHEMATICA®
FUZZY LOGIC

THE MOST FLEXIBLE ENVIRONMENT
FOR EXPLORING FUZZY SYSTEMS

WOLFRAMRESEARCH
www.wolfram.com

For use with *Mathematica* 5.0.
For the latest updates and corrections to this manual visit: documents.wolfram.com.
Comments on this manual are welcomed at: comments@wolfram.com.

Printed in the United States of America.

XXXX XXXXX 0604.rcm

To my friends from the North America Fuzzy Information Processing Society and from the International Fuzzy Systems Association.

Marian S. Stachowicz

To Linda Diane, Shane, Matt, and Christian.

Lance Beall

Table of Contents

Part 2 Demonstration Notebooks

8 Additional Examples

Introduction

1 *Fuzzy Logic* Package

Fuzzy Logic is a collection of notebooks and packages that are designed to introduce fuzzy set theory and fuzzy logic in the *Mathematica* environment. The packages provided in *Fuzzy Logic*, combined with *Mathematica*, provide a powerful tool for studying fuzzy logic and for developing fuzzy applications. The notebooks provided with this package demonstrate how the various fuzzy logic functions are used, and they contain many worked examples showing how this package can be used in real-world applications.

Basic familiarity with *Mathematica* is assumed for use of *Fuzzy Logic*. For more information on Mathematica, see below or contact Wolfram Research, Inc.

New functions added to this updated version of *Fuzzy Logic* package are described in 0_02_New_-Features.nb.

2 The Manual

The manual is a collection of *Mathematica* notebooks that show the capabilities of Fuzzy Logic package and demonstrate the use of all of the functions. Notebooks are interactive documents combining Mathematica input and output, text, and graphics. The materials for the manual come from a wide variety of sources; many of the examples are from works listed in the 1_14_Bibliography.nb notebook. The following is a brief description of the types of examples that can be found in each of the notebooks in the manual.

1_01_CreateFuzzySets.nb

Demonstrates the functions used to create fuzzy sets and options associated with each of the functions.

1_02_CreateFuzzyRelations.nb

Demonstrates the functions used to create fuzzy relations and the options associated with each of these functions.

1_03_FuzzyOperations.nb

Contains examples of different operations that can be applied to fuzzy sets, fuzzy relations, or both.

1_04_FuzzyAggregators.nb

Contains examples of the various package functions for combining two or more fuzzy sets or fuzzy relations, including a full range of t-norm (intersection), s-norm (union), and averaging operators. Both standard and nonstandard operators are included.

1_05_ViewFuzzySets.nb

Demonstrates the various ways to view fuzzy sets in this package, including a variety plotting methods.

1_06_ViewFuzzyRelations.nb

Demonstrates the various ways to view fuzzy relations in this package, including different plot styles and membership matrices.

1_07_Compositions.nb

Shows how to perform a composition between two fuzzy relations.

1_08_FuzzyInference.nb

Shows how to perform fuzzy inferences and demonstrates some applications of fuzzy inferencing for modeling and control.

1_09_FuzzyArithmetic.nb

Shows how to apply fuzzy arithmetic operations to fuzzy numbers. These functions are used only with triangular or trapezoidal fuzzy numbers.

1_10_DiscreteFuzzyArithmetic.nb

Contains examples demonstrating the use of the discrete fuzzy arithmetic operations. These arithmetic operations work on any fuzzy sets or fuzzy numbers.

1_11_Lukasiewicz.nb

Contains examples demonstrating the use of the digital fuzzy sets and the multivalued logic functions.

1_12_FuzzyClustering.nb

Demonstrates the Fuzzy C-Means Clustering algorithm and shows an example.

1_13_Appendix.nb

Contains a partial list of the formulas for the various operations in *Fuzzy Logic* package.

1_14_Bibliography.nb

Contains a bibliography list for selected applications areas and theoretical topics.

3 The Demonstration Notebooks

The notebooks contained in the package consist of worked examples of various fuzzy topics that can be studied using *Fuzzy Logic* package. If you are just learning fuzzy sets, the first three notebooks are a good place to start. These notebooks contain information about fuzzy concepts and the mathematics behind fuzzy sets. The following is a list describing all of *Fuzzy Logic* notebooks.

2_01_SetVersusFuzzySet.nb

Contains an introduction to fuzzy sets and compares and contrasts fuzzy sets with traditional sets.

2_02_StandardOperations.nb

Contains examples that demonstrate the use of the various operations that can be performed on fuzzy sets.

2_03_FuzzyRelations.nb

Introduces fuzzy relations and gives examples of creating, operating on, and using fuzzy relations.

2_04_FuzzyModeling.nb

Demonstrates how fuzzy sets and fuzzy relations can be used in real-world process modeling.

2_05_FuzzyControl.nb

Introduces fuzzy logic control. This notebook contains a truck-backing control example and a step-by-step description of the design process.

2_06_FuzzyNumbers.nb

Contains an introduction to fuzzy numbers and fuzzy arithmetic, as well as a number of examples demonstrating the use of fuzzy arithmetic.

2_07_DigitalFuzzySets.nb

Contains an introduction to analog, discrete, and digital fuzzy sets, as well as a number of examples demonstrating the different digital fuzzy numbers.

2_08_Examples.nb

Describes a wide variety of practical situations in which fuzzy sets and fuzzy logic can be applied. Examples in this notebook are presented in a problem-solution format.

4 The Packages

Mathematica packages are files written in the *Mathematica* programming language. They contain *Mathematica* definitions that extend *Mathematica*'s capabilities in a particular area. The packages contained in *Fuzzy Logic* package contain definitions that allow *Mathematica* to work with fuzzy sets and fuzzy relations. In *Fuzzy Logic* package, we attempted to adhere to the conventions of good *Mathematica* programming, so the packages can be used as models for your own algorithms. *Fuzzy Logic* notebooks make use of the definitions contained in the packages.

The packages are found in several files, all of which can be accessed by loading the `FuzzyLogic`` init file. The following is a list briefly describing the definitions that are contained in each package. The list is provided to allow you to view the code for a particular routine or group of routines. In addition to the packages, a manual is included with the code. The manual is a notebook that can be used either as an introduction to the functions in this package or as a quick reference to look up function names and uses.

`FuzzyLogic`Common``
Contains definitions that are used within one or more other packages. It contains the code that defines fuzzy sets and fuzzy relations.

`FuzzyLogic`Creation``
Contains definitions that can be used to create fuzzy sets and fuzzy relations.

`FuzzyLogic`SetToValue``
Contains definitions that take fuzzy sets or relations as arguments and returns a single value as output.

`FuzzyLogic`SetToList``
Contains functions that take fuzzy sets or fuzzy relations as inputs and returns a list of items as output.

`FuzzyLogic`SetToSet``
Contains programs that take fuzzy sets or fuzzy relations as arguments and returns new fuzzy sets or fuzzy relations.

`FuzzyLogic`Aggregate``
Contains definitions that can be used to aggregate two or more fuzzy sets or fuzzy relations. Definitions of intersections and unions can be found here.

`FuzzyLogic`Average``
Contains functions that average two or more fuzzy sets or fuzzy relations.

`FuzzyLogic`Visual``
Contains functions to display fuzzy sets or fuzzy relations visually.

`FuzzyLogic`Composition``
Contains functions that perform the composition of fuzzy relations.

`FuzzyLogic`Arithmetic``
Contains the definitions for performing fuzzy arithmetic with fuzzy numbers.

`FuzzyLogic`Inference``
Contains the fuzzy logic inferencing functions that are used for fuzzy logic control and modeling.

`FuzzyLogic`Lukasiewicz``
Contains the digital fuzzy sets and the multivalued logic functions.

`FuzzyLogic`Cluster``
Contains the fuzzy clustering functions.

5 Loading the Package

After starting a *Mathematica* session, the proper functions must be loaded from *Fuzzy Logic* package. There are a number of ways to do this.

The most convenient way is to load the subpackage `FuzzyLogic`` init. This can be done with the following command:

```
Needs["FuzzyLogic`"]
```

After entering the command above, all of the functions from *Fuzzy Logic* package will be available. The subpackages containing the function will be automatically loaded when needed.

The `Get` command can be used instead of the `Needs` command, but if the package is already loaded and you try to reload it with the `Get` command, you may see some error messages. For this reason, we recommend using the `Needs` command. If you want to use the `Get` command, you can replace the word `Needs` with `Get`, or you can type the following:

```
<< FuzzyLogic`
```

> *If the package loading command fails, verify that the directory containing Fuzzy Logic directory is on Mathematica's* `$Path`*. You can use the command* `AppendTo[$Path, "directory"]`*. For convenience, this should be in your* `init.m` *file.*

6 Getting More Information about *Mathematica*

For information on standard *Mathematica* operations and packages or for general *Mathematica* information, *Mathematica: A System for Doing Mathematics by Computer*, Fourth Edition by Stephen Wolfram is the definitive reference. There are a number of books that describe programming in the *Mathematica* language; we found Roman Maeder's programming books to be very helpful. In addition to these resources, there are many additional books and magazines that are dedicated to providing information about *Mathematica*. For a comprehensive list of resources or for further information regarding *Mathematica*, check the Wolfram Research web site at `www.wolfram.com`

7 Getting More Information about Fuzzy Sets

There are a large number of good publications about fuzzy sets and fuzzy logic on the market. The sources we used to create this package are listed in the 1_14_Bibliography.nb notebook in the Notebooks folder. We recommend looking at these sources to provide a better understanding of fuzzy concepts and to get ideas for different applications. A partial list of the formulas used in this package is provided in the 1_13_-Appendix.nb notebook (in the Notebooks folder).

8 Acknowledgments

We would like to thank the following people who have been of great assistance during the development of *Fuzzy Logic* package. Dr. Leszek Sczaniecki, formerly Director of Applications at Wolfram Research, Inc., was instrumental in the design and development of the package. He also contributed significantly to the improvements of the present version, primarily the documentation. Thanks to Wolfram Research, Inc., in particular John Novak, Julia Guelfi, and Rebecca Bigelow for helping us develop this package. Thanks to Dr. Maria E. Kochanska, who wrote a special package in Basic for graphical interpretation of fuzzy sets and its operations in 1982. We are grateful to Med. D. Pawel P. Stachowicz for his fabulous *Fuzzy as Hell-1989* and Robert Walker for his *Ann-1992* programs written for fuzzy modeling. A special thanks to Jonathan Andersh, who wrote the educational version of the program. Thanks to Sister Maria Magdalena for helping with proofreading. Thanks to all the members of the Laboratory for Intelligent Systems at the University of Minnesota Duluth for their helpful comments. Thanks to Bei Tang for her beautiful cluster of the fuzzy flowers. Thanks to Dan Holmdahl for his Łukasiewicz logic notebook. Finally, we would like to thank Dr. Stachowicz's *Fuzzy Set Theory and Its Applications* classes and students Chaohui Yang, Guangji Shi, Dan Yao, and Ben Anderson, who used this package for homework assignments and provided useful comments.

9 About the Authors

Marian S. Stachowicz is professor and Jack Rowe Chair at the University of Minnesota Duluth. He heads the Laboratory for Intelligent Systems in the Electrical and Computer Engineering Department. He is also a graduate professor for Computer Science Department UM Duluth campus, ECE Department and Control Science and Dynamical Systems UM Twin Cities campus. He received his M.S. degree in Control and Computer Engineering from LETI, Soviet Union and both his Ph.D. and D.Sc. from A G-H, Poland. Dr. Stachowicz received two awards from the Polish Ministry of Higher Education and Science for the introduction of the digital fuzzy sets into the fuzzy set theory, which has facilitated applications of the fuzzy set theory to computer engineering. His work centers on artificial intelligence and soft computing, decision analysis, and control. Recently, he had worked for Berkeley Initiative in Soft Computing, University of California, Berkeley. He is a member of the North American Fuzzy Information Processing Society, a member of the International Fuzzy Systems Associations, a Senior Member of IEEE, and a consultant for large multinational corporations.

Lance E. Beall has a Bachelor of Computer Engineering degree from the University of Minnesota Duluth, where he graduated summa cum laude. He is currently working for Medtronic and pursuing a Master's Degree at the University of Minnesota Twin Cities.

New Features in This Release

1 Introduction

This notebook demonstrates the new features that were added to *Fuzzy Logic* package.

```
In[1]:= << FuzzyLogic`
```

```
In[2]:= SetOptions[FuzzySet, UniversalSpace → {0, 100, 1}];
```

The first thing to note is that originally the universal space for fuzzy sets in *Fuzzy Logic* package was defined only on the integers. In the new version, the universal space for fuzzy sets and fuzzy relations is defined with three numbers. The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between elements. This gives the user more flexibility in choosing the universal space.

2 New Membership Functions

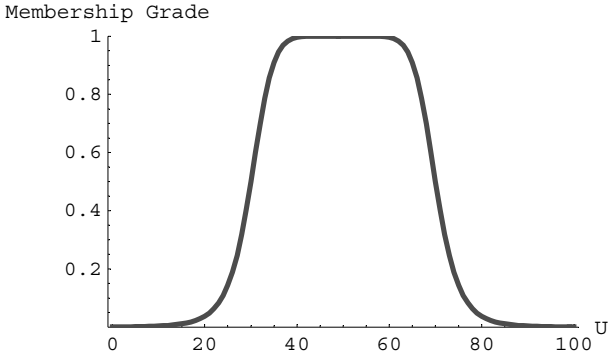
A number of new functions and options were added for creating membership functions. Here is a demonstration of the additions.

Bell-shaped Fuzzy Sets

`FuzzyBell[`*c*`,` *w*`,` *s*`,` *opts*`]` returns a bell-shaped fuzzy set centered at *c* with crossover points at *c* - *w* and *c* + *w* with a slope of *s* / 2*w* at the crossover points.

```
In[3]:= FS1 = FuzzyBell[50, 20, 4];
```

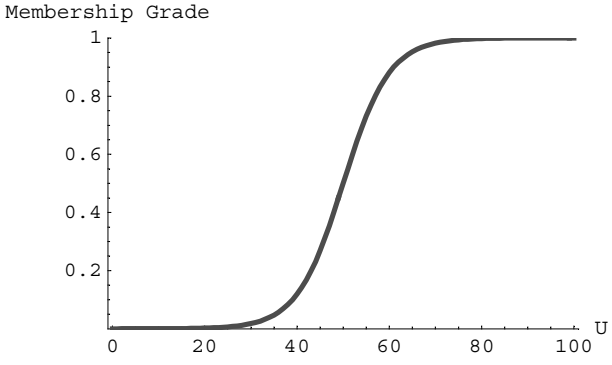
```
In[4]:= FuzzyPlot[FS1, PlotJoined → True];
```



Sigmoidal Fuzzy Sets

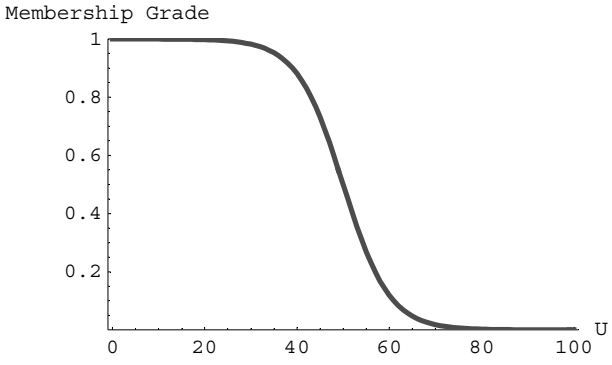
`FuzzySigmoid[`*c*`, `*s*`, `*opts*`]` returns a sigmoidal fuzzy set where *s* controls the slope at the crossover point *c*. A positive slope gives a sigmoidal fuzzy set, which opens to the right, and a negative slope gives a fuzzy set, which opens to the left.

```
In[5]:= FS2 = FuzzySigmoid[50, .2];

In[6]:= FuzzyPlot[FS2, PlotJoined → True];
```




```
In[7]:= FS3 = FuzzySigmoid[50, -.2];
```

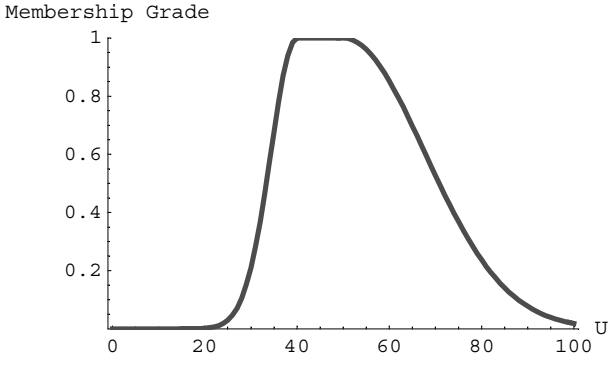
```
In[8]:= FuzzyPlot[FS3, PlotJoined → True];
```



Double-sided Gaussian Fuzzy Sets

`FuzzyTwoGaussian[`*mu1*`,` *sigma1*`,` *mu2*`,` *sigma2*`,` *opts*`]` returns a two-sided Gaussian fuzzy set with centers at *mu1* and *mu2* and widths of *sigma1* and *sigma2*. Between the two mean, the fuzzy set has a membership grade of 1.

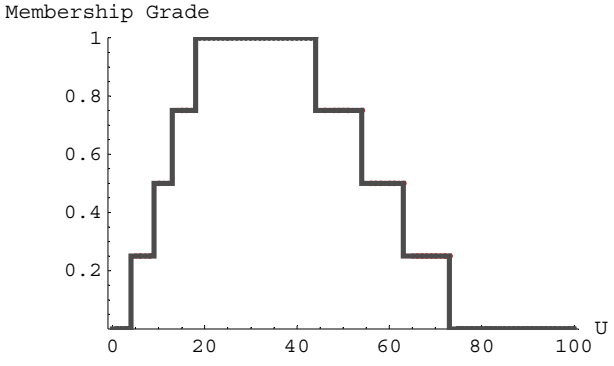
```
In[9]:= FS4 = FuzzyTwoGaussian[40, 8, 50, 25];

In[10]:= FuzzyPlot[FS4, PlotJoined → True];
```



Digital Fuzzy Sets

`DigitalSet[a, b, c, d, (h), opts]` returns a digital fuzzy set with the number of membership grades equal to *n*, where *n* is set by an option for this function. The universal space and the value of *n* may be defined using the `UniversalSpace` and the *n* option, otherwise the default universal space and default *n* value will be given. The values of the membership grades increase linearly from *a* to *b*, then are equal to the closest possible value of *h* from *b* to *c*, and linearly decrease from *c* to *d*. Arguments *a*, *b*, *c*, and *d* must be in increasing order, and *h* must be a value between 0 and 1 inclusive. If a value for *h* is not given, it defaults to 1. In the following example, you can create a fuzzy set with *n* set to 5. This means you will get a L5 fuzzy set or a fuzzy set with five possible membership grades.

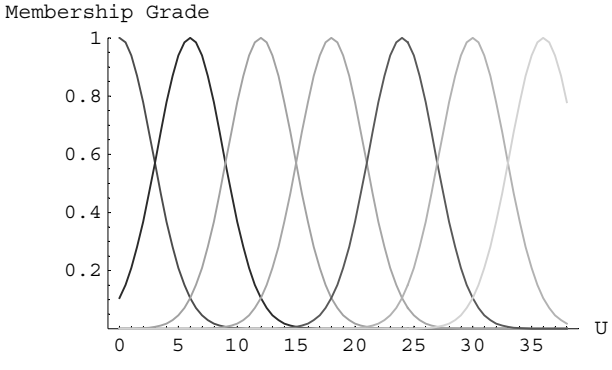
```
In[11]:= D1 = DigitalSet[1, 20, 40, 78, Levels → 5];

In[12]:= FuzzyPlot[D1, Crisp -> True];
```



ChopValue Option

An option was added to the `FuzzyGaussian`, `FuzzyBell`, `FuzzySigmoid`, `FuzzyTwoGaussian`, and `CreateFuzzySets` functions that allows you to specify a `Chop` value to the function. Any membership grades less than this value are taken as zero. When working with Gaussian type functions, every element will have some membership grade, and it is often the case that many of the elements have very small membership grades. Chopping off these elements allows fuzzy sets to be more compact and allows calculations to be quicker. It also aids in creating fuzzy graphs, which are described later. The following example demonstrates the `ChopValue` option.

```
In[13]:= FS5 = FuzzyGaussian  [20, 5, UniversalSpace  → {0, 40, 1}]

Out[13]= FuzzySet[{{0, 1.12535×10^-7}, {1, 5.35535×10^-7}, {2, 2.35258×10^-6},
           {3, 9.54016×10^-6}, {4, 0.0000357128}, {5, 0.00012341}, {6, 0.000393669},
           {7, 0.00115923}, {8, 0.00315111}, {9, 0.00790705}, {10, 0.0183156},
           {11, 0.0391639}, {12, 0.0773047}, {13, 0.140858}, {14, 0.236928}, {15, 0.367879},
           {16, 0.527292}, {17, 0.697676}, {18, 0.852144}, {19, 0.960789}, {20, 1.},
           {21, 0.960789}, {22, 0.852144}, {23, 0.697676}, {24, 0.527292}, {25, 0.367879},
           {26, 0.236928}, {27, 0.140858}, {28, 0.0773047}, {29, 0.0391639}, {30, 0.0183156},
           {31, 0.00790705}, {32, 0.00315111}, {33, 0.00115923}, {34, 0.000393669},
           {35, 0.00012341}, {36, 0.0000357128}, {37, 9.54016×10^-6}, {38, 2.35258×10^-6},
           {39, 5.35535×10^-7}, {40, 1.12535×10^-7}}, UniversalSpace → {0, 40, 1}]

In[14]:= FS6 = FuzzyGaussian  [20, 5, UniversalSpace → {0, 40, 1}, ChopValue → .01]

Out[14]= FuzzySet[{{10, 0.0183156}, {11, 0.0391639}, {12, 0.0773047}, {13, 0.140858},
           {14, 0.236928}, {15, 0.367879}, {16, 0.527292}, {17, 0.697676}, {18, 0.852144},
           {19, 0.960789}, {20, 1.}, {21, 0.960789}, {22, 0.852144}, {23, 0.697676},
           {24, 0.527292}, {25, 0.367879}, {26, 0.236928}, {27, 0.140858},
           {28, 0.0773047}, {29, 0.0391639}, {30, 0.0183156}}, UniversalSpace → {0, 40, 1}]

In[15]:= Var2 = {NB, NM, NS, ZO, PS, PM, PB} =
           CreateFuzzySets  [7, Type → Gaussian [4, ChopValue -> 0.001 ],
            UniversalSpace → {0, 38, 0.5}];

In[16]:= FuzzyPlot  [Var2, PlotJoined  → True ];
```

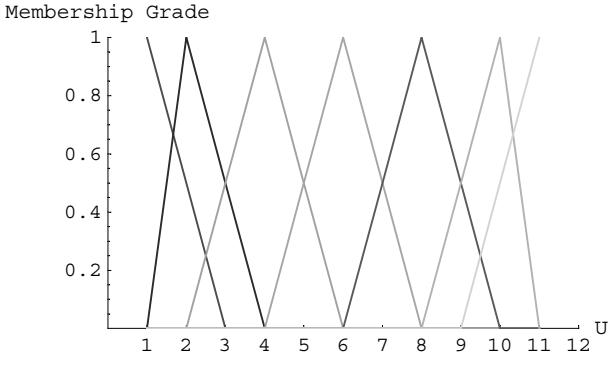


3 Fuzzy Graph

Another visualization function in *Fuzzy Logic* package is a fuzzy graph. A fuzzy graph describes a mapping between an input linguistic variable and an output linguistic variable. In essence, a fuzzy graph serves as an approximation to a function, which is described in words as a collection of fuzzy *if-then* rules. A fuzzy graph can be used to give an idea of what a set of fuzzy rules look like.

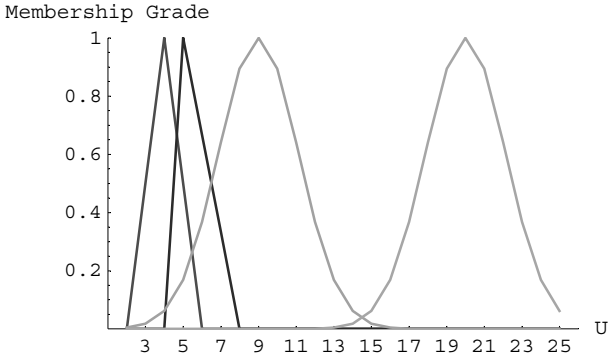
Demonstration

```
In[17]:= SetOptions[FuzzySet, UniversalSpace → {1, 11, 1}];

In[18]:= Input1 = {Tiny, VerySmall, Small, Medium, Big, VeryBig, Huge} =
           {FuzzyTrapezoid[1, 1, 1, 3], FuzzyTrapezoid[1, 2, 2, 4], FuzzyTrapezoid[
             2, 4, 4, 6], FuzzyTrapezoid[4, 6, 6, 8], FuzzyTrapezoid[6, 8, 8, 10],
            FuzzyTrapezoid[8, 10, 10, 11], FuzzyTrapezoid[9, 11, 11, 11]};

In[19]:= FuzzyPlot[Input1, PlotJoined → True];
```

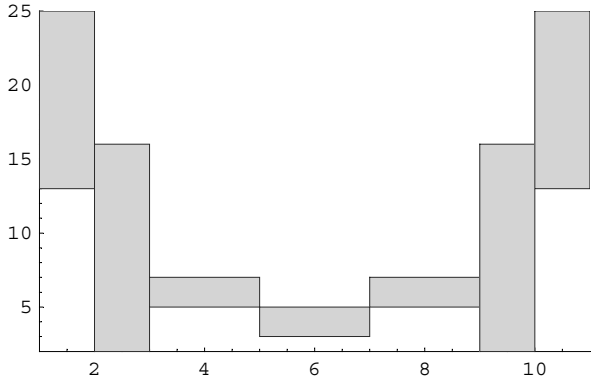



```
In[20]:= SetOptions[FuzzySet, UniversalSpace → {2, 25, 1}];

In[21]:= Output1 = {VeryLow, Low, Middle, High} =
           {FuzzyTrapezoid[2, 4, 4, 6], FuzzyTrapezoid[4, 5, 5, 8],
            FuzzyGaussian[9, 3, ChopValue → .001], FuzzyGaussian[20, 3, ChopValue → .001]};
```

```
In[22]:= FuzzyPlot[Output1, PlotJoined → True];
```




```
In[23]:= Rules1 = {{Tiny, High}, {VerySmall, Middle}, {Small, Low},
           {Medium, VeryLow}, {Big, Low}, {VeryBig, Middle}, {Huge, High}};

In[24]:= FuzzyGraph[Rules1];
```



From the `FuzzyGraph`, you can see that the single-input/single-output fuzzy system described by `Rules1` will produce a curve that is somewhat parabolic.

4 Defuzzifications

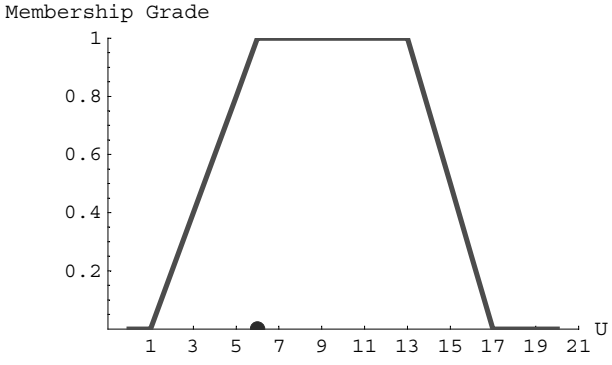
```
In[25]:= MF1 = FuzzyTrapezoid[1, 6, 13, 17, UniversalSpace → {0, 20}];
```

Smallest of Max (Defuzzification)

`SmallestOfMax[`A`]` returns the smallest of maximum defuzzification of fuzzy set A. This function also has a `ShowGraph` option for visualizing the defuzzification.

```
In[26]:= SmallestOfMax[MF1, ShowGraph → True, PlotJoined → True];
```

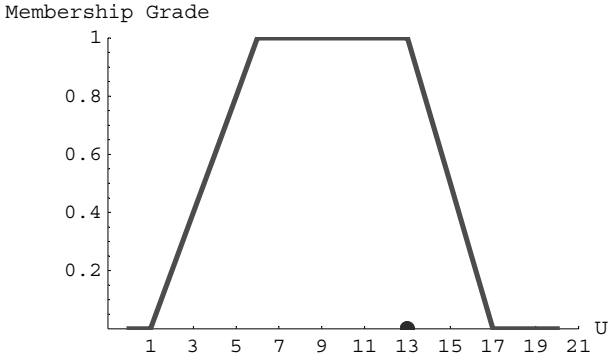
```
Smallest of max is 6.
```



Largest of Max (Defuzzification)

`LargestOfMax[`A`]` returns the largest of maximum defuzzification of fuzzy set A. This function also has a `ShowGraph` option for visualizing the defuzzification.

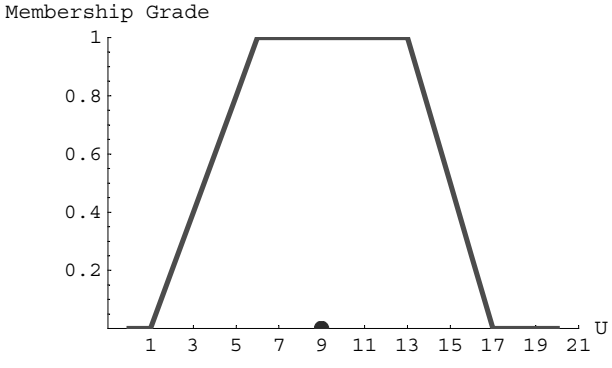
```
In[27]:= LargestOfMax[MF1, ShowGraph → True, PlotJoined → True];

         Largest of max is 13.
```



Bisector of Area (Defuzzification)

`BisectorOfArea[`A`]` returns the bisector of area defuzzification of fuzzy set A. This function also has a `ShowGraph` option for visualizing the defuzzification.

```
In[28]:= BisectorOfArea[MF1, ShowGraph → True, PlotJoined → True];

         Bisector of area is 9.
```



5 Additional Operators

Fuzzy Cardinality

`FuzzyCardinality[A]` returns the fuzzy cardinality of fuzzy set *A*.

```
In[29]:= FuzzyCardinality[MF1]

Out[29]= {{15, 1/5}, {14, 1/4}, {13, 2/5}, {12, 1/2}, {11, 3/5}, {10, 3/4}, {9, 4/5}, {8, 1}}
```

Core

This function was included in the original *Fuzzy Logic* package, but it was named `Nucleus`.

`Core[A]` returns a list of all elements of fuzzy set *A* with a membership grade equal to 1.

```
In[30]:= Core[FS4]

Out[30]= {40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50}
```

Subsethood

`Subsethood[A, B]` returns the degree of subsethood of fuzzy set/fuzzy relation *A* in fuzzy set/fuzzy relation *B*. The value returned will be between 0 and 1, with values closer to 1 indicating that the second fuzzy set/fuzzy relation is closer to being a subset of the first fuzzy set/fuzzy relation.

```
In[31]:= Subsethood[FS1, FS2]

Out[31]= 0.509356

In[32]:= Subsethood[FS2, FS1]

Out[32]= 0.413921
```

Hamming Distance

`HammingDistance[`A`, `B`]` returns the Hamming distance from fuzzy set/fuzzy relation A to fuzzy set/fuzzy relation B.

```
In[33]:= HammingDistance[FS1, FS2]

Out[33]= 49.7321
```

Level Set

LevelSet[A] returns the set of all levels alpha that represent distinct alpha cuts of a fuzzy set/relation A.

```
In[34]:= LevelSet[FS1]

Out[34]= {0.000654931, 0.000769727, 0.000907644, 0.00107397, 0.00127533, 0.00152012,
          0.00181898, 0.00218546, 0.00263693, 0.0031958, 0.00389105, 0.00476048,
          0.00585358, 0.00723556, 0.00899284, 0.0112405, 0.0141328, 0.017877,
          0.0227533, 0.0291409, 0.0375532, 0.0486834, 0.0634603, 0.0831089, 0.109202,
          0.143669, 0.188686, 0.246365, 0.318108, 0.40364, 0.5, 0.601171, 0.699072,
          0.78586, 0.856331, 0.908998, 0.945494, 0.96912, 0.983481, 0.991696, 0.996109,
          0.998321, 0.999345, 0.999775, 0.999934, 0.999985, 0.999997, 1., 1., 1., 1.}
```

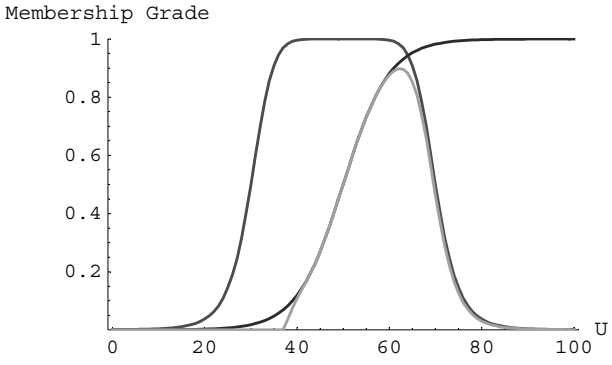
6 New Intersections and Unions

Yu Type Union and Intersection

`Yu[`l`]` is an additional value for `Type` option for the union and intersection operations. The parameter l must be greater than -1.

```
In[35]:= YuInt = Intersection[FS1, FS2, Type → Yu[2]];
```

```
In[36]:= FuzzyPlot[FS1, FS2, YuInt, PlotJoined → True];
```

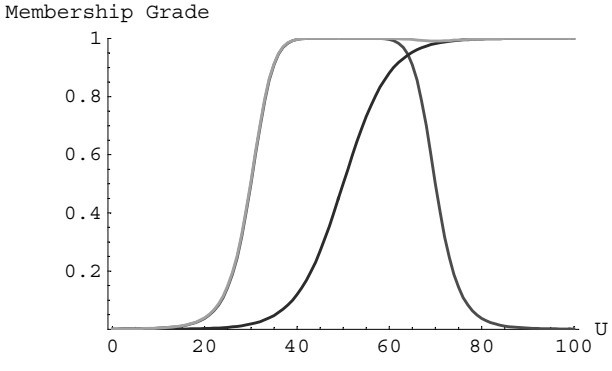


Weber Type Union and Intersection

`Weber[l]` is an additional new value for `Type` option for the union and intersection operations. The parameter *l* must be greater than -1.

```
In[37]:= WeUn = Union[FS1, FS2, Type -> Weber[0.5]];
```

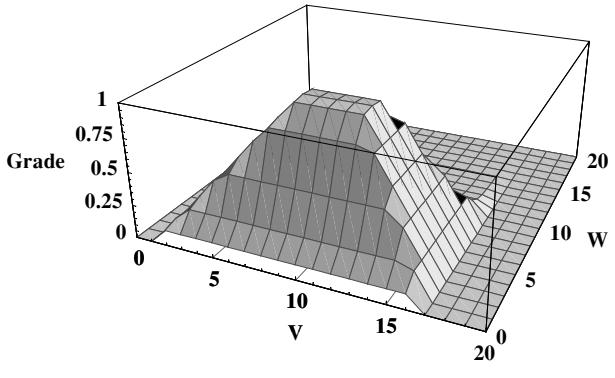
```
In[38]:= FuzzyPlot[FS1, FS2, WeUn, PlotJoined → True];
```



7 Alpha Cuts for Fuzzy Relations

One of the most important concepts of fuzzy sets/fuzzy relations is the concept of an alpha level set and its variant, a strong alpha level set. Alpha level set of fuzzy set/fuzzy relation is the crisp set that contains all the elements of universal space whose membership grades in set/relation are greater than or equal to the specified value of alpha. Strong alpha level set of fuzzy set/fuzzy relation is the crisp set that contains all the elements of universal space whose membership grades in set/relation are greater than the specified value of alpha. The set of all alpha levels that represent distinct alpha cuts of a fuzzy set/fuzzy relation is called a level set of set/relation.

```
In[39]:= SetOptions[FuzzySet, UniversalSpace → {0, 20, 1}];

In[40]:= FS1 = FuzzyTrapezoid[1, 8, 12, 17];

In[41]:= FS2 = FuzzyBell[5, 3, 2];

In[42]:= rel1 = SetsToRelation[Min, FS1, FS2];

In[43]:= FuzzySurfacePlot[rel1];
```



```
In[44]:= AlphaLevelSet[rel1, .2]

Out[44]= {{3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {3, 8}, {3, 9}, {4, 1}, {4, 2},
          {4, 3}, {4, 4}, {4, 5}, {4, 6}, {4, 7}, {4, 8}, {4, 9}, {5, 1}, {5, 2}, {5, 3}, {5, 4},
          {5, 5}, {5, 6}, {5, 7}, {5, 8}, {5, 9}, {6, 1}, {6, 2}, {6, 3}, {6, 4}, {6, 5}, {6, 6},
          {6, 7}, {6, 8}, {6, 9}, {7, 1}, {7, 2}, {7, 3}, {7, 4}, {7, 5}, {7, 6}, {7, 7},
          {7, 8}, {7, 9}, {8, 1}, {8, 2}, {8, 3}, {8, 4}, {8, 5}, {8, 6}, {8, 7}, {8, 8},
          {8, 9}, {9, 1}, {9, 2}, {9, 3}, {9, 4}, {9, 5}, {9, 6}, {9, 7}, {9, 8}, {9, 9},
          {10, 1}, {10, 2}, {10, 3}, {10, 4}, {10, 5}, {10, 6}, {10, 7}, {10, 8}, {10, 9},
          {11, 1}, {11, 2}, {11, 3}, {11, 4}, {11, 5}, {11, 6}, {11, 7}, {11, 8}, {11, 9},
          {12, 1}, {12, 2}, {12, 3}, {12, 4}, {12, 5}, {12, 6}, {12, 7}, {12, 8}, {12, 9},
          {13, 1}, {13, 2}, {13, 3}, {13, 4}, {13, 5}, {13, 6}, {13, 7}, {13, 8}, {13, 9},
          {14, 1}, {14, 2}, {14, 3}, {14, 4}, {14, 5}, {14, 6}, {14, 7}, {14, 8}, {14, 9},
          {15, 1}, {15, 2}, {15, 3}, {15, 4}, {15, 5}, {15, 6}, {15, 7}, {15, 8}, {15, 9},
          {16, 1}, {16, 2}, {16, 3}, {16, 4}, {16, 5}, {16, 6}, {16, 7}, {16, 8}, {16, 9}}

In[45]:= StrongAlphaLevelSet[rel1, .2]

Out[45]= {{3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {3, 8}, {3, 9}, {4, 1}, {4, 2},
          {4, 3}, {4, 4}, {4, 5}, {4, 6}, {4, 7}, {4, 8}, {4, 9}, {5, 1}, {5, 2}, {5, 3}, {5, 4},
          {5, 5}, {5, 6}, {5, 7}, {5, 8}, {5, 9}, {6, 1}, {6, 2}, {6, 3}, {6, 4}, {6, 5}, {6, 6},
          {6, 7}, {6, 8}, {6, 9}, {7, 1}, {7, 2}, {7, 3}, {7, 4}, {7, 5}, {7, 6}, {7, 7},
          {7, 8}, {7, 9}, {8, 1}, {8, 2}, {8, 3}, {8, 4}, {8, 5}, {8, 6}, {8, 7}, {8, 8},
          {8, 9}, {9, 1}, {9, 2}, {9, 3}, {9, 4}, {9, 5}, {9, 6}, {9, 7}, {9, 8}, {9, 9},
          {10, 1}, {10, 2}, {10, 3}, {10, 4}, {10, 5}, {10, 6}, {10, 7}, {10, 8}, {10, 9},
          {11, 1}, {11, 2}, {11, 3}, {11, 4}, {11, 5}, {11, 6}, {11, 7}, {11, 8}, {11, 9},
          {12, 1}, {12, 2}, {12, 3}, {12, 4}, {12, 5}, {12, 6}, {12, 7}, {12, 8}, {12, 9},
          {13, 1}, {13, 2}, {13, 3}, {13, 4}, {13, 5}, {13, 6}, {13, 7}, {13, 8}, {13, 9},
          {14, 1}, {14, 2}, {14, 3}, {14, 4}, {14, 5}, {14, 6}, {14, 7}, {14, 8}, {14, 9},
          {15, 1}, {15, 2}, {15, 3}, {15, 4}, {15, 5}, {15, 6}, {15, 7}, {15, 8}, {15, 9}}

In[46]:= LevelSet[rel1]

Out[46]= {0.00159744, 0.00210406, 0.00282801, 0.00389105, 0.00550197,
          0.00803492, 0.0121951, 0.0193919, 0.032635, 0.0588235, 0.114731, 1/7,
          1/5, 0.240356, 2/7, 2/5, 3/7, 0.5, 4/7, 3/5, 5/7, 4/5, 0.835052, 6/7, 0.987805, 1.}
```

8 Fuzzy Relation Equations

The notion of fuzzy relation equations is associated with the concept of the max-min form of composition of binary relations.

Solve for a Fuzzy Relation

In the following example, you will determine a fuzzy relation `Relat1` given the system's input `A` and output `B`.

```
In[47]:= A = FuzzySet[{{1, .2}, {2, .8}, {3, 1}}, UniversalSpace → {1, 3, 1}]

Out[47]= FuzzySet[{{1, 0.2}, {2, 0.8}, {3, 1}}, UniversalSpace → {1, 3, 1}]

In[48]:= B = FuzzySet[{{1, .5}, {2, .8}, {3, .6}}, UniversalSpace → {1, 3, 1}]

Out[48]= FuzzySet[{{1, 0.5}, {2, 0.8}, {3, 0.6}}, UniversalSpace → {1, 3, 1}]

In[49]:= Relat1 = FindFuzzyRelation[A, B]

Out[49]= FuzzyRelation[{{{1, 1}, 1}, {{1, 2}, 1}, {{1, 3}, 1}, {{2, 1}, 0.5},
           {{2, 2}, 1}, {{2, 3}, 0.6}, {{3, 1}, 0.5}, {{3, 2}, 0.8}, {{3, 3}, 0.6}},
          UniversalSpace → {{1, 3, 1}, {1, 3, 1}}]
```

Solve for a Fuzzy Set

You can view fuzzy relation as a fuzzy system. Then given its output you can determine the input. In the following example, apply `FindFuzzySet` function to determine the input `NewA` for fuzzy relation `Rel2` and output `B`.

```
In[50]:= Rel2 =
          FromMembershipMatrix[{{.7, 1, .4}, {.5, .9, .6}, {.2, .6, .3}}, {{1, 3}, {1, 3}}]

Out[50]= FuzzyRelation[{{{1, 1}, 0.7}, {{1, 2}, 1}, {{1, 3}, 0.4}, {{2, 1}, 0.5},
           {{2, 2}, 0.9}, {{2, 3}, 0.6}, {{3, 1}, 0.2}, {{3, 2}, 0.6}, {{3, 3}, 0.3}},
          UniversalSpace → {{1, 3, 1}, {1, 3, 1}}]

In[51]:= NewA = FindFuzzySet[Rel2, B]

Out[51]= FuzzySet[{{1, 0.5}, {2, 0.8}, {3, 1}}, UniversalSpace → {1, 3, 1}]
```

9 Random Fuzzy Sets and Fuzzy Relations

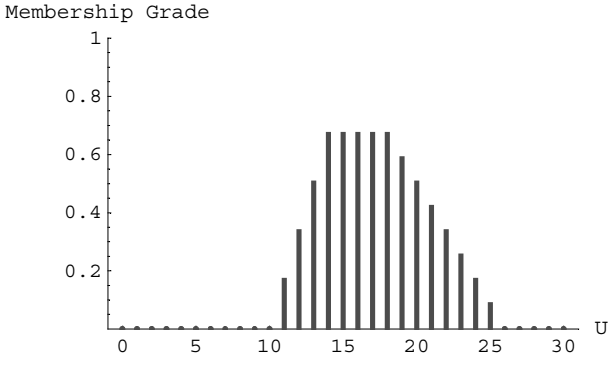
These functions are used to create a random fuzzy set or fuzzy relation. This type of operation might be valuable to test new functions.

Random Fuzzy Set

`RandomFuzzySet[{a, b}]` creates a random fuzzy set with universal space from *a* to *b*. By default, this function will create a random trapezoidal fuzzy set. The option `Type` can be used to create a `Gaussian`, `Triangular`, or `Complete` random fuzzy set. In addition, there is an option called `Normal` that produces a normal random fuzzy set, when set to `True`.

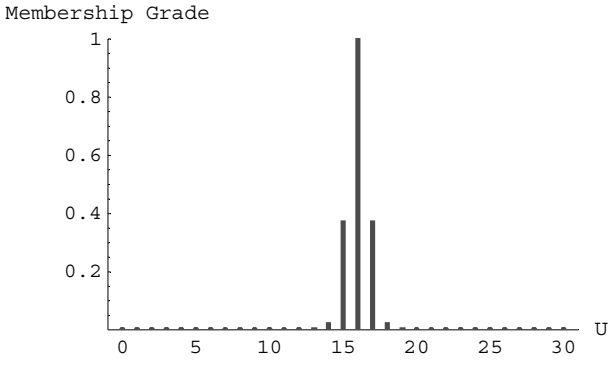
Just as with normal random numbers you can seed the random number generator to produce the same random fuzzy numbers each time. To test the `RandomFuzzySet` function, try reevaluating the function a number of times with different parameters.

```
In[52]:= FS1 = RandomFuzzySet[{0, 30}];

In[53]:= FuzzyPlot[FS1];
```



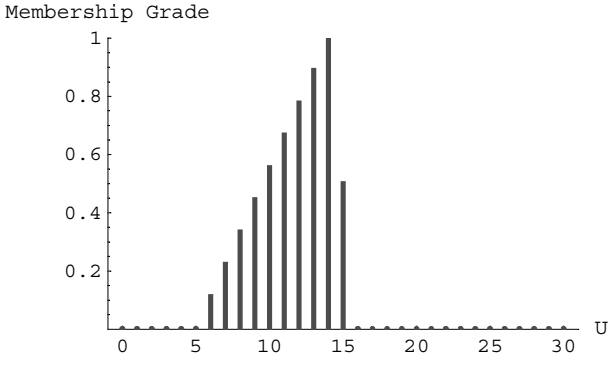

```
In[54]:= FS2 = RandomFuzzySet[{0, 30}, Type → Gaussian];
```

In[55]:= **FuzzyPlot[FS2];**



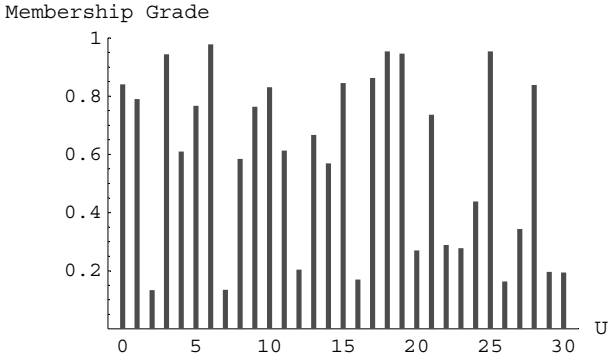
In[56]:= **FS3 = RandomFuzzySet[{0, 30}, Type → Triangular, Normal → True];**

In[57]:= **FuzzyPlot[FS3];**



In[58]:= **FS3 = RandomFuzzySet[{0, 30}, Type → Complete];**

```
In[59]:= FuzzyPlot[FS3];
```

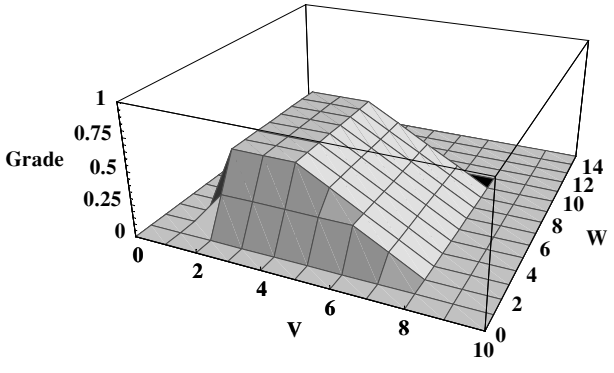


Random Fuzzy Relation

`RandomFuzzySet [{{a1 , b1}, {a2, b2}}]` creates a random fuzzy relation with universal space of *{{a1, b1}, {a2, b2}}*. By default, this function will create a random trapezoidal fuzzy relation. The option `Type` can be used to create a `Complete` random fuzzy relation. In addition, there is an option called `Normal` that produces a normal random fuzzy relation, when set to `True`.

```
In[60]:= fr1 = RandomFuzzyRelation[{{0, 10}, {0, 15}}];
```

```
In[61]:= FuzzySurfacePlot[fr1];
```



10 Fuzzy Inferencing

Rule-based Inference

`RuleBasedInference[{{A1, ... ,An}, ... , {S1, ... ,Sp}}, {Y1, ... ,Yk}, {Ax, ... , Sx, Yx}, {a, ... ,s}, opts]` returns a fuzzy set that is the result of performing rule based inference for multiple-input/single-output systems where *{{A1, ... , An}, ... , {S1, ... , Sp}}* represent linguistic input variables, *{Y1, ... ,Yk}* is the linguistic output variable, rules are given in a list like*{Ax, ... , Sx, Yx}*, and the crisp values for the inputs are given in a list *{a, ... ,s}*. The values for option `Type` are `Mamdani`, `Model`, and `Scaled`.

For an example, see the 1_08_FuzzyInference.nb notebook.

11 Fuzzy Arithmetic

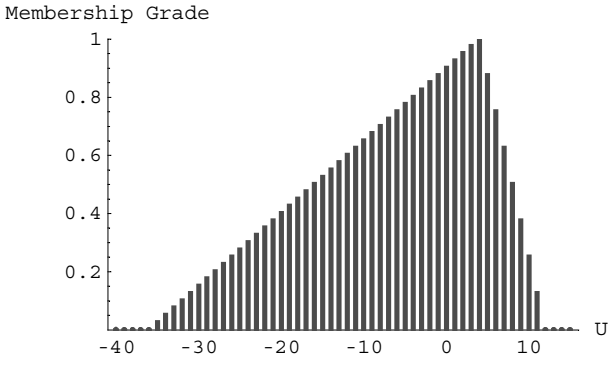
Fuzzy Multiplication and Division

`FuzzyMultiply[{a1, b1, c1, d1}, {a2, b2, c2, d2}]` returns the product of the fuzzy numbers represented by the two lists. The fuzzy product is returned as an unevaluated `FuzzyTrapezoid`. To evaluate the `FuzzyTrapezoid`, use the `ReleaseHold` function.

```
In[62]:= SetOptions[FuzzySet, UniversalSpace → {-40, 15, 1}];

In[63]:= Multi1 = FuzzyMultiply[{-9, 2, 2, 3}, {1, 2, 2, 4}, UniversalSpace -> {-40, 15, 1}]

Out[63]= FuzzyTrapezoid[-36, 4, 4, 12, UniversalSpace → {-40, 15, 1}]
```

In the following, you can plot an approximate result of the fuzzy product. Notice the use of the `ReleaseHold` command.

```
In[64]:= FuzzyPlot[ReleaseHold[Multi1]];
```

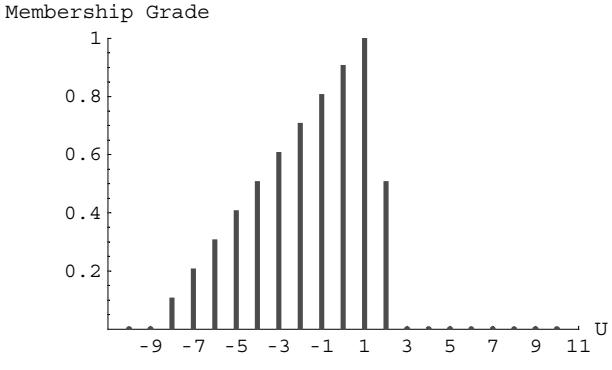


`FuzzyDivide[`*{a1, b1, c1, d1}*`,` *{a2, b2, c2, d2}*`]` returns the division of the fuzzy numbers represented by the two lists. The fuzzy division is returned as an unevaluated `FuzzyTrapezoid`. To evaluate the `FuzzyTrapezoid`, use the `ReleaseHold` function.

```
In[65]:= Div1 = FuzzyDivide[{-9, 2, 2, 3}, {1, 2, 2, 4}, UniversalSpace -> {-10, 10}]

Out[65]= FuzzyTrapezoid[-9, 1, 1, 3, UniversalSpace → {-10, 10, 1}]
```

You can plot the approximate result of the fuzzy division. Notice the use of the `ReleaseHold` command.

```
In[66]:= FuzzyPlot[ReleaseHold[Div1]];
```



12 Fuzzy Clustering

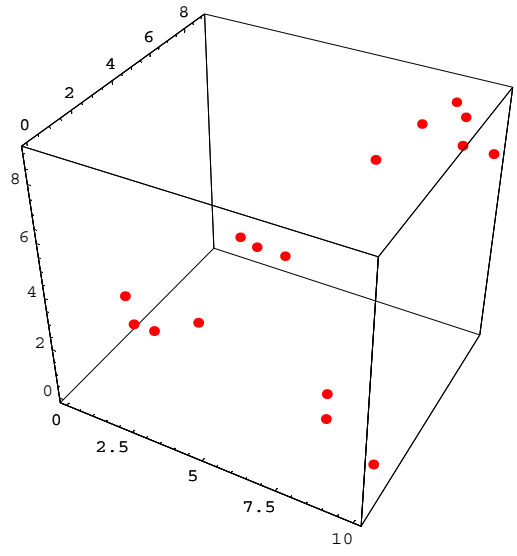
Fuzzy C-Means Clustering

`FCMCluster[`*data*`,` *partmat*`,` *mu*`,` *epsilon*`]` returns a list of cluster centers, a partition matrix indicating the degree to which each data point belongs to a particular cluster center, and a list containing the progression of cluster centers found during the run. The arguments to the function are the data set (*data*), an initial partition matrix (*partmat*), a value determining the degree of fuzziness of the clustering (*mu*), and a value which determines when the algorithm will terminate (*epsilon*). This function runs recursively until the terminating criteria is met. While running, the function prints a value which indicates the accuracy of the fuzzy clustering. When this value is less than the parameter *epsilon*, the function terminates. The parameter *mu* is called the exponential weight and controls the degree of fuzziness of the clusters. As *mu* approaches 1, the fuzzy clusters become crisp clusters, where each data point belongs to only one cluster. As *mu* approaches infinity, the clusters become completely fuzzy, and each point will belong to each cluster to the same degree (1/c) regardless of the data. Studies have been done on selecting the value for *mu*, and it appears that the best choice for *mu* is usually in the interval [1.5, 2.5], where the midpoint, *mu* = 2, is probably the most commonly used value for *mu*.

To demonstrate the FCM clustering algorithm, you can create a data set which consists of four group of data.

```
In[67]:= TrainData3D = {{1, 2, 3}, {2, 2, 2}, {2, 1, 3}, {3, 3, 2},
           {4, 5, 4}, {4, 4, 5}, {5, 5, 4}, {7, 7, 7}, {9, 9, 7}, {8, 8, 8},
           {9, 8, 9}, {10, 9, 7}, {9, 9, 8}, {10, 1, 1}, {8, 2, 1}, {8, 2, 2}};
```

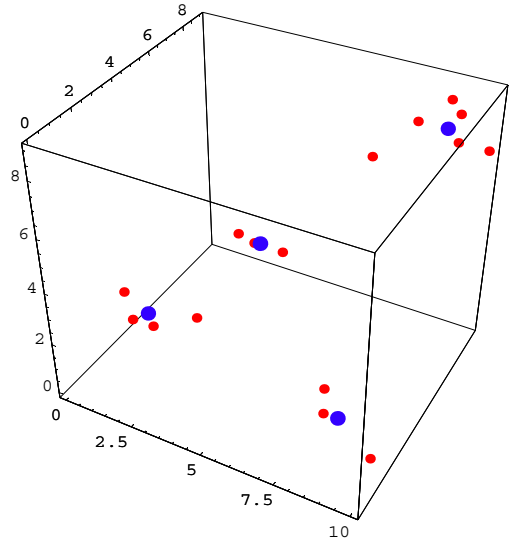
In[68]:=
```
g1 = Show[Graphics3D[
     Map[{Hue[0], PointSize[.02], Point[#]} &, TrainData3D]], Axes -> True];
```



```
In[69]:= Res1a = FCMCluster[TrainData3D, InitializeU[TrainData3D, 4], 2, .01]

           0.859596

           0.390469

           0.580971

           0.33459

           0.159146

           0.0271684

           0.00819179

Out[69]= {{{8.80257, 8.45625, 7.72496}, {8.60842, 1.70315, 1.34927},
           {1.91922, 1.9161, 2.53901}, {4.36615, 4.70241, 4.31657}},
          {{0.00790454, 0.00245483, 0.00787974, 0.0200401, 0.00638034, 0.0193489,
            0.0138087, 0.675572, 0.965732, 0.957304, 0.936261, 0.924364, 0.9852,
            0.0225436, 0.00637423, 0.010978}, {0.0162631, 0.00670802, 0.0208705,
            0.057455, 0.0079704, 0.0244519, 0.0179912, 0.0636494, 0.00973579,
            0.0105433, 0.0178313, 0.0239293, 0.00414031, 0.90297, 0.961731, 0.935038},
           {0.927363, 0.974613, 0.924737, 0.732182, 0.0195252, 0.0661418, 0.0263162,
            0.0556566, 0.00691018, 0.00855686, 0.0136039, 0.0153974, 0.00310599,
            0.033649, 0.0141819, 0.0221196}, {0.0484691, 0.0162238, 0.0465132,
            0.190323, 0.966124, 0.890057, 0.941884, 0.205122, 0.0176218, 0.0235959,
            0.0323038, 0.036309, 0.00755412, 0.0408375, 0.0177126, 0.0318645}},
          {{{7.15306, 6.30536, 5.84949}, {7.93877, 4.64636, 4.82262},
            {5.69922, 4.36545, 3.86212}, {5.30485, 4.48257, 4.57858}},
           {{8.0429, 7.54342, 7.07821}, {7.76901, 4.66307, 4.28611},
            {4.77792, 3.29263, 3.13574}, {4.39454, 3.83761, 3.77279}},
           {{8.5597, 8.22325, 7.613}, {7.92331, 2.62504, 2.34509},
            {4.02478, 2.42384, 2.4866}, {3.92328, 3.5844, 3.54958}},
           {{8.73341, 8.3974, 7.70331}, {8.32764, 1.83272, 1.51549},
            {2.50898, 2.29497, 2.5358}, {3.76241, 3.9794, 3.92443}},
           {{8.77124, 8.42984, 7.71421}, {8.53991, 1.73997, 1.38123},
            {2.05767, 2.03581, 2.48377}, {4.23674, 4.57599, 4.31832}},
           {{8.79512, 8.45028, 7.72315}, {8.59628, 1.70953, 1.35416},
            {1.94406, 1.93933, 2.5207}, {4.34704, 4.68528, 4.32733}},
           {{8.80257, 8.45625, 7.72496}, {8.60842, 1.70315, 1.34927},
            {1.91922, 1.9161, 2.53901}, {4.36615, 4.70241, 4.31657}}}}
```

The last line shows coordinates for four centers. The clustering function also provides the degrees to which each data point belongs to each cluster and the cluster center progression. Because there are problems showing the cluster centers for data of higher dimensions, the `ShowCenters` function only works for 2D clustering. If you want to see the 3D cluster centers with the original data, you could do the following:

```
In[70]:= g2 = Show[g1, Graphics3D[Map[{Hue[0.7], PointSize[.03], Point[#]} &, Res1a[[1]]]]];
```



The clustering function should work for data of any dimension, but it is hard to visualize the results for higher order data.

Part 1
Manual

1 Creating Fuzzy Sets

1.1 Introduction

Fuzzy Logic package provides a number of convenient ways to create fuzzy sets. This chapter demonstrates these functions and the options associated with each function.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

1.2 Basic Objects

`FuzzySet`	object representing a fuzzy set

Fuzzy set object.

FuzzySet is one of the fundamental objects used in the package. Here is its full form:

`FuzzySet[{{u1, X(u1)}, ... ,{un, X(un)}}, UniversalSpace→{a, b, c}]`

where *u1*, ... , *un* are elements of the universal space and *X(u1)*, ... , *X(un)* are the grades of membership of the elements in a fuzzy set.

option name	*default value*	
`UniversalSpace`	`{0, 20, 1}`	universal space for `FuzzySet` object

Option for `FuzzySet`.

```
In[2]:= SetOptions[FuzzySet, UniversalSpace → {0, 20, 1}]

Out[2]= {UniversalSpace → {0, 20, 1}}
```

By default the universal space is set to 0, 1, 2, ..., 20. When creating fuzzy sets, if an explicit universal space is not given, this default value will be used. Universal spaces for fuzzy sets and fuzzy relations are defined with three numbers. The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between discrete elements.

A built-in *Mathematica* function gives the list of default options assigned to a symbol.

```
In[3]:= Options[FuzzySet]

Out[3]= {UniversalSpace → {0, 20, 1}}
```

You can change the default value for `UniversalSpace` globally.

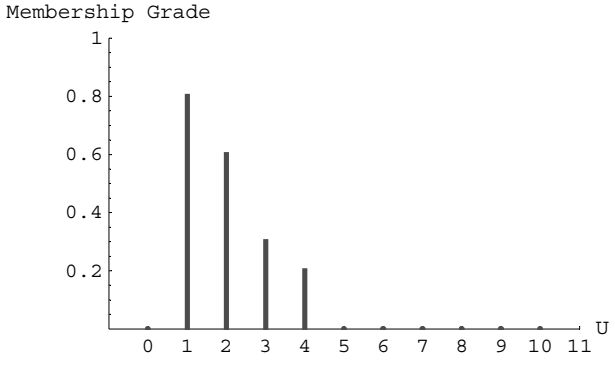
```
In[4]:= SetOptions[FuzzySet, UniversalSpace → {0, 100, 0.5}]

Out[4]= {UniversalSpace → {0, 100, 0.5}}
```

Fuzzy sets can be created manually by entering a list of element and membership grade pairs surrounded by the `FuzzySet` head. Here is a simple example.

```
In[5]:= FS1 = FuzzySet[{{1, .8}, {2, .6}, {3, .3}, {4, .2}}, UniversalSpace → {0, 10, 1}]

Out[5]= FuzzySet[{{1, 0.8}, {2, 0.6}, {3, 0.3}, {4, 0.2}}, UniversalSpace → {0, 10, 1}]
```

When studying fuzzy sets, it is often instructive to look at fuzzy sets graphically. A description of the graphing functions and how to use them appears later in Chapter 5 Fuzzy Set Visualization, but you can use one of the graphing functions here to look at your newly created fuzzy set `FS1`.

```
In[6]:= FuzzyPlot[FS1];
```



1.3 Functions for Creating Fuzzy Sets

This way of creating fuzzy set objects might be quite tedious. *Fuzzy Logic* package provides a number of functions to create special types of fuzzy sets.

`FuzzyTrapezoid[`*a*`,`*b*`, `*c*`, `*d*`,`*h*`]`	return a fuzzy set whose membership grades represent a trapezoid defined by parameters *a*, *b*, *c*, *d*, and *h*
`FuzzyGaussian[`*mu*`, `*sigma*`]`	return a fuzzy set whose membership grades represent a normalized Gaussian function with a mean of *mu* and a width of *sigma*
`FuzzyBell[`*c*`, `*w*`, `*s*`]`	return a bell−shaped fuzzy set centered at *c* with crossover points at $c \pm w$ with a slope of $s / (2\,w)$ at the crossover points
`FuzzyTwoGaussian[`*mu1*`, `*sigma1*`, `*mu2*`, `*sigma2*`]`	return a two−sided Gaussian fuzzy set with centers at *mu1* and *mu2* and widths of *sigma1* and *sigma2*; between the two mean, the fuzzy set has a membership grade of 1
`FuzzySigmoid[`*c*`, `*s*`]`	return a sigmoidal fuzzy set where *s* controls the slope at the crossover point *c*
`CreateFuzzySet[`*func*`, {`*a*`, `*b*`, `*c*`}]`	return a fuzzy set with membership grades defined by*func* in the range *a* to *b* inclusive, and increment *c*
`CreateFuzzySets[`*num*`]`	return a list of *num* overlapping fuzzy sets that span the universal space. The option Type specifies either triangular or Gaussian fuzzy sets
`RandomFuzzySet[{`*a*`, `*b*`}]`	create a random fuzzy set with universal space from *a* to *b*

Functions for creating fuzzy sets.

We will now show a number of examples on how to create fuzzy sets.

`FuzzyTrapezoid[`*a*`, `*b*`, `*c*`, `*d*`, `*h*`, `*opts*`]` returns a fuzzy set with membership grades that linearly increase from 0 to *h* in the range *a* to *b*, are equal to *h* in the range *b* to *c*, and linearly decrease from *h* to 0 in the range *c* to *d*. Arguments *a*, *b*, *c*, and *d* must be in increasing order, and *h* must be a value between 0 and 1, inclusive. If a value for *h* is not specified, the function will use $h = 1$. The universal space may be explicitly defined using the `UniversalSpace` option, or if a universal space is not specified, the default universal space is used.

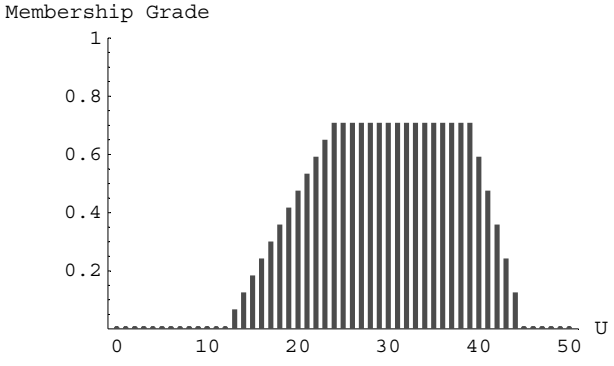
This example shows how to create a trapezoidal fuzzy set with height 0.7. Note that universal space for this fuzzy set is explicitly set.

```
In[7]:= FS2 = FuzzyTrapezoid[12, 24, 39, 45, 0.7, UniversalSpace → {0, 50, 1}]

Out[7]= FuzzySet[
         {{13, 0.0583333}, {14, 0.116667}, {15, 0.175}, {16, 0.233333}, {17, 0.291667},
          {18, 0.35}, {19, 0.408333}, {20, 0.466667}, {21, 0.525}, {22, 0.583333},
          {23, 0.641667}, {24, 0.7}, {25, 0.7}, {26, 0.7}, {27, 0.7}, {28, 0.7},
          {29, 0.7}, {30, 0.7}, {31, 0.7}, {32, 0.7}, {33, 0.7}, {34, 0.7}, {35, 0.7},
          {36, 0.7}, {37, 0.7}, {38, 0.7}, {39, 0.7}, {40, 0.583333}, {41, 0.466667},
          {42, 0.35}, {43, 0.233333}, {44, 0.116667}}, UniversalSpace → {0, 50, 1}]
```

After evaluating this command, we receive a fuzzy set containing a list of the element and membership grade pairs followed by a universal space.

Here we use the function `FuzzyPlot` to look at our newly created fuzzy set.

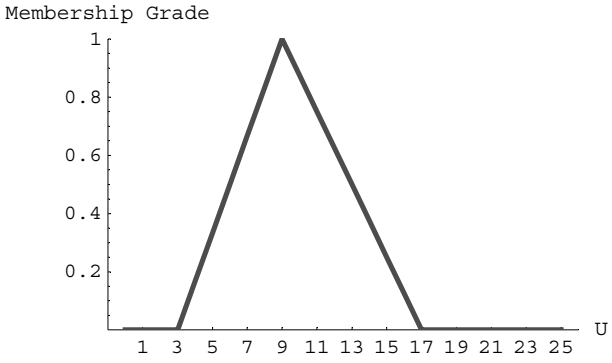
```
In[8]:= FuzzyPlot[FS2];
```



The following is another example of creating a trapezoidal fuzzy set. This time we give the vertices of the trapezoid and allow the function to provide a default setting for the height. Another thing to note about this example is that the second and third parameters for specifying the trapezoid are the same. This technique is used to create triangular fuzzy sets.

```
In[9]:= FS3 = FuzzyTrapezoid[3, 9, 9, 17, UniversalSpace -> {0, 25}]

Out[9]= FuzzySet[{{4, 1/6}, {5, 1/3}, {6, 1/2}, {7, 2/3}, {8, 5/6}, {9, 1}, {10, 7/8}, {11, 3/4},
          {12, 5/8}, {13, 1/2}, {14, 3/8}, {15, 1/4}, {16, 1/8}}, UniversalSpace → {0, 25, 1}]
```

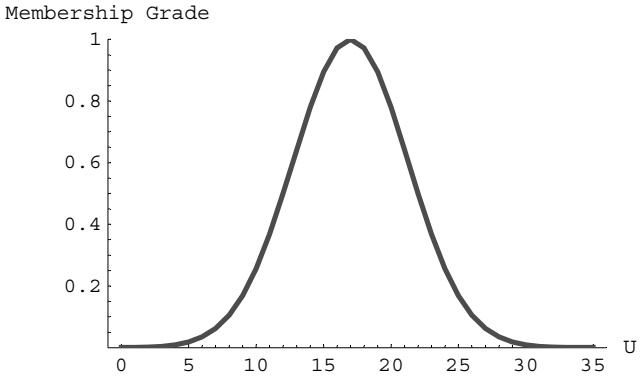
```
In[10]:= FuzzyPlot[FS3, PlotJoined → True];
```



`FuzzyGaussian[`*mu*`,` *sigma*`,` *opts*`]` returns a new fuzzy set whose membership grades represent a normalized Gaussian function with a mean of *mu* and a width of *sigma*. In this example, we create a Gaussian fuzzy set centered at 17 with a width of 6. Note that we chose to specify our own universal space instead of accepting the default setting.

```
In[11]:= FS4 = FuzzyGaussian[17, 6, UniversalSpace → {0, 35, 1}]

Out[11]= FuzzySet[{{0, 0.000326272}, {1, 0.000815988}, {2, 0.00193045},
           {3, 0.00432024}, {4, 0.00914595}, {5, 0.0183156}, {6, 0.0346967},
           {7, 0.0621765}, {8, 0.105399}, {9, 0.169013}, {10, 0.256376}, {11, 0.367879},
           {12, 0.499352}, {13, 0.64118}, {14, 0.778801}, {15, 0.894839}, {16, 0.972604},
           {17, 1.}, {18, 0.972604}, {19, 0.894839}, {20, 0.778801}, {21, 0.64118},
           {22, 0.499352}, {23, 0.367879}, {24, 0.256376}, {25, 0.169013},
           {26, 0.105399}, {27, 0.0621765}, {28, 0.0346967}, {29, 0.0183156},
           {30, 0.00914595}, {31, 0.00432024}, {32, 0.00193045}, {33, 0.000815988},
           {34, 0.000326272}, {35, 0.00012341}}, UniversalSpace → {0, 35, 1}]
```

```
In[12]:= FuzzyPlot[FS4, PlotJoined → True];
```

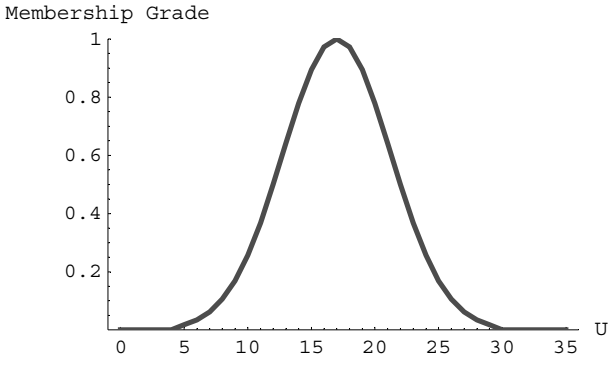


Note that in the example, the membership grades for some of the elements are very small. The option `ChopValue` works with the `FuzzyGaussian`, `FuzzyTwoGaussian`, `FuzzyBell`, and `FuzzySigmoid` functions. It allows you to specify a value for which all membership grades less than that value are set to zero. We demonstrate this option in the following example.

```
In[13]:= FS5 = FuzzyGaussian[17, 6, ChopValue → .01, UniversalSpace → {0, 35, 1}]

Out[13]= FuzzySet[{{5, 0.0183156}, {6, 0.0346967}, {7, 0.0621765}, {8, 0.105399},
           {9, 0.169013}, {10, 0.256376}, {11, 0.367879}, {12, 0.499352}, {13, 0.64118},
           {14, 0.778801}, {15, 0.894839}, {16, 0.972604}, {17, 1.}, {18, 0.972604},
           {19, 0.894839}, {20, 0.778801}, {21, 0.64118}, {22, 0.499352}, {23, 0.367879},
           {24, 0.256376}, {25, 0.169013}, {26, 0.105399}, {27, 0.0621765},
           {28, 0.0346967}, {29, 0.0183156}}, UniversalSpace → {0, 35, 1}]
```

Notice that all the membership grades that were less than 0.01 from the previous example are now set to zero.

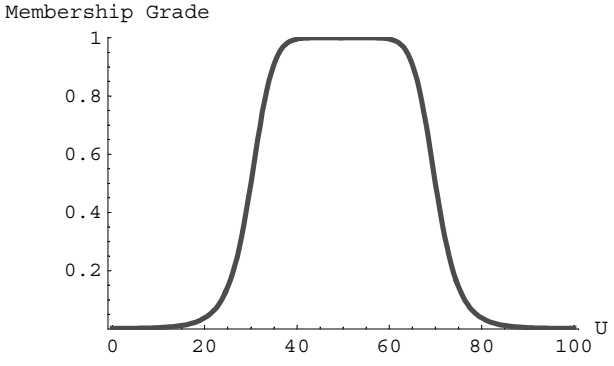
```
In[14]:= FuzzyPlot[FS5, PlotJoined → True];
```



`FuzzyBell[`*c*`,` *w*`,` *s*`,` *opts*`]` returns a bell-shaped fuzzy set centered at *c* with crossover points at $c \pm w$ with a slope $s / (2\ w)$ at the crossover points. Here is an example of creating a bell-shaped fuzzy set centered at 50 with crossover points at 50 ± 20 and slope of 4 / (2 * 20) at the crossover points.

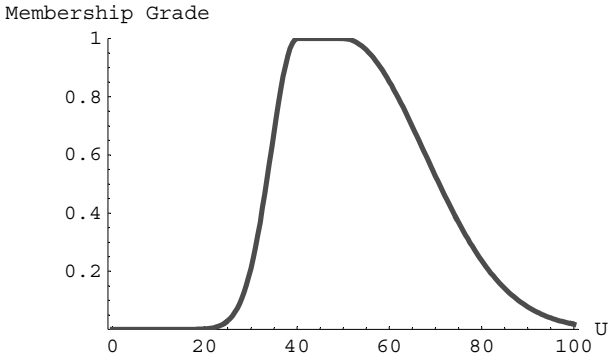
```
In[15]:= FS6 = FuzzyBell[50, 20, 4];
```

```
In[16]:= FuzzyPlot[FS6, PlotJoined → True];
```



The function `FuzzyTwoGaussian[`*mu1*`,` *sigma1*`,` *mu2*`,` *sigma2*`,` *opts*`]` creates a two-sided Gaussian fuzzy set with centers at *mu1* and *mu2*, and widths of *sigma1* and *sigma2*; between the two mean, the fuzzy set has a membership grade of 1. Here we create a two-sided Gaussian fuzzy set with centers at 40 and 50 and widths of 8 and 25, respectively. Between the two mean, the fuzzy set has a membership grade of 1.

```
In[17]:= FS7 = FuzzyTwoGaussian[40, 8, 50, 25];

In[18]:= FuzzyPlot[FS7, PlotJoined → True];
```

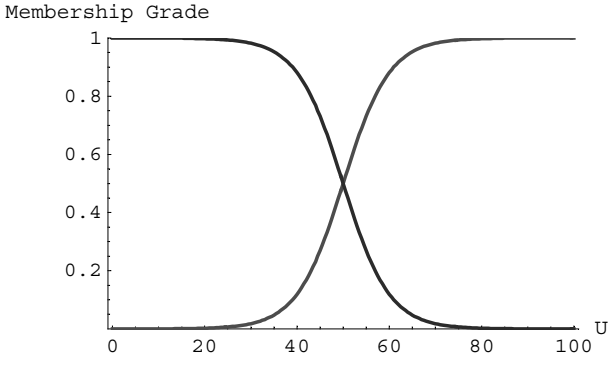


`FuzzySigmoid[c, s, opts]` creates a sigmoidal fuzzy set where *s* controls the slope at the crossover point *c*. Here we create a sigmoidal fuzzy set with the slope at the crossover point `50`. Because the slope of 0.2 is positive, the fuzzy set opens to the right.

```
In[19]:= FS8 = FuzzySigmoid[50, 0.2];
```

Here we create a sigmoidal fuzzy set with the slope at the crossover point 50. Because now the slope of -0.2 is negative, the fuzzy set opens to the left.

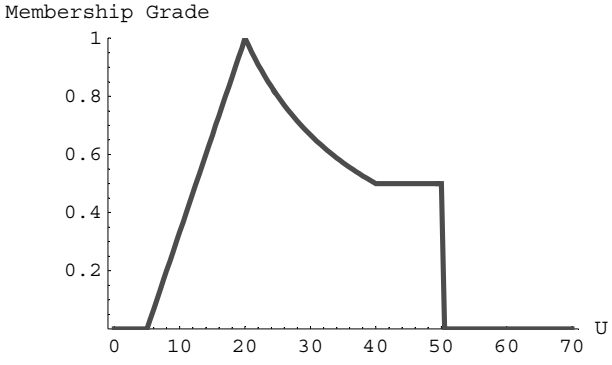
```
In[20]:= FS9 = FuzzySigmoid[50, -0.2];

In[21]:= FuzzyPlot[FS8, FS9, PlotJoined → True];
```



We can create a fuzzy set by specifying a membership function. The function `CreateFuzzySet[func, {a, b, c}]` returns a fuzzy set with membership grades defined by *func* in the range *a* to *b*, inclusive. If no range is given, the function is applied to all of the elements in the universal space. Note that the function must

return values between 0 and 1 over the range provided to create a valid fuzzy set. This is an example of creating a fuzzy set using a piecewise function. The first definition defines the function to use to create the fuzzy set.

```
In[22]:= myfunction[x_] := Which[x ≤ 5, 0, 5 < x < 20, (x - 5)/15, 20 ≤ x ≤ 40, 20/x, x > 40, 0.5]

In[23]:= FS10 = CreateFuzzySet[myfunction, {0, 50, .5}, UniversalSpace -> {0, 70, .5}];

In[24]:= FuzzyPlot[FS10, PlotJoined → True];
```



Sometimes we need to create a collection of `FuzzySet` objects that span a given universal space. `CreateFuzzySets[`*num*`,` *opts*`]` returns a list of *num* overlapping fuzzy sets that span the universal space. The fuzzy sets can be either `Triangular` or `Gaussian`. The `Type` option specifies the shape of the new fuzzy sets, and `Triangular` membership functions are the default setting.

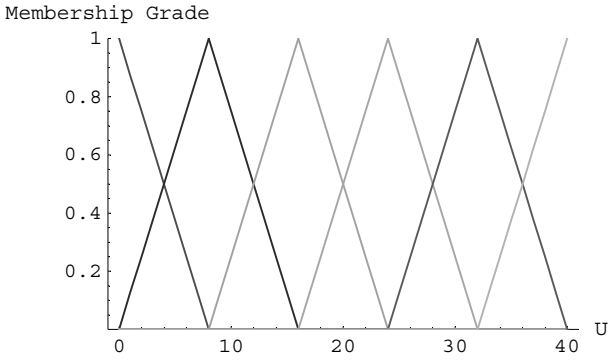
```
In[25]:= Options[CreateFuzzySets]

Out[25]= {Type → Triangular}
```

option name	*default value*	
`Type`	`Triangular`	shape of fuzzy set collection

Option for `CreateFuzzySets`.

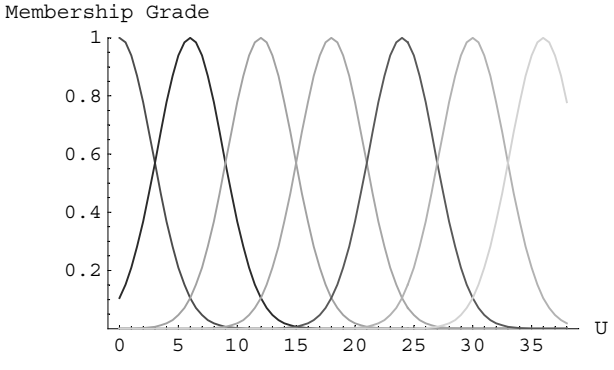
In this example, we divide the universal space into six triangular fuzzy sets. We use an extra option with the `FuzzyPlot` command to show a continuous representation of the fuzzy sets.

```
In[26]:= Var1 = CreateFuzzySets[6, UniversalSpace → {0, 40, 1}];

In[27]:= FuzzyPlot[Var1, PlotJoined → True];
```



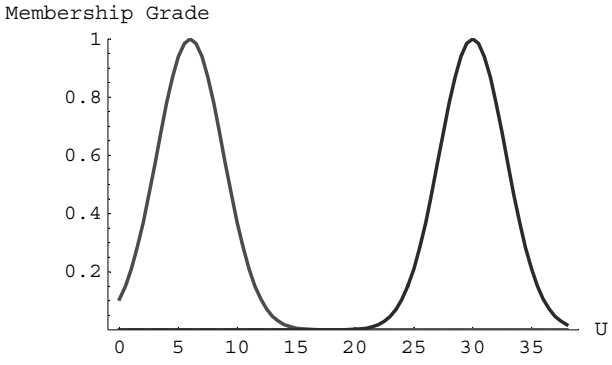
As a final example of fuzzy set creation, we create a set of seven `Gaussian` fuzzy sets. When specifying the `Gaussian` type of fuzzy set, we should provide a width with the `Type` option. If no width is specified, a width of 1 is used. The option `ChopValue` works with the `CreateFuzzySets` function for specified width. It allows you to specify a value for which all membership grades less than that value are set to zero.

```
In[28]:= Var2 = {NB, NM, NS, ZO, PS, PM, PB} = CreateFuzzySets[7,
           Type → Gaussian[4, ChopValue -> 0.001], UniversalSpace → {0, 38, 0.5}];

In[29]:= FuzzyPlot[Var2, PlotJoined → True];
```



Note in the previous example, we assigned variable names to each of the new fuzzy sets. This enables us to reference each individual fuzzy set. This technique is important for the fuzzy inferencing commands that we will talk about later in Chapter 8 Fuzzy Inferencing. To show that we indeed can access the individual fuzzy sets, we graph a couple of the fuzzy sets here.

```
In[30]:= FuzzyPlot[NM, PM, PlotJoined → True];
```



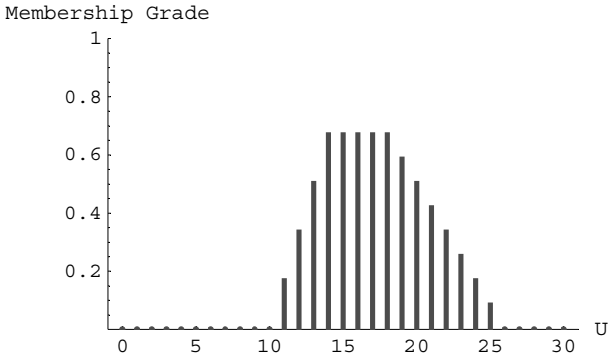
`RandomFuzzySet[{a, b}]` creates a random fuzzy set with universal space from *a* to *b*. By default, this function will create a random trapezoidal fuzzy set. The option `Type` can be used to create a `Gaussian`, `Triangular`, or `Complete` random fuzzy set. In addition, there is an option called `Normal`, which produces a normal `Triangular` random fuzzy set, when set to `True`.

option name	*default value*	
`Type`	`Triangular`	shape of random fuzzy set; admissible values are `Gaussian`, `Triangular`, or `Complete`
`Normal`	`True`	an option that controls whether or not a random set is a normal one

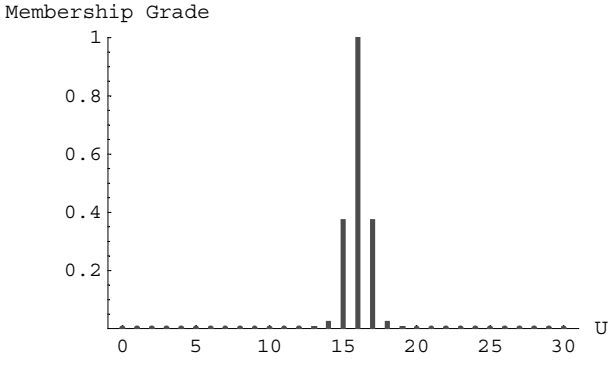
Options for `RandomFuzzySet`.

Just as with normal random numbers we can seed the random number generator to produce the same random fuzzy numbers each time. To test the `RandomFuzzySet` function, try reevaluating the function a number of times with different parameters.

```
In[31]:= RS1 = RandomFuzzySet[{0, 30}];

In[32]:= FuzzyPlot[RS1];
```

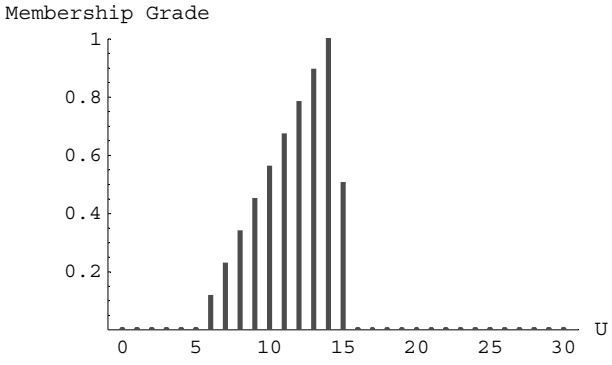



```
In[33]:= RS2 = RandomFuzzySet[{0, 30}, Type → Gaussian];

In[34]:= FuzzyPlot[RS2];
```



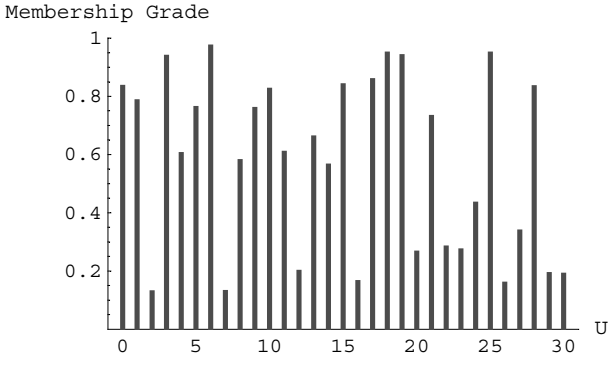

```
In[35]:= RS3 = RandomFuzzySet[{0, 30}, Type → Triangular, Normal → True];
```

```
In[36]:= FuzzyPlot[RS3];
```




```
In[37]:= RS4 = RandomFuzzySet[{0, 30}, Type → Complete];
```

```
In[38]:= FuzzyPlot[RS4];
```



You can use a built-in function `SetOptions` to restore the default setting for the `FuzzySet` object.

```
In[39]:= SetOptions[FuzzySet, UniversalSpace → {0, 20, 1}]

Out[39] {UniversalSpace → {0, 20, 1}}
```

2 Creating Fuzzy Relations

2.1 Introduction

Fuzzy Logic package provides a number of convenient ways to create fuzzy relations. In this chapter, we will demonstrate these functions and the options associated with each function.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

2.2 Basic Objects

`FuzzyRelation`	an object representing a fuzzy relation

Fuzzy relation object.

`FuzzyRelation` is one of the fundamental objects used in the package. Here is its full form:

`FuzzyRelation[`*{{(v1, }, R11}, ... , {{vn, wm}, Rnm}}*,
`UniversalSpace→`*{{a1, b1, c1}, {a2, b2, c2}}*`]`

where *(v1, w1)*, *(v2, w2)*, ... , *(vn, wm)* are elements of the universal space and *R11*, ... , *Rnm* are the values of the function fuzzy relation for these elements. The values are the grade of membership of the elements in a fuzzy relation.

option name	*default value*	
`UniversalSpace`	`{{0,10,1}, {0,10,1}}`	universal space for `FuzzyRelation` object

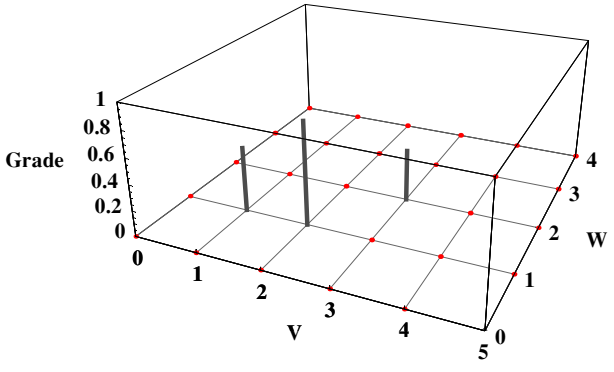
Option for `FuzzyRelation`.

By default the universal space is set to {{0, 10, 1}, {0, 10, 1}}. When creating fuzzy relations, if an explicit universal space is not given, this default value will be used. Universal spaces for fuzzy sets and fuzzy relations are defined with three numbers. The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between discrete elements .

Like fuzzy sets, fuzzy relations can also be created manually by surrounding the appropriate descriptors with the `FuzzyRelation` head. Here we create a simple fuzzy relation.

```
In[2]:= Rel1 = FuzzyRelation[{{{1, 1}, 0.5}, {{2, 1}, 0.8}, {{3, 2}, 0.4}},
          UniversalSpace -> {{0, 5, 1}, {0, 4, 1}}]

Out[2]= FuzzyRelation[{{{1, 1}, 0.5}, {{2, 1}, 0.8}, {{3, 2}, 0.4}},
         UniversalSpace → {{0, 5, 1}, {0, 4, 1}}]
```

```
In[3]:= FuzzyPlot3D[Rel1];
```



2.3 Functions for Creating Fuzzy Relations

Creating fuzzy relations using the basic definition can be quite tedious. *Fuzzy Logic* package provides functions for the most commonly used in practice techniques of creating fuzzy relations.

`FuzzyTrapezoid[{`*ax, bx, cx, dx*`}, {`*ay*`, `*by*`, `*cy*`, `*dy*`}, `*h*`]`	return a trapezoidal fuzzy relation with membership grades that linearly increase from 0 to *h* in both the x and y directions from *a* to *b*, equal *h* from *b* to *c*, and linearly decrease from *h* to 0 from *c* to *d*
`SetsToRelation[`*func*`, `*A*`, `*B*`]`	return a fuzzy relation with elements from the Cartesian product of the universal spaces of fuzzy sets *A* and *B* and membership grades defined by func
`FromMembershipMatrix[`*mat*`, {{`*va*`, `*vb*`}, {`*wa*`, `*wb*`}}]`	return a fuzzy relation by combining the matrix of membership grades, *mat*, with the elements specified by the ranges {*va*,*vb*} and {*wa*,*wb*}
`CreateFuzzyRelation[`*func*`, {{`*va, vb, c*`}, {`*wa, wb, d*`}}]`	return a fuzzy relation with membership grades that are the result of applying*func* to all the elements in the specified range, {{va, *vb*, *c*}, {*wa*, *wb*, *d*}}
`RandomFuzzyRelation[{{`*a1, b1*`}, {`*a2, b2*`}}]`	create a random fuzzy relation with universal space of {{*a1, b1*}, {*a2, b2*}} and increments *c1 = c2* =1

Functions for creating fuzzy relations.

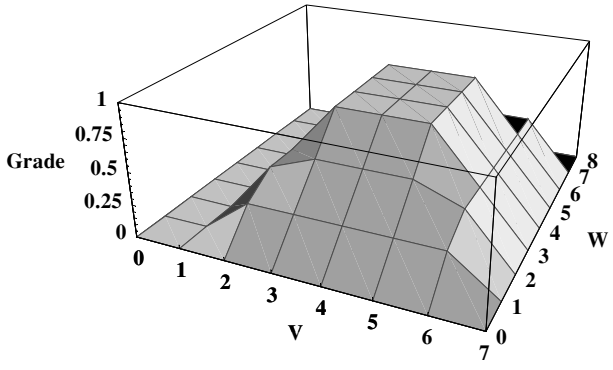
Here are number of examples on how to create fuzzy relations.

`FuzzyTrapezoid[`*{ax, bx, cx, dx}*`,`*{ay, by, cy, dy}*`,`*h*`,`*opts*`]` returns a new trapezoidal fuzzy relation with membership grades that linearly increase from 0 to *h* in both the x and y directions from *a* to *b*, equal *h* from *b* to *c*, and linearly decrease from *h* to 0 from *c* to *d*. If h is not explicitly given, the function uses *h* = 1. As with fuzzy sets, it is possible to explicitly define the universal space or to accept the default setting for fuzzy relations.

```
In[4]:= Rel2 = FuzzyTrapezoid[{1, 3, 5, 7}, {0, 3, 6, 8}, 0.8,
            UniversalSpace -> {{0, 7,1}, {0, 8,1}}];
```

Here we use function FuzzySurfacePlot to look at our newly created fuzzy relation.

```
In[5]:= FuzzySurfacePlot[Rel2];
```



SetsToRelation[*func*, *A*, *B*] returns a new fuzzy relation with elements from the Cartesian product of the universal spaces of fuzzy sets *A* and *B*. The membership grades for the elements are arrived at by applying *func* to the corresponding membership grades of *A* and *B*.

To demonstrate this function, we first need to create some fuzzy sets. We use a couple of the fuzzy set creation functions described earlier in this manual to create two fuzzy sets.

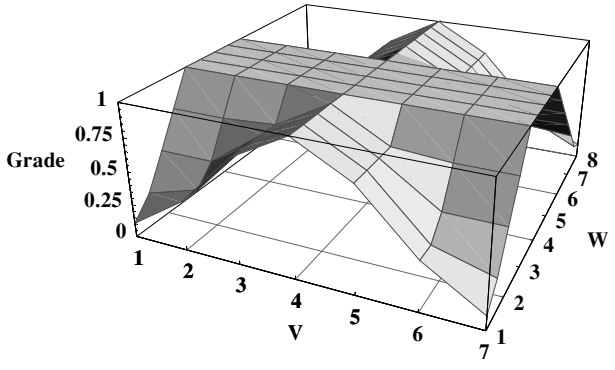
```
In[6]:= FS1 = FuzzyGaussian[4, 2, UniversalSpace -> {1, 7}]

Out[6]= FuzzySet[{{1, 0.105399}, {2, 0.367879}, {3, 0.778801}, {4, 1.},
          {5, 0.778801}, {6, 0.367879}, {7, 0.105399}}, UniversalSpace → {1, 7, 1}]

In[7]:= FS2 = FuzzyTrapezoid[1, 3, 5, 8, UniversalSpace -> {1, 8,0.5}]

Out[7]= FuzzySet[{{1.5, 0.25}, {2., 0.5}, {2.5, 0.75}, {3., 1}, {3.5, 1},
          {4., 1}, {4.5, 1}, {5., 1}, {5.5, 0.833333}, {6., 0.666667}, {6.5, 0.5},
          {7., 0.333333}, {7.5, 0.166667}}, UniversalSpace → {1, 8, 0.5}]
```

With two fuzzy sets, we can now use the SetsToRelation command to create a fuzzy relation. In this example, we use *Mathematica*'s Max function to combine the fuzzy sets.

```
In[8]:= Rel3 = SetsToRelation[Max, FS1, FS2];
```

Here we use function FuzzySurfacePlot to look at our newly created fuzzy relation.

```
In[9]:= FuzzySurfacePlot[Rel3];
```



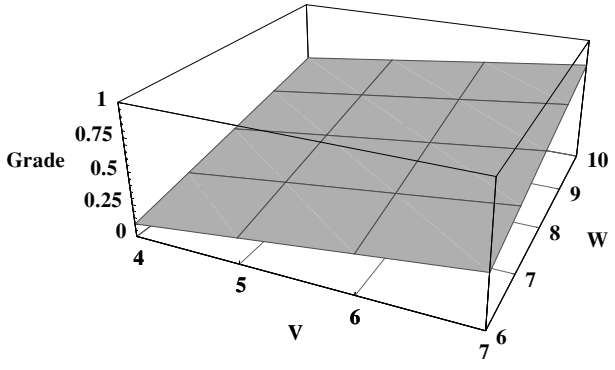
`FromMembershipMatrix[`*mat*`,`*{{va, vb}, {wa, wb}}*`]` returns a new fuzzy relation by combining the matrix of membership grades, *mat*, with the elements specified by the ranges *{va, vb}* and *{wa, wb}*. If no ranges are given, the function assumes the membership grades are for the elements starting at 1 with a size corresponding to the dimension of the matrix, *mat*.

To demonstrate the `FromMembershipMatrix` function, we first create a membership matrix that contains the membership grades that will be used in the relation. We then call our function with the matrix as the argument.

```
In[10]:= MyMembMat := {{0.1, 0.2, 0.3, 0.4, 0.5}, {0.2, 0.3, 0.4, 0.5, 0.6},
           {0.3, 0.4, 0.5, 0.6, 0.7}, {0.4, 0.5, 0.6, 0.7, 0.8}}

In[11]:= Rel4 = FromMembershipMatrix[MyMembMat,{{4,7},{6,10}}];
```

Here we use function `FuzzySurfacePlot` to look at our newly created fuzzy relation.

```
In[12]:= FuzzySurfacePlot[Rel4];
```



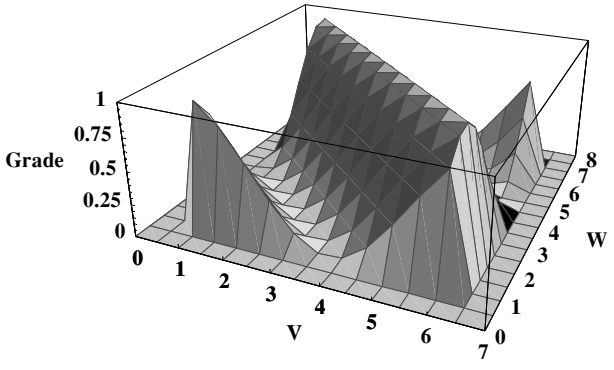
`CreateFuzzyRelation[`*func*`,{{va, vb, c1}, {wa, wb, c2}},`*opts*`]` returns a new fuzzy relation with membership grades that are the result of applying *func* to all the elements in the specified range, *{{va, vb, c1}, {wa, wb, c2}}*. If no range is given, the function is applied to all of the elements in the universal space. Be sure the function provided causes membership grades to be in the correct range, 0 to 1.

The following is an example of creating a fuzzy relation from a function. The first command defines the function, and the second command creates the fuzzy relation. We explicitly declare a universal space in this function rather that accept the default setting. Because fuzzy relations are quite large, we suppress the output by putting a semicolon at the end of the function. To examine the relation, we use one of the *Fuzzy Logic* package's graphing functions. More information about graphing fuzzy relations can be found in Chapter 6 Fuzzy Relation Visualization.

```
In[13]:= MyFunction2[x_, y_]:=1/2*(Sin[x + y]+1)

In[14]:= Rel5 = CreateFuzzyRelation[MyFunction2,{{1,6,0.5},{1,7,0.5}},
              UniversalSpace -> {{0,7,0.5},{0, 8,0.5}}];
```

You can use function `FuzzySurfacePlot` to look at newly created fuzzy relation.

```
In[15]:= FuzzySurfacePlot[Rel5];
```



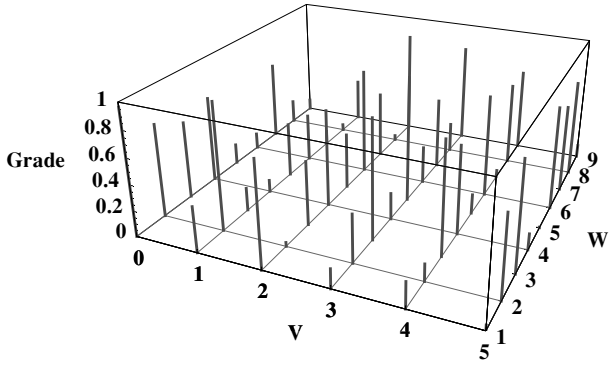
`RandomFuzzyRelation[{{a1, b1}, {a2, b2}}]` creates a random fuzzy relation with universal space of *{{a1, b1}, {a2, b2}}* and increments *c1* = *c2* = 1. By default, this function will create a random trapezoidal fuzzy relation. The option `Type` can be used to create a `Complete` random fuzzy relation. In addition, there is an option called `Normal` that produces a normal random fuzzy relation, when set to `True`.

option name	*default value*	
`Type`	`Trapezoid`	shape of random fuzzy set, admissible values are `Trapezoid` or `Complete`
`Normal`	`True`	an option that controls whether or not a random fuzzy relation is a normal one

Options for `RandomFuzzyRelation`.

```
In[16]:= RR1 = RandomFuzzyRelation[{{0, 5}, {1, 9}}, Normal -> True];

In[17]:= GlobalProjection[RR1]

Out[17]= 1

In[18]:= RR2 = RandomFuzzyRelation[{{0, 5}, {1, 9}}, Type -> Complete];
```

```
In[19]:= FuzzyPlot3D[RR2];
```



3 Fuzzy Operations

3.1 Introduction

Fuzzy Logic package contains a wide variety of operations that can be performed on fuzzy sets and fuzzy relations. This chapter introduces these functions and demonstrates how to use them.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

There are three groups of fuzzy operations, those that apply to fuzzy sets only, those that apply to fuzzy relations only, and those that apply to both fuzzy sets and fuzzy relations.

3.2 Fuzzy Set Operations

This section describes operations that can be performed on individual fuzzy sets.

`Height[`*A*`]`	return the value of the largest membership grade in fuzzy set *A*
`Cardinality[`*A*`]`	return the sum of all of the membership grades in fuzzy set *A*
`RelativeCardinality[`*A*`]`	return the cardinality of fuzzy set *A* divided by the total number of elements in the universal space of *A*
`FuzzyCardinality[`*A*`]`	return the fuzzy cardinality of fuzzy set *A*
`CenterOfArea[`*A*`]`	return the center of area defuzzification of fuzzy set *A*
`MeanOfMax[`*A*`]`	return the mean of maximum defuzzification of fuzzy set *A*
`SmallestOfMax[`*A*`]`	return the smallest of maximum defuzzification of fuzzy set *A*
`LargestOfMax[`*A*`]`	return the largest of maximum defuzzification of fuzzy set *A*

`BisectorOfArea[A]`	return the bisector of area defuzzification of fuzzy set *A*
`Support[A]`	return a list of all elements of fuzzy set *A* with nonzero membership grades
`Core[A]`	return a list of all elements of fuzzy set *A* with a membership grade equal to 1
`EquilibriumSet[A]`	return a list of all elements of fuzzy set *A* with membership grades equal to 0.5

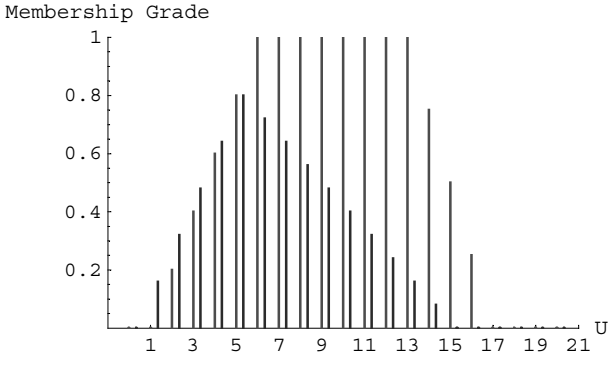
Operations that apply to fuzzy sets only.

To demonstrate the various operations, we first create two fuzzy sets. We use the `FuzzyTrapezoid` function, described earlier in this manual, to create the fuzzy sets.

```
In[2]:= FS1 = FuzzyTrapezoid[1, 6, 13, 17]

Out[2]= FuzzySet[{{2, 1/5}, {3, 2/5}, {4, 3/5}, {5, 4/5}, {6, 1}, {7, 1}, {8, 1}, {9, 1}, {10, 1},
          {11, 1}, {12, 1}, {13, 1}, {14, 3/4}, {15, 1/2}, {16, 1/4}}, UniversalSpace → {0, 20, 1}]

In[3]:= FS2 = FuzzyTrapezoid[0, 5, 5, 15, .8];
```

To examine the fuzzy sets we created, we use the `FuzzyPlot` graphing function. The graphing functions are described in Chapter 5 Fuzzy Set Visualization.

```
In[4]:= FuzzyPlot[FS1, FS2];
```



In this graph, the trapezoid-shaped fuzzy set represents `FS1`, and the triangular fuzzy set represents `FS2`. These fuzzy sets are plotted in red and blue, respectively, in a *Mathematica* session.

The function `Height[A]` returns the value of the largest membership grade in fuzzy set *A*. A fuzzy set with a height of 1 is called a normal fuzzy set.

```
In[5]:= Map[Height, {FS1, FS2}]

Out[5]= {1, 0.8}
```

We see that `FS1` is a normal fuzzy set, while `FS2` is a subnormal fuzzy set.

`Cardinality[A]` returns the sum of all of the membership grades in fuzzy set *A*.

```
In[6]:= Map[Cardinality, {FS1, FS2}] // N

Out[6]= {11.5, 6.}
```

`RelativeCardinality[A]` returns the cardinality of fuzzy set *A* divided by the total number of elements in the universal space of *A*.

```
In[7]:= Map[RelativeCardinality, {FS1, FS2}] // N

Out[7]= {0.547619, 0.285714}
```

`FuzzyCardinality[A]` returns the fuzzy cardinality of fuzzy set *A*. Here we compute the fuzzy cardinalities for `FS1` and `FS2`.

```
In[8]:= FuzzyCardinality[FS1]

Out[8]= {{15, 1/5}, {14, 1/4}, {13, 2/5}, {12, 1/2}, {11, 3/5}, {10, 3/4}, {9, 4/5}, {8, 1}}

In[9]:= FuzzyCardinality[FS2]

Out[9]= {{14, 0.08}, {13, 0.16}, {13, 0.16}, {11, 0.24}, {10, 0.32}, {10, 0.32},
          {8, 0.4}, {7, 0.48}, {5, 0.56}, {4, 0.64}, {4, 0.64}, {2, 0.72}, {1, 0.8}}
```

`CenterOfArea[A]` returns the center of area defuzzification of fuzzy set *A*. We first find the centers of area of for `FS1` and `FS2`.

```
In[10]:= Map[CenterOfArea, {FS1, FS2}] // N

Out[10]= {9.21739, 6.66667}
```

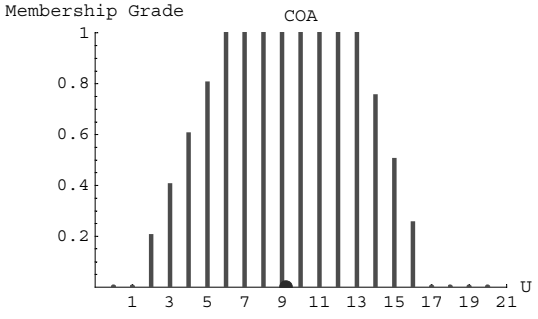
option name	default value	
ShowGraph	False	an option to control whether to display a plot

Option for CenterOfArea, MeanOfMax, SmallestOfMax, LargestOfMax, and BisectorOfArea.

CenterOfArea has an option ShowGraph that by default is set to False. If the option ShowGraph is set to True, this function returns a plot of fuzzy set with a dot along the base axis indicating where the center of area is located. The graph produced by the ShowGraph option can be customized using any of the options available with *Mathematica*'s standard Plot function. To demonstrate this, we give the plot a label with the PlotLabel option.

In[11]:= **CenterOfArea[FS1, ShowGraph → True, PlotLabel → "COA"];**

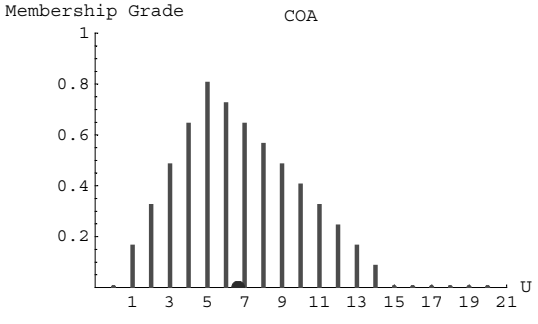
```
Center of area is 9.21739.
```




```
Center of area is 9.21739.
```

In[12]:= **CenterOfArea[FS2, ShowGraph → True, PlotLabel → "COA"];**

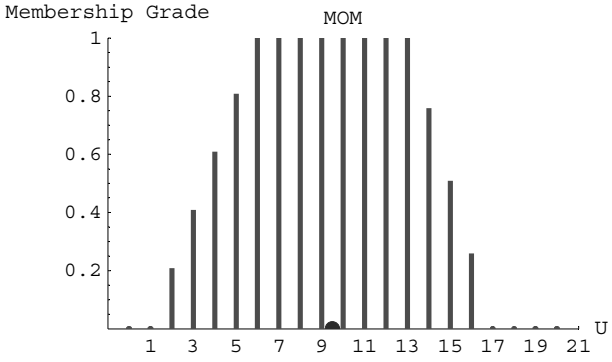
```
Center of area is 6.66667.
```



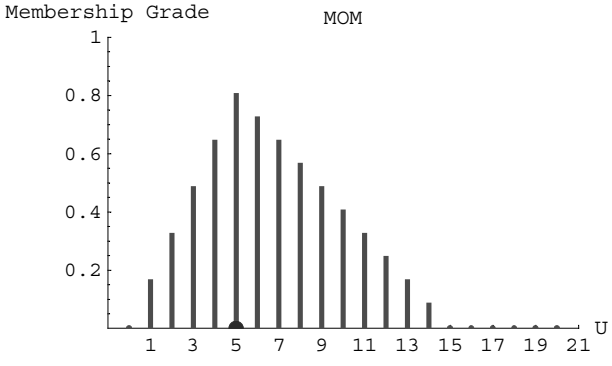

```
Center of area is 6.66667.
```

`MeanOfMax[`A`]` returns the mean of maximum defuzzification of fuzzy set A. This function also has a `ShowGraph` option for visualizing the defuzzification similarly to `CenterOfArea`.

```
In[13]:= MeanOfMax[FS1, ShowGraph → True, PlotLabel → "MOM"];

         Mean of max is 9.5.
```

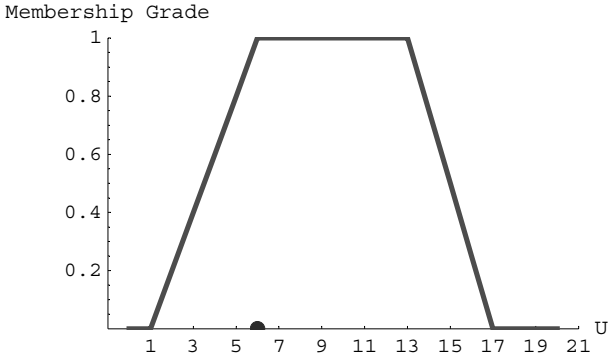



```
         Mean of max is 9.5.

In[14]:= MeanOfMax[FS2, ShowGraph → True, PlotLabel → "MOM"];

         Mean of max is 5..
```



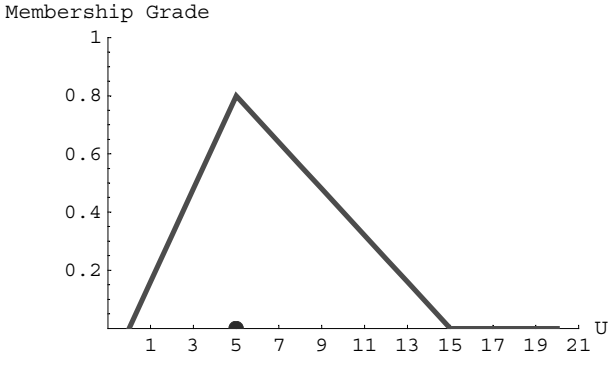
`SmallestOfMax[`A`]` returns the smallest of maximum defuzzification of fuzzy set A. This function also has a `ShowGraph` option for visualizing the defuzzification (see `CenterOfArea` above).

```
In[15]:= SmallestOfMax[FS1, ShowGraph → True, PlotJoined → True];

         Smallest of max is 6..
```



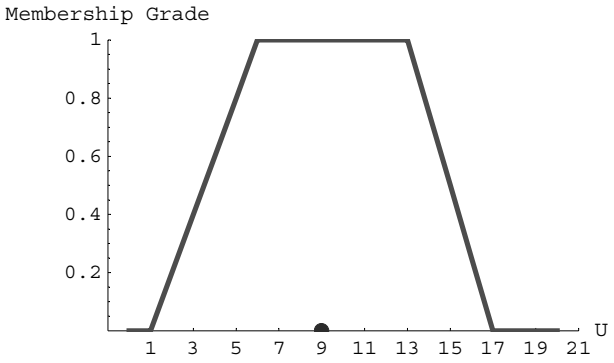
`LargestOfMax[`A`]` returns the largest of maximum defuzzification of fuzzy set A. This function also has a `ShowGraph` option for visualizing the defuzzification (see `CenterOfArea` above).

```
In[16]:= LargestOfMax[FS2, ShowGraph → True, PlotJoined → True];

         Largest of max is 5..
```



`BisectorOfArea[`A`]` returns the bisector of area defuzzification of fuzzy set A. This function also has a `ShowGraph` option for visualizing the defuzzification (see `CenterOfArea` above).

```
In[17]:= BisectorOfArea[FS1, ShowGraph → True, PlotJoined → True];

          Bisector of area is 9..
```



`Support[A]` returns a list of all elements of fuzzy set A with nonzero membership grades.

```
In[18]:= Support[FS1]

Out[18]= {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

`Core[A]` returns a list of all elements of fuzzy set A with a membership grade equal to 1.

```
In[19]:= Core[FS1]

Out[19]= {6, 7, 8, 9, 10, 11, 12, 13}
```

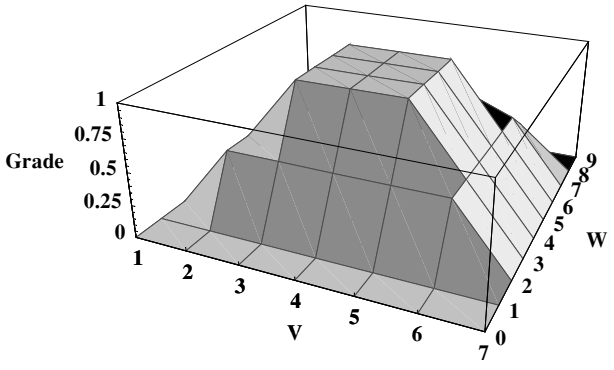
`EquilibriumSet[A]` returns a list of all elements of fuzzy set A with membership grades equal to 0.5.

```
In[20]:= EquilibriumSet[FS1]

Out[20]= {15}
```

3.3 Fuzzy Relation Operations

This section describes operations that can be performed on individual fuzzy relations. To demonstrate these operations, we create a new fuzzy relation with the `FuzzyTrapezoid` function. We use the `FuzzySurfacePlot` function to show the new fuzzy relation.

```
In[21]:= Rel1 = FuzzyTrapezoid[{1, 3, 5, 7}, {1, 3, 6, 9}, UniversalSpace → {{1, 7}, {0, 9}}];

In[22]:= FuzzySurfacePlot[Rel1];
```



`FirstProjection[`*A*`]`	return a fuzzy set that is the projection of fuzzy relation *A* onto the first space of the relation
`SecondProjection[`*A*`]`	return fuzzy set that is the projection of fuzzy relation *A* onto the second space of the relation
`GlobalProjection[`*A*`]`	return a value that represents the maximum membership grade in fuzzy relation *A*

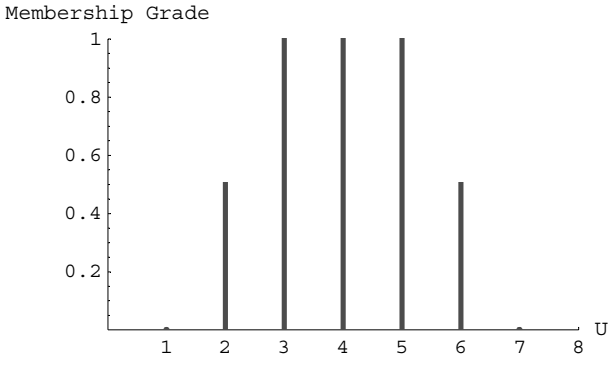
Operations that apply to fuzzy relations only.

`FirstProjection[`*A*`]` returns a fuzzy set that is the projection of fuzzy relation *A* onto the first space of the relation.

```
In[23]:= Proj1 = FirstProjection[Rel1]

Out[23]= FuzzySet[{{2, 1/2}, {3, 1}, {4, 1}, {5, 1}, {6, 1/2}}, UniversalSpace → {1, 7, 1}]
```

We can plot the first projection using `FuzzyPlot`.

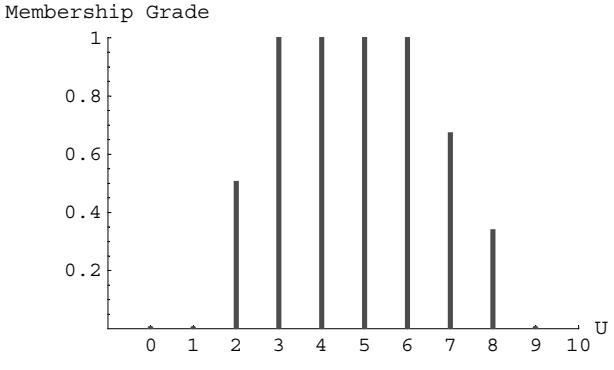
```
In[24]:= FuzzyPlot[Proj1];
```



`SecondProjection[A]` returns a fuzzy set that is the projection of fuzzy relation *A* onto the second space of the relation.

```
In[25]:= Proj2 = SecondProjection[Rel1]

Out[25]= FuzzySet[{{2, 1/2}, {3, 1}, {4, 1}, {5, 1}, {6, 1}, {7, 2/3}, {8, 1/3}},
          UniversalSpace → {0, 9, 1}]
```

```
In[26]:= FuzzyPlot[Proj2];
```



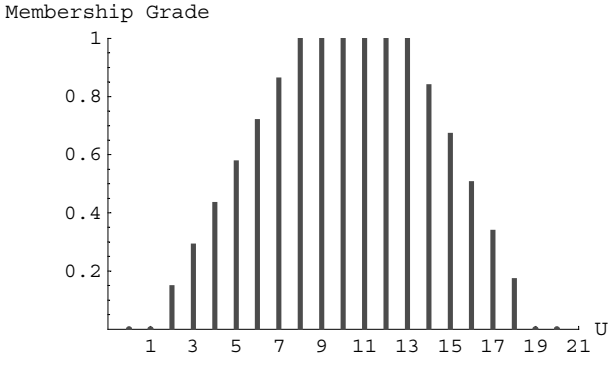
`GlobalProjection[A]` returns a value that represents the maximum membership grade in fuzzy relation *A*. This is analogous to the `Height` function for fuzzy sets.

```
In[27]:= GlobalProjection[Rel1]

Out[27]= 1
```

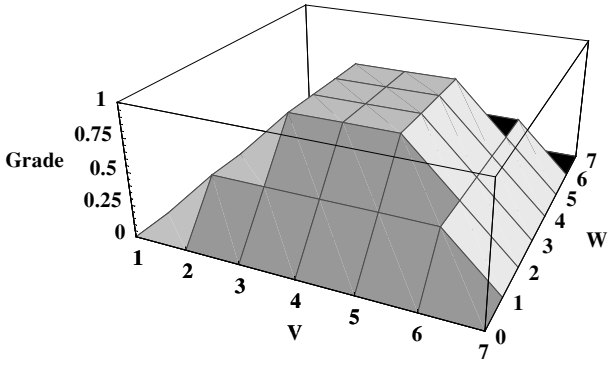
3.4 Fuzzy Set or Fuzzy Relation Operations

This section describes operations that can be performed on either individual fuzzy sets or individual fuzzy relations. To demonstrate these operations, we first create one new fuzzy set and one new fuzzy relation. In some of the examples that follow, we demonstrate functions operating on just a fuzzy set or just a fuzzy relation, but keep in mind that all the operators in this section work for both fuzzy sets and fuzzy relations.

```
In[28]:= FS3 = FuzzyTrapezoid[1, 8, 13, 19];

In[29]:= FuzzyPlot[FS3];
```




```
In[30]:= Rel =
           FuzzyTrapezoid[{1, 3, 5, 7}, {0, 2, 5, 7}, 0.8, UniversalSpace → {{1, 7}, {0, 7}}];

In[31]:= FuzzySurfacePlot[Rel];
```



`AlphaLevelSet[`*A*`, `*alpha*`]`	return a list of all elements of *A* with membership grades greater than or equal to *alpha*, where *A* is a fuzzy set or a fuzzy relation
`StrongAlphaLevelSet[`*A*`, `*alpha*`]`	return a list of all elements of *A* with membership grades greater than *alpha*, where *A* is a fuzzy set or a fuzzy relation
`LevelSet[`*A*`]`	return a list of all alpha levels of *A* , where *A* is a fuzzy set or fuzzy relation
`Subsethood[`*A*`, `*B*`]`	return the degree of subsethood of *A* in *B*, where *A* and *B* are fuzzy sets or fuzzy relations
`HammingDistance[`*A*`, `*B*`]`	return the Hamming distance from *A* to *B*, where *A* and *B* are fuzzy sets or fuzzy relations
`Concentrate[`*A*`]`	concentrate the membership grades of *A*, where *A* is a fuzzy set or a fuzzy relation
`Dilate[`*A*`]`	dilate the membership grades of *A*, where *A* is a fuzzy set or fuzzy relation
`IntensifyContrast[`*A*`]`	intensify the contrast between the membership grades of *A*
`Normalize[`*A*`]`	adjust the membership grades of *A* to create a normalized fuzzy set or fuzzy relation with a height of 1
`Complement[`*A*`]`	return the complement of *A*, where *A* can be a fuzzy set or a fuzzy relation
`FuzzyModify[`*func*`, `*A*`]`	modify the membership grades of *A* according to the supplied function*func*

Operations that apply to both fuzzy sets and fuzzy relations.

`AlphaLevelSet[`*A*`, `*alpha*`]` returns a list of all elements of fuzzy set *A* with membership grades greater than or equal to *alpha*.

```
In[32]:= AlphaLevelSet[FS1, 0.6]

Out[32]= {4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
```

`StrongAlphaLevelSet[`*A*`, `*alpha*`]` returns a list of all elements of fuzzy set *A* with membership grades greater than *alpha*.

```
In[33]:= StrongAlphaLevelSet[FS1, 0.6]

Out[33]= {5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
```

`LevelSet[A]` returns a list of all *alpha* of *A*, where *A* is a fuzzy set .

```
In[34]:= LevelSet[FS1]
```

$$\text{Out[34]=}\ \left\{\frac{1}{5}, \frac{1}{4}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{3}{4}, \frac{4}{5}, 1\right\}$$

`Subsethood[A, B]` returns the degree of subsethood of fuzzy set/fuzzy relation *A* in fuzzy set/fuzzy relation *B*. The value returned will be between 0 and 1, with values closer to 1 indicating that the second fuzzy set/fuzzy relation is closer to being a subset of the first fuzzy set/fuzzy relation.

```
In[35]:= Subsethood[FS1, FS2]

Out[35]= 0.486957

In[36]:= Subsethood[FS2, FS1]

Out[36]= 0.933333
```

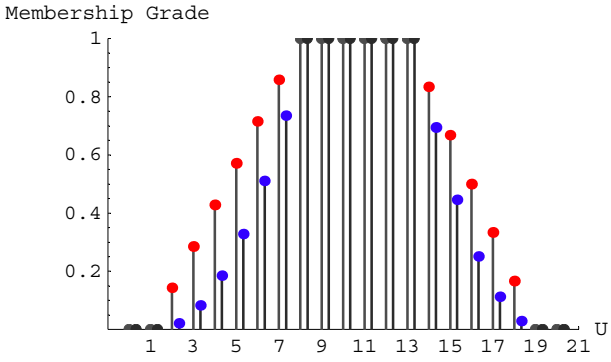
`HammingDistance[A, B]` returns the Hamming distance from fuzzy set/fuzzy relation *A* to fuzzy set/fuzzy relation *B*.

```
In[37]:= HammingDistance[FS1, FS2]

Out[37]= 6.3
```

`Concentrate[A]` concentrates the membership grades of *A*, where *A* is a fuzzy set or a fuzzy relation. The concentration function squares the membership grades for each element of *A*.

We find the concentration of our newly created fuzzy set here, and plot the original fuzzy set with its concentration on the same graph to show the effects of the operation.

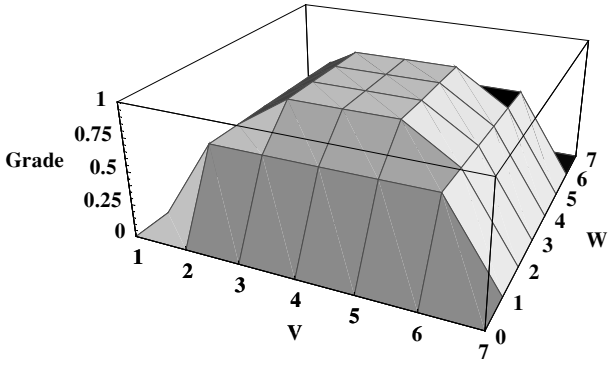
```
In[38]:= Con = Concentrate[FS3];

In[39]:= FuzzyPlot[FS3, Con, ShowDots -> True];
```



In the graph, the concentrated fuzzy set is included in the original fuzzy set, and it looks like a concentrated version of the original fuzzy set.

`Dilate[A]` dilates the membership grades of A, where A is a fuzzy set or fuzzy relation. The dilation function takes the square root of the membership grades for each element in A.

```
In[40]:= Dil = Dilate[Rel];

In[41]:= FuzzySurfacePlot[Dil];
```



From the graph we can clearly see that the new fuzzy relation is a dilated version of the original fuzzy relation.

`IntensifyContrast[A]` intensifies the contrast between the membership grades of A. The new membership grades are calculated as follows:

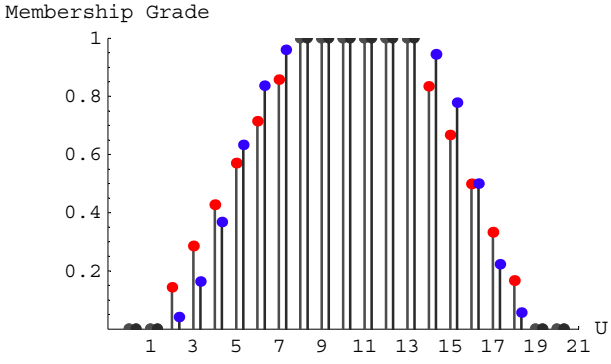
$$2\,x_n^2 \quad \text{if} \quad x_n \le 0.5,$$

$$1 - 2\,(1 - x_n)^2 \quad \text{otherwise,}$$

where x_n - original membership grade.

```
In[42]:= IC = IntensifyContrast[FS3];
```

```
In[43]:= FuzzyPlot[FS3, IC, ShowDots -> True];
```

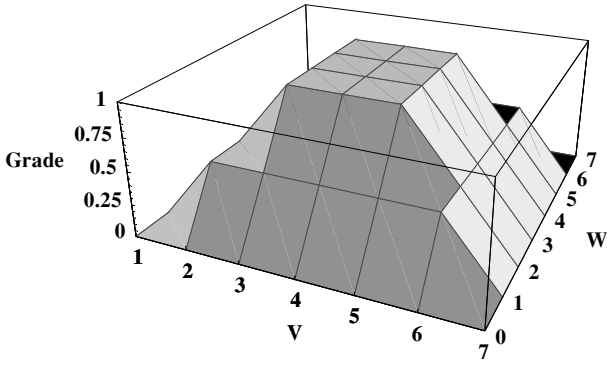


`Normalize[A]` adjusts the membership grades of A to create a normalized fuzzy set or fuzzy relation with a height of 1.

```
In[44]:= Norm = Normalize[Rel];
```

```
General::spell1 : Possible spelling error:
   new symbol name "Norm" is similar to existing symbol "Nor".
```

```
In[45]:= FuzzySurfacePlot[Norm];
```



`Complement[A]` returns the complement (standard or nonstandard) of A, where A can be a fuzzy set or a fuzzy relation.

option name	*default value*	
`Type`	`Standard`	specifies the type of complement

An option for `Complement`.

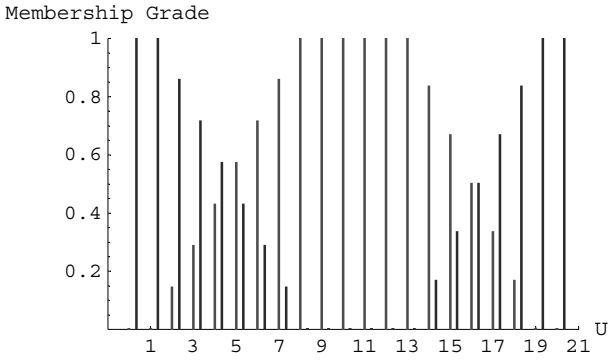
The `Type` option specifies the type of complement to be performed. The `Standard` complement is the default, but the following parameterized complements are also available.

`Type`	*parameter*
`Sugeno[`*alpha*`]`	*alpha* in the range –1 to infinity
`Yager[`w`]`	w in the range 0 to infinity

Admissible values for `Type` for `Complement` for nonstandard cases.

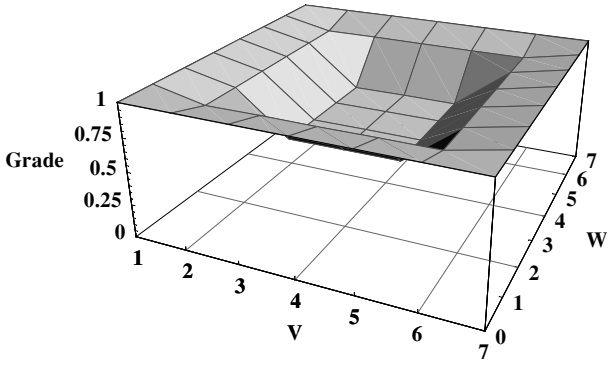
We find the `Complement` of our fuzzy set. Since we have not specified a `Type`, the function performs the `Standard` complement.

```
In[46]:= Compl1 = Complement[FS3];

In[47]:= FuzzyPlot[FS3, Compl1];
```



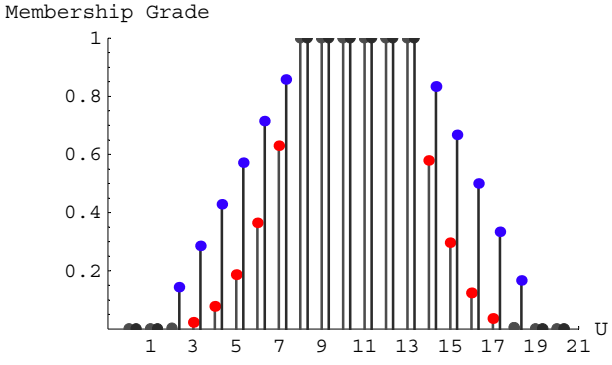
Here we use a `Yager` complement of our fuzzy relation with the parameter *w* set to 2.

```
In[48]:= Compl2 = Complement[Rel, Type → Yager[2]];

In[49]:= FuzzySurfacePlot[Compl2];
```



You can define your own operations with the help of `FuzzyModify`. `FuzzyModify[`*func*`, `*A*`]` modifies the membership grades of *A* according to the supplied function *func*. *A* may be either a fuzzy set or a fuzzy relation. In this example, we first define a function that cubes its input. We then perform the `FuzzyModify` operation using the cubing function to modify the membership grades of our original fuzzy set.

```
In[50]:= MyModFun[x_] := x^3

In[51]:= FM = FuzzyModify[MyModFun, FS3]

Out[51]= FuzzySet[{{2, 1/343}, {3, 8/343}, {4, 27/343}, {5, 64/343}, {6, 125/343},
          {7, 216/343}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}, {13, 1}, {14, 125/216},
          {15, 8/27}, {16, 1/8}, {17, 1/27}, {18, 1/216}}, UniversalSpace → {0, 20, 1}]

In[52]:= FuzzyPlot[FM, FS3, ShowDots -> True];
```



`Equality[`A`, `B`]`	return `True` if A and B are equal, and `False` otherwise
`Included[`A`, `B`]`	return `True` if A is included in B and `False` otherwise

Distinctions for fuzzy sets and fuzzy relations.

`Equality[`A`, `B`]` returns `True` if A and B are equal, and `False` otherwise.

```
In[53]:= Equality[FS3, FS3]

Out[53]= True
```

`Included[`A`, `B`]` returns `True` if A is included in B and `False` otherwise. To be included, the membership grades of A must be less than or equal to the membership grades of B for all elements.

```
In[54]:= Included[FS3, FS3]

Out[54]= True
```

4 Aggregation Operations

4.1 Introduction

Fuzzy Logic package provides a number of functions for aggregating two or more fuzzy sets or fuzzy relations. In this chapter, we will introduce these functions and demonstrate how to use them.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

To use the various aggregation operations, we need a few fuzzy sets or fuzzy relations. For convenience, we demonstrate the aggregation operations in this chapter using a pair of fuzzy sets. Keep in mind, however, that the aggregation operations will also work with two or more fuzzy sets or fuzzy relations.

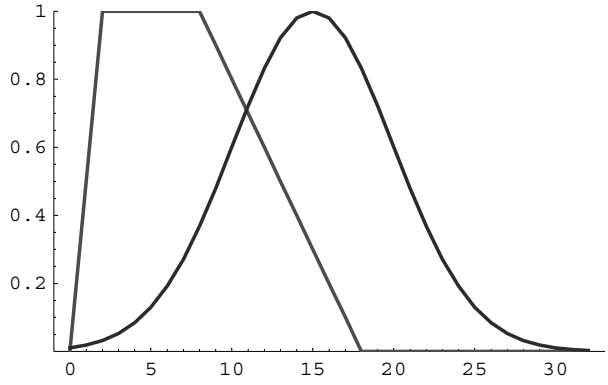
When using the aggregation operations, the fuzzy sets or fuzzy relations being combined must be defined on the same universal space. It does not make sense to combine items defined on different universal spaces.

We create two fuzzy sets to use in our demonstrations with the `FuzzyTrapezoid` and `FuzzyGaussian` functions that were described earlier in Chapter 1 Creating Fuzzy Sets.

```
In[2]:= SetOptions[FuzzySet, UniversalSpace → {0, 32, 1}];

In[3]:= FS1 = FuzzyTrapezoid[0, 2, 8, 18, 1];

In[4]:= FS2 = FuzzyGaussian[15, 7];
```

To look at these two fuzzy sets, we use the `FuzzyPlot` command. On the graph, `FS1` is represented by the trapezoidal fuzzy set with a height of 1. `FS2` is the Gaussian fuzzy set.

```
In[5]:= plot1 = FuzzyPlot[FS1, FS2, PlotJoined → True, AxesLabel → None];
```



4.2 Intersection (t-norm) and Union (s-norm) Operations

`Intersection[`*A1*`,`*A2*`, ... ,`*An*`]`	return the intersection of *A1, A2, ... , An,* where the *A*' s are either fuzzy sets or fuzzy relations
`Union[`*A1*`,`*A2*`, ... ,`*An*`]`	return the union of *A1, A2, ... , An,* where the *A*' s are either fuzzy sets or fuzzy relations

Basic aggregation operations.

`Intersection[`*A1*`, `*A2*`, ... , `*An*`]` returns the intersection of *A1, A2, ... , An,* where the `A`'s are either fuzzy sets or fuzzy relations. This function has a `Type` option for specifying what kind of intersection should be taken. The default setting is the `Standard` or `Min` intersection. Other valid types of intersections are shown in the following table.

Type	*parameter*
`Hamacher[v]`	*v* in the range 0 to infinity
`Frank[s]`	*s* in the range 0 to infinity, except 1
`Yager[w]`	*w* in the range 0 to infinity
`DuboisPrade[a]`	*a* in the range 0 to 1
`Dombi[alpha]`	*alpha* in the range 0 to infinity

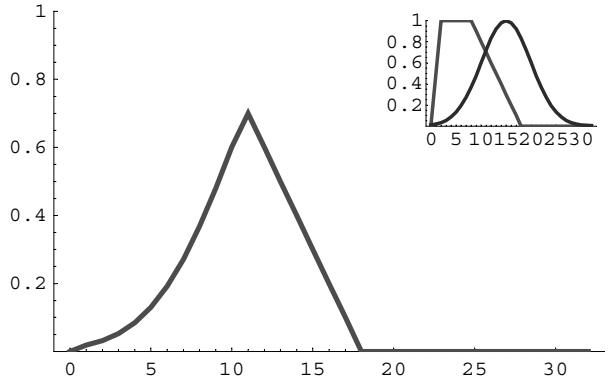
Values of `Type` option for Intersection.

For our first example, we use the default intersection to combine the two fuzzy sets that we created earlier. This produces the `Standard` or `Min` intersection of the two fuzzy sets.

```
In[6]:= Int1 = Intersection[FS1, FS2];
```

For the standard case, the intersection takes the minimum of `FS1` and `FS2`. We can plot the original fuzzy sets, `FS1` and `FS2`, along with their intersection using the `FuzzyPlot` function. The `PlotJoined` option is set to `True` to produce a continuous representation of the discrete fuzzy sets. We use *Mathematica*'s `Epilog` function to show the plot of the original fuzzy sets with the plot of the intersection. We also could have used the `FuzzyPlot` command to plot all of these fuzzy sets on one graph, and we use that technique below.

```
In[7]:= FuzzyPlot[Int1, PlotJoined → True,
          Epilog → Rectangle[{12, .6}, {40, 1}, plot1], AxesLabel → None];
```

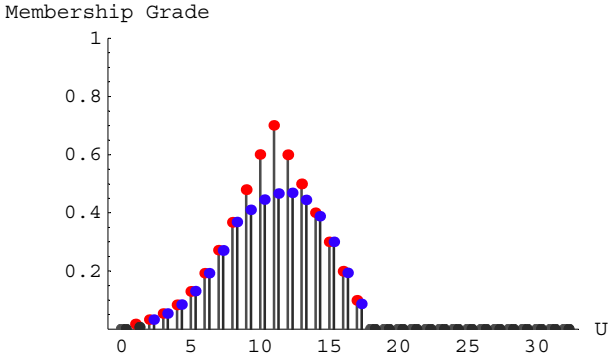


We now experiment with some of the nonstandard intersections by changing the `Type` option. Here we take the `Hamacher` intersection of the two fuzzy sets where the parameter *w* is set to 2.

```
In[8]:= Int2 = Intersection[FS1, FS2, Type → Hamacher[2]]

Out[8]= FuzzySet[{{1, 0.00614272}, {2, 0.031778}, {3, 0.0529305}, {4, 0.0846367},
          {5, 0.129923}, {6, 0.191463}, {7, 0.270868}, {8, 0.367879}, {9, 0.410335},
          {10, 0.444751}, {11, 0.466047}, {12, 0.467919}, {13, 0.443425}, {14, 0.387226},
          {15, 0.3}, {16, 0.192843}, {17, 0.0860875}}, UniversalSpace → {0, 32, 1}]
```

We compare this intersection with the `Standard` intersection by plotting the two results on the same graph. When we do this, we see that our `Hamacher` intersection returns values slightly less than the standard `Min` intersection. In general, this is true for all nonstandard intersections.

```
In[9]:= FuzzyPlot[Int1, Int2, ShowDots -> True];
```



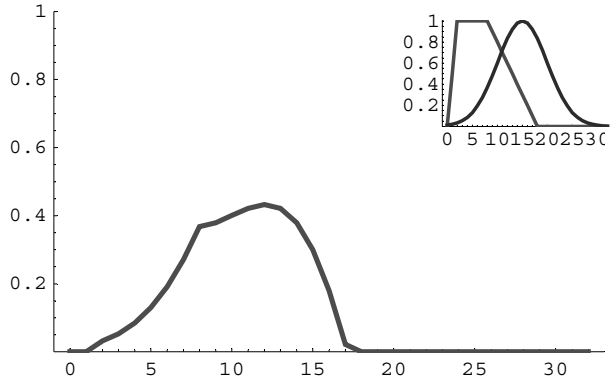
There is a group of t-norm operators that are collectively called products. Each of these operations is simply a specific case of one of the intersections listed before, so we have not included explicit functions for these operations. If you desire to use one of these operators, however, you may use the appropriate value for the option `Type` of union from the following table.

product type	*equivalent* `Intersection` *type*
Drastic Product	`Hamacher[Infinity],Yager[0]`
Bounded Product	`Yager[1]`
Einstein Product	`Hamacher[2]`
Algebraic Product	`Hamacher[1],DuboisPrade[1]`
Hamacher Product	`Hamacher[0]`

Relation between types of products and option Type for Intersection.

As an example, if you wanted to find the bounded product of `FS1` and `FS2`, you could use a `Yager` intersection like this.

```
In[10]:= BP = Intersection[FS1, FS2, Type → Yager[1]]

Out[10]= FuzzySet[{{2, 0.031778}, {3, 0.0529305}, {4, 0.0846367}, {5, 0.129923},
           {6, 0.191463}, {7, 0.270868}, {8, 0.367879}, {9, 0.379652}, {10, 0.400373},
           {11, 0.421422}, {12, 0.432208}, {13, 0.42161}, {14, 0.379799},
           {15, 0.3}, {16, 0.179799}, {17, 0.0216104}}, UniversalSpace → {0, 32, 1}]

In[11]:= FuzzyPlot[BP, PlotJoined → True,
           Epilog → Rectangle[{14, .6}, {40, 1}, plot1], AxesLabel → None];
```



`Union[`*A1*`,`*A2*`,...,`*An*`,`*opts*`]` returns the union of *A1, A2, ... , An,* where the *A*'s can be fuzzy sets or fuzzy relations. This function has a Type option for specifying what type of union should be taken. The default is the `Standard` or `Max` union. Other valid union types are shown in the following table.

`Type`	*parameter*
`Hamacher[`*v*`]`	*v* in the range 0 to infinity
`Frank[`*s*`]`	*s* in the range 0 to infinity, except 1
`Yager[`*w*`]`	*w* in the range 0 to infinity
`DuboisPrade[`*a*`]`	*a* in the range 0 to 1
`Dombi[`*alpha*`]`	*alpha* in the range 0 to infinity

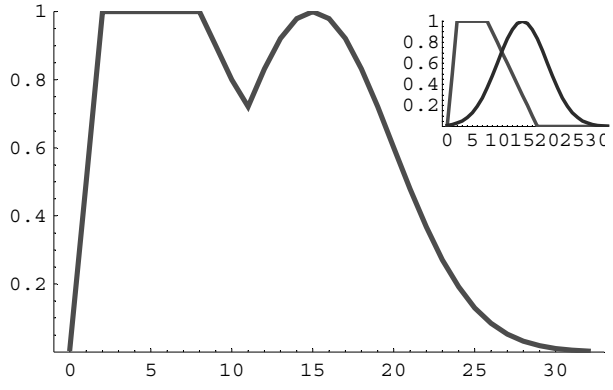
Values of `Type` option for `Union`.

For our first `Union` example, we use the default `Standard` union to combine our two fuzzy sets.

```
In[12]:= Un1 = Union[FS1, FS2]

Out[12]= FuzzySet[{{0, 0.0101342}, {1, 1/2}, {2, 1}, {3, 1}, {4, 1}, {5, 1}, {6, 1}, {7, 1},
           {8, 1}, {9, 9/10}, {10, 4/5}, {11, 0.721422}, {12, 0.832208}, {13, 0.92161},
           {14, 0.979799}, {15, 1.}, {16, 0.979799}, {17, 0.92161}, {18, 0.832208},
           {19, 0.721422}, {20, 0.600373}, {21, 0.479652}, {22, 0.367879},
           {23, 0.270868}, {24, 0.191463}, {25, 0.129923}, {26, 0.0846367},
           {27, 0.0529305}, {28, 0.031778}, {29, 0.0183156}, {30, 0.0101342},
           {31, 0.00538311}, {32, 0.00274504}}, UniversalSpace → {0, 32, 1}]
```

We look at the results by plotting the original fuzzy sets along with the union. On the graph, since the union is graphed last, it covers the original fuzzy sets in certain areas. If you remember what our original fuzzy sets looked like, however, you can see that the standard union takes the maximum membership grade of the two fuzzy sets for each element.

```
In[13]:= FuzzyPlot[Un1, PlotJoined → True,
           Epilog → Rectangle[{14, .6}, {40, 1}, plot1], AxesLabel → None];
```

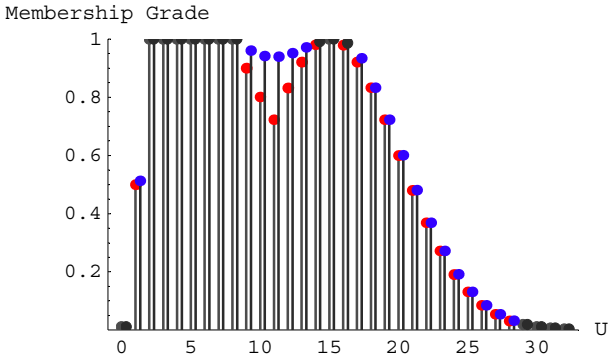


Now let's look at a nonstandard union. This time we use the `Frank` union with parameter *s* set to 3. The formulas for all the aggregation operations can be found in the appendix.

```
In[14]:= Un2 = Union[FS1, FS2, Type → Frank[3]]

Out[14]= FuzzySet[{{0, 0.0101342}, {1, 0.511569}, {2, 1.}, {3, 1.}, {4, 1.}, {5, 1.},
           {6, 1.}, {7, 1.}, {8, 1.}, {9, 0.960128}, {10, 0.940352}, {11, 0.938509},
           {12, 0.950529}, {13, 0.970518}, {14, 0.990518}, {15, 1.}, {16, 0.985729},
           {17, 0.933408}, {18, 0.832208}, {19, 0.721422}, {20, 0.600373}, {21, 0.479652},
           {22, 0.367879}, {23, 0.270868}, {24, 0.191463}, {25, 0.129923}, {26, 0.0846367},
           {27, 0.0529305}, {28, 0.031778}, {29, 0.0183156}, {30, 0.0101342},
           {31, 0.00538311}, {32, 0.00274504}}, UniversalSpace → {0, 32, 1}]
```

We compare this latest union with the `Standard` union from the first example by plotting the two results together.

```
In[15]:= FuzzyPlot[Un1, Un2, ShowDots -> True];
```



From the graph, you see that the `Frank` union returns membership grades greater than or equal to the membership grades returned by the `Standard` union. This condition is true for all of the nonstandard unions.

There is a group of s-norm operators, which are collectively called sums. Each of these operations is simply a specific case of one of the unions listed before, so we do not include explicit functions for these operations. If you desire to use one of these operators, however, you may use the appropriate value for the option `Type` of union from the following table.

sum type	*equivalent* `Union` *type*
Drastic Sum	`Hamacher[Infinity],Yager[0]`
Bounded Sum	`Yager[1]`
Einstein Sum	`Hamacher[2]`
Algebraic Sum	`Hamacher[1],DuboisPrade[1]`
Hamacher Sum	`Hamacher[0]`

Relation between types of sums and option `Type` for `Union`.

As an example, to find the drastic sum of `FS1` and `FS2`, you would enter the following.

```
In[16]:= DS = Union[FS1, FS2, Type → Hamacher[Infinity]]

Out[16]= FuzzySet[{{0, 0.0101342}, {1, 1}, {2, 1}, {3, 1}, {4, 1}, {5, 1}, {6, 1}, {7, 1},
           {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}, {13, 1}, {14, 1}, {15, 1}, {16, 1},
           {17, 1}, {18, 0.832208}, {19, 0.721422}, {20, 0.600373}, {21, 0.479652},
           {22, 0.367879}, {23, 0.270868}, {24, 0.191463}, {25, 0.129923}, {26, 0.0846367},
           {27, 0.0529305}, {28, 0.031778}, {29, 0.0183156}, {30, 0.0101342},
           {31, 0.00538311}, {32, 0.00274504}}, UniversalSpace → {0, 32, 1}]
```

4.3 Averaging Operations

Averaging operators are aggregation operators that fall in the region between intersections and unions. In this section, we examine some of the common averaging operations.

`FuzzyArithmeticMean[`*A1, A2, ... , An*`]`
return the arithmetic mean of *A1, A2, ... , An,* where the *A*' s are fuzzy sets or fuzzy relations

FuzzyGeometricMean[*A1, A2,... , An*]
return the geometric mean of *A1, A2, ... , An,* where the *A*' s are fuzzy sets or fuzzy relations

FuzzyHarmonicMean[*A1, A2,... , An*]
return the harmonic mean of A1, A2, ... , An, where the A' s are fuzzy sets or fuzzy relations

`GeneralizedMean[`*A1*`,` *A2*`,... ,` *An*`,` *alpha*`]`
return the generalized mean of *A1, A2, ... , An,* where the *A*' s are fuzzy sets or fuzzy relations

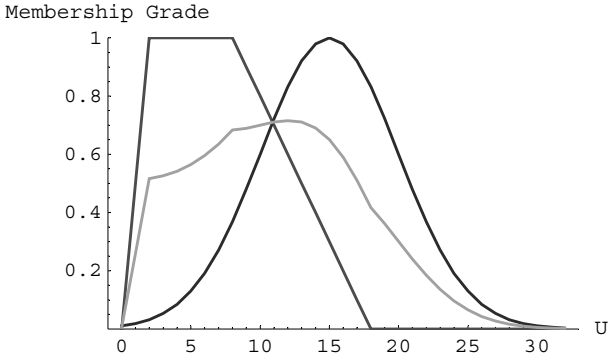
Averaging aggregate operations.

`FuzzyArithmeticMean[`*A1*`,` *A2*`, ... ,` *An*`]` returns the arithmetic mean of *A1, A2, ... , An,* where the *A*'s are fuzzy sets or fuzzy relations.

```
In[17]:= AM = FuzzyArithmeticMean[FS1, FS2]

Out[17]= FuzzySet[{{0, 0.00506711}, {1, 0.259158}, {2, 0.515889}, {3, 0.526465},
           {4, 0.542318}, {5, 0.564961}, {6, 0.595731}, {7, 0.635434}, {8, 0.68394},
           {9, 0.689826}, {10, 0.700187}, {11, 0.710711}, {12, 0.716104},
           {13, 0.710805}, {14, 0.689899}, {15, 0.65}, {16, 0.589899}, {17, 0.510805},
           {18, 0.416104}, {19, 0.360711}, {20, 0.300187}, {21, 0.239826}, {22, 0.18394},
           {23, 0.135434}, {24, 0.0957314}, {25, 0.0649613}, {26, 0.0423183},
           {27, 0.0264653}, {28, 0.015889}, {29, 0.00915782}, {30, 0.00506711},
           {31, 0.00269155}, {32, 0.00137252}}, UniversalSpace → {0, 32, 1}]
```

We look at how this averaging operation behaves by plotting the original fuzzy sets with their average. The membership grades for the average fuzzy set falls between the membership grades of the two original fuzzy sets.

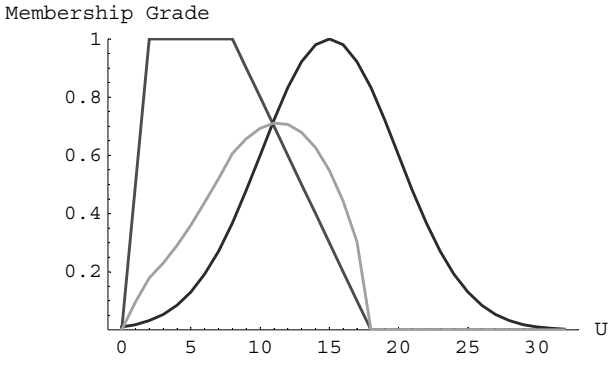
```
In[18]:= FuzzyPlot[FS1, FS2, AM, PlotJoined → True];
```



`FuzzyGeometricMean[`*A1*`, `*A2*`, ... , `*An*`]` returns the geometric mean of *A1, A2, ... , An,* where the *A*'s are fuzzy sets or fuzzy relations.

```
In[19]:= GM = FuzzyGeometricMean[FS1, FS2]

Out[19]= FuzzySet[{{1, 0.0956965}, {2, 0.178264}, {3, 0.230066}, {4, 0.290924},
           {5, 0.360448}, {6, 0.437565}, {7, 0.52045}, {8, 0.606531}, {9, 0.657029},
           {10, 0.693036}, {11, 0.71063}, {12, 0.706629}, {13, 0.678826}, {14, 0.626035},
           {15, 0.547723}, {16, 0.442673}, {17, 0.30358}}, UniversalSpace → {0, 32, 1}]
```

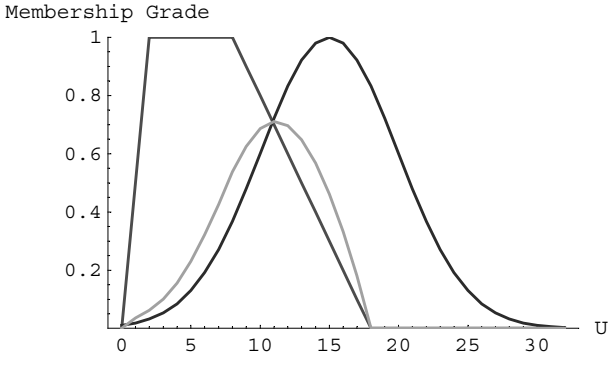
```
In[20]:= FuzzyPlot[FS1, FS2, GM, PlotJoined → True];
```



`FuzzyHarmonicMean[`*A1*`,`*A2*`, ... ,`*An*`]` returns the harmonic mean of *A1, A2, ... , An,* where the *A*'s are fuzzy sets or fuzzy relations.

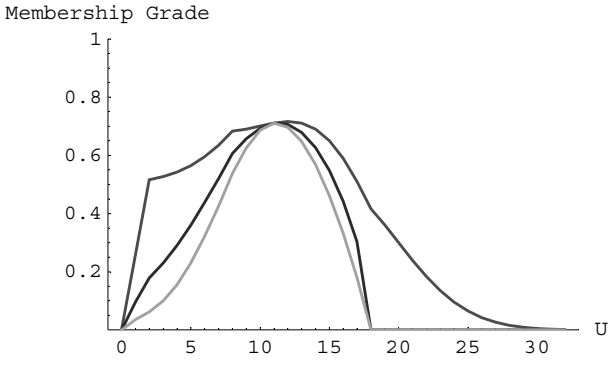
```
In[21]:= HM = FuzzyHarmonicMean[FS1, FS2]

Out[21]= FuzzySet[{{1, 0.0353368}, {2, 0.0615986}, {3, 0.100539}, {4, 0.156065},
           {5, 0.229967}, {6, 0.321391}, {7, 0.426273}, {8, 0.537883}, {9, 0.625791},
           {10, 0.685958}, {11, 0.71055}, {12, 0.69728}, {13, 0.648286}, {14, 0.568082},
           {15, 0.461538}, {16, 0.332192}, {17, 0.180423}}, UniversalSpace → {0, 32, 1}]
```

```
In[22]:= FuzzyPlot[FS1, FS2, HM, PlotJoined → True];
```



Now we compare the various averaging operations by plotting all the results of the averaging operations on the same graph.

```
In[23]:= FuzzyPlot[AM, GM, HM, PlotJoined → True];
```

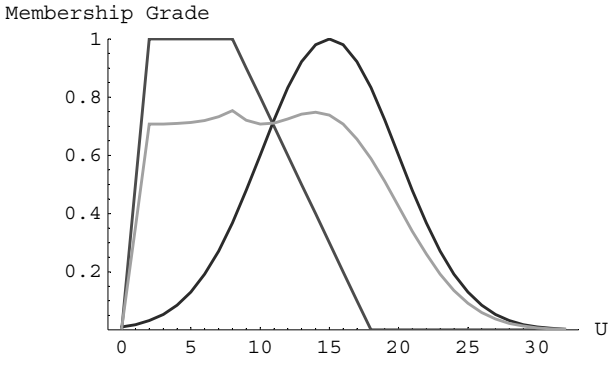


`GeneralizedMean[`*A1*`,`*A2*`, ... ,`*An*`,`*alpha*`]` returns the generalized mean of *A1, A2, ... , An,* where the *A*'s are fuzzy sets or fuzzy relations. This function takes a parameter *alpha*.

This function is a general averaging function which encompasses the full range of averaging operations. The averaging operations shown earlier are specific cases of the generalized mean. This function provides you with a great deal of flexibility when choosing how to average fuzzy sets. Here is an example where the parameter *alpha* is 2.

```
In[24]:= GenM = GeneralizedMean[FS1, FS2, 2]

Out[24]= FuzzySet[{{0, 0.00716598}, {1, 0.353791}, {2, 0.707464}, {3, 0.708097},
           {4, 0.709635}, {5, 0.71305}, {6, 0.719951}, {7, 0.732588}, {8, 0.753437},
           {9, 0.721133}, {10, 0.707265}, {11, 0.710792}, {12, 0.725455}, {13, 0.741406},
           {14, 0.748333}, {15, 0.738241}, {16, 0.707109}, {17, 0.655502},
           {18, 0.58846}, {19, 0.510123}, {20, 0.424528}, {21, 0.339165}, {22, 0.26013},
           {23, 0.191533}, {24, 0.135385}, {25, 0.0918692}, {26, 0.0598472},
           {27, 0.0374275}, {28, 0.0224705}, {29, 0.0129511}, {30, 0.00716598},
           {31, 0.00380643}, {32, 0.00194104}}, UniversalSpace → {0, 32, 1}]
```

```
In[25]:= FuzzyPlot[FS1, FS2, GenM, PlotJoined → True];
```



4.4 Difference Operations

`Difference[`A`, `B`]`	return the general difference between A and B, where A and B are fuzzy sets or fuzzy relations
`AbsoluteDifference[`A`, `B`]`	return the absolute difference between A and B, where A and B are fuzzy sets or fuzzy relations
`SymmetricDifference[`A`, `B`]`	return the symmetric difference between A and B, where A and B are fuzzy sets or fuzzy relations

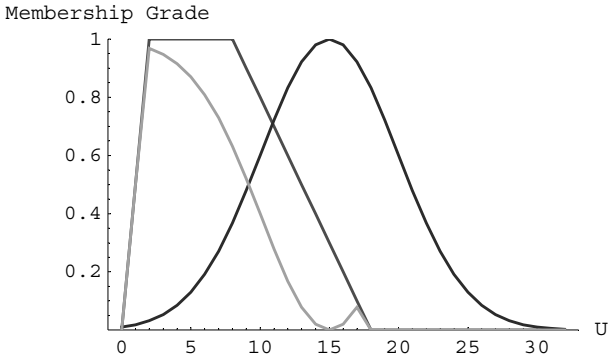
Difference aggregate operations.

`Difference[`A`, `B`]` returns the general difference between A and B, where A and B are fuzzy sets or fuzzy relations.

```
In[26]:= Diff = Difference[FS1, FS2]

Out[26]= FuzzySet[{{1, 1/2}, {2, 0.968222}, {3, 0.947069}, {4, 0.915363}, {5, 0.870077},
           {6, 0.808537}, {7, 0.729132}, {8, 0.632121}, {9, 0.520348}, {10, 0.399627},
           {11, 0.278578}, {12, 0.167792}, {13, 0.0783896}, {14, 0.0202013},
           {16, 0.0202013}, {17, 0.0783896}}, UniversalSpace → {0, 32, 1}]
```

To get a feel of what kind of result the differences produce, we plot the original fuzzy sets along with the difference on the same plot.

```
In[27]:= FuzzyPlot[FS1, FS2, Diff, PlotJoined → True];
```

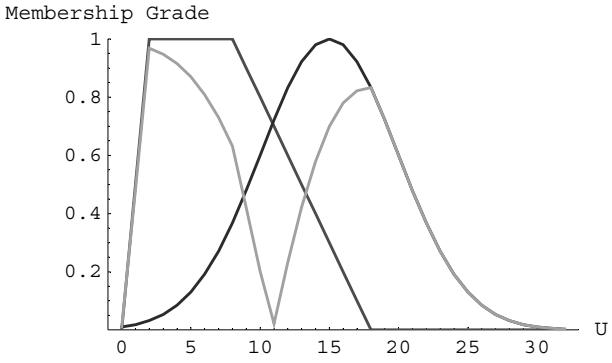


`AbsoluteDifference[`A`, `B`]` returns the absolute difference between A and B, where A and B are fuzzy sets or fuzzy relations.

```
In[28]:= AD = AbsoluteDifference[FS1, FS2]

Out[28]= FuzzySet[{{0, 0.0101342}, {1, 0.481684}, {2, 0.968222}, {3, 0.947069}, {4, 0.915363},
           {5, 0.870077}, {6, 0.808537}, {7, 0.729132}, {8, 0.632121}, {9, 0.420348},
           {10, 0.199627}, {11, 0.0214223}, {12, 0.232208}, {13, 0.42161}, {14, 0.579799},
           {15, 0.7}, {16, 0.779799}, {17, 0.82161}, {18, 0.832208}, {19, 0.721422},
           {20, 0.600373}, {21, 0.479652}, {22, 0.367879}, {23, 0.270868}, {24, 0.191463},
           {25, 0.129923}, {26, 0.0846367}, {27, 0.0529305}, {28, 0.031778}, {29, 0.0183156},
           {30, 0.0101342}, {31, 0.00538311}, {32, 0.00274504}}, UniversalSpace → {0, 32, 1}]
```

We again look at the results with the `FuzzyPlot` function.

```
In[29]:= FuzzyPlot[FS1, FS2, AD, PlotJoined → True];
```

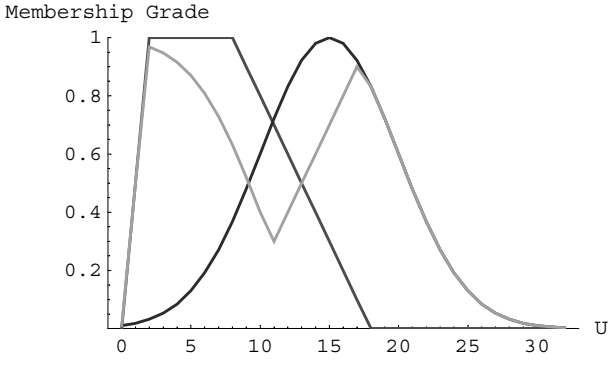


`SymmetricDifference[`A`,` B`]` returns the symmetric difference between A and B, where A and B are fuzzy sets or fuzzy relations.

```
In[30]:= SD = SymmetricDifference[FS1, FS2]
```

```
Out[30]= FuzzySet[{{0, 0.0101342}, {1, 1/2}, {2, 0.968222}, {3, 0.947069}, {4, 0.915363},
           {5, 0.870077}, {6, 0.808537}, {7, 0.729132}, {8, 0.632121}, {9, 0.520348},
           {10, 0.399627}, {11, 3/10}, {12, 2/5}, {13, 1/2}, {14, 3/5}, {15, 7/10}, {16, 4/5},
           {17, 9/10}, {18, 0.832208}, {19, 0.721422}, {20, 0.600373}, {21, 0.479652},
           {22, 0.367879}, {23, 0.270868}, {24, 0.191463}, {25, 0.129923}, {26, 0.0846367},
           {27, 0.0529305}, {28, 0.031778}, {29, 0.0183156}, {30, 0.0101342},
           {31, 0.00538311}, {32, 0.00274504}}, UniversalSpace → {0, 32, 1}]
```

```
In[31]:= FuzzyPlot[FS1, FS2, SD, PlotJoined → True];
```



4.5 User-Defined Aggregators

`GeneralAggregator[`*fun*`, A1, A2, ... , An]`
return combined membership grades of fuzzy sets or relations *A1, A2, ... , An* using function *fun*

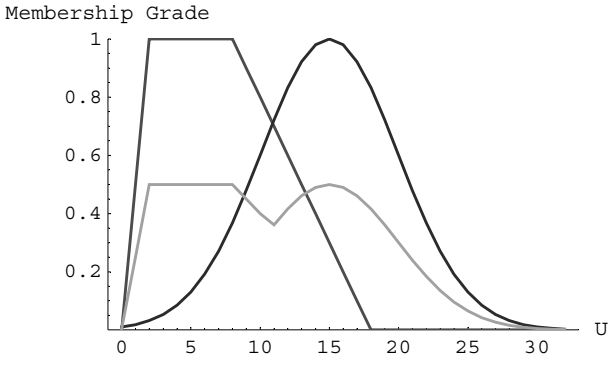
General aggregation operation.

`GeneralAggregator[`*fun*`, A1, A2, ... , An]` uses the function *fun* to combine the membership grades of *A1, A2, ... , An,* where the *A*'s are either fuzzy sets or fuzzy relations. This is the most general aggregate operation and virtually every other aggregation operation can be expressed in its terms. Here we demonstrate this function by first creating a function for combining the membership grades of the elements and then calling `GeneralAggregator` with this function and our two fuzzy sets.

```
In[32]:= MyAgg[x_, y_] := 1/2 Max[x, y]

In[33]:= GA = GeneralAggregator[MyAgg, FS1, FS2]

Out[33]= FuzzySet[{{0, 0.00506711}, {1, 1/4}, {2, 1/2}, {3, 1/2}, {4, 1/2}, {5, 1/2}, {6, 1/2},
           {7, 1/2}, {8, 1/2}, {9, 9/20}, {10, 2/5}, {11, 0.360711}, {12, 0.416104},
           {13, 0.460805}, {14, 0.489899}, {15, 0.5}, {16, 0.489899}, {17, 0.460805},
           {18, 0.416104}, {19, 0.360711}, {20, 0.300187}, {21, 0.239826}, {22, 0.18394},
           {23, 0.135434}, {24, 0.0957314}, {25, 0.0649613}, {26, 0.0423183},
           {27, 0.0264653}, {28, 0.015889}, {29, 0.00915782}, {30, 0.00506711},
           {31, 0.00269155}, {32, 0.00137252}}, UniversalSpace → {0, 32, 1}]
```

To see how our new aggregator behaves, we look at a plot of the results from the previous calculation.

```
In[34]:= FuzzyPlot[FS1, FS2, GA, PlotJoined → True];
```



You can use a built-in function `SetOptions` to restore the default setting for the `FuzzySet` object.

```
In[35]:= SetOptions[FuzzySet, UniversalSpace → {0, 20, 1}]

Out[35]= {UniversalSpace → {0, 20, 1}}
```

5 Fuzzy Set Visualization

5.1 Introduction

Fuzzy Logic package provides a number of functions that can be used to visualize fuzzy sets. In this chapter, we demonstrate these functions and the options associated with each function.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

5.2 Visualization Functions

To demonstrate the various fuzzy set graphing functions, we must first create some fuzzy sets to view. We use some of the fuzzy set creation functions from the *Fuzzy Logic* package to accomplish this. The following functions create a trapezoidal fuzzy set, a Gaussian fuzzy set, and a crisp set in that order.

```
In[2]:= FS1 = FuzzyTrapezoid[1, 6, 12, 17, 0.9];

In[3]:= FS2 = FuzzyGaussian[11, 5];

In[4]:= FS3 = FuzzyTrapezoid[4, 4, 12, 12];
```

FuzzyPlot[*A1*, *A2*, ... , *An*]	plot fuzzy sets *A1*, *A2*, ... , *An*
FuzzyGraph[A]	plot a fuzzy graph for a set of control rules A

Functions to visualize fuzzy sets.

FuzzyPlot[*A1*, *A2*, ... , *An*] plots fuzzy sets *A1*, *A2*, ... , *An*. This function is only for use with fuzzy sets, and the fuzzy sets it graphs must by defined on the same universal space.

option name	*default value*	
`ShowDots`	`False`	plot a dot at the top of each vertical line when this option is set to `True`
`PlotJoined`	`False`	plot a continuous line when this option is set to `True`
`Crisp`	`False`	plot a crisp set when this option is set to `True`

Options for `FuzzyPlot`.

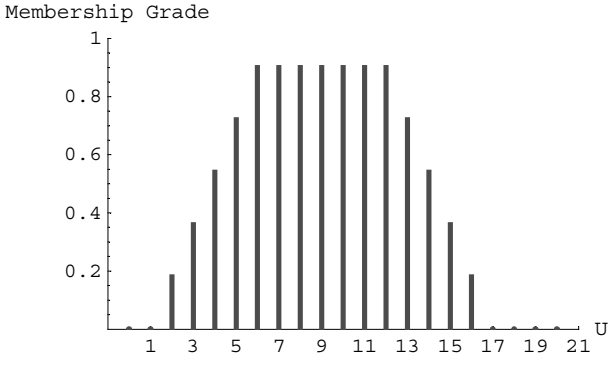
There are numerous options for this function, which we will demonstrate in the following examples.

If the `FuzzyPlot` function is called with one or more fuzzy sets and no options, the function will return a plot of vertical lines with heights representing the membership grades of the corresponding elements.

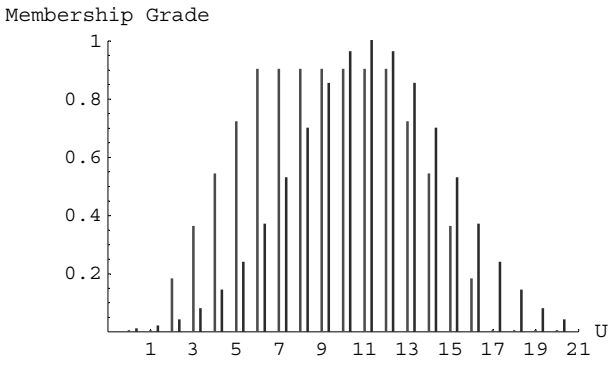
When more than one fuzzy set is graphed at a time, each successive fuzzy set is graphed in a new color and shifted slightly to the right to avoid overwriting a previous fuzzy set. Therefore, even though a line representing the membership grade is offset to the right of the discrete point on the graph, the line corresponds to the discrete point to the left of the plotted line.

Now let us look at some examples.

```
In[5]:= FuzzyPlot[FS1];
```



```
In[6]:= FuzzyPlot[FS1, FS2];
```

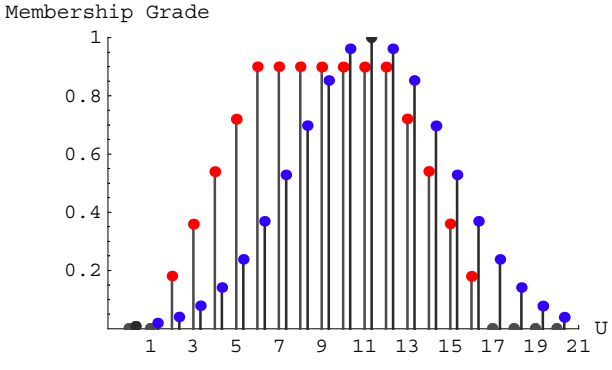


To resize a graph, click the graph and drag one of the dark squares to a new position.

When plotting more than one fuzzy set on the same graph, the `FuzzyPlot` command uses the following order of colors: red, blue, green, orange, purple, turquoise, yellow, dark pink, light blue, and yellow-green. For more than ten fuzzy sets plotted on the same graph, the color pattern above repeats, so the eleventh fuzzy set will again be red.

As more and more fuzzy sets are plotted on the same graph, the graph becomes more and more cluttered, and the fuzzy sets become increasingly difficult to distinguish from one another. Setting the `ShowDots` option to `True` causes a dot to be placed at the top of each of the vertical lines representing the membership grades. This option sometimes makes it easier to distinguish fuzzy sets in a cluttered graph.

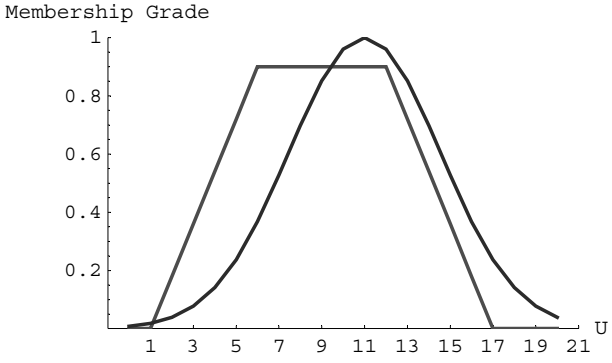
```
In[7]:= FuzzyPlot[FS1, FS2, ShowDots → True];
```



Even with the `ShowDots` option set to `True`, some graphs with multiple fuzzy sets still become too cluttered to easily interpret. Furthermore, if the universal space of the fuzzy sets you are plotting is large, the vertical line-plotting method does not work well. If this is the case, setting the `PlotJoined` option to `True`

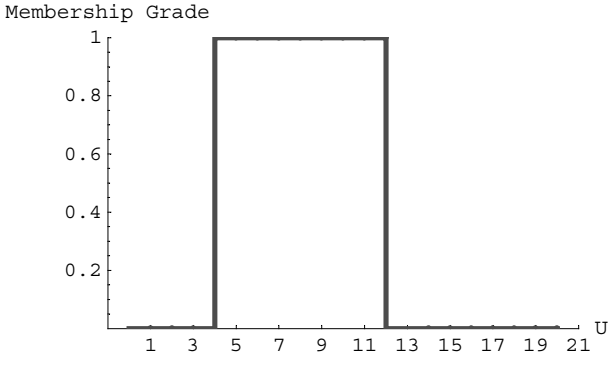
is often your best bet. This will cause the fuzzy sets to be plotted as a continuous line. A linear interpolation is used for the points between the integer elements, and successive fuzzy sets are not offset to the right as is the case for the discrete plots talked about earlier in this section.

```
In[8]:= FuzzyPlot[FS1, FS2, PlotJoined → True];
```



It is often instructive to work with fuzzy sets along with crisp or classical sets. Therefore, Fuzzy Logic package includes a graphing option that allows crisp sets to be plotted in an accurate manner. To use the crisp plotting option, set the `Crisp` option to `True`. We demonstrate this option by plotting a fuzzy set that is also a crisp set (all membership grades are either 1 or 0).

```
In[9]:= FuzzyPlot[FS3, Crisp → True];
```

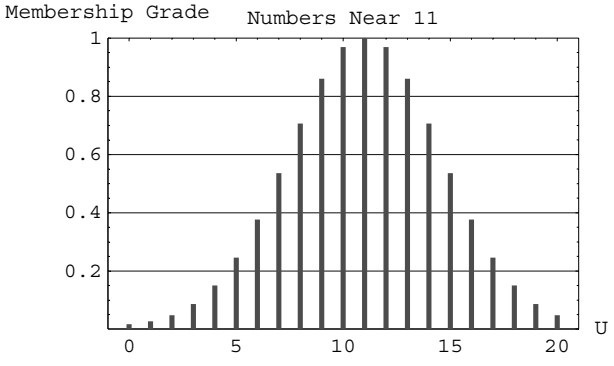


In addition to the options mentioned above, the `FuzzyPlot` function also accepts any of the options that *Mathematica*'s standard `Plot` function accepts. If the default options for the fuzzy plotting functions do not produce a readable graph, or if you want the graphs presented in a different way, these additional options can be used. Here is a list of the options for *Mathematica*'s Plot function.

```
AspectRatio      DisplayFunction     PlotDivision
Axes             Epilog              PlotLabel
AxesLabel        FormatType          PlotPoints
AxesOrigin       Frame               PlotRange
AxesStyle        FrameLabel          PlotRegion
Background       FrameStyle          PlotStyle
ColorOutput      FrameTicks          Prolog
Compiled         GridLines           RotateLabel
DefaultColor     ImageSize           TextStyle
DefaultFont      MaxBend             Ticks
```

Here is one example that uses some of the standard *Mathematica* plotting options.

```
In[10]:= FuzzyPlot[FS2, Frame → True,
           GridLines → {None, Automatic}, PlotLabel → "Numbers Near 11"];
```

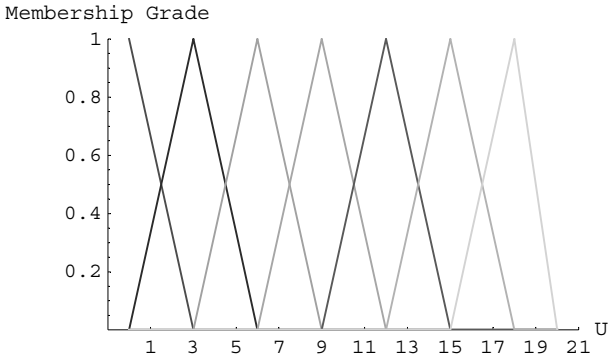


Another visualization function in the *Fuzzy Logic* package is a fuzzy graph. A fuzzy graph can be used to give an idea of what a set of fuzzy rules look like.

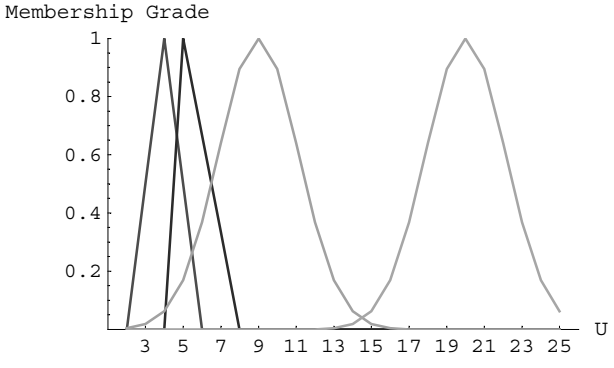
Let's look at an example to see how the `FuzzyGraph` function works. The first thing we need is a set of rules that relate fuzzy inputs to fuzzy outputs. We start by creating the fuzzy sets that are used to represent the input and output variables.

```
In[11]:= SetOptions[FuzzySet, UniversalSpace → {0, 20, 1}];

In[12]:= Input1 =
           {Tiny, VerySmall, Small, Medium, Big, VeryBig, Enormous} = CreateFuzzySets[7];
```

```
In[13]:= FuzzyPlot[Input1, PlotJoined → True];
```




```
In[14]:= SetOptions[FuzzySet, UniversalSpace → {2, 25, 1}];

In[15]:= Output1 = {VeryLow, Low, Middle, High} =
           {FuzzyTrapezoid[2, 4, 4, 6], FuzzyTrapezoid[4, 5, 5, 8],
            FuzzyGaussian[9, 3, ChopValue → .001], FuzzyGaussian[20, 3, ChopValue → .001]};

In[16]:= FuzzyPlot[Output1, PlotJoined → True];
```

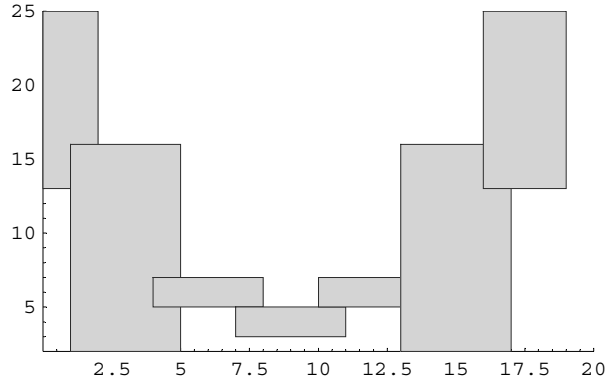


Next we create a set of fuzzy rules relating the inputs to the outputs.

```
In[17]:= Rules1 = {{Tiny, High}, {VerySmall, Middle}, {Small, Low},
           {Medium, VeryLow}, {Big, Low}, {VeryBig, Middle}, {Enormous, High}};
```

Now we can use the fuzzy graph function to get a general idea of what the system we created will look like. An expression of the form `{Tiny, High}` is referred to as a Cartesian granule. In this sense, a fuzzy graph may be viewed as a disjunction of Cartesian granules.

```
In[18]:= FuzzyGraph[Rules1];
```



In this case, we see that the rules we created map to a function that is parabolic in shape. In essence, a fuzzy graph serves as an approximation to a function, which is described in words as a collection of fuzzy *if-then* rules.

6 Fuzzy Relation Visualization

6.1 Introduction

Fuzzy Logic package provides a number of functions that can be used to visualize fuzzy relations. In this chapter, we demonstrate these functions and the options associated with each function.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

6.2 Visualization Functions

To demonstrate the various functions for visualizing fuzzy relations, we need first to create some fuzzy relations. We use the `FuzzyTrapezoid` function here to create two fuzzy relations.

```
In[2]:= Rel1 = FuzzyTrapezoid[{1, 3, 5, 7}, {1, 4, 6, 8}, UniversalSpace → {{1, 7}, {1, 8}}];
```

```
In[3]:= Rel2 = FuzzyTrapezoid[{1, 1, 4, 7}, {1, 1, 5, 8}, UniversalSpace → {{1, 7}, {1, 8}}];
```

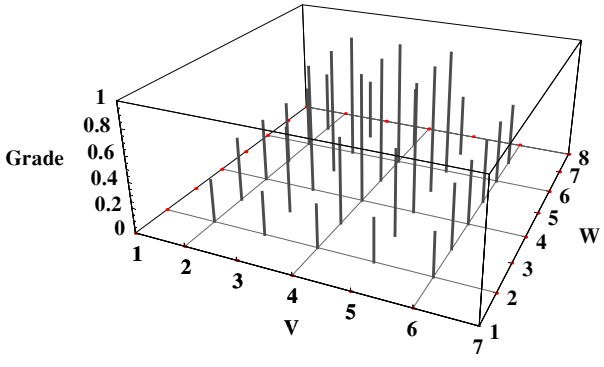
`FuzzyPlot3D[`*A1*`, `*A2*`, ... , `*An*`]`	plot the fuzzy relations *A1*, *A2*, ... , *An*, as a collection of vertical lines with heights representing the membership grades of the elements
FuzzySurfacePlot[*A1*, *A2*, ... , *An*]	plot a surface plot of fuzzy relations *A1*, *A2*, ... , *An*
`ToMembershipMatrix[`*A*`]`	return the membership matrix of fuzzy relation *A*

Functions to visualize fuzzy relations.

Fuzzy Plot 3D

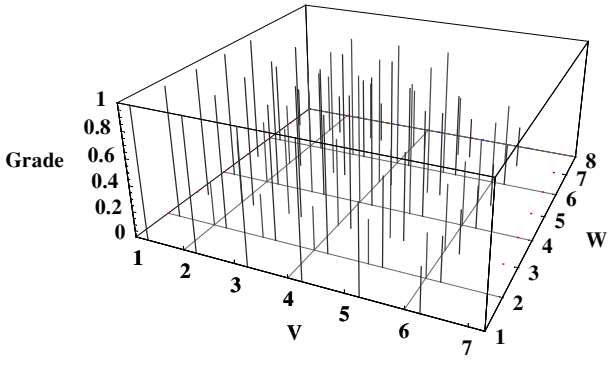
`FuzzyPlot3D[`*A1*`,` *A2*`, ... ,` *An*`,` *opts*`]` plots the fuzzy relations *A1*, *A2*, ... , *An* as a collection of vertical lines with heights representing the membership grades of the elements. This function is only for fuzzy relations, and all of the fuzzy relations *A1*, *A2*, ... , *An* must be defined on the same universal space. Here we demonstrate the function with our two fuzzy relations, `Rel1` and `Rel2`.

```
In[4]:= FuzzyPlot3D[Rel1];
```



As with fuzzy sets, when multiple fuzzy relations are shown on the same graph, the graph of each successive fuzzy relation is shifted a little to the right along the V axis. Even though there is a shift in the location of each line, the line still corresponds to the discrete point to the left of the plotted line. We can see this in the next example.

```
In[5]:= FuzzyPlot3D[Rel1, Rel2];
```

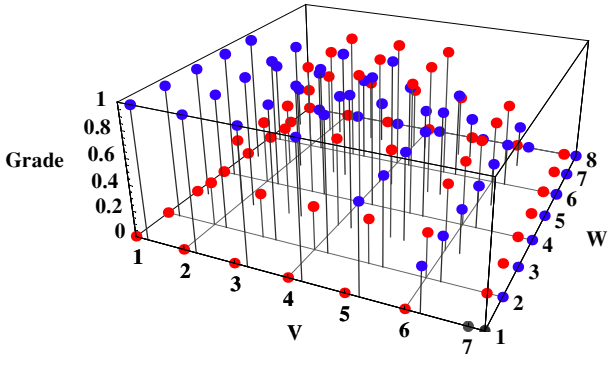


option name	*default value*	
`ShowDots`	`False`	plot a dot at the top of each vertical line when this option is set to `True`

Option for `FuzzyPlot3D`.

As with fuzzy sets, graphs of multiple fuzzy relations tend to become cluttered, and it becomes difficult to visualize the different fuzzy relations. One option, which may help clear up the graph, is the `ShowDots` option. When set to `True`, this option plots a dot at the top of each vertical line. This is shown in the following example.

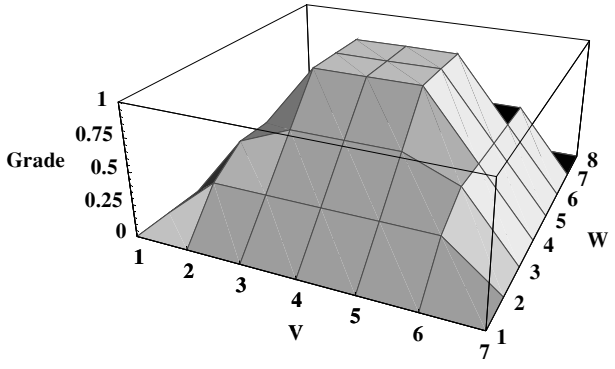
```
In[6]:= FuzzyPlot3D[Rel1, Rel2, ShowDots → True];
```



Fuzzy Surface Plot

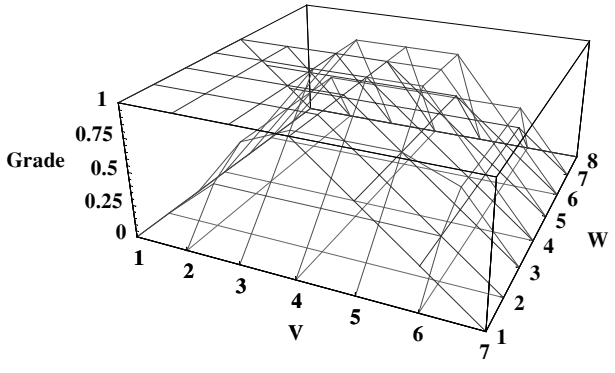
`FuzzySurfacePlot[`*A1*`,` *A2*`, ... ,` *An*`,` *opts*`]` displays a surface plot of fuzzy relations *A1*, *A2*, ... , *An*. The discrete plots shown earlier are fairly messy for plotting large fuzzy relations or for plotting multiple fuzzy relations. An alternative graphing solution is to plot fuzzy relations as surfaces. Let's replot the first fuzzy relation as a surface now with the `FuzzySurfacePlot` function.

```
In[7]:= FuzzySurfacePlot[Rel1];
```



The `FuzzySurfacePlot` function plots a fuzzy relation as a solid object. Therefore, plotting multiple fuzzy relations on the same surface plot causes some fuzzy relations to block out regions of other fuzzy relations. To get around this problem, the `FuzzySurfacePlot` function comes with an option that plots the surfaces as meshes rather than solid objects. Setting the `HideSurfaces` option to `True` produces a mesh plot. To demonstrate, we will replot the first fuzzy relation with the second fuzzy relation, this time with the `HideSurfaces` option set to `True`.

```
In[8]:= FuzzySurfacePlot[Rel1, Rel2, HideSurfaces → True];
```

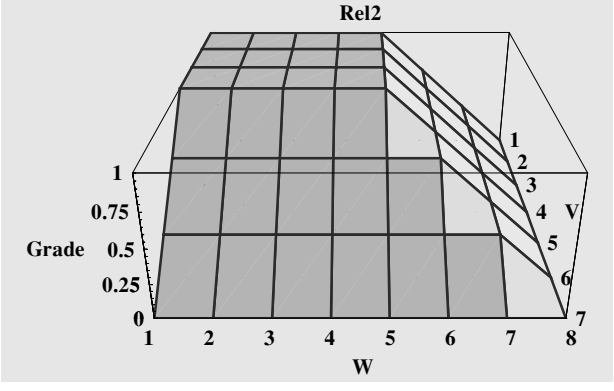


In addition to the options described in this chapter, the `FuzzyPlot3D` and `FuzzySurfacePlot` functions also accept any of the options that *Mathematica*'s standard `Plot3D` function accepts. Here is a list of these additional options.

```
AmbientLight    Compiled          PlotRange
AspectRatio     DefaultColor      PlotRegion
Axes            DefaultFont       Plot3Matrix
AxesEdge        DisplayFunction   Prolog
AxesLabel       Epilog            Shading
AxesStyle       FaceGrids         SphericalRegion
Background      HiddenSurface     Ticks
Boxed           Lighting          ViewCenter
BoxRatios       LightSources      ViewPoint
BoxStyle        MeshStyle         ViewVertical
ClipFill        PlotLabel
ColorOutput     PlotPoints
```

For more information on these options, see *The Mathematica Book* by Stephen Wolfram. We demonstrate a few of these options in the following example.

```
In[9]:= FuzzySurfacePlot[Rel2, ViewPoint → {2, 0, 1},
          AxesEdge → {{1, -1}, {1, -1}, {1, -1}}, MeshStyle → {Thickness[.006], Hue[.7]},
          Background → GrayLevel[.9], PlotLabel → "Rel2"];
```



Membership Matrix

One last technique for visualizing a fuzzy relation is to look at its membership matrix. `ToMembership Matrix[A]` displays the membership matrix of fuzzy relation *A*. A membership matrix is a matrix that shows the membership grades of all the elements of a fuzzy relation. This is a convenient way to view fuzzy relations. We demonstrate this function on our first fuzzy relation.

```
In[10]:= ToMembershipMatrix[Rel1] // MatrixForm
```

Out[10]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & \frac{2}{3} & 1 & 1 & 1 & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & \frac{2}{3} & 1 & 1 & 1 & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & \frac{2}{3} & 1 & 1 & 1 & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

7 Compositions

7.1 Introduction

The *Fuzzy Logic* package provides a few different ways to perform compositions between two fuzzy relations. In this chapter, we introduce and demonstrate these functions and the options associated with them.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

7.2 Composition Function

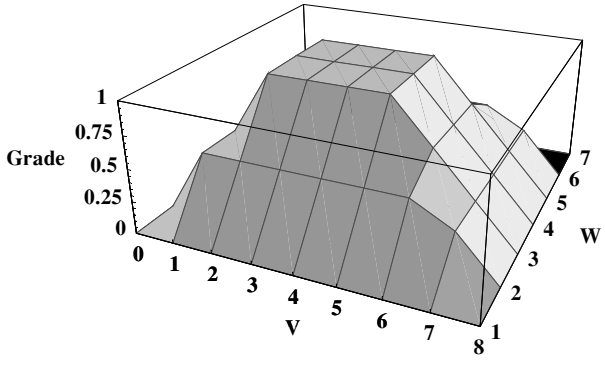
To demonstrate the various composition functions, we must first create some fuzzy relations. We use the `FuzzyTrapezoid` function here to create two fuzzy relations. We must be careful to create two fuzzy relations that can be combined by a composition operation. To take the composition of two fuzzy relations, the second range of the first fuzzy relation's universal space must be the same as the first range in the second fuzzy relation's universal space. This is a necessary condition to perform a composition of two fuzzy relations.

```
In[2]:= Rel1 = FuzzyTrapezoid[{0, 2, 5, 8}, {1, 3, 5, 7}, 1, UniversalSpace → {{0, 8}, {1, 7}}];
```

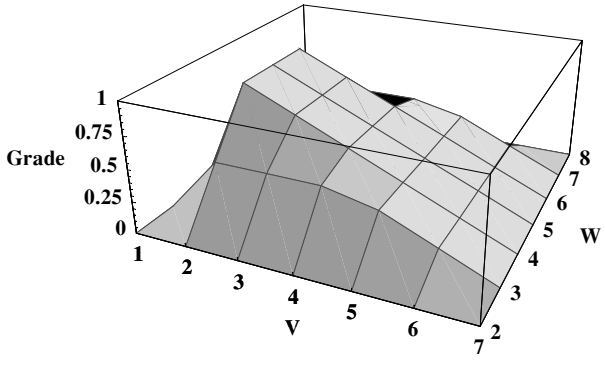
```
In[3]:= Rel2 =
          FuzzyTrapezoid[{1, 2, 2, 7}, {2, 4, 6, 8}, 0.9, UniversalSpace → {{1, 7}, {2, 8}}];
```

We can view the two fuzzy relations we created using one of the *Fuzzy Logic* package's plotting functions. Here we use the `FuzzySurfacePlot` function to look at the fuzzy relations.

```
In[4]:= FuzzySurfacePlot[Rel1];
```




```
In[5]:= FuzzySurfacePlot[Rel2];
```



`Composition[`*A*`, `*B*`]`	return a fuzzy relation that is the result of performing a composition between fuzzy relations *A* and *B*

Function `Composition`.

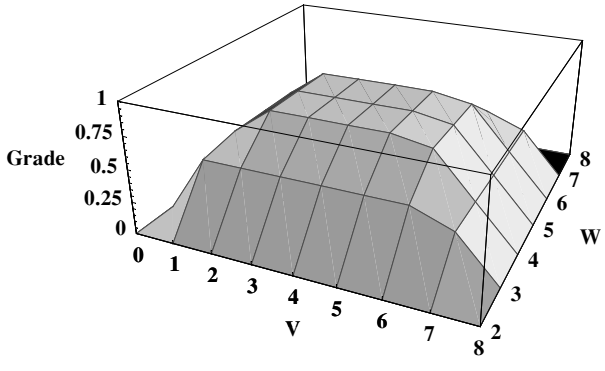
`Composition[`*A*`, `*B*`, `*opts*`]` returns a fuzzy relation that is the result of performing a composition between fuzzy relations *A* and *B*. There is a `Type` option with this function that defaults to a `MaxMin` composition. Other types available are `MaxProduct` and `MaxStar[`*func*`]`, which allows you to define your own function for combining the relations.

option name	*default value*	
`Type`	`MaxMin`	an option for fuzzy `Composition` with admissible values `MaxMin`, `MaxProduct`, and `MaxStar`

Option for fuzzy set `Composition`.

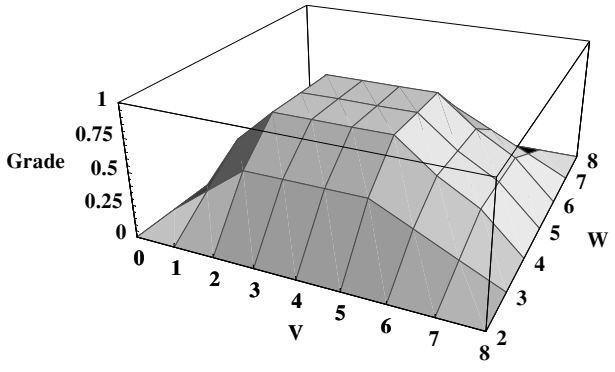
The `MaxMin` composition performs an operation similar to matrix multiplication. The membership grades of the new fuzzy relation created by this operation are found by combining the membership grades from each row of the first fuzzy relation with the membership grades of each column of the second fuzzy relation. In the `MaxMin` composition, the membership grades are combined by finding the maximum of the minimums of the corresponding membership grades in the rows of the first fuzzy relation and the columns of the second fuzzy relation. Let's see how this function performs.

```
In[6]:= Com11 = Composition[Rel1, Rel2];

In[7]:= FuzzySurfacePlot[Com11];
```



In the `MaxProduct` composition, the membership grades are found by taking the maximum of the products of corresponding membership grades in the rows of the first fuzzy relation and the columns of the second fuzzy relation. Let's see how this function behaves.

```
In[8]:= Com22 = Composition[Rel1, Rel2, Type → MaxProduct];

In[9]:= FuzzySurfacePlot[Com22];
```



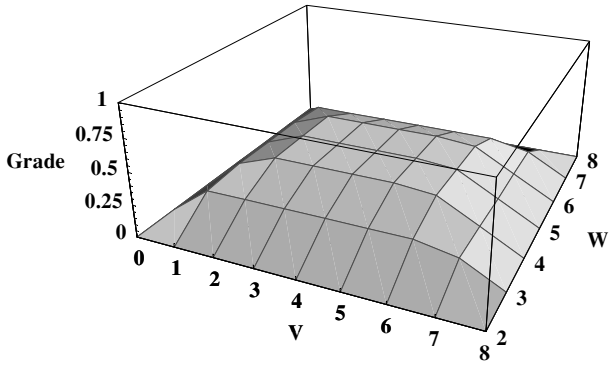
For the `MaxStar` composition, the membership grades are found by taking the maximum of the result of applying a user-defined function to corresponding membership grades in the rows of the first fuzzy relation and the columns of the second fuzzy relation. Let's see how to use this function.

First we create a function of our own for combining membership grades. We create a function that combines membership grades by taking half of the `Min` of the membership grades.

```
In[10]:= MyFun[x__] := Min[x]/2
```

Now we call the `Composition` function with `Type` set to `MaxStar` with our function, `MyFun`, as a parameter.

```
In[11]:= Com33 = Composition[Rel1, Rel2, Type → MaxStar[MyFun]];

In[12]:= FuzzySurfacePlot[Com33];
```



8 Fuzzy Inferencing

8.1 Introduction

Fuzzy Logic package provides three different functions for performing fuzzy inferencing. The inferencing functions are a little more involved than the other functions in this package, and consequently they need a little more setup. In this chapter, we demonstrate how to set up and use the inferencing functions.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

8.2 Inference Functions

`CompositionBasedInference[`*A*`,` *B*`]`
: return a fuzzy set that is the result of performing a maxmin composition between a fuzzy set *A* and a fuzzy relation *B*

`RuleBasedInference[{`*A1, ... , An*`}, {`*B1, ... , Bm*`}, {`*C1, ... , Ck*`}, {{`*Ax*`,` *Bx*`,` *Cx*`}, ... }, `*a*`,` *b*`]`
: return a fuzzy set that is the result of performing rule based inference for two−input/single−output systems, where {*A1, ... , An*} and {*B1, ... , Bm*} represent linguistic input variables, {*C1, ... ,Ck*} is the linguistic output variable, rules are given in a list of triples like {*Ax, Bx, Cx*}, and the crisp values for the inputs are *a* and *b*

`RuleBasedInference[` {{*A1, ... , An*}, ... , {*S1, ... , Sp*}}, {*Y1, ... ,Yk*}, {*Ax, ... , Sx, Yx*}, {*a, ... , s*}]	return a fuzzy set that is the result of performing rule based inference for multiple–input/single–output systems, where {{*A1, ... , An*}, ... , {*S1, ... , Sp*}} represent linguistic input variables,{*Y1, ... ,Yk*} is the linguistic output variable, rules are given in a list like {*Ax, ... , Sx, Yx*} and the crisp values for the inputs are given in a list {*a, ... ,s*}

Functions for fuzzy inferencing.

option name	*default value*	
`Type`	`Mamdani`	set the type of rule based fuzzy inference; the admissible values are `Mamdani`, `Model`, and `Scaled`

Option for `RuleBasedInference`.

8.3 Composition-Based Inference

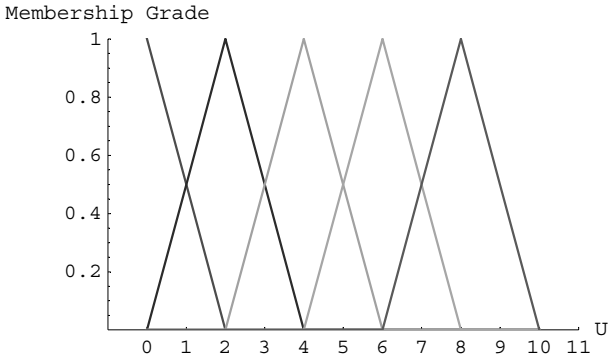
The `CompositionBasedInference` function works for single-input/single-output systems. To use the `CompositionBasedInference` function, a fuzzy relation must be created which models a system's input-output response. The fuzzy inference function then takes a fuzzy set as input and performs a composition to arrive at the output. What follows is a description of what is needed to perform a `CompositionBasedInference` and a demonstration of the inferencing operation.

Defining Inputs

To create the fuzzy relation needed to perform a `CompositionBasedInference`, we first need to define our collection of input fuzzy sets. Each fuzzy set used to define the collection of input fuzzy sets can be created individually using one of the techniques discussed in the Chapter 1 Creating Fuzzy Sets, or the entire collection of fuzzy sets can be created all at once using the `CreateFuzzySets` function as shown here. We look at our input fuzzy sets with the `FuzzyPlot` function.

```
In[2]:= INPUT = {NegBig, NegSmall, Zero, PosSmall, PosBig} =
          CreateFuzzySets[5, UniversalSpace → {0, 10}];
```

```
In[3]:= FuzzyPlot[INPUT, PlotJoined → True];
```

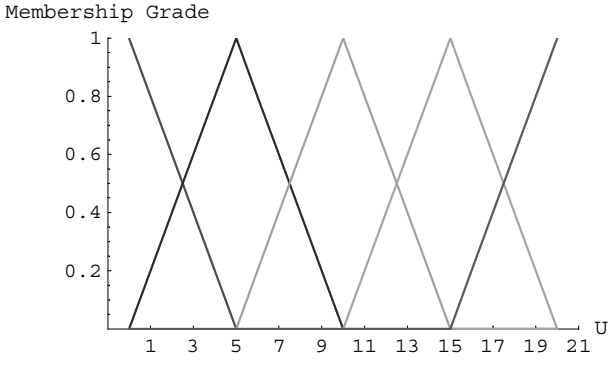


In this example, we have created five fuzzy sets which are named `NegBig`, `NegSmall`, `Zero`, `PosSmall`, and `PosBig`. Notice that these fuzzy sets are labeled with linguistic terms that describe the input's universal space.

Defining Outputs

With the input membership functions defined, we must create a set of output fuzzy sets. We again use the `CreateFuzzySets` function for convenience.

```
In[4]:= OUTPUT = {NBO, NSO, ZO, PSO, PBO} = CreateFuzzySets[5];

In[5]:= FuzzyPlot[OUTPUT, PlotJoined → True];
```



Defining Rules

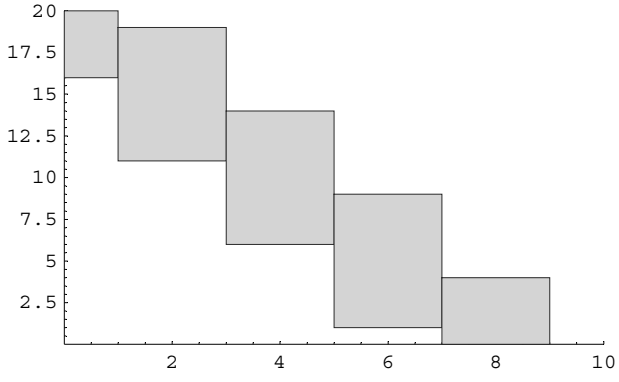
The final item we need, to create the fuzzy relation that will define our system model, is a set of rules relating the input conditions to the output responses. These rules are formed as a list of *if-then* pairs, where the input condition is the first element of the pair, and the output response is the second element. Here is an example of a set of rules for the input and output we defined above.

```
In[6]:= TheRules =
          {{NegBig, PBO}, {NegSmall, PSO}, {Zero, ZO}, {PosSmall, NSO}, {PosBig, NBO}};
```

The rules shown above constitute an *if-then* pair. For example, the first rule, {`NegBig`, `PBO`}, would be translated as follows:

> if the input is `NegBig` (Negative Big), then the output should be `PBO` (Positive Big).

```
In[7]:= FuzzyGraph[TheRules];
```

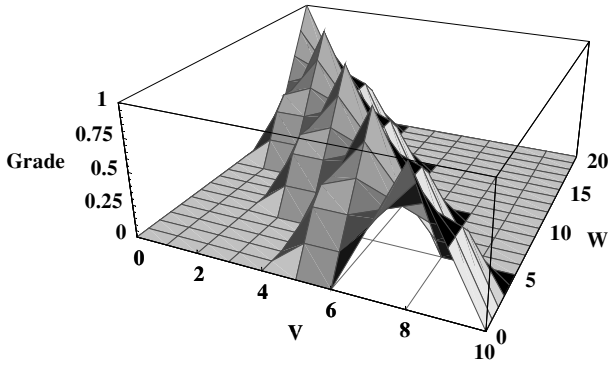


In general, there can be a rule corresponding to each input membership function, but there should not be more than one rule for any one input condition.

Building the Model

To create the fuzzy relation that will serve as our fuzzy model, we call the `BuildModel` function with a set of rules. The `BuildModel` function creates a fuzzy relation that can serve as a system model. This function works by applying the `SetsToRelation` function to each of the rule pairs and performing a union with the resulting fuzzy relations to form one universal relation describing the set of rules. Let's create a fuzzy model to represent our system. We can view the fuzzy model we created with the `FuzzySurfacePlot` function.

```
In[8]:= FuzzyModel = BuildModel[TheRules];

In[9]:= FuzzySurfacePlot[FuzzyModel];
```



Note that the input and output fuzzy sets to which the rules refer must have been created prior to using the `BuildModel` function.

Inferencing

To perform a fuzzy inference using the model just created, we need to provide a fuzzy set input. The universal space of our input fuzzy set must be equal to the universal space of the input used to build the model. Here we create a fuzzy set that will serve as input to the inferencing function.

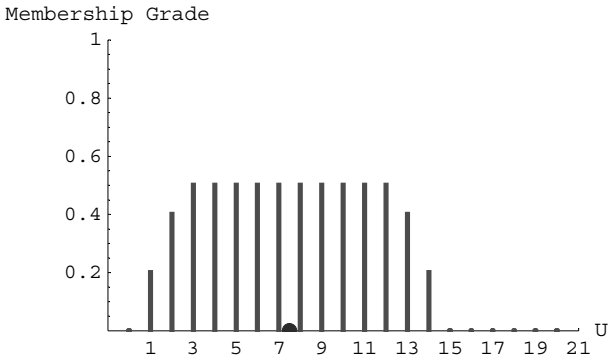
```
In[10]:= MyModelInp = FuzzyTrapezoid[5, 5, 5, 5, UniversalSpace → {0, 10}]

Out[10]= FuzzySet[{{5, 1}}, UniversalSpace → {0, 10, 1}]
```

Now we can use the CompositionBasedInference function to evaluate the model response to the input.

`CompositionBasedInference[`*A*`, `*B*`]` returns a fuzzy set that is the result of performing a `MaxMin` composition-based inference, where *A* is the input fuzzy set, and *B* is the model fuzzy relation. The output fuzzy set that can be defuzzified using either the `CenterOfArea` or `MeanOfMax` function. Here we use the `MeanOfMax` defuzzification to find a crisp output for our inferencing operation.

```
In[11]:= MeanOfMax[CompositionBasedInference[MyModelInp, FuzzyModel], ShowGraph -> True];
```

```
Mean of max is 7.5
```



For this example, we see that for an input fuzzy set centered around 5, we receive an output response of 7.5.

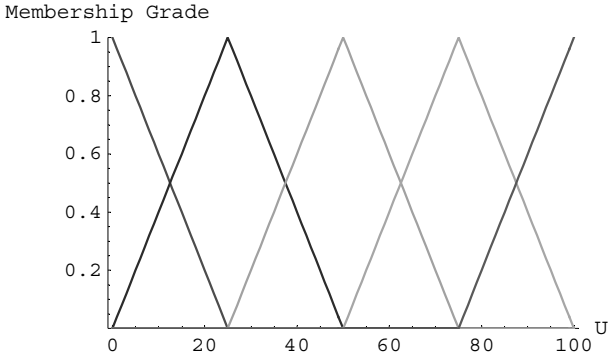
8.4 Rule-Based Inference

The `RuleBasedInference` functions work with two-input/one-output systems and with multiple-input/one-output systems. The following is a description of what is needed to perform a `RuleBasedInference` and a demonstration of the inferencing operation using a two-input/one-output system.

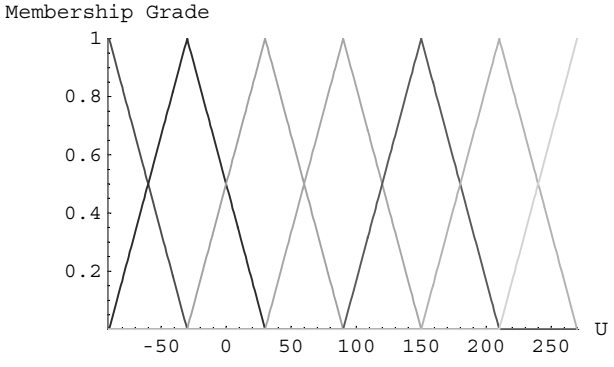
Defining Inputs

As with the `CompositionBasedInference`, the first thing we need to do to perform a `RuleBasedInference` is define our input fuzzy sets. Unlike the `CompositionBasedInference`, we need to create two sets of input fuzzy sets. We again use the `CreateFuzzySets` function to define our input fuzzy sets, and we use the `FuzzyPlot` function to look at the fuzzy sets.

```
In[12]:= FirstInput = {LE, LC, CE, RC, RI} = CreateFuzzySets[5, UniversalSpace → {0, 100}];

In[13]:= FuzzyPlot[FirstInput, PlotJoined → True];
```




```
In[14]:= SecondInput =
           {RB, RU, RV, VE, LV, LU, LB} = CreateFuzzySets[7, UniversalSpace → {-90, 270}];

In[15]:= FuzzyPlot[SecondInput, PlotJoined → True];
```

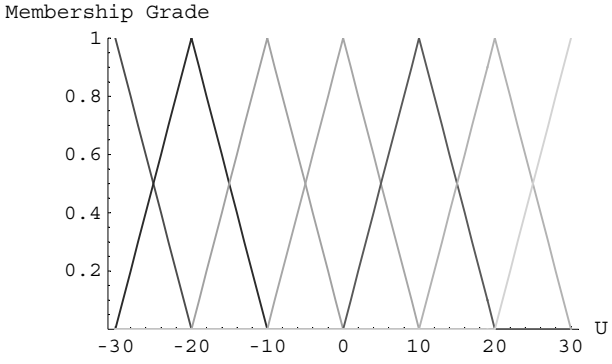


Notice how we divided our two inputs into different numbers of membership functions. The choice for the number of membership functions used to define an input is entirely up to the designer. Notice also that the names of the membership functions are different. It is important that all of the input and output membership functions have distinct names for the inferencing to work correctly.

Defining Outputs

Again we need to define a collection of output fuzzy sets. We define these just as we did the input, making sure that we use distinct names for the individual fuzzy sets.

```
In[16]:= TheOutput =
           {NB, NM, NS, ZE, PS, PM, PB} = CreateFuzzySets[7, UniversalSpace → {-30, 30}];

In[17]:= FuzzyPlot[TheOutput, PlotJoined → True];
```



Defining Rules

The final piece needed to perform a `RuleBasedInference` is a list of rules. The `RuleBasedInference` function expects the rules to be a list of triplets. The first two items in the triplet correspond to the input conditions, and the third item corresponds to the output. Here is an example.

```
In[18]:= ControlRules = {{LE, RB, PS}, {LC, RB, PM}, {CE, RB, PM}, {RC, RB, PB}, {RI, RB, PB},
           {LE, RU, NS}, {LC, RU, PS}, {CE, RU, PM}, {RC, RU, PB}, {RI, RU, PB}, {LE, RV, NM},
           {LC, RV, NS}, {CE, RV, PS}, {RC, RV, PM}, {RI, RV, PB}, {LE, VE, NM}, {LC, VE, NM},
           {CE, VE, ZE}, {RC, VE, PM}, {RI, VE, PM}, {LE, LV, NB}, {LC, LV, NM}, {CE, LV, NS},
           {RC, LV, PS}, {RI, LV, PM}, {LE, LU, NB}, {LC, LU, NB}, {CE, LU, NM}, {RC, LU, NS},
           {RI, LU, PS}, {LE, LB, NB}, {LC, LB, NB}, {CE, LB, NM}, {RC, LB, NM}, {RI, LB, NS}};
```

The list of rules we just created can again be interpreted as an *if-then* statement. For example, the first rule, {`LE`, `RB`, `PS`}, could be interpreted in the following manner:

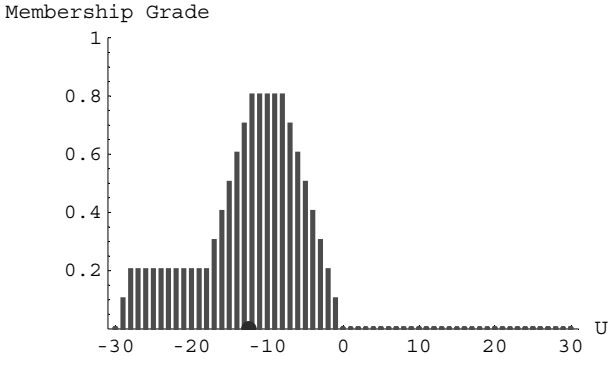
> if the `FirstInput` is `LE` and the `SecondInput` is `RB`, then `TheOutput` response should be `PS`.

The order of the inputs should be the same throughout the list because the order does matter when performing the fuzzy inference.

Inferencing

`RuleBasedInference[{A1, ... , An}, {B1, ... ,Bn}, {C1, ... ,Cn}, {{Ai, Bj, Ck}, ...}, a, b]` returns a fuzzy set that is the result of performing a rule based inference Mamdani (`MaxMin`) type, where *{A1, ... , An}* and *{B1, ... , Bn}* represent the collections of two input membership functions; *{C1, ... , Cn}* represent the collection of output membership functions; the *{Ai, Bj, Ck}* triplets represent rules relating the two inputs to the one output; and *a* and *b* represent the crisp inputs. Here we use the `CenterOfArea` defuzzification to come up with a crisp output. Crisp inputs of 70 and 210 give a crisp output of -12.4138.

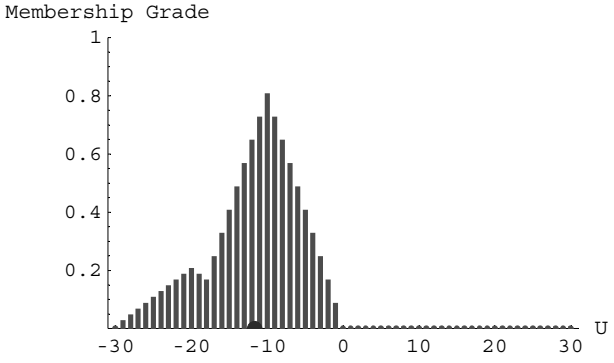
```
In[19]:= CenterOfArea[RuleBasedInference[FirstInput,
           SecondInput, TheOutput, ControlRules, 70, 210], ShowGraph -> True];

         Center of area is -12.4138.
```



The `RuleBasedInference` example shown in this section contains the specifications for a fuzzy logic controller, which can be used to back a truck up to a loading dock. To see a complete simulation of the truck-backer example, see the `FuzzyControl` notebook.

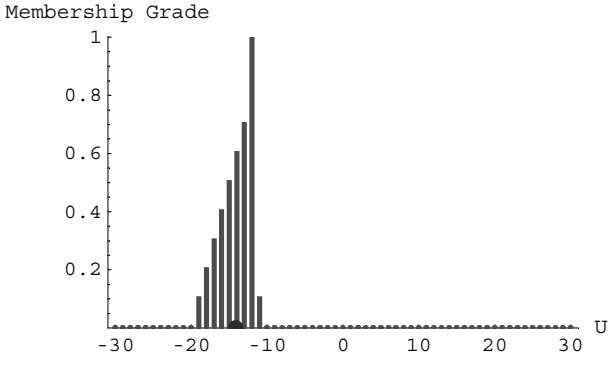
The scaled inference is like the Mamdani except instead of clipping the output membership functions, the output membership functions are scaled to have a height equivalent to the input fuzzification value. Here is an example.

```
In[20]:= CenterOfArea[RuleBasedInference[{FirstInput, SecondInput},
           TheOutput, ControlRules, {70, 210}, Type -> Scaled], ShowGraph -> True];

         Center of area is -11.6522.
```



The Model inference, instead of clipping the output membership functions, raises the membership function values to 1 for all elements of the output membership function greater than the input fuzzification value. Also, instead of Max-Union the Min-Intersection is taken for all of the output membership functions. Here is an example.

```
In[21]:= CenterOfArea[RuleBasedInference[{FirstInput, SecondInput},
           TheOutput, ControlRules, {70, 210}, Type → Goedel], ShowGraph -> True];

         Center of area is -14.1282.
```



9 Fuzzy Arithmetic

9.1 Introduction

Fuzzy Logic package provides a number of functions for performing fuzzy arithmetic. In this chapter, we will demonstrate these functions and the options associated with each function.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

The arithmetic operations described in this chapter are designed to work with triangular or trapezoidal fuzzy numbers. The arithmetic functions are a little different than the rest of the *Fuzzy Logic* package's functions in that they operate on lists of numbers that represent vertices of fuzzy sets, rather than fuzzy sets themselves. This makes working with these operations a little awkward, but we show a convenient way to work with these functions in this section. For fuzzy arithmetic operations that work on actual fuzzy sets, see Chapter 10 Discrete Fuzzy Arithmetic.

First we need to set up an appropriate universal space. Since we are working with fuzzy numbers, and since they can be positive or negative, our universal space should be semiotic around zero. The universal space should also be big enough so that the results of the arithmetic operations will still fall within the universal space. We change the default setting for universal space using the `SetOptions` command.

```
In[2]:= SetOptions[FuzzySet, UniversalSpace → {-30, 30}];
```

Now that the universal space is defined, we need some fuzzy numbers. As mentioned earlier, these operations work on lists of vertices, so we create two lists now, which represent two triangular fuzzy numbers.

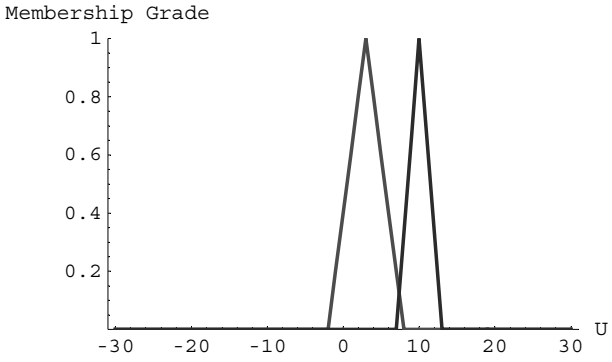
```
In[3]:= num1 := {-2, 3, 3, 8}
```

```
In[4]:= num2 := {7, 10, 10, 13}
```

We can convert the list of vertices, `num1` and `num2`, into actual fuzzy sets using the `FuzzyTrapezoid` function. You can look at the results with the `FuzzyPlot` function.

```
In[5]:= A1 = FuzzyTrapezoid[num1];
```

```
In[6]:= A2 = FuzzyTrapezoid[num2];

In[7]:= FuzzyPlot[A1, A2, PlotJoined → True];
```



9.2 Fuzzy Arithmetic Functions

`FuzzyPlus[{a1, b1, c1, d1}, {a2, b2, c2, d2}]`
return the sum of the fuzzy numbers represented by the two lists

`FuzzyMinus[{a1, b1, c1, d1}, {a2, b2, c2, d2}]`
return the fuzzy difference between the two fuzzy numbers represented by the two lists

`FuzzyConstantTimes[{a, b, c, d}, k]`
return the fuzzy number that is the result of the multiplication of constant k by the fuzzy number represented by the list {*a*, *b*, *c*, *d*}

`FuzzyImage[{a, b, c, d}]`
return the image of the fuzzy number represented by the list {*a*, *b*, *c*, *d*}

`FuzzyMultiply[{a1, b1, c1, d1}, {a2, b2, c2, d2}]`	return the product of the fuzzy numbers represented by the two lists
`FuzzyDivide[{a1, b1, c1, d1}, {a2, b2, c2, d2}]`	return the division of the fuzzy numbers represented by the two lists

Fuzzy arithmetic operations.

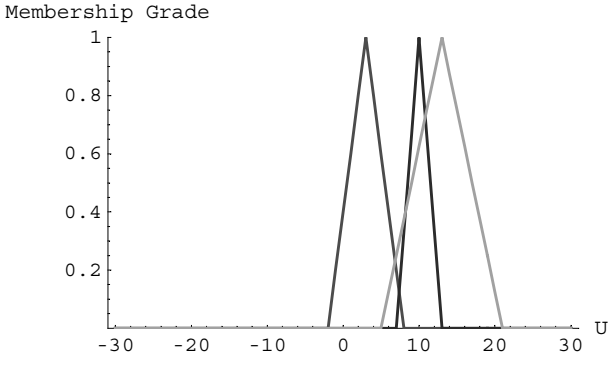
Fuzzy Addition

`FuzzyPlus[{a1, b1, c1, d1}, {a2, b2, c2, d2}]` returns the sum of the fuzzy numbers represented by the two lists. The fuzzy sum is returned as an unevaluated `FuzzyTrapezoid`. To evaluate the `FuzzyTrapezoid`, use the `ReleaseHold` function.

```
In[8]:= Sum1 = FuzzyPlus[num1, num2]

Out[8]= FuzzyTrapezoid[5, 13, 13, 21, UniversalSpace → {-30, 30, 1}]
```

The result of the fuzzy addition of the two fuzzy numbers is an unevaluated fuzzy trapezoid. To evaluate the result as a normal fuzzy set, you must use *Mathematica*'s `ReleaseHold` function. Below we plot our two original fuzzy sets with their sum. Notice the use of the `ReleaseHold` function.

```
In[9]:= FuzzyPlot[A1, A2, ReleaseHold[Sum1], PlotJoined → True];
```



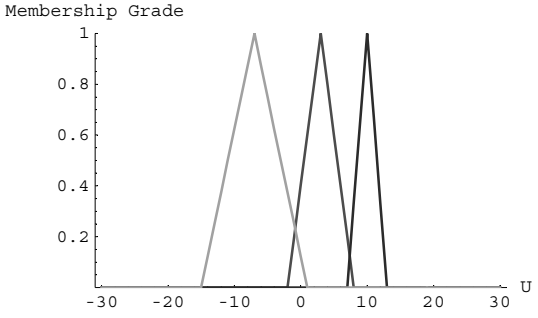
Fuzzy Subtraction

`FuzzyMinus[{a1, b1, c1, d1}, {a2, b2, c2, d2}]` returns the fuzzy difference between the two fuzzy numbers represented by the two lists. The fuzzy difference is returned as an unevaluated `FuzzyTrapezoid`. To evaluate the `FuzzyTrapezoid`, use `ReleaseHold`.

```
In[10]:= Diff1 = FuzzyMinus[num1, num2]

Out[10]= FuzzyTrapezoid[-15, -7, -7, 1, UniversalSpace → {-30, 30, 1}]
```

The fuzzy difference between the two fuzzy numbers is returned in the same form as the fuzzy sum. Here we plot the original fuzzy sets with their fuzzy difference.

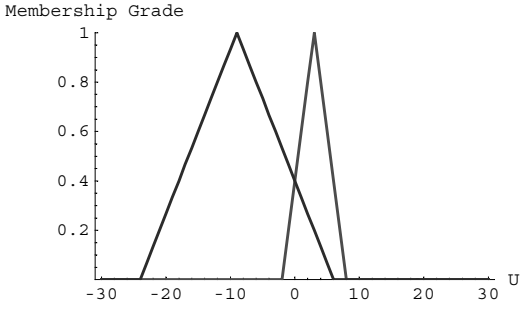
```
In[11]:= FuzzyPlot[A1, A2, ReleaseHold[Diff1], PlotJoined → True];
```



Multiplication by a Constant

`FuzzyConstantTimes[{a, b, c, d}, k]` returns the fuzzy number that is the result of the multiplication of constant *k* by the fuzzy number represented by the list, *{a, b, c, d}*. The result is returned as an unevaluated `FuzzyTrapezoid`. To evaluate the `FuzzyTrapezoid`, use `ReleaseHold`.

```
In[12]:= Prod1 = FuzzyConstantTimes[num1, -3]

Out[12]= FuzzyTrapezoid[-24, -9, -9, 6, UniversalSpace → {-30, 30, 1}]

In[13]:= FuzzyPlot[A1, ReleaseHold[Prod1], PlotJoined → True];
```



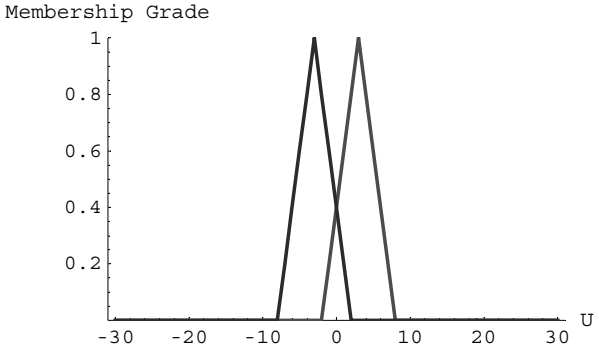
Fuzzy Image

`FuzzyImage[{a, b, c, d}]` returns the image of the fuzzy number represented by the list *{a, b, c, d}*. The result is returned as an unevaluated `FuzzyTrapezoid`. To evaluate the `FuzzyTrapezoid`, use `ReleaseHold`.

```
In[14]:= Im1 = FuzzyImage[num1]

Out[14]= FuzzyTrapezoid[-8, -3, -3, 2, UniversalSpace → {-30, 30, 1}]
```

The image of a fuzzy set is a new fuzzy set that is a mirror image of the first one flipped around zero. Here is a plot of our first fuzzy set and its image.

```
In[15]:= FuzzyPlot[A1, ReleaseHold[Im1], PlotJoined → True];
```



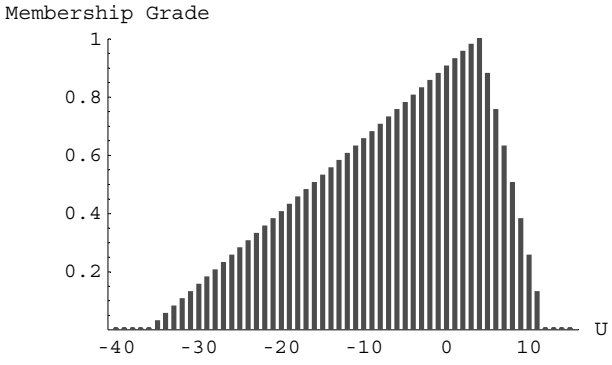
Fuzzy Multiplication

`FuzzyMultiply[{a1, b1, c1, d1}, {a2, b2, c2, d2}]` returns the product of the fuzzy numbers represented by the two lists. The fuzzy product is returned as an unevaluated `FuzzyTrapezoid`. To evaluate the `FuzzyTrapezoid`, use the `ReleaseHold` function.

```
In[16]:= Multi1 = FuzzyMultiply[{-9, 2, 2, 3}, {1, 2, 2, 4}, UniversalSpace -> {-40, 15, 1}]

Out[16]= FuzzyTrapezoid[-36, 4, 4, 12, UniversalSpace → {-40, 15, 1}]
```

The result of the fuzzy multiplication of the two fuzzy numbers is an unevaluated fuzzy trapezoid. To evaluate the result as a normal fuzzy set, we must use the `ReleaseHold` function. Next we plot the approximate result of the fuzzy product.

```
In[17]:= FuzzyPlot[ReleaseHold[Multi1]];
```



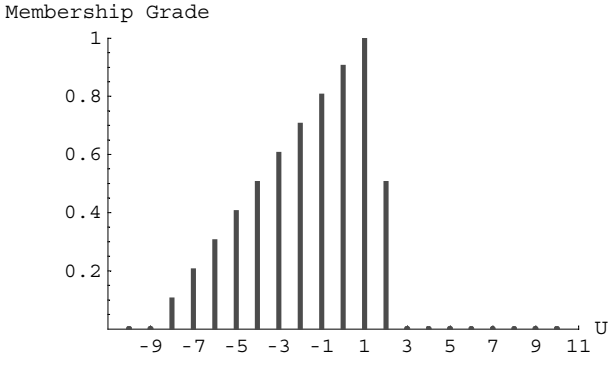
Fuzzy Division

`FuzzyDivide[{a1, b1, c1, d1}, {a2, b2, c2, d2}]` returns the division of the fuzzy numbers represented by the two lists. The fuzzy division is returned as an unevaluated `FuzzyTrapezoid`. To evaluate the `FuzzyTrapezoid`, use the `ReleaseHold` function.

```
In[18]:= Div1 = FuzzyDivide[{-9, 2, 2, 3}, {1, 2, 2, 4}, UniversalSpace -> {-10, 10}]

Out[18]= FuzzyTrapezoid[-9, 1, 1, 3, UniversalSpace → {-10, 10, 1}]
```

The result of the fuzzy division of the two fuzzy numbers is an unevaluated fuzzy trapezoid. To evaluate the result as a normal fuzzy set, we must use the `ReleaseHold` function. Next we plot the approximate result of the fuzzy division.

```
In[19]:= FuzzyPlot[ReleaseHold[Div1]];
```



10 Discrete Fuzzy Arithmetic

10.1 Introduction

Fuzzy Logic package provides a number of functions for performing fuzzy arithmetic. In this chapter, we will demonstrate these functions and the options associated with each function.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

In this package, all of the fuzzy sets are defined in the discrete space. This enables us to perform some quick, discrete fuzzy arithmetic operations on fuzzy sets. In this chapter, we examine a couple of arithmetic operations, which are well-defined in real space, and compare the discrete results with the real space results.

To begin, we must create some fuzzy sets. We will do this later, but first let's extend the default universal space, so we have more room to work.

```
In[2]:= SetOptions[FuzzySet, UniversalSpace → {-40, 40}];
```

When working with the fuzzy arithmetic operations in this package, it is important to choose a proper universal space. The universal space should range from a negative value to the corresponding positive value. This is because fuzzy arithmetic operations often involve working with the image of the fuzzy set, a fuzzy set flipped around zero.

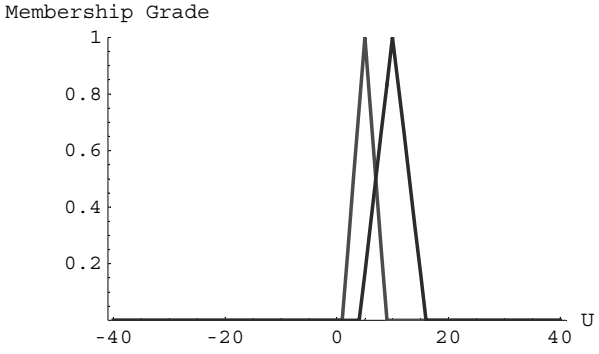
Another consideration is that the result of an arithmetic operation should be contained in the same universal space as the numbers used in the calculation. By choosing a large universal space, you can ensure that the results will fall in the universal space, but the calculations can take longer, and the graphs may be difficult to read. A general rule for setting the universal space is to find the smallest universal space that is still big enough to represent all possible results.

Now we create some fuzzy sets. The first two fuzzy sets we create are triangular fuzzy sets and the second two are Gaussian fuzzy sets. These two types of fuzzy sets are both commonly used to represent fuzzy numbers. For more information on creating fuzzy sets, see the Chapter 1 Creating Fuzzy Sets.

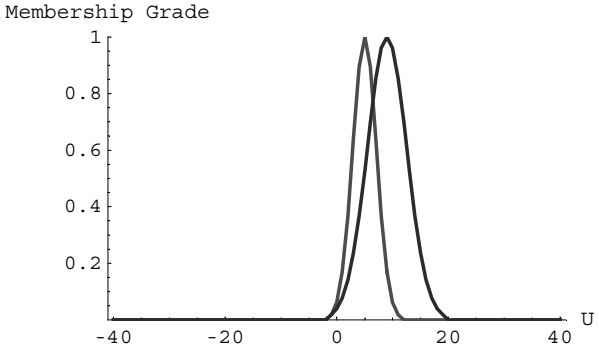
```
In[3]:= TriFS1 = FuzzyTrapezoid[1, 5, 5, 9];

In[4]:= TriFS2 = FuzzyTrapezoid[4, 10, 10, 16];

In[5]:= GausFS3 = FuzzyGaussian[5, 3, ChopValue -> 0.01];

In[6]:= GausFS4 = FuzzyGaussian[9, 5, ChopValue -> 0.01];
```

We use the FuzzyPlot function to see what our initial fuzzy sets look like.

```
In[7]:= FuzzyPlot[TriFS1, TriFS2, PlotJoined → True];
```




```
In[8]:= FuzzyPlot[GausFS3, GausFS4, PlotJoined → True];
```



10.2 Discrete Arithmetic on Triangular Fuzzy Numbers

`DiscreteFuzzyPlus[`A`, `B`]`	return a fuzzy set that is the result of applying fuzzy addition to fuzzy sets A and B
`DiscreteFuzzyMinus[`A`, `B`]`	return a fuzzy set that is the result of applying fuzzy subtraction to fuzzy set A and B
`DiscreteFuzzyMultiply[`A`, `B`]`	return a fuzzy set that is the result of performing a discrete fuzzy multiplication of fuzzy sets A and B
`DiscreteFuzzyImage[`A`]`	return a fuzzy set that is the image of fuzzy set A
`MAX[`A`, `B`]`	return the MAX operation of the fuzzy numbers A and B
`MIN[`A`, `B`]`	return the MIN operation of the fuzzy numbers A and B

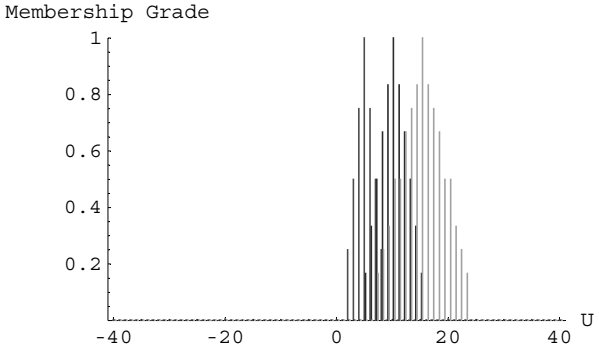
Discrete arithmetic functions for triangular fuzzy numbers.

As mentioned earlier, triangular fuzzy sets are common representations of fuzzy numbers. Triangular fuzzy numbers are convenient to work with because in real space, adding two triangular fuzzy numbers involves adding the corresponding vertices used to define the triangular fuzzy numbers. Similar simple formulas can be used to subtract or find the image of triangular fuzzy numbers. Because of these properties, we demonstrate the discrete arithmetic operations with triangular fuzzy numbers and compare the results with the expected results if the fuzzy sets were defined in real space.

Fuzzy Addition

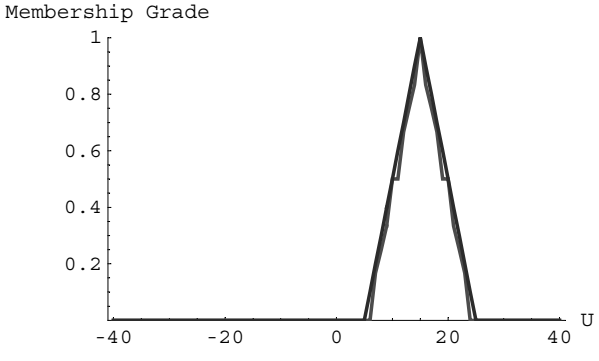
`DiscreteFuzzyPlus[A, B]` returns a fuzzy set that is the result of applying fuzzy addition to fuzzy sets *A* and *B*. We add our first two fuzzy sets and plot the results.

```
In[9]:= Sum1 = DiscreteFuzzyPlus[TriFS1, TriFS2];

In[10]:= Sum111 = TriFS1 ~ FP ~ TriFS2;

In[11]:= FuzzyPlot[TriFS1, TriFS2, Sum1];
```



Based on earlier discussion, the sum of the two triangular fuzzy sets in real space should be a triangular fuzzy set with vertices at {1 + 4, 5 + 10, 5 + 10, 9 + 16} or {5, 15, 15, 25}. To evaluate the results of the discrete fuzzy addition, we plot the expected real space result with our discrete result.

```
In[12]:= ExpectedSum = FuzzyTrapezoid[5, 15, 15, 25];

In[13]:= FuzzyPlot[Sum1, ExpectedSum, PlotJoined → True];
```

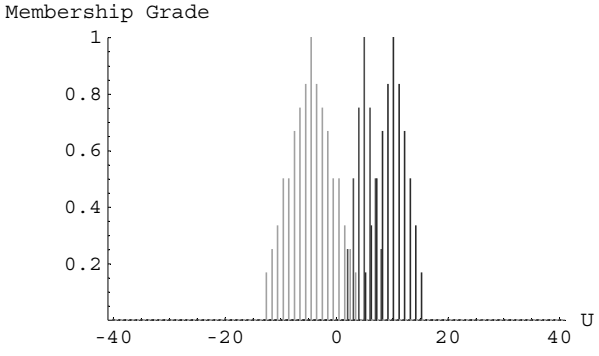


From the graph we see that the discrete fuzzy sum, is bounded above by the expected fuzzy set for real space. This is the case for all of the discrete fuzzy arithmetic operations.

Fuzzy Subtraction

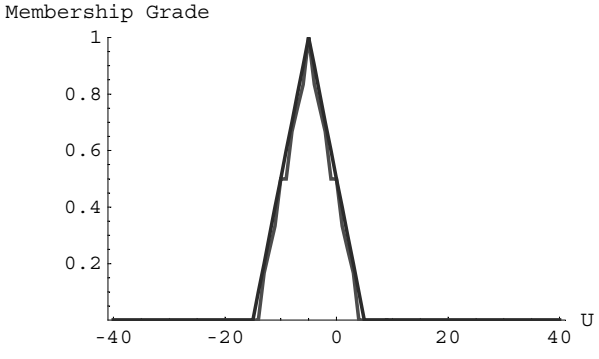
`DiscreteFuzzyMinus[A, B]` returns a fuzzy set that is the result of applying fuzzy subtraction to fuzzy set *A* and *B*. We will subtract our first fuzzy set from the second one and plot the results.

```
In[14]:= Diff1 = DiscreteFuzzyMinus[TriFS1, TriFS2];

In[15]:= FuzzyPlot[TriFS1, TriFS2, Diff1];
```



In real space the difference between the two triangular fuzzy sets would be a triangular fuzzy set with vertices at {1 - 16, 5 - 10, 5 - 10, 9 - 4} or {-15, -5, -5, 5}. To evaluate the results of the discrete fuzzy subtraction, we plot this expected result with the discrete result.

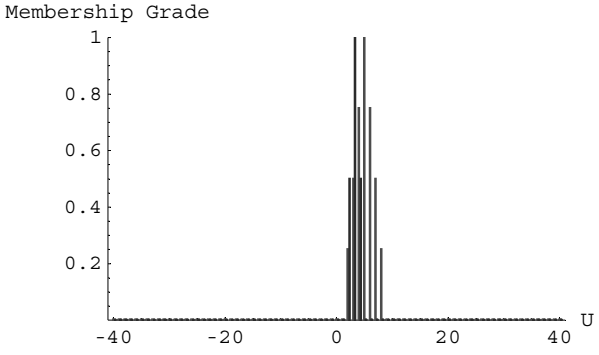
```
In[16]:= ExpectedDiff = FuzzyTrapezoid[-15, -5, -5, 5];

In[17]:= FuzzyPlot[Diff1, ExpectedDiff, PlotJoined → True];
```



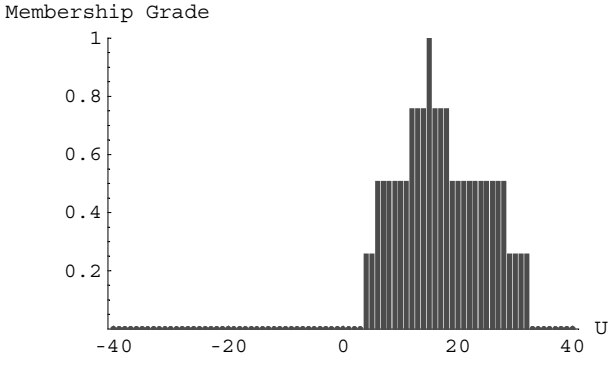
Fuzzy Multiplication

`DiscreteFuzzyMultiply[`A`,` B`]` returns a fuzzy set that is the result of performing a discrete fuzzy multiplication of fuzzy sets A and B. Since fuzzy multiplication causes a fuzzy number to be quite spread out, we use a smaller fuzzy number, `TriFS3`, to demonstrate the fuzzy multiplication.

```
In[18]:= TriFS3 = FuzzyTrapezoid[1, 3, 3, 5];

In[19]:= Prod1 = DiscreteFuzzyMultiply[TriFS1, TriFS3];

In[20]:= FuzzyPlot[TriFS1, TriFS3];
```




```
In[21]:= FuzzyPlot[Prod1];
```

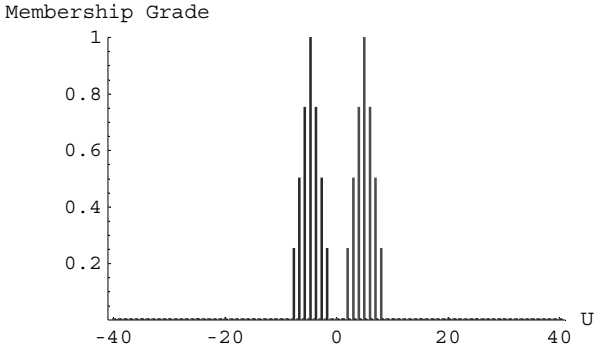


Fuzzy Image

`DiscreteFuzzyImage[`*A*`]` returns a fuzzy set that is the image of fuzzy set *A*. We will find the image of our first fuzzy set and plot the results here.

```
In[22]:= Im1 = DiscreteFuzzyImage[TriFS1];
```

```
In[23]:= FuzzyPlot[TriFS1, Im1];
```



In the graph, the fuzzy set to the right is our original fuzzy set, and the fuzzy set to the left is the image of the original fuzzy set. In the discrete case, the image of a fuzzy set will look the same as it would in real space.

MAX and MIN Functions

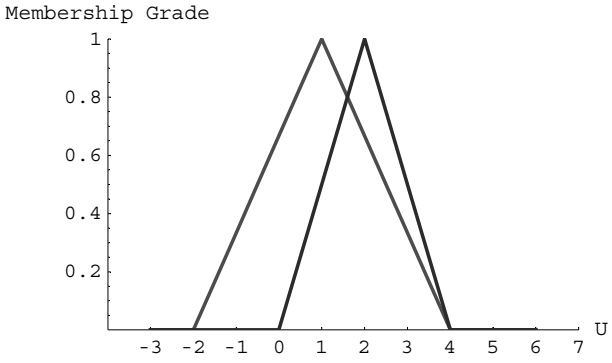
To introduce a meaningful ordering of fuzzy numbers, we may extend the lattice operations min and max on real numbers to corresponding operations on fuzzy numbers, `MIN` and `MAX`. A partial ordering for comparable fuzzy numbers is defined as $A \preceq B$ iff MIN(A, B) = A or, alternatively $A \preceq B$ iff MAX(A, B) = B.

`MAX[`*A*`,` *B*`]` returns the fuzzy maximum of the fuzzy numbers *A* and *B*.

```
In[24]:= fs1 = FuzzyTrapezoid[-2, 1, 1, 4, UniversalSpace → {-3, 6, .5}];
```

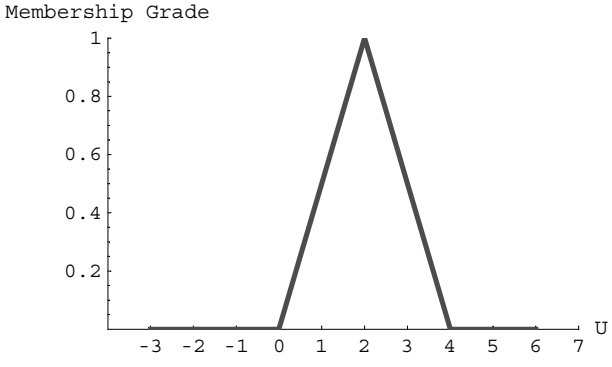
```
In[25]:= fs2 = FuzzyTrapezoid[0, 2, 2, 4, UniversalSpace → {-3, 6, .5}];
```

```
In[26]:= FuzzyPlot[fs1, fs2, PlotJoined → True];
```




```
In[27]:= mx1 = MAX[fs1, fs2];
```

```
In[28]:= FuzzyPlot[mx1, PlotJoined → True];
```

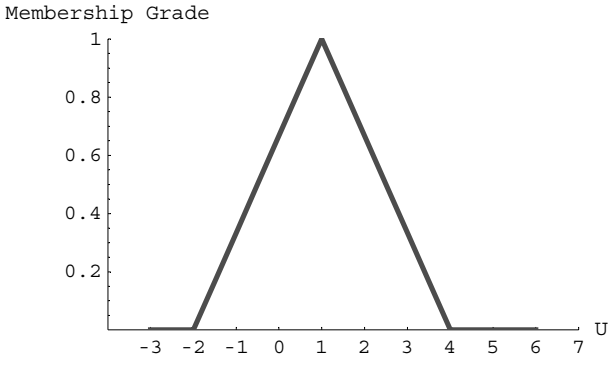



```
In[29]:= Equality[mx1, fs2]

Out[29]= True
```

`MIN[`A`,` B`]` returns the fuzzy minimum of the fuzzy numbers A and B.

```
In[30]:= mn1 = MIN[fs1, fs2];
```

```
In[31]:= FuzzyPlot[mn1, PlotJoined → True];
```




```
In[32]:= Equality[mn1, fs1]

Out[32]= True
```

In our example `fs1 < fs2`.

Note that the symbols `MIN` and `MAX`, which denote the introduced operations on fuzzy numbers, must be distinguished from the symbols min and max, which denote the operations of minimum and maximum on real numbers, respectively.

10.3 Discrete Arithmetic on Gaussian Fuzzy Numbers

Using Gaussian fuzzy sets is another common way to represent fuzzy numbers. Gaussian fuzzy numbers are convenient to work with because in real space, adding two Gaussian fuzzy numbers involves adding the means and standard deviations used to define the original Gaussian fuzzy numbers. To subtract two Gaussian fuzzy numbers, subtract the means and add the standard deviations. Because of these properties, we further demonstrate the discrete arithmetic operations using Gaussian fuzzy numbers.

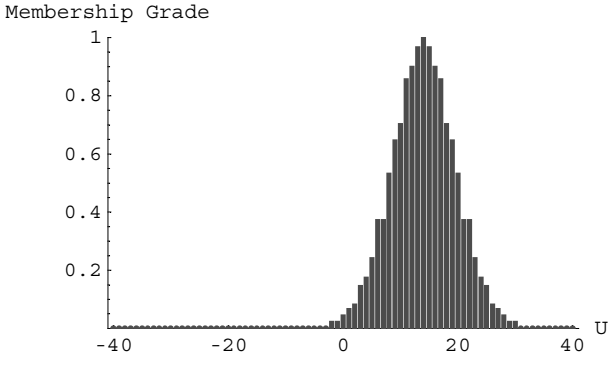
`DiscreteFuzzyPlus[A, B]`	return a fuzzy set that is the result of applying fuzzy addition to fuzzy sets A and B
`DiscreteFuzzyMinus[A, B]`	return a fuzzy set that is the result of applying fuzzy subtraction to fuzzy set A and B
`DiscreteFuzzyMultiply[A, B]`	return a fuzzy set that is the result of performing a discrete fuzzy multiplication of fuzzy sets A and B
`DiscreteFuzzyImage[A]`	return a fuzzy set that is the image of fuzzy set A

Discrete arithmetic functions for Gaussian fuzzy numbers.

Fuzzy Addition

We start by adding the two Gaussian fuzzy numbers created earlier, and we plot the results.

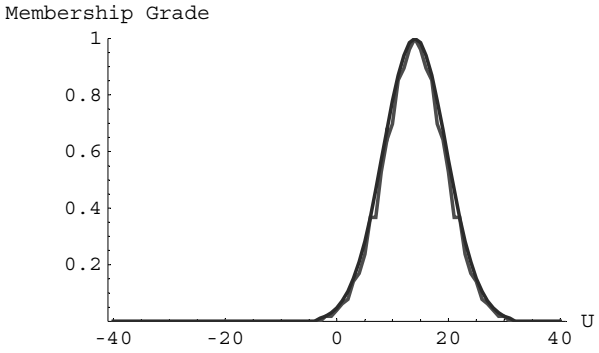
```
In[33]:= Sum2 = DiscreteFuzzyPlus[GausFS3, GausFS4];

In[34]:= FuzzyPlot[Sum2];
```



Notice that when two Gaussian fuzzy numbers are added, the universal space is doubled.

As discussed above, after adding two Gaussian fuzzy sets defined in real space, the result is a Gaussian fuzzy set with mean and standard deviation equal to the sum of the means and standard deviations of the original fuzzy sets. In our example, we expect to get a result with mean of 5 + 9 = 14 and a standard deviation of 3 + 5 = 8. We can examine the results of our discrete fuzzy addition by plotting it with the expected results.

```
In[35]:= ExpectedSum2 = FuzzyGaussian[14, 8, ChopValue -> 0.01, UniversalSpace -> {-40, 40}];

In[36]:= FuzzyPlot[Sum2, ExpectedSum2, PlotJoined → True];
```

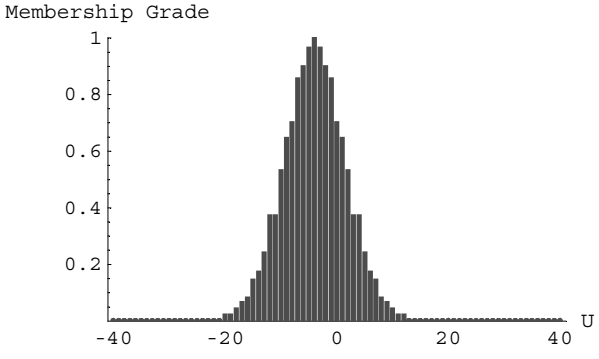


Again, as with triangular fuzzy numbers, the discrete fuzzy sum is bounded above by the expected result in real space. This will be true for all fuzzy arithmetic operations.

Fuzzy Subtraction

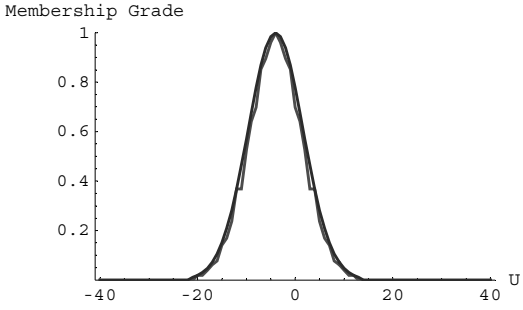
Here we subtract the two Gaussian fuzzy numbers and plot the result.

```
In[37]:= Diff2 = DiscreteFuzzyMinus[GausFS3, GausFS4];

In[38]:= FuzzyPlot[Diff2];
```



The expected fuzzy difference for our example would be a Gaussian fuzzy set with a mean of 5 - 9 = -4 and a standard deviation of 3 + 5 = 8. We plot the discrete result with this expected result below.

```
In[39]:= ExpectedDiff2 = FuzzyGaussian[-4, 8, ChopValue -> 0.01];

In[40]:= FuzzyPlot[Diff2, ExpectedDiff2, PlotJoined → True];
```



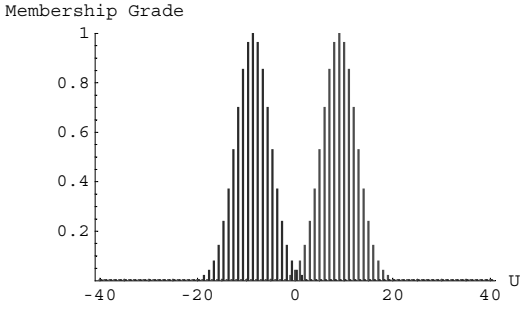
Fuzzy Multiplication

We can perform a discrete fuzzy multiplication with Gaussian fuzzy numbers, but since Gaussian fuzzy sets are defined for every element in the universal space, the result would be defined on a large universal space. If we multiplied our two Gaussian fuzzy sets from this notebook, the universal space would extend from -1600 to 1600, and the calculations would take a long time. For this reason, we will not demonstrate here the `DiscreteFuzzyMultiply` function on the Gaussian fuzzy sets.

Fuzzy Image

Here we find the image of our last fuzzy set and we plot the results.

```
In[41]:= Im2 = DiscreteFuzzyImage[GausFS4];

In[42]:= FuzzyPlot[GausFS4, Im2];
```



The image is a fuzzy set that is the mirror image of the original fuzzy set, flipped about zero.

You can use a built-in function `SetOptions` to restore the default setting for the `FuzzySet` object.

```
In[43]:= SetOptions[FuzzySet, UniversalSpace → {0, 20, 1}]

Out[43]= {UniversalSpace → {0, 20, 1}}
```

11 Łukasiewicz Sets and Logic

11.1 Introduction

The goal of this notebook is to demonstrate how the *Fuzzy Logic* package can be used to perform Łukasiewicz logic. Łukasiewicz sets are very similar to fuzzy sets, but elements in Łukasiewicz sets take on one of n-values, where n is a natural number greater than or equal to 2. Łukasiewicz fuzzy sets are often referred to as Ln sets, where n represents the number of values allowed for membership grades. For performing three valued logic, you would be using L3 sets. In this notebook we will demonstrate the functions contained in the *Fuzzy Logic* package that deal with Łukasiewicz sets and n-valued logic.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

11.2 Creating Łukasiewicz Sets

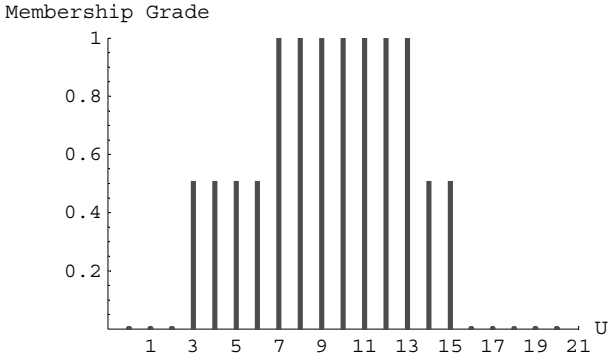
DigitalSet[*a*, *b*, *c*, *d*, *h*, Levels -> *n*]	return a Łukasiewicz set with membership grades that linearly increase from zero to *h* in the range *a* to *b*, equals *h* in the range *b* to *c*, and linearly decrease from *h* to zero in the range *c* to *d*
ToDigital[*A*, *n*]	take a fuzzy set and an integer as input and returns a Łukasiewicz set using n–valued logic

Functions for creating Łukasiewicz sets.

DigitalSet[*a*, *b*, *c*, *d*, *h*, Levels -> *n*] returns a digital fuzzy set with the number of levels equal to *n*. The universal space and the value of *n* may be defined using the UniversalSpace and the *n* option, otherwise the default universal space and default *n* will be given. The values of the membership grades increase linearly from *a* to *b*, then are equal to the closest possible value of *h* from *b* to *c*, and linearly decrease from *c* to *d*. Arguments *a*, *b*, *c*, and *d* should be in increasing order, and *h* must be a value between 0

and 1, inclusive. If a value for *h* is not given, it defaults to 1. In the following example, we create a fuzzy set with defaults universal space and *n*. This means we will get a L3 fuzzy set or a digital fuzzy set with three possible membership grades.

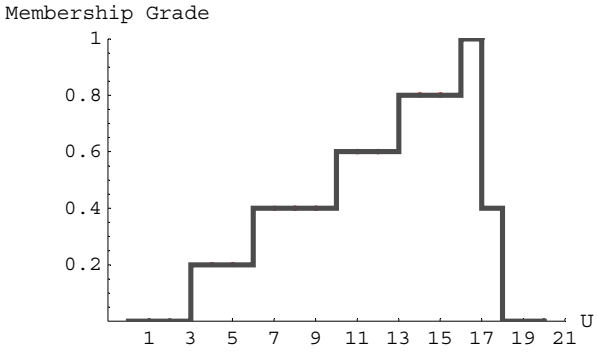
```
In[2]:= L1 = DigitalSet[1, 8, 12, 17, UniversalSpace -> {0, 20, 1}]

Out[2]= FuzzySet[{{3, 1/2}, {4, 1/2}, {5, 1/2}, {6, 1/2}, {7, 1}, {8, 1}, {9, 1}, {10, 1},
          {11, 1}, {12, 1}, {13, 1}, {14, 1/2}, {15, 1/2}}, UniversalSpace → {0, 20, 1}]

In[3]:= FuzzyPlot[L1];
```



We see from the plot that the fuzzy set above contains only three membership grades 0, 0.5, or 1. As another example, we can use the same specifications as the previous example, but this time let's set *n* to be 6.

```
In[4]:= L2 = DigitalSet[1, 17, 17, 19, Levels → 6]

Out[4]= FuzzySet[{{3, 1/5}, {4, 1/5}, {5, 1/5}, {6, 2/5}, {7, 2/5},
          {8, 2/5}, {9, 2/5}, {10, 3/5}, {11, 3/5}, {12, 3/5}, {13, 4/5}, {14, 4/5},
          {15, 4/5}, {16, 1}, {17, 1}, {18, 2/5}}, UniversalSpace → {0, 20, 1}]
```

```
In[5]:= FuzzyPlot[L2, Crisp -> True];
```



This time we see that there are six membership grade levels. From this example, you can see that as n gets larger, the Łukasiewicz sets look more and more like typical fuzzy sets. As you might expect, as n goes to infinity, Łukasiewicz logic becomes fuzzy logic.

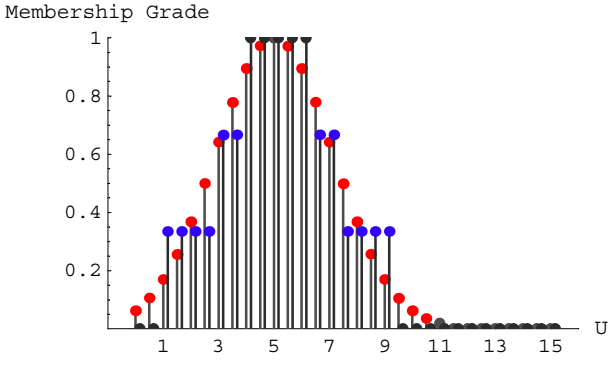
`ToDigitalSet[`A`,` n`]` takes a fuzzy set and an integer as input, and it returns a Łukasiewicz set using n-valued logic. Let's look at an example.

```
In[6]:= FS1=FuzzyGaussian[5,3,ChopValue->0.01, UniversalSpace->{0,15,0.5}];
```

Here we create a digital fuzzy set with four membership level grades from a discrete fuzzy set and plot two sets together.

```
In[7]:= LS1=ToDigital[FS1,4];
```

```
In[8]:= FuzzyPlot[FS1,LS1,ShowDots->True];
```



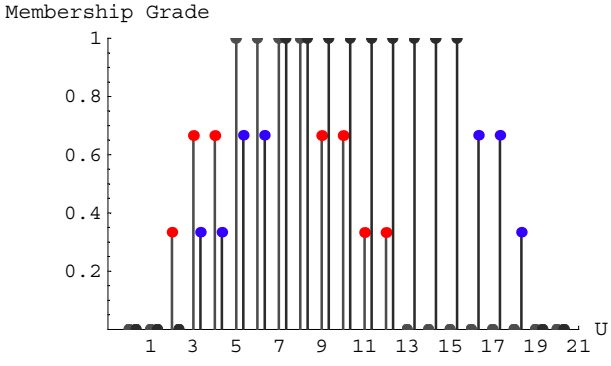
11.3 Operations on Ln Sets

`Union[`*A1, A2, ... , An*`]`	return a Ln set that is the union of Ln sets *A1, A2, ... , An*
`Intersection[`*A1, A2, ... , An*`]`	return a Ln set that is the intersection of Ln sets *A1, A2, ... , An*
`Implication[`*A, B*`]`	return a Ln set that is the implication of Ln sets *A* and *B*
`Complement[`*A*`]`	return a Ln set that is the complement of Ln set *A*
`PrimitiveMatrix[`*n, op*`]`	return a logic table that shows the results of the operation *op* for *n* valued logic sets

Operations on Ln sets.

Łukasiewicz logic is a form of n-valued logic. We demonstrate some of the logic operations using four valued logic in the following examples. Since the Ln set is created as a fuzzy set with n membership values, operations may be performed using the standard operations defined in the *Fuzzy Logic* manual. One additional function, which has been added to the package, is the implication function. Original Łukasiewicz Logic operations were all based on the operations of the negation and implication primitives. To demonstrate some of the operations, we start by creating two L4 sets.

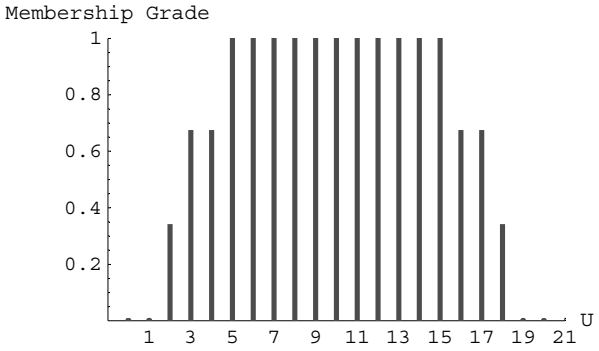
```
In[9]:= Luk1 := DigitalSet[1, 5, 8, 13, Levels → 4]

In[10]:= Luk2 := DigitalSet[2, 7, 15, 19, Levels → 4]

In[11]:= FuzzyPlot[Luk1, Luk2, ShowDots -> True];
```



We can now perform some of the various logic operations on our Łukasiewicz sets.

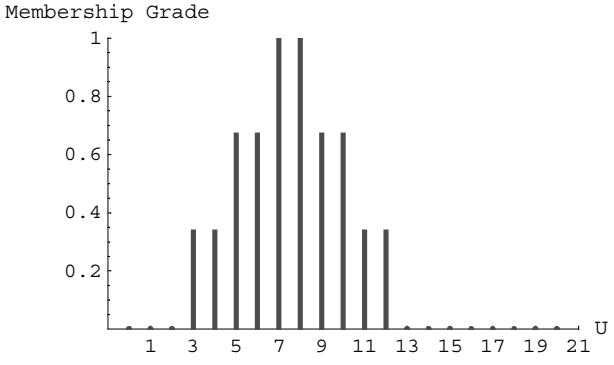
The `Union` operation takes the max(a, b), where a and b are the membership grades of corresponding elements of two n-valued sets. The result will also be a n-valued set.

```
In[12]:= Un1 := Union[Luk1, Luk2]

In[13]:= FuzzyPlot[Un1];
```



The `Intersection` operation takes the min(a, b), where a and b are the membership grades of corresponding elements of two n-valued sets. The result will also be a n-valued set.

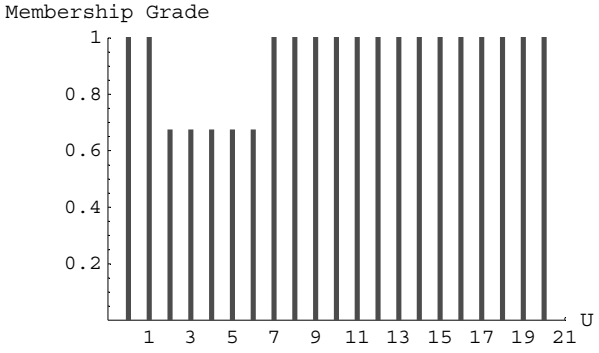
```
In[14]:= Int1 := Intersection[Luk1, Luk2]

In[15]:= FuzzyPlot[Int1];
```



The `Implication` operation is defined as min(1, 1 + b - a), where a and b are the membership grades of corresponding elements of two n-valued sets.

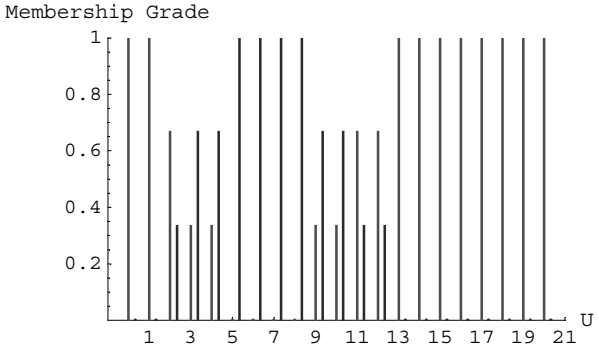
```
In[16]:= Imp1 = Implication[Luk1, Luk2]

Out[16]= FuzzySet[{{0, 1}, {1, 1}, {2, 2/3}, {3, 2/3}, {4, 2/3}, {5, 2/3}, {6, 2/3},
           {7, 1}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}, {13, 1}, {14, 1}, {15, 1},
           {16, 1}, {17, 1}, {18, 1}, {19, 1}, {20, 1}}, UniversalSpace → {0, 20, 1}]
```

```
In[17]:= FuzzyPlot[Impl];
```



The `Complement` operation returns the complement of an n-valued set, which is also an n-valued set.

```
In[18]:= Comp1 := Complement[Luk1]

In[19]:= FuzzyPlot[Comp1, Luk1];
```



`PrimitiveMatrix[`*n*`, `*op*`]` returns a logic table that shows the results of the operation *op* for *n*-valued logic sets.

```
In[20]:= PrimitiveMatrix[3]
```

Out[20]//DisplayForm=

a	b	Implication	Bicondition	Intersection	Union
0	0	1	1	0	0
0	$\frac{1}{2}$	1	$\frac{1}{2}$	0	$\frac{1}{2}$
0	1	1	0	0	1
$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$
$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1
1	0	0	0	0	1
1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1	1	1

12 Fuzzy Clustering

12.1 Introduction

Clustering involves the task of dividing data points into homogeneous classes or clusters so that items in the same class are as similar as possible and items in different classes are as dissimilar as possible. Clustering can also be thought of as a form of data compression, where a large number of samples are converted into a small number of representative prototypes or clusters. Depending on the data and the application, different types of similarity measures may be used to identify classes, where the similarity measure controls how the clusters are formed. Some examples of values that can be used as similarity measures include distance, connectivity, and intensity.

In non-fuzzy or hard clustering, data is divided into crisp clusters, where each data point belongs to exactly one cluster. In fuzzy clustering, the data points can belong to more than one cluster, and associated with each of the points are membership grades which indicate the degree to which the data points belong to the different clusters. This chapter demonstrates the fuzzy c-means clustering algorithm.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

12.2 Fuzzy C-Means Clustering (FCM)

The FCM algorithm is one of the most widely used fuzzy clustering algorithms. This technique was originally introduced by Professor Jim Bezdek in 1981. The FCM algorithm attempts to partition a finite collection of elements $X=\{x_1, x_2, \ldots, x_n\}$ into a collection of c fuzzy clusters with respect to some given criterion. Given a finite set of data, the algorithm returns a list of c cluster centers V, such that

$$V = v_i, i = 1, 2, \ldots, c$$

and a partition matrix U such that

$$U = u_{ij},\ i = 1, ..., c,\ j = 1, ..., n$$

where u_{ij} is a numerical value in [0, 1] that tells the degree to which the element $\mathbf{x}_j$ belongs to the i-th cluster.

The following is a linguistic description of the FCM algorithm, which is implemented in *Fuzzy Logic* package. The functions that implement this algorithm can be found in the *Clustering.m* file.

Step 1: Select the number of clusters c ($2 \le c \le n$), exponential weight μ ($1 < \mu < \infty$), initial partition matrix U^0, and the termination criterion ϵ. Also, set the iteration index l to 0.

Step 2: Calculate the fuzzy cluster centers $\{v_i^l \mid i=1, 2, ..., c\}$ by using U^l.

Step 3: Calculate the new partition matrix U^{l+1} by using $\{v_i^l \mid i=1, 2, ..., c\}$.

Step 4: Calculate the new partition matrix $\Delta = \| U^{l+1} - U^l \| = \max_{i,j} | u_{ij}^{l+1} - u_{ij}^l |$. If $\Delta > \epsilon$, then set l = l + 1 and go to step 2. If $\Delta \le \epsilon$, then stop.

We will demonstrate here how to set up and use the clustering functions.

`FCMCluster[`*data*`, `*partmat*`, `*mu*`, `*epsilon*`]`	return a list of cluster centers, a partition matrix indicating the degree to which each data point belongs to a particular cluster center, and a list containing the progression of cluster centers found during the run
`InitializeU[`*data*`, `*n*`]`	return a random initial partition matrix for use with the `FCMCluster` function where n is the number of cluster centers desired
`ShowCenters[`*graph*`, `*res*`]`	display a 2 D plot showing a graph of a set of data points along with large dots indicating the cluster centers found by the `FCMCluster` function
`ShowCentersProgression[`*graph*`, `*res*`]`	display a 2 D plot showing a graph of a set of data points along with a plot of how the cluster centers migrated during the application of the `FCMCluster` function

Function for cluster analysis.

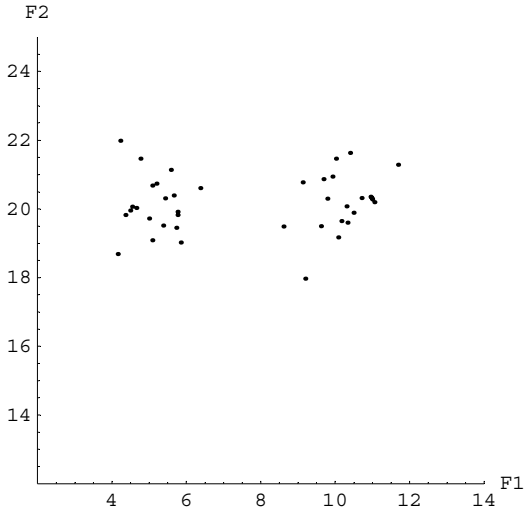
12.3 Example

To demonstrate the FCM clustering algorithm, we will create a 2D data set that consists of two groups of data. One group of data is centered around the point {5, 20} and the other is centered around {10, 20}. Below are the functions used to create the data set of 40 points, each of has two features F1 and F2.

```
In[2]:= SeedRandom[1234]

In[3]:= Needs["Statistics`MultinormalDistribution`"];

In[4]:= e1 = RandomArray[MultinormalDistribution[{0, 0}, {{.5, 0}, {0, .5}}], 20];

In[5]:= e2 = RandomArray[MultinormalDistribution[{0, 0}, {{.5, 0}, {0, .5}}], 20];

In[6]:= TrainData1 = Join[Table[{5, 20} + e1[[i]], {i, 20}], Table[{10, 20} + e2[[i]], {i, 20}]]

Out[6]= {{5.37737, 19.5085}, {5.73233, 19.4497}, {4.66011, 20.0309}, {5.66351, 20.3851},
         {4.22536, 21.9799}, {5.08249, 20.6785}, {5.08281, 19.0859}, {4.54482, 20.0675},
         {4.35949, 19.8239}, {5.19668, 20.7287}, {4.48769, 19.9573}, {5.76696, 19.8192},
         {4.1538, 18.6845}, {6.3741, 20.6019}, {5.58993, 21.1309}, {5.76799, 19.9128},
         {5.85273, 19.0219}, {5.00223, 19.7178}, {5.42941, 20.3109}, {4.77241, 21.4551},
         {8.61372, 19.4807}, {10.7079, 20.3151}, {10.9941, 20.2807}, {11.6882, 21.2844},
         {9.12723, 20.7716}, {10.3058, 20.074}, {10.4027, 21.6238}, {10.9551, 20.3493},
         {9.79065, 20.2991}, {9.6922, 20.8628}, {10.0233, 21.4553}, {11.0519, 20.1996},
         {9.20106, 17.9669}, {10.4986, 19.8826}, {9.61325, 19.4909}, {10.3346, 19.5939},
         {9.92963, 20.9454}, {10.1744, 19.6404}, {10.9785, 20.3215}, {10.0844, 19.1679}}
```

The following is a plot of the data set, which we will use to test the FCM clustering algorithm.

```
In[7]:= g1 = ListPlot[TrainData1, PlotRange -> {{2, 14}, {12, 25}},
              AxesLabel -> {"F1", "F2"},
       PlotStyle -> {AbsolutePointSize[2]}, AspectRatio -> 1];
```



`FCMCluster[`*data*`,` *partmat*`,` *mu*`,` *epsilon*`]` returns a list of cluster centers, a partition matrix indicating the degree to which each data point belongs to a particular cluster center, and a list containing the progression of cluster centers found during the run. The arguments to the function are the data set (*data*), an initial partition matrix (*partmat*), a value determining the degree of fuzziness of the clustering (*mu*), and a value which determines when the algorithm will terminate (*epsilon*). This function runs recursively until the terminating criteria is met. While running, the function prints a value that indicates the accuracy of the fuzzy clustering. When this value is less than the parameter epsilon, the function terminates. The parameter *mu* is called the exponential weight and controls the degree of fuzziness of the clusters. As *mu* approaches 1, the fuzzy clusters become crisp clusters, where each data point belongs to only one cluster. As *mu* approaches infinity, the clusters become completely fuzzy, and each point will belong to each cluster to the same degree ($1/c$) regardless of the data. Studies have been done on selecting the value for *mu*, and it appears that the best choice for *mu* is usually in the interval [1.5, 2.5], where the midpoint, $mu = 2$, is probably the most commonly used value for *mu*.

We can use the `FCMCluster` function to find clusters in the data set created earlier. In order to create the initial partition matrix that will be used by the `FCMCluster` function, we will use the `InitializeU` function described below.

`InitializeU[`*data*`,` *n*`]` returns a random initial partition matrix for use with the `FCMCluster` function, where *n* is the number of cluster centers desired. The following is an example using the `FCMCluster` function to find two cluster centers in the data set created earlier. Notice that the function runs until the terminating criteria goes under 0.01, which is the value specified for *epsilon*.

```
In[8]:= Res1a = FCMCluster[TrainData1, InitializeU[TrainData1, 2], 1.5, 0.01]

          0.503451

          0.16241

          0.333521

          0.179176

          0.0119194

          0.000282141

Out[8]= {{{10.2204, 20.2151}, {5.16335, 20.1134}},
         {{0.000295259, 0.00135735, 0.0000705869, 0.000242704, 0.0123289, 0.000149924,
           0.0014715, 0.000142399, 0.000447395, 0.000221595, 0.000213201, 0.00050857,
           0.00607655, 0.0128432, 0.00297573, 0.000415074, 0.00656587, 0.0000440822,
           0.0000228615, 0.00389807, 0.939566, 0.999935, 0.999686, 0.9944, 0.991388,
           0.999999, 0.995415, 0.999725, 0.99992, 0.998902, 0.996166, 0.999602, 0.921725,
           0.999957, 0.998047, 0.999782, 0.999304, 0.999828, 0.999701, 0.998032},
          {0.999705, 0.998643, 0.999929, 0.999757, 0.987671, 0.99985, 0.998528,
           0.999858, 0.999553, 0.999778, 0.999787, 0.999491, 0.993923, 0.987157, 0.997024,
           0.999585, 0.993434, 0.999956, 0.999977, 0.996102, 0.0604338, 0.0000647235,
           0.000313873, 0.0056, 0.00861153, 1.05662*10^-6, 0.00458459, 0.000275437,
           0.000079949, 0.0010978, 0.00383382, 0.000397518, 0.0782754, 0.0000434256,
           0.00195332, 0.000218114, 0.000696407, 0.000172122, 0.000299415, 0.00196828}},
         {{{7.63886, 20.0333}, {7.57786, 20.2694}}, {{7.835, 20.0932}, {7.52867, 20.2236}},
          {{8.47023, 20.1361}, {6.89488, 20.1792}}},
          {{9.99462, 20.2004}, {5.39357, 20.1148}}, {{10.2179, 20.2143}, {5.16427, 20.1132}},
          {{10.2204, 20.2151}, {5.16335, 20.1134}}}}
```

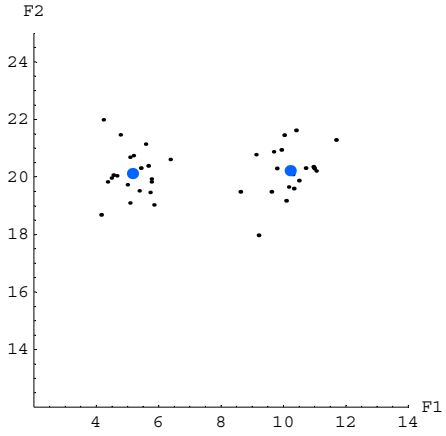
The clustering function should work for data of any dimension, but it is hard to visualize the results for higher order data.

There are two functions in *Fuzzy Logic* package that are useful in visualizing the results of the FCMCluster algorithm, and they are described below.

`ShowCenters[`*graph*`,` *res*`]` displays a 2D plot showing a graph of a set of data points along with large dots indicating the cluster centers found by the `FCMCluster` function. The variable *graph* is a plot of the data points and *res* is the results from the `FCMCluster` function.

The following is an example showing the cluster centers found from the previous example. Notice that the cluster centers are located where you would expect near the centers of the two clusters of data.

```
In[9]:= p1 = ShowCenters[g1, Res1a];
```

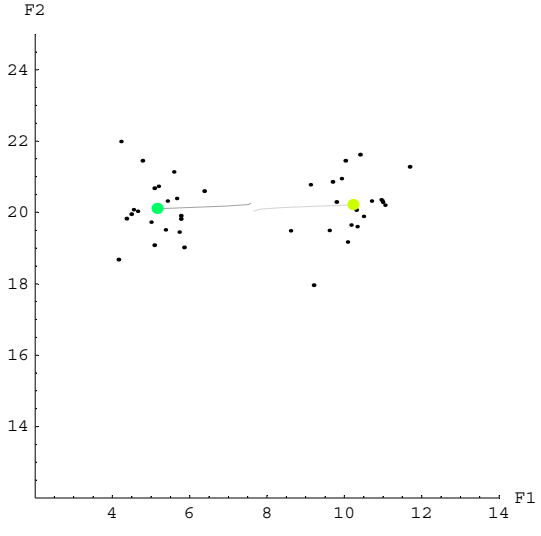


Another similar function in *Fuzzy Logic* package is the `ShowCenterProgression` function.

`ShowCentersProgression[`*graph*`,` *res*`]` displays a 2D plot showing a graph of a set of data points along with a plot of how the cluster centers migrated during the application of the `FCMCluster` function. The variable *graph* is a plot of the data points and *res* is the results from the `FCMCluster` function.

The following is the result of the previous example. Notice that the cluster centers start near the center of the data and move to their final spots near the centers of the two data clusters.

```
In[10]:= ShowCenterProgression[g1, Res1a];
```



The clustering function just returns cluster centers and the degrees to which the different data points belong to each cluster center. They don't translate directly into membership functions. There are different ways we could turn the cluster centers into membership functions, such as taking the projection of the membership grades of the points onto one of the axis or by just placing fuzzy sets at the cluster centers or some other way.

Appendix

Fuzzy Operator Formulas

The following is a list of formulas used to operate on the membership grades of fuzzy set `A` for selected general operations on fuzzy sets. In the following formulas, x_n represents the new membership grades after applying the operators, and y_n represents the original membership grades for the elements of fuzzy set `A`.

Concentrate

$$x_n = y_n^2$$

Dilate

$$x_n = \sqrt{y_n}$$

Intensify Contrast

$$x_n = \begin{cases} 2\,y_n^2 & \text{if } y_n \le 1/2 \\ 1 - 2\,(1 - y_n)^2 & \text{otherwise} \end{cases}$$

Normalize

$$x_n = \frac{y_n}{\text{Height}[A]}$$

Complement

Standard

$$x_n = 1 - y_n$$

Sugeno[α], w $\in$ (-1, ∞)

$$x_n = \frac{1 - y_n}{1 - \alpha\, y_n}$$

Yager[w], w $\in$ (0, ∞)

$$x_n = (1 - y_n^{w})^{1/w}$$

Intersection Formulas

The following is a list of the formulas used for various intersections between two fuzzy sets, A and B. The following formulas indicate how the membership grades for corresponding elements in fuzzy sets A and B should be combined. In the formulas, a represent the membership grade for an element in fuzzy set A, and b represents the membership grade for an element in fuzzy set B.

Standard

$$i(a, b) = \text{Min}[a, b]$$

Hamacher[v], v $\in$ (0, ∞)

$$i(a, b) = \frac{a\, b}{v - (1 - v)\,(a + b - a * b)}$$

Frank[s], s $\in$ (0, ∞) $\wedge$ s $\neq$ 1.

$$i(a, b) = \text{Log}_s\left[1 + \frac{(s^a - 1)\,(s^b - 1)}{s - 1}\right]$$

Yager[w], w $\in$ (0, ∞)

$$i(a, b) = 1 - \text{Min}[1, ((1 - a)^{w} + (1 - b)^{w})^{1/w}]$$

DuboisPrade[α], $\alpha \in [0, 1]$

$$i(a, b) = \frac{a\,b}{\text{Max}[a, b, \alpha]}$$

Dombi[α], $\alpha \in (0, \infty)$

$$i(a, b) = \frac{1}{1 + \left(\left(\frac{1}{a} - 1\right)^{\alpha} + \left(\frac{1}{b} - 1\right)^{\alpha}\right)^{1/\alpha}}$$

Weber[λ], $\lambda \in (-1, \infty)$

$$i(a, b) = \text{Max}\left[0, \frac{a + b + \lambda\,a\,b - 1}{1 + \lambda}\right]$$

Yu[λ], $\lambda \in (-1 , \infty)$

$$i(a, b) = \text{Max}[\,0\,,\ (1 + \lambda)\,(a + b - 1) - \lambda\,a\,b\,]$$

Union Formulas

The following is a list of the formulas used for various unions between two fuzzy sets, `A` and `B`. The following formulas indicate how the membership grades for corresponding elements in fuzzy sets `A` and `B` should be combined. In the formulas, `a` represent the membership grade for an element in fuzzy set `A`, and `b` represents the membership grade for the corresponding element in fuzzy set `B`.

Standard

$$u(a, b) = \text{Max}[a, b]$$

Hamacher[v], $v \in (0, \infty)$

$$u(a, b) = \frac{a + b - (2 - v)\,a\,b}{1 - (1 - v)\,a\,b}$$

Frank[s], $s \in (0, \infty) \wedge s \neq 1$

$$u(a, b) = 1 - \text{Log}_s\left[1 + \frac{(s^{1-a} - 1)\,(s^{1-b} - 1)}{s - 1}\right]$$

Yager[w], $w \in (0, \infty)$

$$u(a, b) = \text{Min}[1, (a^w + b^w)^{1/w}]$$

DuboisPrade[α], $\alpha \in [0, 1]$

$$u(a, b) = 1 - \frac{(1 - a)\,(1 - b)}{\text{Max}[1 - a, 1 - b, \alpha]}$$

Dombi[α], $\alpha \in (0, \infty)$

$$u(a, b) = \frac{1}{1 + \left(\left(\frac{1}{a} - 1\right)^{-\alpha} + \left(\frac{1}{b} - 1\right)^{-\alpha}\right)^{-1/\alpha}}$$

Weber[λ], $\lambda \in (-1 , \infty)$

$$u(a, b) = \text{Min}\left[1 , a + b - \frac{\lambda}{1 - \lambda}\, a\, b\right]$$

Yu[λ], $\lambda \in (-1 , \infty)$

$$u(a, b) = \text{Min}[1 , a + b + \lambda\, a\, b]$$

Averaging Formulas

Following is a list of the formulas for taking the various averages used in this package. The averaging operations are denoted by the letter `h`, and $a_1, \ldots, a_n$ represent the membership grades for corresponding elements in the `n` fuzzy sets being averaged.

Arithmetic Mean

$$h(a_1, a_2, \ldots, a_n) = \frac{a_1 + a_2 + \ldots + a_n}{n}$$

Geometric Mean

$$h(a_1, a_2, \ldots, a_n) = (a_1\, a_2 \ldots a_n)^{1/n}$$

Harmonic Mean

$$h(a_1, a_2, \ldots, a_n) = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \ldots + \frac{1}{a_n}}$$

Generalized Mean $[\alpha]$, $\alpha \in (-\infty, \infty)$

$$h_\alpha(a_1, a_2, \ldots, a_n) = \left(\frac{a_1^{\alpha} + a_2^{\alpha} + \ldots + a_n^{\alpha}}{n}\right)^{1/\alpha}$$

Miscellaneous Formulas

Gaussian Fuzzy Sets

$$e^{-((x-m)/s)^2}$$

where m is the mean, `s` is the width, and `x` is the element.

Bell Fuzzy Sets

$$\frac{1}{1 + \text{Abs}\left[\frac{x - c}{w}\right]^{2s}}$$

where `c` is the center, crossover points are at `c ± w`, a slope at the crossover points of `s/2w`, and `x` is the element.

Sigmoid Fuzzy Sets

$$\frac{1}{1 + e^{-s(x-c)}}$$

where s controls the slope at crossover point `c`, and `x` is the element.

Bibliography

J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms,* Plenum Press, New York, 1981.

J. C. Bezdek, ed., *Analysis of Fuzzy Information*, Vol. 1, 2, 3, CRC Press, Boca Raton, FL, 1987.

E. Czogala, *Probabilistic Sets in Decision Making and Control,* Verlag TUV Reinland, Koeln, 1984.

D. Driankov, H. Hellendoorn, and M. Reinfrank, *An Introduction to Fuzzy Control,* Springer-Verlag, Berlin Heidelberg, 1993.

D. Dubois and H. Prade, *Fuzzy Sets and Systems: Theory and Applications,* Academic Press, Cambridge, MA, 1980.

D. Dubois and H. Prade, *Possibility Theory*, Plenum Press, New York, 1988.

J. A. Freeman, Fuzzy systems for control applications: the truck backer-upper, *The Mathematica Journal*, vol. 4, pp. 64-69, 1994.

J. A. Goguen, L-fuzzy sets, *J. Math. Anal. Appl.*, vol. 18, pp. 145-174, 1967.

M. M. Gupta, G.N. Saridis, and B. R. Gaines, eds., *Fuzzy Automata and Decision Processes*, North-Holland, New York, 1977.

M. M. Gupta, *Advances in Fuzzy Set Theory and Application,* North-Holland, Amsterdam, 1987.

K. Hirota, ed., *Industrial Applications of Fuzzy Technology*, Springer-Verlag, Tokyo, 1993.

J.-S. R. Jang, C.-T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing,* Prentice Hall, Upper Saddle River, NJ, 1997.

J. Kacprzyk, *Multistage Decision Making under Fuzziness*, Verlag TUV Reinland, Koeln, 1983.

J. Kacprzyk and M. Fedrizzi, eds., *Combining Fuzzy Imprecision with Probabilistic Uncertainty in Decision Making,* Springer-Verlag, Berlin, 1988.

A. Kandel, *Fuzzy Techniques in Pattern Recognition,* John Wiley, New York, 1982.

A. Kandel, *Fuzzy Expert Systems*, CRC Press, Boca Raton, FL, 1992.

A. Kandel and G. Langholz, eds., *Fuzzy Control Systems*, CRC Press, Boca Raton, FL, 1994.

W. Karwowski and A. Mital, eds., *Applications of Fuzzy Set Theory in Human Factors*, Elsevier Science Publishing Company, Inc., Amsterdam, 1986.

A. Kaufmann and M. M. Gupta, *Fuzzy Mathematical Models in Engineering and Management Science*, North-Holland, Amsterdam, 1988.

A. Kaufmann and M. M. Gupta, *Introduction to Fuzzy Arithmetic Theory and Application,* Van Nostrand Reinhold, New York, 1991.

G. J. Klir, and T. A. Folger, *Fuzzy Sets, Uncertainty, and Information*, Prentice Hall, Englewood Cliffs, NJ, 1988.

G. J. Klir, and Bo Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice Hall, Upper Saddle River, NJ, 1995.

G. J. Klir, Ute H. St. Clair, and Bo Yuan , *Fuzzy Set Theory*, Prentice Hall, Upper Saddle River, NJ, 1997.

B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice Hall, Englewood Cliffs, NJ, 1991.

B. Kosko, *Fuzzy Thinking, the New Science of Fuzzy Logic*, Hyperion, New York, 1993.

T. Kubinski, Nazwy nieostre, *Studia Logica,* vol. 7, pp. 115-179, 1958.

T. Kubinski, An attempt to bring logic near to colloquial language, *Studia Logica*, vol. 10, pp. 61-75, 1960.

H. Kwakernaak, Fuzzy random variables, Part I: Definitions and Theorems, *Information Science,* vol. 15, pp. 1-15, 1978.

C. C. Lee, Fuzzy logic in control systems: fuzzy logic controller, part I, *IEEE Trans.System, Man, Cybernerics,* vol. 20, no. 2, pp. 404-418, 1990.

C. C. Lee, Fuzzy logic in control systems: fuzzy logic controller, part II, *IEEE Trans.System, Man, Cybernerics,* vol. 20, no. 2, pp. 419-435, 1990.

R. Lowen, The relation between filter and net convergence in topological spaces, *Fuzzy Math.,* vol. 3, no. 4, pp. 41-53, 1983.

R. Maeder, *Programming in Mathematica*, Second Edition, Redwood City, California, 1991.

E. H. Mamdani, A fuzzy rule-based method of controlling dynamic processes, *Proc. 20th IEEE Conf. on Decision and Control*, San Diego, 1981.

R. J. Marks II, ed., *Fuzzy Logic Technology and Applications*, IEEE Press, New York, 1994.

D. McNeil and P. Freiberger, *Fuzzy Logic,* Touchstone-Simon and Schuster, New York, 1994.

C. Negoita and D. Ralescu, *Simulation, Knowledge-Based Computing and Fuzzy Statistics,* Van Nostrand Reinhold Company Inc., New York, 1987.

H. T. Nguyen and E. A. Walker, *A First Course in Fuzzy Logic,* CRC Press, 1997.

K. M. Passino and S. Yurkovich, *Fuzzy Control,* Addison-Wesley, Menlo Park, CA, 1998.

W. Pedrycz, *Fuzzy Sets Engineering*, CRC Press, Boca Baton, FL, 1995.

W. Pedrycz and F. Gomide, *An Introduction to Fuzzy Sets: Analysis and Design,* The MIT Press, 1998.

T. J. Ross, *FuzzyLogic with Engineering Applications.* McGraw-Hill, Hightstown, NJ, 1995.

E. Sanchez, Resolution of composite fuzzy relation equations, Information and Control, vol. 30, pp. 38-48, 1976.

B. Soucek and The IRIS Group, eds., *Fuzzy, Holographic, and Parallel Intelligence*, John Wiley & Sons, Inc., New York, 1992.

M. S. Stachowicz and M. E. Kochanska, *Graphic interpretation of fuzzy sets and fuzzy relations*, *Mathematics at the Service of Man,* Edited by A. Ballester, D. Cardus, and E. Trillas, based on materials of Second World Conf., Universidad Politecnica Las Palmas, Spain, 1982.

M. S. Stachowicz and M. E. Kochanska, Graphic interpretation of fuzzy knowledge, *Proc. First International Fuzzy Systems Association Congress*, Palma de Mallorca, July 1-6, 1985.

M. S. Stachowicz and M. E. Kochanska, Interpretation of fuzzy relations, *Robotyka i Maszynowa Inteligencja,* Prace Naukowe I. C. T., P. W. z. 65, s.: Konferencje nr. 24, vol. 1, Edited by W. Findeisen, A. Morecki, et al., WPW, Wroclaw, 1985.

M. S. Stachowicz and M.E. Kochanska, Fuzzy modeling of the process, *Proc.of Second International Fuzzy Systems Association Congress,* Tokyo, 1987.

M. S. Stachowicz and K. J. Reid, *Fuzzy sets and intelligent control, Matrials Processing in the Computer Age,* Edited by V. R. Voller, M. S. Stachowicz, and B. G. Thomas, The Mineral, Metals & Materials Society, 1991.

M. Sugeno, ed., *Industrial Application of Fuzzy Control,* North-Holland, New York, 1985.

T. Terano, K. Asai, and M. Sugeno, eds., *Fuzzy Systems Theory and Its Applications,* Academic Press, Inc., San Diego, 1992.

T. Terano, K. Asai, and M. Sugeno, eds., *Applied Fuzzy Systems*, AP Professional, Cambridge, MA, 1994.

K. Tanaka, *An Introduction to Fuzzy Logic for Practical Applications,* Springer-Verlag, New York, 1997.

R. M. Tong, Synthesis of fuzzy models for industrial processes, *Int. J. General Systems,* vol. 4, pp. 143-162, 1978.

Ch. de la Valle Poussin, *Integrales de Lebesque, fonction d'ensemble, classes de Baire* , 2-e ed., Gauthier-Villars, Paris, 1950.

Li-Xin Wang, *Adaptive Fuzzy Systems and Control,* Prentice-Hall, Englewood Cliffs, NJ, 1994.

Li-Xin Wang, *A Course in Fuzzy Systems and Control,* Prentice-Hall, Englewood Cliffs, NJ, 1997.

Z. Wang and G. J. Klir, *Fuzzy Measure Theory,* Plenum Press, New York, 1992.

S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer,* Second Edition, Addison-Wesley, Reading, MA, 1991.

R. R. Yager, An approach to inference in approximate reasoning, *Int. J. Man-Machine Studies,* vol. 13, pp. 323-338, 1980.

R. R. Yager, D. P. Filev, *Essentials of Fuzzy Modeling and Control,* J. Wiley & Sons, Inc., New York, 1994.

J. Yen and R. Langari , *Fuzzy Logic - Intelligence, Control, and Information*, Prentice Hall, Upper Saddle River, NJ, 1999

L. A. Zadeh, Fuzzy sets, *Information and Control,* vol. 8, pp. 338-353, 1965.

L. A. Zadeh, Probability measure of fuzzy events, *J. Math. Anal. Appl.*, vol. 12, pp. 421-427, 1968.

L. A. Zadeh, Similarity relations and fuzzy ordering, *Information Science,* vol. 3, pp. 171-200, 1971.

L. A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, *IEEE Trans. Systems, Man , Cybernetics*, vol. 3, pp. 22-44, 1973.

L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning –I, *Information Sciences*, vol. 8, pp. 199-249, 1975.

L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning –II, *Information Sciences*, vol. 8, pp. 301-357, 1975.

L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning –III, *Information Sciences*, vol. 9, pp. 43-80, 1975.

L. A. Zadeh, A computational approach to fuzzy quantifiers in natural languages, *Comput. and Math.*, vol. 9, no. 1, pp. 149-184, 1983.

M. Zemankova-Leech and A. Kandel, *A Fuzzy Relational Data Bases – A Key to Expert Systems*, Verlag TUV Reinland, Koln, 1984.

H. J. Zimmermann, *Fuzzy Set Theory and Its Applications*, 2nd ed., Kluwer Academic Publishers, Boston, MA, 1991.

H. J. Zimmermann, *Fuzzy Set Theory and Its Applications*, 3rd ed., Kluwer Academic Publishers, Boston, MA, 1996.

Part 2
Demonstration Notebooks

1 Sets Versus Fuzzy Sets

1.1 Introduction

This notebook deals with fuzzy sets defined as functions that relate values from the membership function space to the elements from the object space. Based on the structure of the membership function's value, you can define crisp sets and fuzzy sets. An attempt is made to perform a uniform graphical interpretation of the above mentioned types of sets. It is assumed that each of these sets is a finite set. The elements from the objects space are arranged in sequence. A vertical lines are drawn, whose lengths correspond to the value of the membership function.

Apart from presentation of crisp sets and fuzzy sets, the basic properties are illustrated by the same graphical method using the *Fuzzy Logic* package, along with standard *Mathematica* functions.

To demonstrate fuzzy set theory, we use many functions from the *Fuzzy Logic* package, along with standard *Mathematica* functions. The *Fuzzy Logic* package contains numerous functions for working with fuzzy sets and fuzzy logic; this notebook demonstrates only a few of the functions that it contains.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

1.2 Characteristic Function and Membership Function

For a long time humankind has endeavored to understand the laws of the surrounding world and has made continuous attempts to describe the phenomena occurring in the world. Naturally we want to achieve the most adequate descriptions by means of exact and precise terms. Mathematical language should be the best tool to express such descriptions; however, the language of set theory and extensional logic is sometimes insufficient.

Let's recall that in classical set theory, which originated in the work of George Cantor during the years 1871-1883, the notions "element" and "set" and the relation "is an element of" are undefined concepts. Thus, sets are defined by a simple statement describing whether a particular element having a certain property

belongs to a particular set. When we consider set X contained in an universal space U, also referred to as the universe of discourse, we can state unequivocally whether each element u of space U is or is not an element of X. Set X is well described by the so-called characteristic function X, introduced by Charles de la Vallee-Poussin [Poussin 1950]. This function, defined on the universal space U, assumes a value of 1 for those elements u that belong to set X, and value of 0 for those elements u that do not belong to set X .

$$X : U \to \{ 0 , 1 \}$$
$$X(u) = 1, \quad u \text{ is a member of } X$$
$$X(u) = 0, \quad u \text{ is not a member of } X$$

By using mathematical apparatus based on classical set theory, we can describe only "sharp" situations, that is, situations in which there is no doubt as to what is true and what is false, in which there is a sharp boundary between elements having a certain property and other elements of the universal space.

Consider, for example, space U consisting of natural numbers less than or equal to 12.

$$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$$

Then, the set of prime numbers could be described as follows.

$$\texttt{PRIME} = \{u \text{ contained in } U \mid u \text{ is a prime number}\}$$

The elements of set PRIME are defined unequivocally in the following manner.

```
In[2]:= PRIME = {2, 3, 5, 7, 11}

Out[2]= {2, 3, 5, 7, 11}
```

Frequently, we have to deal with "unsharp" phenomena—imprecise situations in which it is difficult to find a boundary between truth and falsehood. Let us consider, for example, the same set of objects U as previously, and within it let us distinguish the set of small numbers SMALL. How can we do it? We can, of course, say that 4 is less than 5, but does that mean that 4 is small and 5 is not? If there are any difficulties with assigning membership in such a simple case, then we can imagine how difficult it is to describe complex systems involving human linguistic descriptions. Aristotelian two-valued logic is ineffective in such cases.

In the 1950s, Kubinski used the notion of a vague term and the notion of an unsharp set [Kubinski 1958, 1960]. This notion was applied to sets in which transition from full membership to non-membership was gradual. Such sets had no sharp boundaries.

In the case of the considered set SMALL, the transition from full membership (e.g., 1 is a member of SMALL) to the lack of membership (e.g., 12 is not a member of SMALL) is smooth. Thus SMALL can not be described by the characteristic function assuming values in the set {0,1}.

Sets X, like SMALL, which have unsharp boundaries, are well characterized by a function that assigns a real number from the closed interval from 0 to 1 to each element u in the set U.

$$X : U \rightarrow [\,0\,,\,1\,]$$

This function, introduced by Professor Lotfi Zadeh in 1965 and called a membership function [Zadeh 1965], describes to what degree an element u belongs to set X; this is called a fuzzy set. To avoid a complex description like, "a fuzzy set X characterized by the membership function X," J. Goguen [Goguen 1967] proposed to identify a fuzzy set with a function that characterizes it and introduced the following definition:

> A fuzzy set X defined in the universal space U is a function defined in U which assumes values in the range [0, 1]. A fuzzy set X will be written as a set of pairs {u, X(u)}:
>
> X = {{u , X(u)}}, u in the set U

where u is an element of the universal space U, and X(u) is the value of the function X for this element. The value X(u) is the grade of membership of the element u in a fuzzy set X.

An empty fuzzy set is a function that for every u in the set U assumes the value of zero. Using the notation just described, we can write this set in the following way.

Empty Set = {{u , 0}}, u in the set U

Space U, treated as a fuzzy set, is a function equal to 1. Presenting set U in the form we just described, we have the following.

U = {{u , 1}}, u in the set U

A classical "sharp" set, A (a subset of U treated as a fuzzy set), is a function assuming the values 0 for u not contained in A and 1 for u contained in A. Set A is thus identified with its characteristic function.

Bearing in mind the concept of a fuzzy set, we shall now resume the presentation of set SMALL of small numbers in set U consisting of natural numbers less than or equal to 12.

Assume:

SMALL(1) = 1 SMALL (2) = 1 SMALL(3) = .9 SMALL(4) = .6
SMALL(5) = .4 SMALL(6) = .3 SMALL(7) = .2 SMALL(8) = .1
SMALL(u) = 0 for u >= 9

According to notion described earlier:

SMALL = {{1, 1}, {2, 1}, {3, .9}, {4, .6}, {5, .4}, {6, .3}, {7, .2}, {8, .1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}

The important point to note is that such a fuzzy set can be defined precisely by associating with each u its grade of membership in SMALL. Note also that the above assignment defining set SMALL is wholly subjective. This subjectivity in the evaluation of the grade of membership of particular elements in the set is characteristic of fuzzy sets.

In spite of allegations that such descriptions are subjective, it has an important advantage. It enables you to take into account the human experience and intelligence by translating imprecise natural language and human reasoning into a mathematical model for a system.

Subjectivity is also a feature distinguishing otherwise similar probability from membership functions. Lack of precision in determining data is sometimes caused by a lack of clearly defined criteria for classifying variables or parameters. This is a starting point for developing a method for solving such problems. Thus, we can repeat what Kwakernaak [Kwakernaak 1978] said: "with indeterminacy of a fuzzy type, we are dealing with a situation in which we admit the possibility that element u from space U fulfills the condition imposed only to a certain degree. The second type of indeterminacy, randomness, occurs when a given element u in the set U either fully fulfills the condition or does not fulfill it at all, but it is not possibly to determine which of these situations applies. Randomness is thus a result of cognitive indetermination.

It is important to note that in the case of a fuzzy set, it is not meaningful to speak of an object as belonging to or not belonging to that set, except for elements whose grades of membership in the set are unity or zero. A membership function describes to what degree element u belongs to set X, so we may say that everything is a matter of degree.

1.3 Graphic Interpretation of Sets

Let's restrict our consideration to a case when the universal space U is at the very most a countable set.

$$U = \{ u_i \}, \; i = 1, 2, ...$$

From the standpoint of practical applications, this restriction will not be troublesome. In most applications, especially when using computational techniques, a finite number of the elements of set U is taken into consideration.

An assumption of denumerability of the universal space permits a very simple graphic interpretation of the fuzzy sets defined in U. The elements of the collection of objects of space U, due to its denumerability, can be arranged in a sequence. You can plot for each element a segment of height corresponding to the value of the membership function of the given element in the fuzzy set under consideration.

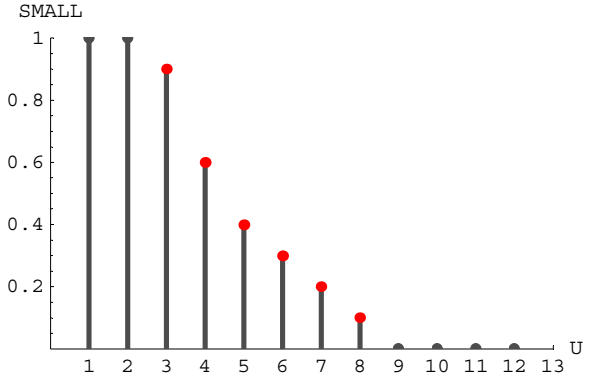
Universal spaces for fuzzy sets and fuzzy relations are defined with three numbers in this package. The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between discrete elements .

For example, the set considered earlier, fuzzy set SMALL of small numbers, defined in the space

$$U = \{ u_i \} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$$

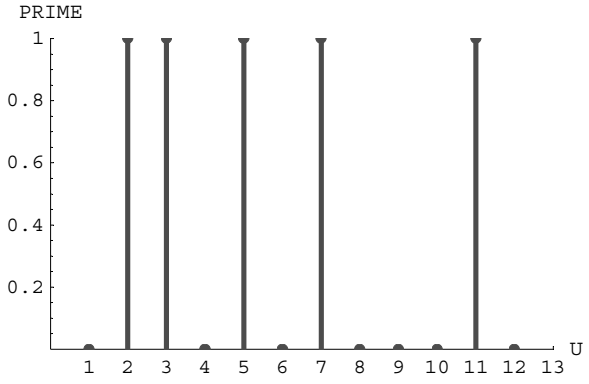
may be presented in the following way.

```
In[3]:= SetOptions[FuzzySet, UniversalSpace → {1, 12, 1}]

Out[3]= {UniversalSpace → {1, 12, 1}}

In[4]:= SetOptions[FuzzyPlot, PlotJoined -> False, ShowDots → True]

Out[4]= {PlotJoined → False, Crisp → False, ShowDots → True}

In[5]:= SMALL = FuzzySet[{{1, 1}, {2, 1}, {3, .9}, {4, .6}, {5, .4},
          {6, .3}, {7, .2}, {8, .1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}];

In[6]:= FuzzyPlot[SMALL, AxesLabel → {"U", "SMALL"}];
```



Set PRIME, the prime numbers, which is a classical subset of U, may be presented in the following way.

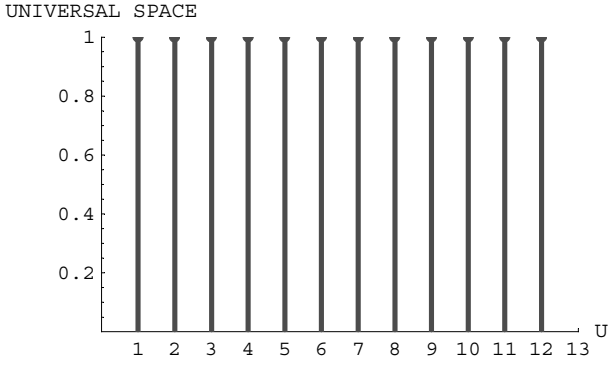
```
In[7]:= PRIME = FuzzySet[{{1, 0}, {2, 1}, {3, 1}, {4, 0},
            {5, 1}, {6, 0}, {7, 1}, {8, 0}, {9, 0}, {10, 0}, {11, 1}, {12, 0}}];

In[8]:= FuzzyPlot[PRIME, AxesLabel → {"U", "PRIME"}];
```



Universal Set

In any application of the theory of sets or fuzzy sets, all sets under investigation will likely be subsets of a fixed set. We call this set the universal space or universe of discourse. We denote this set by U. Universal space U, treated as a fuzzy set, is a function equal to 1 for all elements.

```
In[9]:= UNIVERSALSPACE = FuzzySet[{{1, 1}, {2, 1}, {3, 1}, {4, 1},
            {5, 1}, {6, 1}, {7, 1}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}}];

In[10]:= FuzzyPlot[UNIVERSALSPACE, AxesLabel → {"U", "UNIVERSAL SPACE"}];
```



Finite and Infinite Universal Space

Universal sets can be finite or infinite. Any universal set is finite if it consists of a specific number of different elements, that is, if in counting the different elements of the set, the counting can come to an end, otherwise a set is infinite.

Examples:

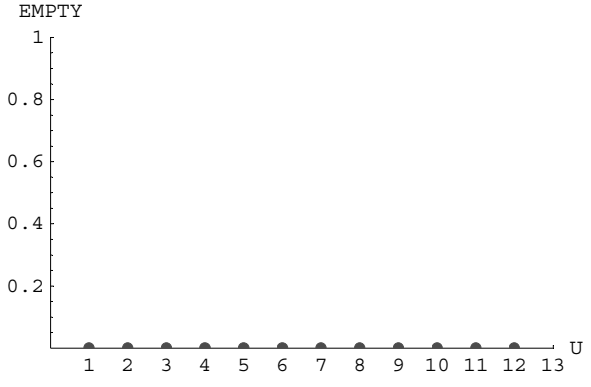
1. Let N be the universal space of the days of the week. N = {Mo, Tu, We, Th, Fr, Sa, Su}. N is finite.
2. Let M = {1, 3, 5, 7, 9, ...}. M is infinite.
3. Let L = {u | u is a lake in Minnesota}. Although it may be difficult to count the number of lakes in Minnesota, L is still a finite universal set.

Empty Set

It is convenient to introduce the concept of the empty set, that is, a set that contains only elements with a grade of membership equal to 0. For example, let `EMPTY` be a set of people in Minnesota who are older than 120. According to known statistics, `EMPTY` is an empty set. This set is sometimes called the null set.

```
In[11]:= EMPTY = FuzzySet[{}];
```

```
In[12]:= FuzzyPlot[EMPTY, AxesLabel → {"U", "EMPTY"}];
```



You can restore the default settings of the *Fuzzy Logic* package for `FuzzySet` and `FuzzyPlot`.

```
In[13]:= SetOptions[FuzzySet, UniversalSpace → {0, 20, 1}];
```

```
In[14]:= SetOptions[FuzzyPlot, PlotJoined → False, ShowDots → False];
```

References

J. A. Goguen, L-fuzzy sets, *J. Math. Anal. Appl.*, vol. 18, pp. 145-174, 1967.

G. J. Klir, and T. A. Folger, *Fuzzy Sets, Uncertainty, and Information*, Prentice Hall, Englewood Cliffs, NJ, 1988.

T. Kubinski, Nazwy nieostre, *Studia Logica*, vol. 7, pp. 115-179, 1958.

T. Kubinski, An attempt to bring logic near to colloquial language, *Studia Logica,* vol. 10, pp. 61-75, 1960.

H. Kwakernaak, Fuzzy random variables, Part I: Definitions and Theorems, *Information Science,* vol. 15, pp. 1-15, 1978.

M. S. Stachowicz and M. E. Kochanska, Graphic interpretation of fuzzy sets and fuzzy relations, *Mathematics at the Service of Man.* Edited by A. Ballester, D. Cardus, and E. Trillas, based on materials of Second World Conf., Universidad Politecnica Las Palmas, Spain, 1982.

Ch. de la Valle Poussin, *Integrales de Lebesque, fonction d'ensemble, classes de Baire,* 2-e ed., Paris, Gauthier-Villars, 1950.

L. A. Zadeh, Fuzzy sets, *Information and Control*, vol. 8, pp. 338-353, 1965.

L. A. Zadeh, Probability measure of fuzzy events, *J. Math. Anal. Appl.*, vol. 12, pp. 421-427, 1968.

L. A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, *IEEE Transactions on Systems, Man , Cybernetics*, vol. 3, pp. 22-44, 1973.

L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning –I, *Information Sciences,* vol. 8, pp. 199-249, 1975.

L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning –II, *Information Sciences,* vol. 8, pp. 301-357, 1975.

L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning –III, *Information Sciences,* vol. 9, pp. 43-80, 1975.

H. J. Zimmermann, *Fuzzy Set Theory and Its Applications,* 2nd ed., Kluwer Academic Publishers, Boston, MA, 1991.

2 Standard Operations

2.1 Introduction

This notebook deals with fuzzy operations defined in the same universal space. Professor Lotfi A. Zadeh [Zadeh 1965] formulated a fuzzy set theory in the terms of the standard operations: complement, union, intersection, and difference.

In this notebook, we present a graphical interpretation [Stachowicz and Kochanska, 1982] of the standard fuzzy set terms using standard *Mathematica* functions and functions from the *Fuzzy Logic* package.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

With the necessary routines loaded, we are ready to investigate fuzzy set theory.

2.2 Fuzzy Operations

Inclusion

Let X and Y be fuzzy sets defined in the same universal space U. We say that the fuzzy set X is included in the fuzzy set Y if and only if:

For every u in the set U we have $X(u) \leq Y(u)$

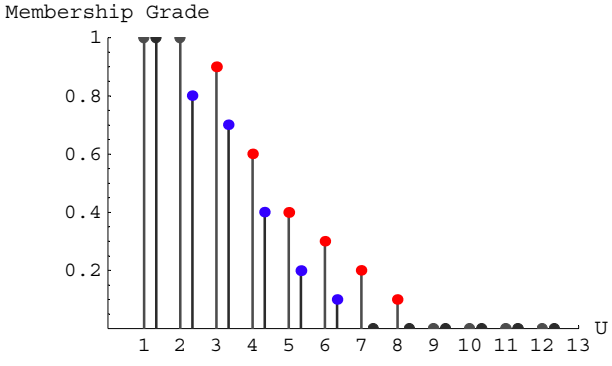
To illustrate inclusion of fuzzy sets, we consider the space U and fuzzy sets defined in U. We demonstrate this property in the following example.

```
In[2]:= SetOptions[FuzzySet, UniversalSpace → {1, 12}]

Out[2]= {UniversalSpace → {1, 12, 1}}
```

```
In[3]:= SetOptions[FuzzyPlot, ShowDots → True]

Out[3]= {PlotJoined → False, Crisp → False, ShowDots → True}

In[4]:= SMALL = FuzzySet[{{1, 1}, {2, 1}, {3, .9}, {4, .6}, {5, .4},
            {6, .3}, {7, .2}, {8, .1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}];

In[5]:= VERYSMALL = FuzzySet[{{1, 1}, {2, .8}, {3, .7}, {4, .4},
            {5, .2}, {6, .1}, {7, 0}, {8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}];
```

VERYSMALL is included in SMALL as defined by the previous definition of inclusion. This inclusion will be directly seen in the next figure.

```
In[6]:= Included[VERYSMALL, SMALL]

Out[6]= True

In[7]:= FuzzyPlot[SMALL, VERYSMALL];
```



It should also be noted that each fuzzy set X defined in the universal space U is a fuzzy subset of the fuzzy set U. We have U(u) = 1, and X(u) is contained in the interval [0, 1] for each element u in the set U, hence X(u) ≤ U(u). Similarly the empty set is a fuzzy subset of all other fuzzy sets.

```
In[8]:= UNIVERSALSPACE = FuzzySet[{{1, 1}, {2, 1}, {3, 1}, {4, 1},
            {5, 1}, {6, 1}, {7, 1}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}}];

In[9]:= EMPTY = FuzzySet[{}];

In[10]:= Included[SMALL, UNIVERSALSPACE]

Out[10]= True
```

```
In[11]:= Included[VERYSMALL, UNIVERSALSPACE]

Out[11]= True

In[12]:= Included[EMPTY, SMALL]

Out[12]= True
```

Comparability

Two fuzzy sets A and B are said to be comparable if the following condition holds:

$$A \subset B \text{ or } B \subset A$$

That is, if one of the fuzzy sets is a subset of the other set, they are comparable. Two sets A and B are said to be incomparable in the following case.

$$A \not\subset B \text{ and } B \not\subset A$$

Example 2.1

Let A = {{a, 1}, {b, 1}, {c, 0}} and B = {{a, 1}, {b, 1}, {c, 1}}. Then A is comparable to B, since A is a subset of B.

Example 2.2

Let C = {{a, 1}, {b, 1}, {c, 0.5}} and D = {{a, 1}, {b, 0.9}, {c, 0.6}}. Then C and D are not comparable since C is not a subset of D and D is not a subset of C.

Property Related to Inclusion

In mathematics, many statements can be proven to be true by the use of previous definitions or assumptions. Here we prove a theorem of fuzzy sets using the definition of inclusion.

Theorem:

If $A \subset B$ and $B \subset C$ then $A \subset C$.

Proof:

Notice that we must show that

$$A(u) \leq C(u) \qquad \text{for all u in the set U.}$$

Since A ⊂ B then

$$A(u) \le B(u) \qquad \text{for all u in the set U.}$$

By hypothesis, B ⊂ C; hence

$$B(u) \le C(u) \qquad \text{for all u in the set U.}$$

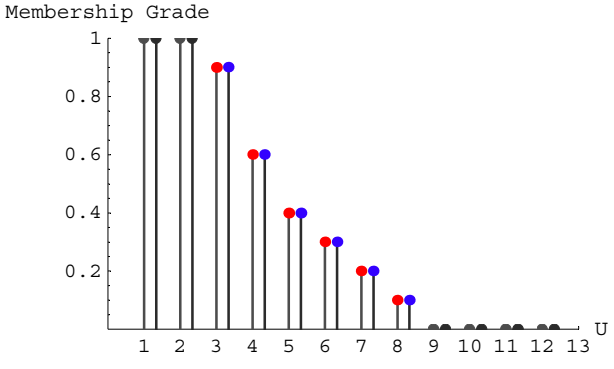
So, we have shown that for all u in the set U, if A(u) ⊂ B(u) ⊂ C(u), then accordingly A ⊂ C.

Equality

Let X and Y be fuzzy sets defined in the same space U. We say that sets X and Y are equal, which is denoted X = Y if and only if for all u in the set U, X(u) = Y(u).

We demonstrate this property in the following example.

```
In[13]:= SMALL = FuzzySet[{{1, 1}, {2, 1}, {3, .9}, {4, .6}, {5, .4},
            {6, .3}, {7, .2}, {8, .1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}];

In[14]:= STILLSMALL = FuzzySet[{{1, 1}, {2, 1}, {3, .9}, {4, .6},
            {5, .4}, {6, .3}, {7, .2}, {8, .1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}];

In[15]:= FuzzyPlot[SMALL, STILLSMALL];
```



We see that by definition, SMALL = STILLSMALL.

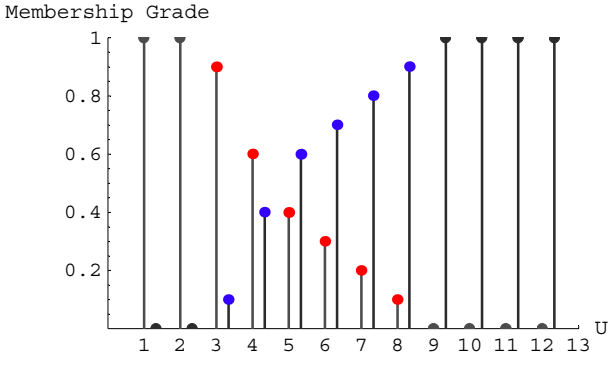
If equality X(u) = Y(u) is not satisfied even for one element u in the set U, then we say that X is not equal to Y.

Complementation

Let X be a fuzzy set defined in the space U. We say that the fuzzy set Y is a complement of the fuzzy set X, if and only if, for all u in the set U, Y(u) = 1 - X(u).

The complement of the fuzzy set X is often denoted by X'. For example, the complement of the fuzzy set SMALL in the space U is a fuzzy set NOTSMALL. We show both SMALL and NOTSMALL in the following graph.

```
In[16]:= NOTSMALL = Complement[SMALL]

Out[16]= FuzzySet[{{3, 0.1}, {4, 0.4}, {5, 0.6}, {6, 0.7}, {7, 0.8},
           {8, 0.9}, {9, 1}, {10, 1}, {11, 1}, {12, 1}}, UniversalSpace → {1, 12, 1}]

In[17]:= FuzzyPlot[SMALL, NOTSMALL];
```

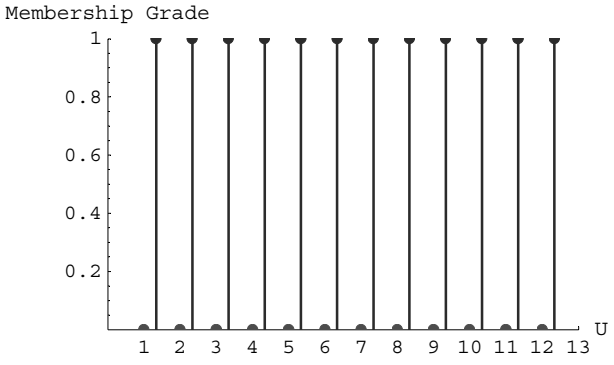


We see that there are many elements that can have some nonzero grades of membership in both a fuzzy set, SMALL, and its complement, NOTSMALL.

The empty set and the universal set, treated as fuzzy sets, are complements of one another.

$$\emptyset' = U$$
$$U' = \emptyset$$

```
In[18]:= EMPTY = Complement[UNIVERSALSPACE];
         FuzzyPlot[EMPTY, UNIVERSALSPACE];
```



Union

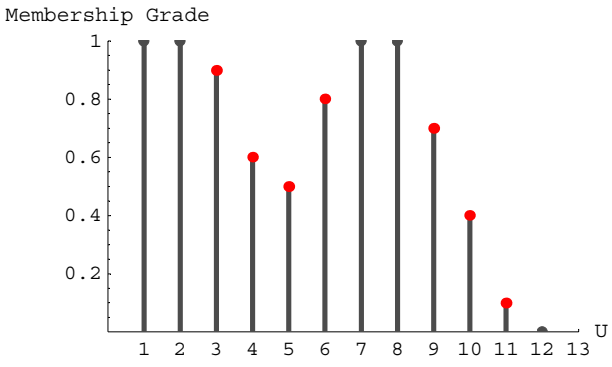
Let X and Y be fuzzy sets defined in the space U. We define the union of those sets as the smallest (in the sense of the inclusion) fuzzy set that contains both X and Y. The union of X and Y will be denoted by $X \bigcup Y$. Thus the following relation must be satisfied for the union operation.

For all u in the set U, $(X \bigcup Y)(u) = \text{Max}(X(u), Y(u))$

For example, for fuzzy sets `SMALL` and `MEDIUM`, which we define here, the `Union` returns the following.

```
In[20]:= MEDIUM = FuzzySet[{{1, 0}, {2, 0}, {3, 0}, {4, .2}, {5, .5},
             {6, .8}, {7, 1}, {8, 1}, {9, .7}, {10, .4}, {11, .1}, {12, 0}}];

In[21]:= UNION = SMALL ⋃ MEDIUM

Out[21]= FuzzySet[{{1, 1}, {2, 1}, {3, 0.9}, {4, 0.6}, {5, 0.5}, {6, 0.8}, {7, 1},
           {8, 1}, {9, 0.7}, {10, 0.4}, {11, 0.1}}, UniversalSpace → {1, 12, 1}]
```

```
In[22]:= FuzzyPlot[UNION];
```



Example

Thus, if at a point u = 6, the following is true

$$\texttt{SMALL}(6) = 0.3 \quad \text{and} \quad \texttt{MEDIUM}(6) = 0.8$$

then at u = 6, the union is 0.8.

As in the case of non-fuzzy sets, the notion of the union is closely related to that of the connective "or." Thus, if A is a class of "Young" men, B is a class of "Bald" men, and "David is Young" or "David is Bald," then David is associated with the union of A and B.

David is a member of set A or David is a member of set B implies David is a member of $A \bigcup B$.

Properties Related to Union

A collection of the properties related to union follow. We demonstrate each property using some of the fuzzy sets we have created in this notebook and with functions from the *Fuzzy Logic* package.

Identity:

$$X \bigcup \emptyset = X$$

```
In[23]:= Equality[SMALL⋃EMPTY, SMALL]

Out[23]= True
```

Identity:

$$X \bigcup U = U$$

```
In[24]:= Equality[SMALL⋃UNIVERSALSPACE, UNIVERSALSPACE]

Out[24]= True
```

Idempotence:

$$X \bigcup X = X$$

```
In[25]:= Equality[SMALL⋃SMALL, SMALL]

Out[25]= True
```

Commutativity:

$$X \bigcup Y = Y \bigcup X$$

```
In[26]:= Equality[SMALL⋃MEDIUM, MEDIUM⋃SMALL]

Out[26]= True
```

Associativity:

$$X \bigcup (Y \bigcup Z) = (X \bigcup Y) \bigcup Z$$

```
In[27]:= BIG = FuzzySet[{{1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0},
            {6, .1}, {7, .2}, {8, .4}, {9, .6}, {10, .8}, {11, 1}, {12, 1}}];

In[28]:= Equality[SMALL⋃(MEDIUM⋃BIG), (SMALL⋃MEDIUM)⋃BIG]

Out[28]= True
```

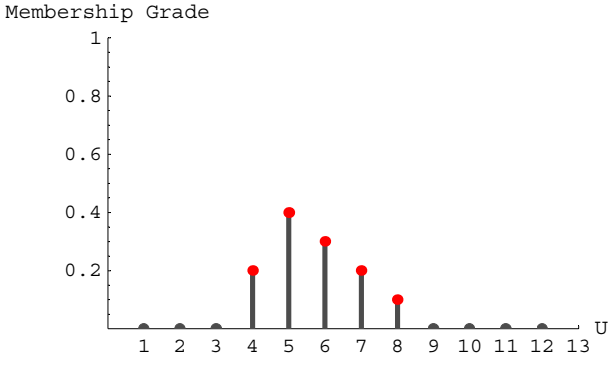
Intersection

Let X and Y be fuzzy sets in the space U. The intersection of those sets, denoted by $X \bigcap Y$, is defined as the greatest (in the sense of the inclusion) fuzzy set included both in X and Y. Thus the relation the intersection must satisfy the following property.

For all u in the set U, $(X \bigcap Y)(u) = \text{Min}(X(u), Y(u))$.

For the sets SMALL and MEDIUM from the previous example, we find the intersection is the following manner.

```
In[29]:= INTERSECTION  = SMALL⋂MEDIUM

Out[29]= FuzzySet[{{4, 0.2}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}},
          UniversalSpace → {1, 12, 1}]

In[30]:= FuzzyPlot[INTERSECTION];
```



Additional Properties Related to Intersection and Union

This section contains a collection of additional properties related to fuzzy intersection and union.

Absorption by Empty Set:

$$X \cap \phi = \phi$$

```
In[31]:= Equality [SMALL⋂EMPTY, EMPTY]

Out[31]= True
```

Identity:

$$X \cap U = X$$

```
In[32]:= Equality [SMALL⋂UNIVERSALSPACE, SMALL ]

Out[32]= True
```

Idempotence:

$$X \cap X = X$$

```
In[33]:= Equality [SMALL ⋂ SMALL, SMALL ]

Out[33]= True
```

Commutativity:

$$X \cap Y = Y \cap X$$

```
In[34]:= Equality [SMALL ⋂ BIG, BIG ⋂ SMALL]

Out[34]= True
```

Associativity:

$$X \cap (Y \cap Z) = (X \cap Y) \cap Z$$

```
In[35]:= Equality [SMALL ⋂ (MEDIUM ⋂ BIG), (SMALL ⋂ MEDIUM) ⋂ BIG]

Out[35]= True
```

Distributivity:

$$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$$

```
In[36]:= Equality [SMALL ⋂ (MEDIUM ⋃ BIG), SMALL ⋂ MEDIUM ⋃ SMALL ⋂ BIG]

Out[36]= True
```

Distributivity:

$$X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$$

```
In[37]:= Equality[SMALL ⋃ MEDIUM ⋂ BIG, (SMALL ⋃ MEDIUM) ⋂ (SMALL ⋃ BIG)]

Out[37]= True
```

For crisp sets, we have the following properties.

Law of excluded middle:

$$A \cup A' = U$$

Law of contradiction:

$$A \cap A' = \emptyset$$

In fuzzy logic these properties do not apply.

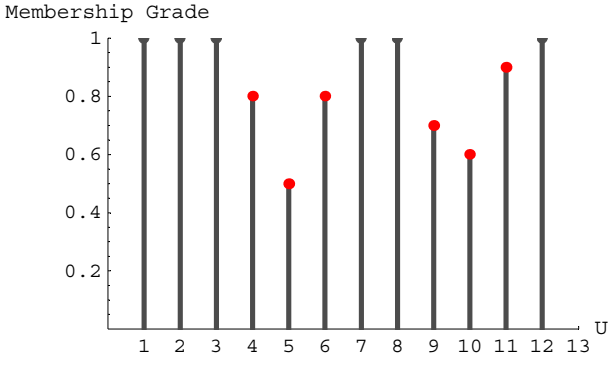
$$X \cup X' \neq U$$

$$X \cap X' \neq \emptyset$$

This is demonstrated in the following graphs.

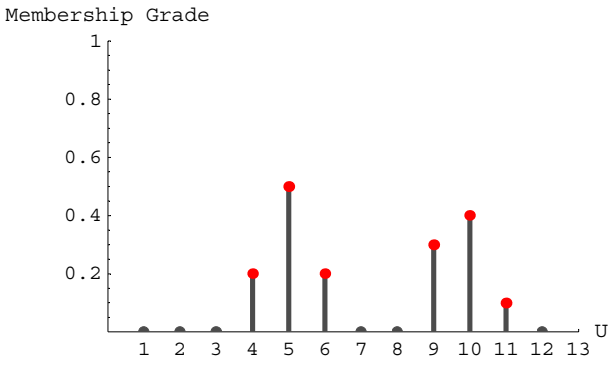
```
In[38]:= MYUNION = MEDIUM ⋃ Complement[MEDIUM];
```

```
In[39]:= FuzzyPlot [MYUNION ];
```




```
In[40]:= MYINTERSECTION  = MEDIUM ⋂ (Complement [MEDIUM ]);
```

```
In[41]:= FuzzyPlot [MYINTERSECTION ];
```



The theory of fuzzy sets was formulated in terms of the standard complement, union, and intersection operators. These original operators possess particular significance. When the range of grade of membership is restricted to the set {0,1}, these functions perform like the corresponding operators for Cantor's sets.

If any error e is associated with the grade of membership A(u) and B(u), then the maximum error associated with the grade of membership of u in A', $A \cup B$, and $A \cap B$ remains e.

Difference

The difference of two fuzzy sets is defined as the intersection of the minuend and the complement of the subtrahend.

$$X - Y = X \cap Y'$$

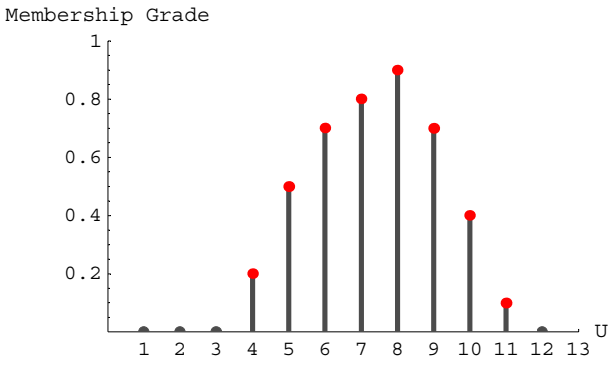
For the difference, each of the elements u in the set U satisfies the following relation.

$$\text{For all u in the set U, } (X - Y)(u) = \text{Min}(X(u), 1 - Y(u))$$

For our sets `SMALL` and `MEDIUM`, we have the following results.

```
In[42]:= MYDIFFERENCE  = Difference  [MEDIUM, SMALL]

Out[42]= FuzzySet[{{4, 0.2}, {5, 0.5}, {6, 0.7}, {7, 0.8}, {8, 0.9},
           {9, 0.7}, {10, 0.4}, {11, 0.1}}, UniversalSpace → {1, 12, 1}]
```

```
In[43]:= FuzzyPlot [MYDIFFERENCE ];
```



Properties Related to Difference

This section contains properties related to the fuzzy difference. Properties are demonstrated using previously defined fuzzy sets and functions from the Fuzzy Logic package.

$$X - Y \cup Z = (X - Y) \cup (X - Z)$$

```
In[44]:= Equality[Difference[SMALL, MEDIUM⋃BIG],
          (Difference[SMALL, MEDIUM]) ⋂ (Difference[SMALL, BIG])]

Out[44]= True
```

$$X - Y \cup Z = (X - Y) \cup (X - Z)$$

```
In[45]:= Equality[Difference[SMALL, MEDIUM⋂BIG],
          Difference[SMALL, MEDIUM] ⋃Difference[SMALL, BIG]]

Out[45]= True
```

$$X - Y \cup Z = (X - Y) - Z$$

```
In[46]:= Equality[Difference[SMALL, MEDIUM⋃BIG],
          Union[Difference[Difference[SMALL, MEDIUM], BIG]]]

Out[46]= True
```

For crisp sets we have the following properties.

$$X \cup (Y - X) = X \cup Y$$

$$X - (X - Y) = X \cup Y$$

With fuzzy sets, this is not the case. For fuzzy sets, we have these properties.

$$X \cup (Y - X) = X \cup (Y \cap X') = (X \cup Y) \cap (X \cup X')$$

$$X - (X - Y) = X - X \cap Y' = X \cap (X \cup Y')' = X \cap (X \cup Y) = (X \cup X') \cap (X \cup Y)$$

You can restore the default settings of the *Fuzzy Logic* for `FuzzySet` and `FuzzyPlot`.

```
In[47]:= SetOptions[FuzzySet, UniversalSpace → {0, 20}];

In[48]:= SetOptions[FuzzyPlot, ShowDots → False];
```

References

G. J. Klir and T. A. Folger, *Fuzzy Sets, Uncertainty, and Information*, Prentice Hall, Englewood Cliffs, NJ, 1988.

M. S. Stachowicz and M. E. Kochanska, Graphic interpretation of fuzzy sets and fuzzy relations, *Mathematics at the Service of Man.* Edited by A. Ballester, D. Cardus, and E. Trillas, based on materials of Second World Conference, Universidad Politecnica Las Palmas, Spain, 1982.

L. A. Zadeh, Fuzzy sets, *Information and Control*, vol. 8, pp. 338-353, 1965.

H. J. Zimmermann, *Fuzzy Set Theory and Its Applications,* 2nd ed., Kluwer Academic Publishers, Boston, MA, 1991.

3 Fuzzy Relations

3.1 Introduction

Fuzzy relations play an important role in fuzzy modeling, fuzzy diagnosis, and fuzzy control. They also have applications in fields such as psychology, medicine, economics, and sociology. In this notebook we define and discuss fuzzy relations. Beginning with a definition of fuzzy relations, we then talk about expressing fuzzy relations in terms of matrices and graphical visualizations. Later we discuss the various properties of fuzzy relations and operations that can be performed with fuzzy relations.

We illustrate the basic properties of fuzzy relations using the graphical functions of the *Fuzzy Logic* package, along with standard *Mathematica* functions.

This loads the package.

```
<< FuzzyLogic`
```

3.2 Fuzzy Relation Form

A fuzzy relation is characterized by the same two items as a fuzzy set. First is a list containing element and membership grade pairs, $\{\{v_1, w_1\}, R_{11}\}, \{\{v_1, w_2\}, R_{12}\}, \ldots, \{\{v_n, w_m\}, R_{nm}\}\}$. Note that the elements of the relation are defined as ordered pairs, $\{v_1, w_1\}, \{v_1, w_2\}, \ldots, \{v_n, w_m\}$. These elements are again grouped with their membership grades, $\{R_{11}, R_{12}, \ldots, R_{nm}\}$, which are values that range from 0 to 1, inclusive.

The second item characterizing fuzzy relations is the universal space. For relations, the universal space consists of a pair of ordered pairs, $\{\{V_{min}, V_{max}, c_1\}, \{W_{min}, W_{max}, c_2\}\}$. The first pair defines the universal space to be used for the first set under consideration in the relation, and the second pair defines the universal space for the second set. The following is an example showing how fuzzy relations are represented in this package.

Universal spaces for fuzzy sets and fuzzy relations are defined with three numbers in this package. The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between discrete elements. Here is an example.

```
FuzzyRelation[{{{1, 1}, 0.1}, {{1, 2}, 0.5}, {{2, 1}, 0.9}, {{2, 2},
0.7}, {{3, 1}, 0.4}, {{3, 2}, 1}}, UniversalSpace→{{0, 3, 1}, {0, 2,
1}}]
```

Assuming that V and W are two collections of objects, an arbitrary fuzzy set R, defined in the Cartesian product VxW, will be called a fuzzy relation in the space VxW.

R is thus a function defined in the space VxW, which takes values from the interval [0, 1].

$$R : V \times W \rightarrow [0, 1]$$

In the case where V = W, we have a binary fuzzy relation on single set V.

We can start our discussion by considering a countable collection of objects.

$V = \{v_i\}, i = 1, 2, ...$
$W = \{w_j\}, j = 1, 2, ...$

A fuzzy relation R can be represented in the following way:

$$R = \{\{\{v_i, w_j\}, R(v_i, w_j)\}\},\ i = 1, 2, ... ; j = 1, 2, ...$$

A matrix or graphic interpretation is a more convenient way to study fuzzy relations. We examine fuzzy relations in this notebook in a manner analogous to that introduced earlier for fuzzy sets.

Let V = {1, 2, 3} and W = {1, 2, 3, 4}.

A fuzzy relation R in VxW has the following definition.

```
R = FuzzyRelation[{{{1, 1}, 1}, {{1, 2}, .2}, {{1, 3}, .7}, {{1, 4}, 0},
    {{2, 1}, .7}, {{2, 2}, 1}, {{2, 3}, .4}, {{2, 4}, .8}, {{3, 1}, 0},
    {{3, 2}, .6}, {{3, 3}, .3}, {{3, 4}, .5}}, UniversalSpace → {{1, 3}, {1, 4}}];
```

This relation can be represented in the following two forms:

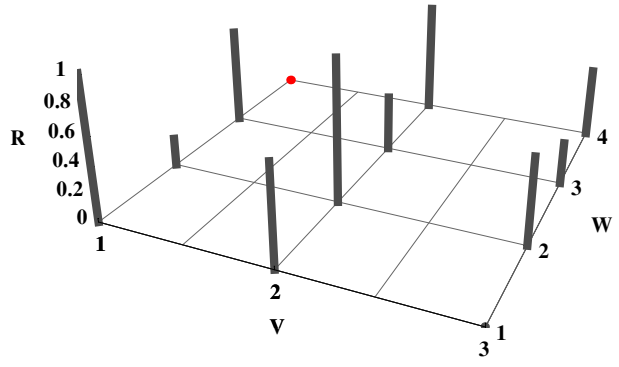
as a membership matrix

```
ToMembershipMatrix[R] // MatrixForm
```

$$\begin{pmatrix} 1 & 0.2 & 0.7 & 0 \\ 0.7 & 1 & 0.4 & 0.8 \\ 0 & 0.6 & 0.3 & 0.5 \end{pmatrix}$$

as a graph

```
FuzzyPlot3D[R, AxesLabel → {"V", "W", "R    "}, Boxed → False];
```



Note that the elements of the fuzzy relation are defined as ordered pairs; v_i is the first and w_j the second element of an ordered pair $\{v_i, w_j\}$. The membership grades of the elements are represented by the heights of the vertical lines at the corresponding elements.

3.3 Projection of a Fuzzy Relation

Let R be a fuzzy relation in the Cartesian product VxW. The fuzzy set R(1) defined in V by the following expression is called the first projection of fuzzy relation R.

For all v in the set V, R(1)(v) = Max(R(v, w)), for all w in the set W

The fuzzy set R(2) defined in W in the following expression is called the second projection of fuzzy relation R.

For all w in the set W, R(1)(w) = Max(R(v, w)), for all v in the set V

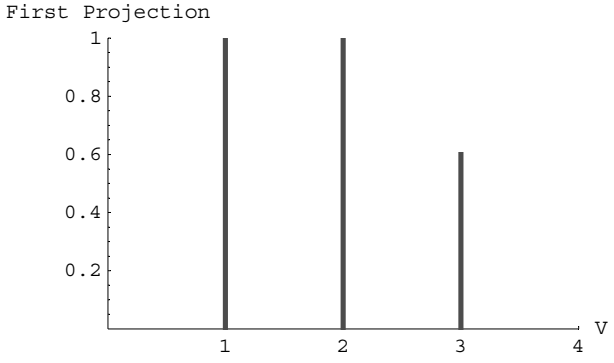
The number defined by the following expression is called the global projection of fuzzy relation R.

h(R) = Max(Max(R(v, w))), for all v in the set V and all w in the set W

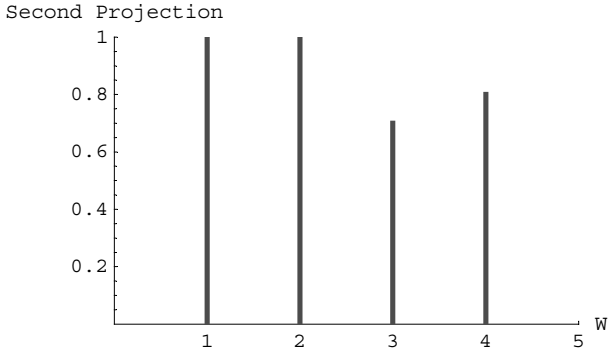
The fuzzy relation R: h(R) = 1 is called normal. If h(R) < 1, the relation is called subnormal.

For the fuzzy relation we defined earlier, the projections are shown here.

```
R1 = FirstProjection[R];

FuzzyPlot[R1, AxesLabel → {"V", "First Projection"}];
```




```
R2 = SecondProjection[R];

FuzzyPlot[R2, AxesLabel → {"W", "Second Projection"}];
```




```
h = GlobalProjection[R]

1
```

We see that this relation is normal since its global projection is 1.

3.4 Fuzzy Operations

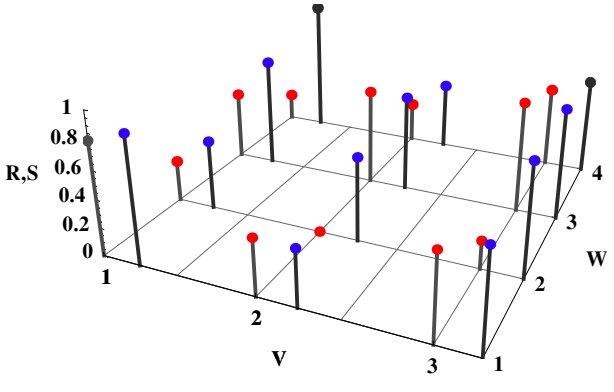
Here we present the basic properties of fuzzy relations and the basic operations that can be performed on fuzzy relations. We begin with operations defined for fuzzy relations in the same universal space—the same Cartesian product of two collections of objects.

Envelope of a Fuzzy Relation

If R and S are two fuzzy relations of VxW such that for all v and w in VxW, $R(v, w) \leq S(v, w)$, we say that fuzzy relation S is an envelope of fuzzy relation R or that fuzzy relation R is an enclosure of fuzzy relation S.

In the example that follows, fuzzy relations R and S are defined in VxW where sets V and W were defined earlier. Relations R and S satisfy the envelope condition stated here, so we can say that fuzzy relation R is an enclosure of S.

```
RMat = {{.8, .3, .5, .2}, {.4, 0, .7, .3}, {.6, .2, .8, .6}};

SMat = {{.9, .5, .8, 1}, {.4, .6, .7, .5}, {.7, .8, .8, .7}};

R = FromMembershipMatrix[RMat, {{1, 3}, {1, 4}}];

S = FromMembershipMatrix[SMat, {{1, 3}, {1, 4}}];

FuzzyPlot3D[R, S, AxesLabel → {"V", "W", "R,S  "}, ShowDots → True, Boxed → False];
```



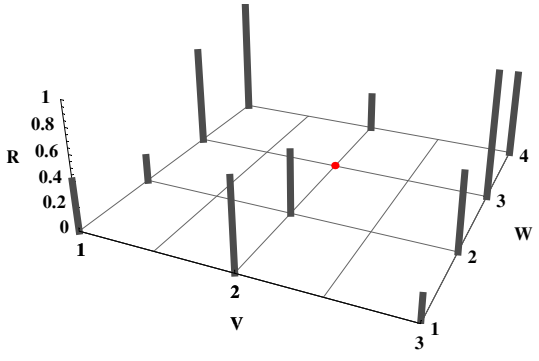
Union of Fuzzy Relations

Let R and S be fuzzy relations in VxW. The union of two fuzzy relations R and S, denoted by R $\cup$ S, is a fuzzy relation in V x W , such that for all (v, w) in V x W. Then

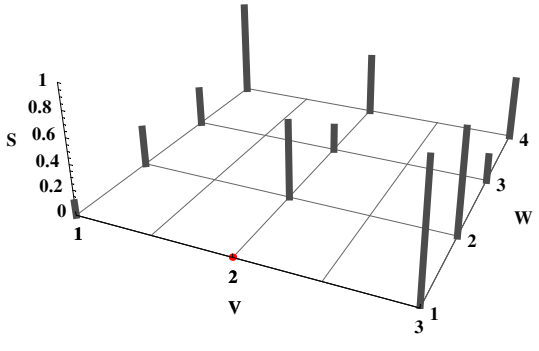
$$(R \cup S)(v, w) = \text{Max}(R(v, w), S(v, w)).$$

We demonstrate this operator in the following example.

```
RMATRIX = {{.4, .2, .8, 1}, {.7, .5, 0, .3}, {.2, .6, 1, .7}};

R = FromMembershipMatrix[RMATRIX, {{1, 3}, {1, 4}}];

FuzzyPlot3D[R, AxesLabel → {"V", "W", "R  "}, Boxed → False];
```




```
SMATRIX = {{.1, .3, .3, .8}, {0, .6, .2, .5}, {1, .8, .2, .5}};

S = FromMembershipMatrix[SMATRIX, {{1, 3}, {1, 4}}];

FuzzyPlot3D[S, AxesLabel → {"V", "W", "S  "}, Boxed → False];
```

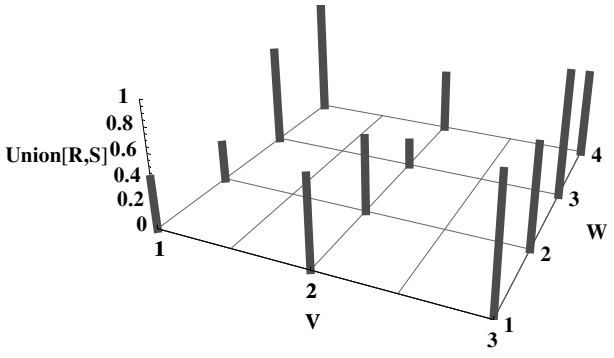


The union of fuzzy relations `R` and `S` is the following relation.

```
UNION = Union[R, S];

ToMembershipMatrix[UNION] // MatrixForm
```

$$\begin{pmatrix} 0.4 & 0.3 & 0.8 & 1 \\ 0.7 & 0.6 & 0.2 & 0.5 \\ 1 & 0.8 & 1 & 0.7 \end{pmatrix}$$

```
FuzzyPlot3D[UNION, AxesLabel → {"V", "W", "Union[R,S]"}, Boxed → False];
```



The operation of the union of fuzzy relations can be generalized to n relations. If R_1, R_2, ... , R_n are fuzzy relations in VxW then their union is a relation defined in VxW such that for all (v, w) in VxW, R(v, w) = Max(R_i(v, w)).

Intersection of Fuzzy Relations

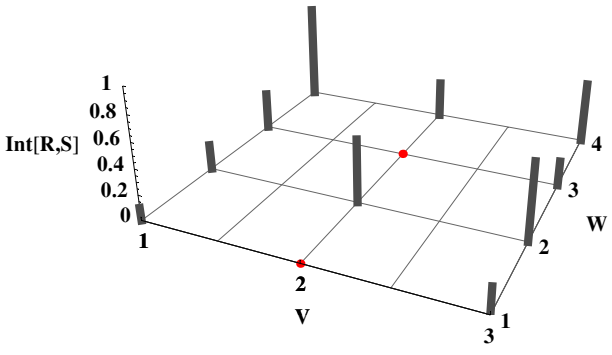
The intersection of two fuzzy relations, R and S, in VxW is the fuzzy relation in VxW such that for all (v, w) in VxW, $(R \cap S)(v, w) = \text{Min}(R(v, w), S(v, w))$.

The intersection of the fuzzy relations from the previous example is shown here.

```
INTERSECTION = Intersection[R, S];

ToMembershipMatrix[INTERSECTION] // MatrixForm
```

$$\begin{pmatrix} 0.1 & 0.2 & 0.3 & 0.8 \\ 0 & 0.5 & 0 & 0.3 \\ 0.2 & 0.6 & 0.2 & 0.5 \end{pmatrix}$$

```
FuzzyPlot3D[INTERSECTION, AxesLabel → {"V", "W", "Int[R,S]   "}, Boxed → False];
```



The intersection of the fuzzy relation can also be generalized to the intersection of n relations in the same way it was generalized for the union operation.

Algebraic Product of Fuzzy Relations

The algebraic product of the two fuzzy relations R and S in the space VxW is defined as a fuzzy set in VxW, whose elements satisfy the relation, for all (v, w) in VxW, AlgProduct(R, S)(v, w) = R(v, w) * S(v, w).

For the relations `R` and `S` presented earlier, the algebraic product is the following fuzzy relation.

```
AlgProduct = Intersection[R, S, Type → Hamacher[1]];

ToMembershipMatrix[AlgProduct] // MatrixForm
```

$$\begin{pmatrix} 0.04 & 0.06 & 0.24 & 0.8 \\ 0 & 0.3 & 0 & 0.15 \\ 0.2 & 0.48 & 0.2 & 0.35 \end{pmatrix}$$

Complement of a Fuzzy Relation

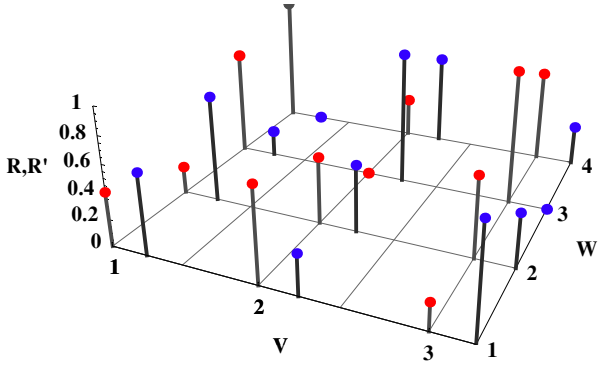
The complement of the fuzzy relation R is denoted by R'; the relation R' is defined in the same space as R. It is defined as, for all (v, w) in VxW, R'(v, w) = 1 - R(v, w).

The complement of fuzzy relation R is shown in the following graph.

```
COMPLEMENT = Complement[R];

ToMembershipMatrix[COMPLEMENT] // MatrixForm
```

$$\begin{pmatrix} 0.6 & 0.8 & 0.2 & 0 \\ 0.3 & 0.5 & 1 & 0.7 \\ 0.8 & 0.4 & 0 & 0.3 \end{pmatrix}$$

```
FuzzyPlot3D[R, COMPLEMENT, ShowDots → True,
  AxesLabel → {"V", "W", "  R,R'  "}, Boxed → False];
```



3.5 Composition of Two Fuzzy Relations

So far we have considered operations on fuzzy relations defined in the same space as the Cartesian product of two collections of objects. Now we shall consider the composition of two fuzzy relations.

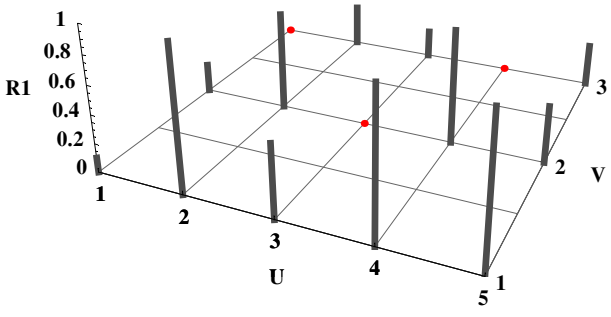
Let R_1 be a fuzzy relation in UxV and R_2 a fuzzy relation in VxW. We shall present two ways of composing such relations.

Max-Min Composition

The max-min composition of relations R_1 and R_2 denoted by MaxMin(R_1, R_2) is a fuzzy relation in UxW, such that for all (u, w) in UxW, MaxMin(R_1, R_2)(u, w) = Max(Min(R_1(u, v), R_2(v, w))) over all v in the set V.

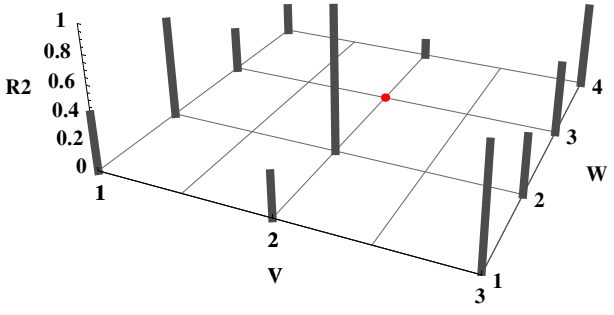
Consider for example U = {1, 2, 3, 4, 5}, V = {1, 2, 3}, W = {1, 2, 3, 4}, and the following fuzzy relations.

```
R1MAT = {{.1, .2, 0}, {1, .7, .3}, {.5, 0, .2}, {1, .8, 0}, {1, .4, .3}};

R1 = FromMembershipMatrix[R1MAT, {{1, 5}, {1, 3}}];

FuzzyPlot3D[R1, AxesLabel → {"U", "V", "R1   "}, Boxed → False];
```




```
R2MAT = {{.4, .7, .3, .2}, {.3, 1, 0, .1}, {.8, .4, .5, .6}};

R2 = FromMembershipMatrix[R2MAT, {{1, 3}, {1, 4}}];
```

```
FuzzyPlot3D[R2, AxesLabel → {"V", "W", "R2   "}, Boxed → False];
```

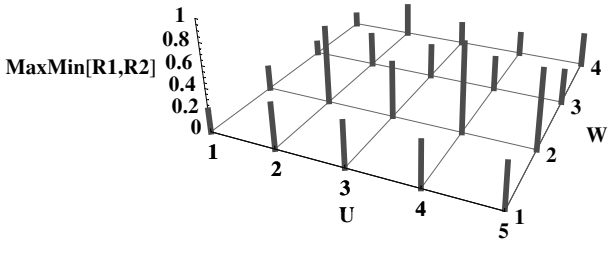


We can now find the composition of fuzzy relations `R1` and `R2`.

```
MAXMIN = Composition[R1, R2, Type → MaxMin];

ToMembershipMatrix[MAXMIN] // MatrixForm
```

$$\begin{pmatrix} 0.2 & 0.2 & 0.1 & 0.1 \\ 0.4 & 0.7 & 0.3 & 0.3 \\ 0.4 & 0.5 & 0.3 & 0.2 \\ 0.4 & 0.8 & 0.3 & 0.2 \\ 0.4 & 0.7 & 0.3 & 0.3 \end{pmatrix}$$

```
FuzzyPlot3D[MAXMIN, AxesLabel → {"U", "W", "MaxMin[R1,R2]   "}, Boxed → False];
```



Note that the new fuzzy relation, `MAXMIN`, is defined on the space UxW.

Max-Star Composition

We continue by considering the two fuzzy relations, R_1 in UxV and R_2 in VxW.

The max-star composition of such relations is the composition created by replacing the min operation in the previous composition definition by any other operation that is associative and monotonic non-decreasing in each argument. In the definition of composition, this operation is denoted by a star, hence the name of the composition. The max-star composition is defined such that for all (u, w) in UxW, MaxStar(R_1, R_2)(u, w) = Max(R_1(u, v)*R_2(v, w)) over all v in the set V.

If we take the algebraic product for the star operation, then we will obtain a composition referred to as the max-product, which is defined in the following way:

For all (u, w) in UxW,

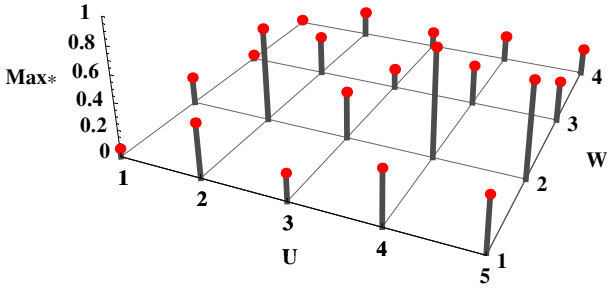
MaxProduct(R_1, R_2)(u, w) = Max(R_1(u, v) × R_2(v, w)) over all v in the set V.

For relation `R1` and `R2` from previous example, we obtain the fuzzy relation.

```
MAXSTAR = Composition[R1, R2, Type → MaxProduct];

ToMembershipMatrix[MAXSTAR] // MatrixForm
```

$$\begin{pmatrix} 0.06 & 0.2 & 0.03 & 0.02 \\ 0.4 & 0.7 & 0.3 & 0.2 \\ 0.2 & 0.35 & 0.15 & 0.12 \\ 0.4 & 0.8 & 0.3 & 0.2 \\ 0.4 & 0.7 & 0.3 & 0.2 \end{pmatrix}$$

```
FuzzyPlot3D[MAXSTAR, ShowDots → True, AxesLabel → {"U", "W", "Max*  "}, Boxed → False];
```



Again the new fuzzy relation is defined on the space UxW.

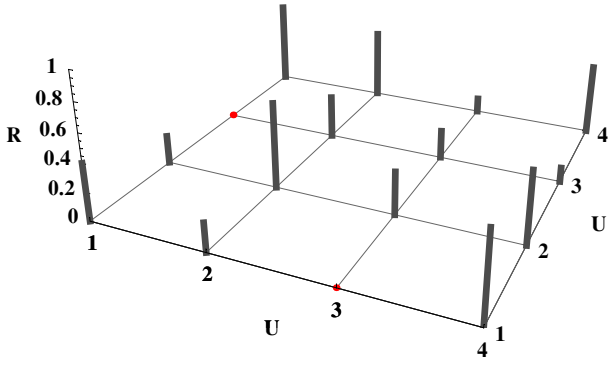
3.6 Binary Relations

We now consider binary fuzzy relations in U, that is, fuzzy relations defined in the Cartesian product UxU.

Symmetry

We say that a binary fuzzy relation R in U is symmetric if for all (u, v) in UxU, (R(u, v) = a) ⇒ (R(v, u) = a). An example of a symmetric, binary fuzzy relation in U = {u_1, u_2, u_3, u_4} is shown here.

```
RMATRIX = {{.4, .2, 0, .6}, {.2, .6, .3, .5}, {0, .3, .2, .1}, {.6, .5, .1, .5}};

R = FromMembershipMatrix[RMATRIX, {{1, 4}, {1, 4}}];

FuzzyPlot3D[R, AxesLabel → {"U", "U", "R  "}, Boxed → False];
```



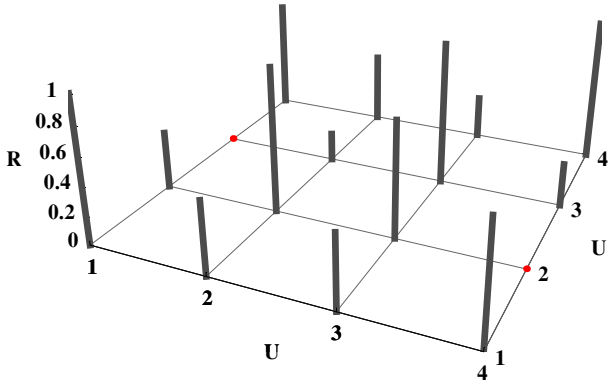
Note that the membership grades are symmetric about the main diagonal.

Reflexivity

A binary fuzzy relation R in U is reflexive if for all u in the set U, R(u, u) =1. The following relation is a reflexive relation.

```
RMATRIX = {{1, .4, 0, .8}, {.5, 1, .2, .5}, {.5, .8, 1, .3}, {.8, 0, .3, 1}};

R = FromMembershipMatrix[RMATRIX, {{1, 4}, {1, 4}}];
```

```
FuzzyPlot3D[R, AxesLabel → {"U", "U", "R  "}, Boxed → False];
```

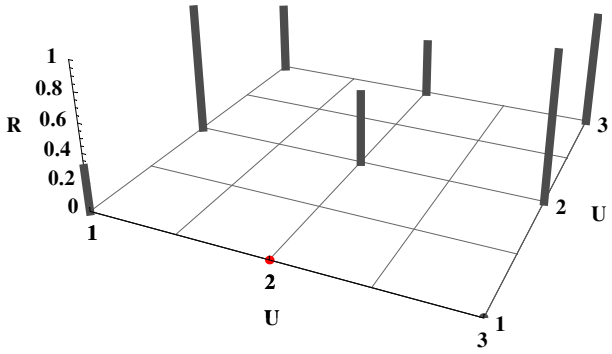


Note that the membership grades of the main diagonal all have a height of 1.

Transitivity

If a binary fuzzy relation R in U satisfies the condition that for all (u, v) in UxU, R(u, v) ≥ Max(Min(R(u, w), R(w, v))) for all w in the set U, then we say that the relation is transitive. A transitive relation is shown in the following example.

```
RMATRIX = {{.3, .9, .5}, {0, .5, .4}, {0, 1, .8}};

R = FromMembershipMatrix[RMATRIX, {{1, 3}, {1, 3}}];

FuzzyPlot3D[R, AxesLabel → {"U", "U", "R  "}, Boxed → False];
```



Before we check that this is really a transitive relation, note that the condition for transitivity can be written as follows:

For all (u, v) in UxU, R(u, v) ≥ Composition(R, R)(u, v)

From this condition, we can formulate that a binary fuzzy relation R in U is transitive if the following condition is true:

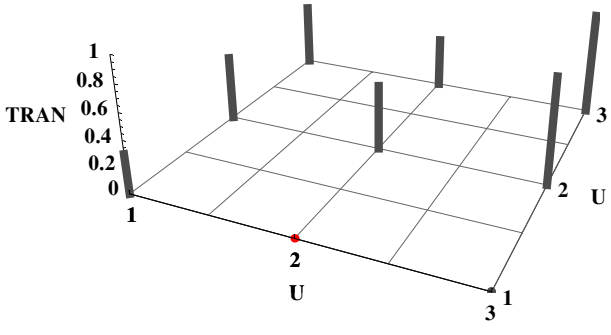
Composition(R, R) must be a subset of R

For the binary fuzzy relation we used earlier, the composition, is the following fuzzy relation.

```
TRAN = Composition[R, R, Type → MaxMin];

ToMembershipMatrix[TRAN] // MatrixForm
```

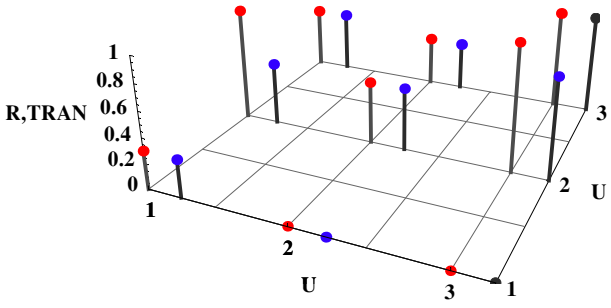
$$\begin{pmatrix} 0.3 & 0.5 & 0.5 \\ 0 & 0.5 & 0.4 \\ 0 & 0.8 & 0.8 \end{pmatrix}$$

```
FuzzyPlot3D[TRAN, AxesLabel → {"U", "U", "TRAN  "}, Boxed → False];
```



From the following graph, where we show both the composition and the original fuzzy relation, we see that the composition is indeed a subset of the original fuzzy relation, R. This proves that this binary fuzzy relation is indeed transitive.

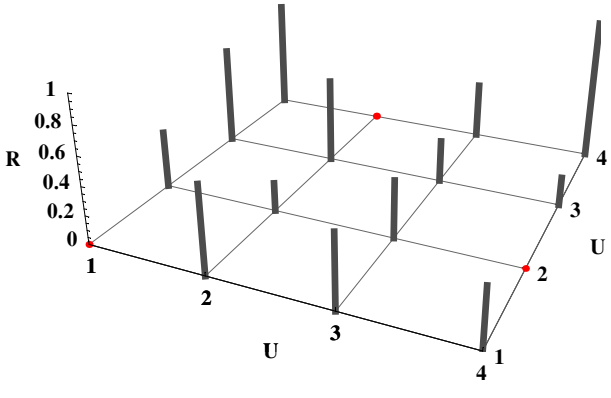
```
FuzzyPlot3D[R, TRAN,
  AxesLabel → {"U", "U", "R,TRAN  "}, ShowDots → True, Boxed → False];
```



Antisymmetry

A fuzzy binary relation R in U is antisymmetric if for all (u, v) in UxU, R(u, v) does not equal R(v, u) or if R(u, v) = R(v, u) = 0. An example of an antisymmetric relation is shown here. It is a fuzzy relation in U = {u_1, u_2, u_3, u_4}.

```
RMATRIX = {{0, .4, .7, .8}, {.6, .2, .6, 0}, {.5, .4, .3, .4}, {.4, 0, .2, 1}};

R = FromMembershipMatrix[RMATRIX];

FuzzyPlot3D[R, AxesLabel → {"U", "U", "R  "}, Boxed → False];
```



References

G. J. Klir, and T. A. Folger, *Fuzzy Sets, Uncertainty, and Information*, Prentice Hall, Englewood Cliffs, NJ, 1988.

M. S. Stachowicz and M. E. Kochanska, Graphic interpretation of fuzzy sets and fuzzy relations, *Mathematics at the Service of Man.* Edited by A. Ballester, D. Cardus, and E. Trillas, based on materials of Second World Conf., Universidad Politecnica Las Palmas, Spain, 1982.

L. A. Zadeh, Fuzzy sets, *Information and Control*, vol. 8, pp. 338-353, 1965.

H. J. Zimmermann, *Fuzzy Set Theory and Its Applications*, 2nd ed., Kluwer Academic Publishers, Boston, MA, 1991.

4 Fuzzy Modeling

4.1 Introduction

The goal of this notebook is to demonstrate how the *Fuzzy Logic* package can be used for modeling. We plan to show how fuzzy sets can be used to represent a real system or process. In this demonstration, we use modeling data from a paper in which the authors investigate the effects of using various fuzzy operators for constructing models [Stachowicz and Kochanska , 1987]. We only look at one example from the paper, but it may be instructive to try to duplicate some of the other models with different operators.

To demonstrate fuzzy modeling, we use many functions from the *Fuzzy Logic* package, along with standard *Mathematica* functions. The *Fuzzy Logic* package contains numerous functions for working with fuzzy sets and fuzzy logic; this notebook demonstrates only a few of those functions. For specific information on the functions used in this notebook, refer to the *Introduction* and *Manual*.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

4.2 Representing the Model Input

The Process

In this notebook, we will be modeling a theoretical relationship between a set of input and a set of output. We scaled the input and output numbers used in the original paper by 100, so if their output was 400, ours will be 4.

We start with a list of the input-output pairs for the process. The first number in each pair represents the input to the system, and the second item is the system output. It is this input-output relationship that we intend to model. In this example, we start with a list of specific points, but for fuzzy modeling, exact points are not required. Instead of the first input being exactly {1, 20}, it could be thought of in the following way: if the input is approximately one then the output is around twenty.

```
In[2]:= OriginalData = {{1, 20}, {2, 12}, {3, 9}, {4, 6},
          {5, 5}, {6, 4}, {7, 5}, {8, 6}, {9, 9}, {10, 12}, {11, 20}};
```

Dividing the Input Range

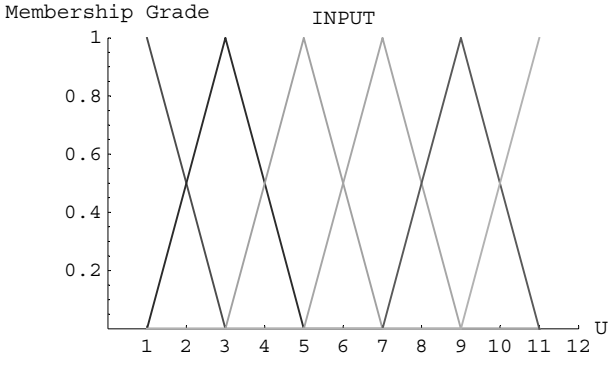
One of the first things to consider when designing a model is the range for the input variables. From the original data, we see that the input ranges from 1 to 11; we use this range as the universal space for our input variable. Once we have a reasonable range, we can consider how to divide that range into descriptive linguistic terms. For this model, we are using 6 membership functions to divide up the input universal space. We will repeat that using the `CreateFuzzySets` function. The following command divides the universal space into six even triangular fuzzy sets. Notice how we assign linguistically significant names to the six membership functions.

```
In[3]:= INPUT = {Inull, Izero, Ismall, Imedium, Ibig, Isuperbig} =
          CreateFuzzySets[6, UniversalSpace → {1, 11}];
```

Viewing the Membership Functions

We can take a look at our membership functions with the `FuzzyPlot` command. Notice in this example how we use one of *Mathematica*'s standard plotting options, `PlotLabel`, with our fuzzy plotting function. Remember, all of *Mathematica*'s standard plotting options can be used with the `FuzzyPlot` function, so you can customize the look of your plots.

```
In[4]:= FuzzyPlot[INPUT, PlotJoined → True, PlotLabel → "INPUT"];
```



4.3 Representing the Model Output

Dividing the Output Range

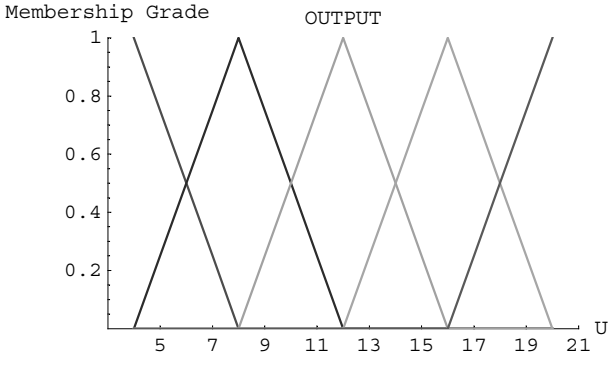
Like the input, the range of the output variable must be divided up into various membership functions. Looking at our original data, we see that the output ranges from 4 to 20 over the input range; we use this as our universal space. For this example, the authors of the original paper chose to use five fuzzy sets to represent the output. We will do the same thing by again using the `CreateFuzzySets` function. Notice how we give the five fuzzy sets linguistically significant names from `NZero` to `NSuperbig`. It is important when naming membership functions to use distinct names.

```
In[5]:= OUTPUT = {NZero, NSmall, NMedium, NBig, NSuperbig} =
          CreateFuzzySets[5, UniversalSpace → {4, 20}];
```

Viewing the Membership Functions

We plot the output membership functions using the `FuzzyPlot` function.

```
In[6]:= FuzzyPlot[OUTPUT, PlotJoined → True, PlotLabel → "OUTPUT"];
```



4.4 Creating Linguistic Control Rules

With our inputs and outputs defined, we need only specify a set of rules to create a model. In this package, rules are organized as a list of ordered pairs. The first item in the pair represents the input condition or the *if* part of the implication. The second item in the pair represents the output condition or the *then* part of the implication. For example, looking at the list of rules that follow, we see that the first pair in the list is {`In`-`ull`, `NSuperbig`}. This rule would be equivalent to the verbal statement:

if the input is `Inull` then the output is `NSuperbig`

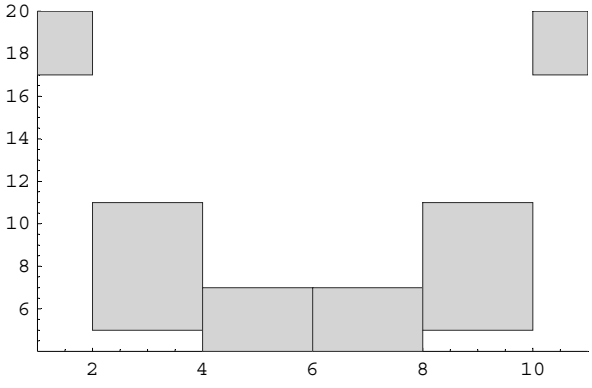
You see that this type of modeling is intuitively easy. The specified rules may be based on test data or simply on observations. By examining our original data, we can see the basis for the first rule. The first data point is {1, 20}, which corresponds to a very small input (`Inull`) and a very large output (`NSuperbig`) . Here is the complete list of rules that were used in the paper [Stachowicz and Kochanska, 1987]. Note that there is a rule associated with each of the input membership functions.

```
In[7]:= TheRules = {{Inull, NSuperbig}, {Izero, NSmall}, {Ismall, NZero},
          {Imedium, NZero}, {Ibig, NSmall}, {Isuperbig, NSuperbig}};
```

Viewing the Linguistic Rules

We plot the linguistic rules using the `FuzzyGraph` function.

```
In[8]:= FuzzyGraph[TheRules];
```



4.5 Building the Model

With all the necessary ingredients defined, we can create our model. To do that, we call `BuildModel` with our list of rules. The `BuildModel` function creates a fuzzy relation from each of the rules we specified using the `SetsToRelation` function. It then combines all of the fuzzy relations using the `Union` operator. It is possible to use other operators to combine the rules, and that is the subject of the paper we are following [Stachowicz and Kochanska, 1987]. The end result of the `BuildModel` function is a single fuzzy relation, which should give a good representation of the process we are modeling.

Creating the Model Fuzzy Relation

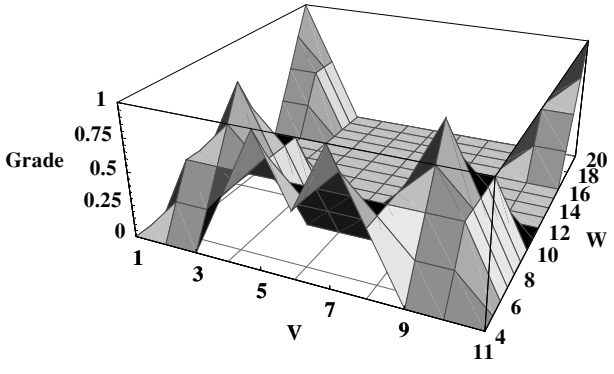
This statement creates a fuzzy relation that will model our process.

```
In[9]:= ModelRel = BuildModel[TheRules];
```

Displaying the Model Fuzzy Relation

We look at the fuzzy relation that will serve as the model for our process using the `FuzzySurfacePlot` function.

```
In[10]:= FuzzySurfacePlot[ModelRel];
```



4.6 Using the Model

Inferencing

To use our model to evaluate inputs, we use the `CompositionBasedInference` function. It takes as arguments a fuzzy set, which represents the input, and a fuzzy relation, which is our model. It performs a `MaxMin` type `Composition` with them to come up with fuzzy output. In this example, we use a `MeanOfMax` defuzzification to get a crisp value.

We use a singleton fuzzy set as input to the inferencing function. Here we set up a function that calls the inferencing function with the proper input. We need only provide this function an integer in the range 1 to 11, and `MyModeler` will provide the crisp output of the model.

```
In[11]:= MyModeler[inp_] := MeanOfMax[CompositionBasedInference[
           FuzzySet[{{inp, 1}}, UniversalSpace → {1, 11}], ModelRel]]
```

Model Results

Table of Results

To see what kind of results the model provides, we make a table of input-output pairs for the entire range of inputs. We accomplish this with *Mathematica*'s `Table` function, and we will look at the results in `TableForm`.

```
In[12]:= ModelResults = Table[{i, N[MyModeler[i]]}, {i, 1, 11}];
```

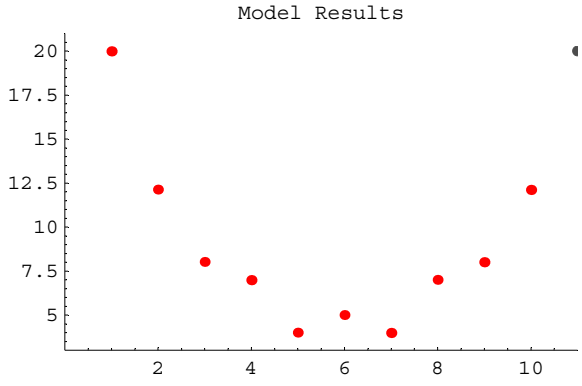
```
In[13]:= TableForm[Prepend[ModelResults, {"Input", "ModelOutput"}]]
```

```
Out[13]//TableForm=
      Input      ModelOutput
      1          20.
      2          12.125
      3          8.
      4          7.
      5          4.
      6          5.
      7          4.
      8          7.
      9          8.
      10         12.125
      11         20.
```

Graph of Results

We plot these results to see how well our model performs. We do this with the `ListPlot` function.

```
In[14]:= ListPlot[Transpose[ModelResults][[2]], PlotStyle → {Hue[0], PointSize[.02]},
           PlotLabel → "Model Results", PlotRange → {{0, 11}, {3, 21}}, AxesOrigin → {0, 3}];
```



4.7 Evaluating the Model

To see how well our model performed, we compare the original data with the results from our model. We can compare the results in table form and with a graph.

Table Comparison

Here we create a table showing input, original output, model output, and absolute error.

```
In[15]:= CompareTable = Table[{i, OriginalData[[i, 2]], ModelResults[[i, 2]],
           Abs[OriginalData[[i, 2]] - ModelResults[[i, 2]]]}, {i, 1, 11}];
```

```
In[16]:= TableForm[Prepend[CompareTable, {"Input", "Process", "Model", "AbsErr"}]]
```

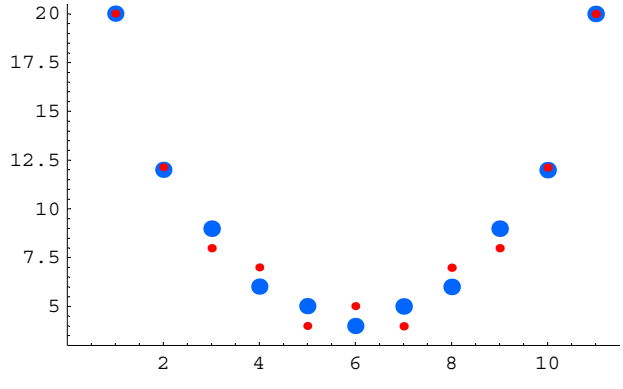
Out[16]//TableForm=

Input	Process	Model	AbsErr
1	20	20.	0.
2	12	12.125	0.125
3	9	8.	1.
4	6	7.	1.
5	5	4.	1.
6	4	5.	1.
7	5	4.	1.
8	6	7.	1.
9	9	8.	1.
10	12	12.125	0.125
11	20	20.	0.

Graphical Comparison

We again use `ListPlot` to show the results. This function will plot the original process data as large blue dots and our model data as smaller red dots. You can see that the model performs quite well for its simple design. It is possible to create entirely different models by changing operators, defuzzification strategies, rules, or membership functions. Feel free to create a different model and compare your results to this one.

```
In[17]:= Show[ListPlot[Transpose[OriginalData][[2]],
           DisplayFunction → Identity, PlotStyle → {Hue[0.6], PointSize[.03]},
           PlotRange → {{0, 11.5}, {3, 20.5}}, AxesOrigin → {0, 3}],
          ListPlot[Transpose[ModelResults][[2]], DisplayFunction → Identity,
           PlotStyle → {Hue[0], PointSize[.015]}, PlotRange → {{0, 11.5}, {3, 20.5}},
           AxesOrigin → {0, 3}], DisplayFunction → $DisplayFunction];
```



References

M. S. Stachowicz and M. E. Kochanska, Analysis of the application of fuzzy relations in modeling, *Proc.-North American Fuzzy Information Society '86*, New Orleans, 1986.

M. S. Stachowicz and M. E. Kochanska, Fuzzy modeling of the process, *Proc.of Second International Fuzzy Systems Association Congress,* Tokyo, 1987.

L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning–I, *Information Sciences*, vol. 8, pp. 199-249, 1975.

L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning–II, *Information Sciences*, vol. 8, pp. 301-357, 1975.

L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning–III, *Information Sciences*, vol. 9, pp. 43-80, 1975.

5 Fuzzy Logic Control

5.1 Introduction

In this notebook, we look at how to design a fuzzy logic control strategy to back a truck up to a loading dock.

The goal of the controller is to come up with a steering strategy to back-up a truck so that it arrives perpendicular to a dock. The truck may start at any initial position and orientation in the parking lot. To model this scenario, we assume our parking lot is a graph that is 100 units wide and 100 units tall. Our loading dock is located at the top-center of the graph at position {50, 100}. Our controller decides what steering angle is needed at each stage to back the truck up correctly. We limit the steering angle to be between -30 and 30 degrees.

To implement our fuzzy logic controller, we use many of the commands from *Fuzzy Logic* package as well as a few additional routines to model the truck [Freeman 1994].

This loads the package.

```
In[1]:= << FuzzyLogic`
```

5.2 Defining Input Membership Functions

Truck Position

Set Universal Space

One of the first questions to ask when designing a fuzzy logic controller is the following: What are my inputs and outputs? Once this question is answered, the next item to address is the range of the inputs and outputs. When talking about fuzzy sets, this range is referred to as the universal space.

In our example, the first input we use is the truck's x-position on the graph. Since we decided on a parking lot 100 units wide, we need a universal space that ranges from 0 to 100. To simplify the later construction of our membership functions, we change the default universal space now to be the desired range, 0 to 100, with the following command.

```
In[2]:= SetOptions[FuzzySet, UniversalSpace → {0, 100}];
```

Define Linguistic Variables

We next need to decide on the number and shape of the membership functions to use. There is no optimal solution for choosing membership functions, only guidelines. We do not worry about that here, we simply follow the example used by Kosko [Kosko 1992].

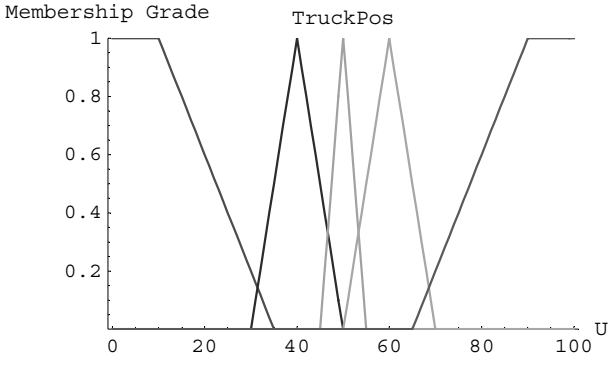
We use the `FuzzyTrapezoid` function to construct our membership functions. The names of our fuzzy sets have the following linguistic meanings: `LE`–Left , `LC`–LeftCenter, `CE`–Center , `RC`–RightCenter, and `RI`–Right. These variables are used to describe the truck's x-position on the graph. We collect all the membership functions together under the heading `TruckPos`, which is the first input to our controller.

```
In[3]:= LE = FuzzyTrapezoid[0, 0, 10, 35];
        LC = FuzzyTrapezoid[30, 40, 40, 50];
        CE = FuzzyTrapezoid[45, 50, 50, 55];
        RC = FuzzyTrapezoid[50, 60, 60, 70];
        RI = FuzzyTrapezoid[65, 90, 100, 100];

In[8]:= TruckPos = {LE, LC, CE, RC, RI};
```

Plot the Membership Functions

Next, we plot the membership functions for our first input with the `FuzzyPlot` function. The `PlotJoined` option is set to `True` to produce the line graph shown.

```
In[9]:= FuzzyPlot[TruckPos, PlotJoined → True, PlotLabel → "TruckPos"];
```



Truck Angle

Set Universal Space

The second input is the truck's orientation or the angle between a line down the center of the truck and a line horizontal to the parking lot. For this input, the universal space ranges from -90 degree, where the back of the truck is facing away from the dock, to 270 degrees, where it is again facing away. The truck angle should end up near 90 degrees, which indicates that the back is facing the dock and the truck is perpendicular to the dock. For angles between -90 and 90 degrees, the back of the truck will be facing toward the right of the graph, and for angles between 90 and 270 degrees, the back of the truck will face the left side of the graph.

We set the universal space for our second input just as we did for our first input.

```
In[10]:= SetOptions[FuzzySet, UniversalSpace → {-90, 270}];
```

Define Linguistic Variables

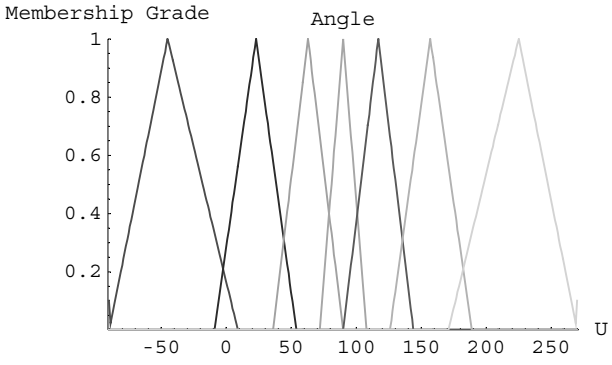
We again use Kosko's membership functions [Kosko 1992] for this input. The linguistic variables have the following meanings: `RB`–Right Below, `RU`–Right Upper, `RV`–Right Vertical, `VE`–Vertical, `LV`–Left Vertical, `LU`–Left Upper, and `LB`–Left Below. We group all of the membership functions under the name `Angle`, which is the second input to our controller.

```
In[11]:= RB = FuzzyTrapezoid[-90, -45, -45, 9] ⋃ FuzzySet[{{-90, .1}}];
         RU = FuzzyTrapezoid[-9, 23, 23, 54];
         RV = FuzzyTrapezoid[36, 63, 63, 90];
         VE = FuzzyTrapezoid[72, 90, 90, 108];
         LV = FuzzyTrapezoid[90, 117, 117, 144];
         LU = FuzzyTrapezoid[126, 157, 157, 189];
         LB = FuzzyTrapezoid[171, 225, 225, 270] ⋃ FuzzySet[{{270, .1}}];

In[18]:= Angle = {RB, RU, RV, VE, LV, LU, LB};
```

Plot Membership Functions

Here is a plot of the second input's membership functions.

```
In[19]:= FuzzyPlot[Angle, PlotJoined → True, PlotLabel → "Angle"];
```



5.3 Defining Output Membership Functions

Set Universal Space

The output of our controller is a steering angle limited to the range -30 to 30 degrees. This then is the universal space we need for our output membership functions.

```
In[20]:= SetOptions[FuzzySet, UniversalSpace → {-30, 30}];
```

Defining Linguistic Variables

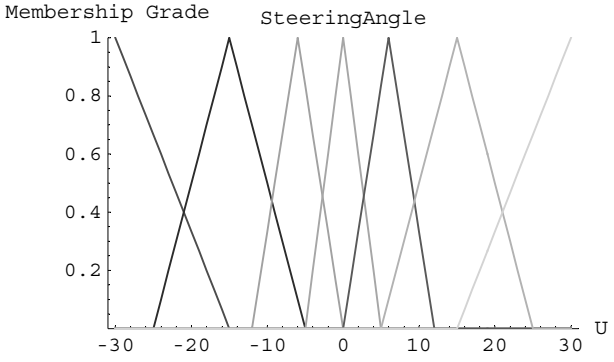
We again use the membership function definitions from the Kosko book [Kosko 1992] to define our output membership functions. This time we name our membership functions : `NB`-Negative Big, `NM`-Negative Medium, `NS`-Negative Small, `ZE`-Zero, `PS`-Positive Small, `PM`-Positive Medium, `PB`-Positive Big. These names are used frequently in defining fuzzy logic controllers. We group all the output membership functions under the name `SteeringAngle`, which is our single output.

```
In[21]:= NB = FuzzyTrapezoid[-30, -30, -30, -15];
         NM = FuzzyTrapezoid[-25, -15, -15, -5];
         NS = FuzzyTrapezoid[-12, -6, -6, 0];
         ZE = FuzzyTrapezoid[-5, 0, 0, 5];
         PS = FuzzyTrapezoid[0, 6, 6, 12];
         PM = FuzzyTrapezoid[5, 15, 15, 25];
         PB = FuzzyTrapezoid[15, 30, 30, 30];

In[28]:= SteeringAngle = {NB, NM, NS, ZE, PS, PM, PB};
```

Plot Membership Functions

Here we see how our output membership functions looks.

```
In[29]:= FuzzyPlot[SteeringAngle, PlotJoined → True, PlotLabel → "SteeringAngle"];
```



5.4 Defining Control Rules

The final item needed for a fuzzy logic controller is a set of rules or a rule base. Typically rules for fuzzy logic controllers appear in *if-then* form. Our fuzzy inference system accepts rules as a list of triplets. The first two item in each triplet are the input conditions and the third item is the output condition.

This list represents rules of the form: if input1 = #1 and input2 = #2, then output = #3. For example, looking at the following control rules, the first rule, {LE,RB,PS}, represents the linguistic rule: if TruckPos is LE (Left) and Angle is RB (Right Below), then SteeringAngle should be PS (Positive Small). The rules are formed with linguistic terms, which follow human intuition.

```
In[30]:= ControlRules = {{LE, RB, PS}, {LC, RB, PM}, {CE, RB, PM}, {RC, RB, PB}, {RI, RB, PB},
           {LE, RU, NS}, {LC, RU, PS}, {CE, RU, PM}, {RC, RU, PB}, {RI, RU, PB}, {LE, RV, NM},
           {LC, RV, NS}, {CE, RV, PS}, {RC, RV, PM}, {RI, RV, PB}, {LE, VE, NM}, {LC, VE, NM},
           {CE, VE, ZE}, {RC, VE, PM}, {RI, VE, PM}, {LE, LV, NB}, {LC, LV, NM}, {CE, LV, NS},
           {RC, LV, PS}, {RI, LV, PM}, {LE, LU, NB}, {LC, LU, NB}, {CE, LU, NM}, {RC, LU, NS},
           {RI, LU, PS}, {LE, LB, NB}, {LC, LB, NB}, {CE, LB, NM}, {RC, LB, NM}, {RI, LB, NS}};
```

5.5 Simulation Functions

This section contains functions that will allow us to simulate the truck backing. This first function is essentially the fuzzy logic controller. The two inputs to this function, `phi` and `x` represent the two inputs to the controller, the angle and truck position. The function will return a crisp output that is the controller's recommended steering angle.

```
In[31]:= Steer[phi_, x_] := CenterOfArea[
           RuleBasedInference[TruckPos, Angle, SteeringAngle, ControlRules, x, phi]]
```

These next two functions are for modeling the truck-backing. They contain functions that model the movement of the truck and draw the parking lot and truck graphics. These functions are from a *Mathematica Journal* article by James Freeman [Freeman 1994].

```
In[32]:= simulateTruck[x0_, y0_, phi0_] :=
          Module[{x = x0, y = y0, phi = phi0, newPhi, result = {}},
           While[y ≤ 95, newPhi = phi + Steer[phi, x]; AppendTo[result,
             {x, y, phi} = N[{x + 4 Cos[newPhi π / 180], y + 4 Sin[newPhi π / 180], newPhi}]];]; result]
```

```
In[33]:= showTruck[{x_, y_, phi_}, {l_, w_}] :=
           Module[{s = N[Sin[phi π / 180]], c = N[Cos[phi π / 180]]}, Show[Graphics[{Hue[0.7],
             Line[Transpose[{x, y} + {{-s w / 2, s w / 2, s w / 2 - c l, s w / 4 - c l, s w / 4 - 1.2 c l,
                 -s w / 4 - 1.2 c l, -s w / 4 - c l, -s w / 2 - c l, -s w / 2}, {c w / 2, -c w / 2, -c w / 2 - s l,
                 -c w / 4 - s l, -c w / 4 - 1.2 s l, c w / 4 - 1.2 s l, c w / 4 - s l, c w / 2 - s l, c w / 2}}]],
             GrayLevel[0], Line[{{0, 100}, {100, 100}}], Line[{{100, 100}, {100, 0}}],
             Line[{{50, 100}, {50, 95}}]}, Axes → True, AspectRatio → Automatic,
            PlotRange → {{0, 100}, {0, 100}}, AxesOrigin → {0, 0}]];
```

5.6 Test Run 1

Run Simulation

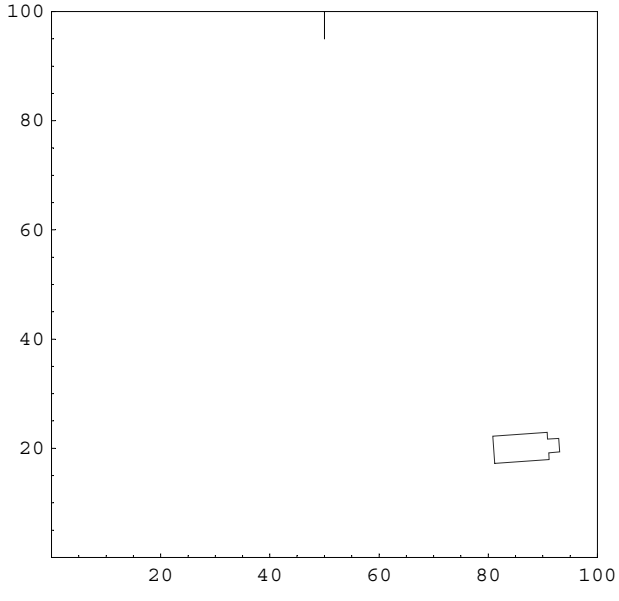
We provide the function `simulateTruck` with an initial x-position, y-position, and truck angle. For the first simulation, we start the truck at position {85, 20} with an initial angle of 190 degrees.

```
In[34]:= simlist = simulateTruck[85, 20, 190];
```

Model Results

The function below will draw a graph showing the location and angle of the truck in the parking lot. To see an animation of the truck backing, click the cell bracket containing all of the truck graphics; this will highlight the bracket. Now select Animate Selected Graphics from the Graph pull down menu. This will cause the truck to appear to move in one of the graphs. The buttons that appear at the bottom border during animations can be used to control the speed and direction of the animation. To stop the animation click anywhere in the notebook.

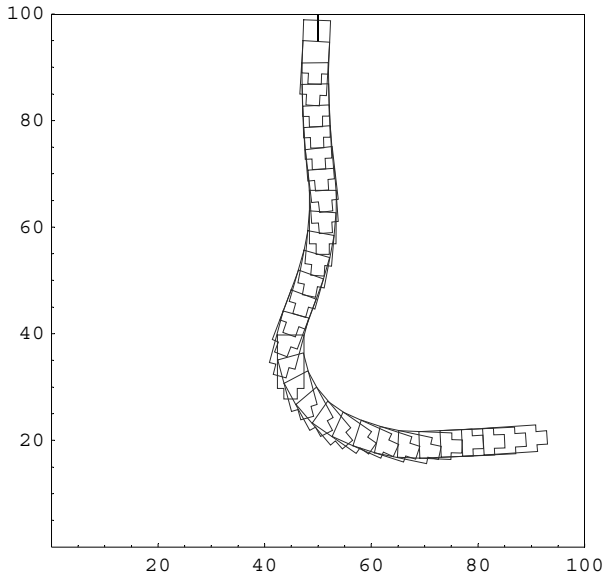
```
In[35]:= graph1 = (showTruck[#1, {10, 5}] &) /@ simlist;
```



Show Complete Trajectory

The complete trajectory of the truck can be seen using the `Show` command.

```
In[36]:= Show[graph1];
```



5.7 Test Run 2

Run Simulation

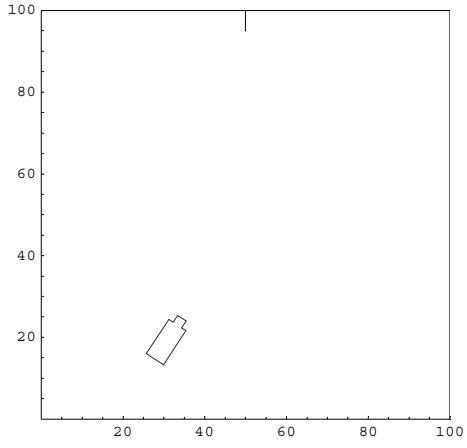
We can run another simulation from a different starting position and angle. You can test this program with your own initial values for `x`, `y`, and `phi`. The controller should work if there is enough room from any position and angle.

```
In[37]:= test2 = simulateTruck[30, 18, 260];
```

Model Results

We again model the results. You can again animate the graphic cells in this section using the animation option.

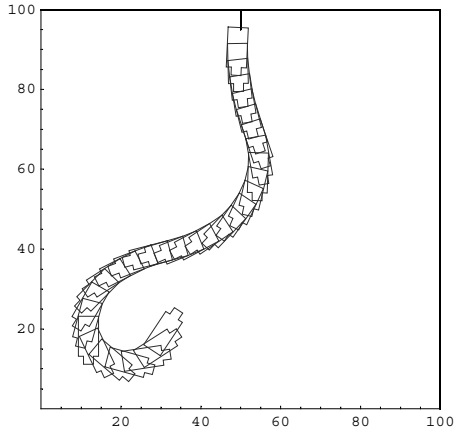
```
In[38]:= graph2 = (showTruck[#1, {10, 5}] &) /@ test2;
```



Complete Trajectory

Again, we show the complete trajectory of our second test run.

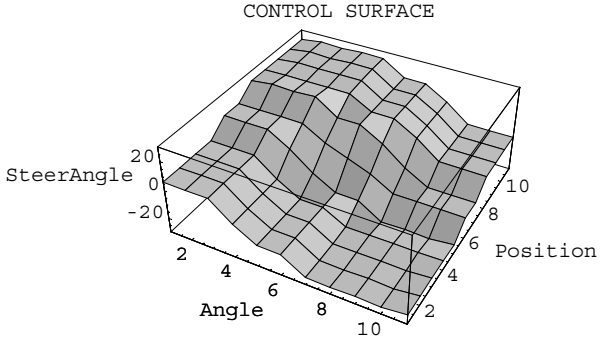
```
In[39]:= Show[graph2];
```



5.8 Control Surface

A convenient way to examine a two-input/one-output control strategy is to look at a control surface. This is a 3D graph in which the inputs form the base of the graph, and the output is represented by the height of the graph above each input pair. We have constructed such a surface here.

```
In[40]:= ControlTable =
           Table[CenterOfArea[RuleBasedInference[TruckPos, Angle, SteeringAngle,
              ControlRules, i, j]], {i, 0, 100, 10}, {j, -90, 270, 36}];

In[41]:= ListPlot3D[ControlTable, AxesLabel → {"Angle", "Position", "SteerAngle"},
           PlotRange → {{1, 11}, {1, 11}, {-30, 30}}, PlotLabel → "CONTROL SURFACE"];
```



References

J. A. Freeman, *Fuzzy systems for control applications: the truck backer-upper*, *The Mathematica Journal*, vol. 4, pp. 64-69, 1994.

B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice Hall, Englewood Cliffs, NJ, 1992.

6 Fuzzy Numbers

6.1 Introduction

This notebook deals with discrete fuzzy arithmetic operations. We'll be using the *Fuzzy Logic* package, along with standard *Mathematica* functions.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

6.2 Creating Fuzzy Numbers

When working with fuzzy numbers and performing fuzzy arithmetic, we should use a large universal space because the intervals over which fuzzy numbers are defined widen as arithmetic operations are performed. Also, like traditional numbers, fuzzy numbers can be negative or positive, so the universal space should be symmetric around zero. Taking this into consideration, we investigate fuzzy arithmetic with a universal space from -100 to 100. We change the default universal space to this range with the following command.

```
In[2]:= SetOptions[FuzzySet, UniversalSpace → {-100, 100}];
```

The following function is used to create fuzzy numbers. The function accepts an integer as its argument and returns a triangular fuzzy set centered around the integer. This is a common representation of a fuzzy number.

```
In[3]:= FuzzyNumber[x_?IntegerQ] := FuzzyTrapezoid[x - 3, x, x, x + 3]
```

We can create a set of fuzzy numbers by using the function we just defined and *Mathematica*'s `Table` function. Here we create fuzzy numbers ranging from `NegSIX` (Negative Six) to `SIX`.

```
In[4]:= {NegSIX, NegFIVE, NegFOUR, NegTHREE, NegTWO, NegONE, ZERO,
           ONE, TWO, THREE, FOUR, FIVE, SIX} = Table[FuzzyNumber[k], {k, -6, 6}];
```

6.3 Fuzzy Arithmetic

Shortening Arithmetic Names

The *Fuzzy Logic* package comes with a number of convenient functions for performing discrete arithmetic, but the names of these functions are quite long. To save some typing, we rename the functions here to shorten their names.

```
In[5]:= FP := DiscreteFuzzyPlus
        FM := DiscreteFuzzyMinus
        FX := DiscreteFuzzyMultiply
        FI := DiscreteFuzzyImage
```

Using Infix Form

When performing arithmetic operations, it is traditional to write out the operation from left to right, with the sign for the operation to be performed located between the two numbers on which it is to be performed. To simulate the traditional form, we use *Mathematica*'s infix form in the following examples. Instead of surrounding the operands with square brackets and the function name (e.g., `Func[A, B]`), the infix form locates the function name between the operands and surrounds the operand with tildes (e.g., `A~Func~B`). Let's look at the following example to see how this works. In this example, we perform fuzzy addition on the fuzzy numbers `NegFOUR` and `THREE`.

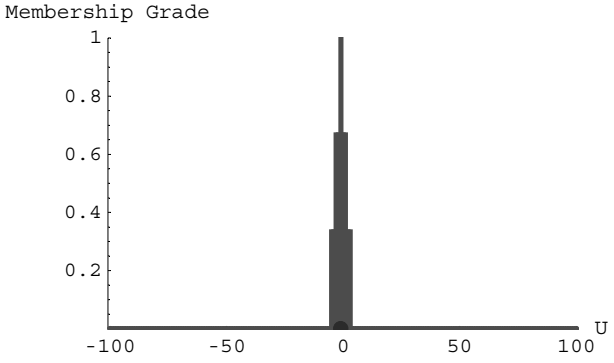
```
In[9]:= Sum1 = NegFOUR ~ FP ~ THREE
```

```
Out[9]= FuzzySet[{{-5, 1/3}, {-4, 1/3}, {-3, 2/3}, {-2, 2/3}, {-1, 1},
          {0, 2/3}, {1, 2/3}, {2, 1/3}, {3, 1/3}}, UniversalSpace → {-100, 100, 1}]
```

After defuzzifying the result with the `MeanOfMax` method, we see that the result is a fuzzy number centered around -1. This is the answer we would expect to get when performing the operation -4 + 3.

```
In[10]:= MeanOfMax[Sum1, ShowGraph → True];
```

```
Mean of max is -1.
```



Example 1

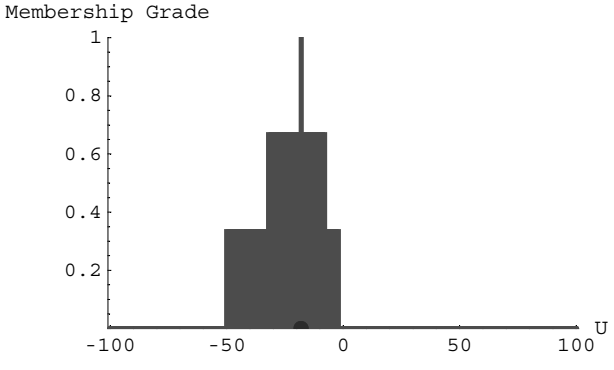
The following example demonstrates the fuzzy equivalent of the operation: (5 - (-1)) * (-3)

```
In[11]:= Tot2 = (FIVE ~ FM ~ NegONE) ~ FX ~ NegTHREE;
```

By using the `MeanOfMax` defuzzification, we receive -18, which would be the solution to the equivalent nonvisual operation.

```
In[12]:= MeanOfMax[Tot2, ShowGraph → True];
```

```
Mean of max is -18.
```



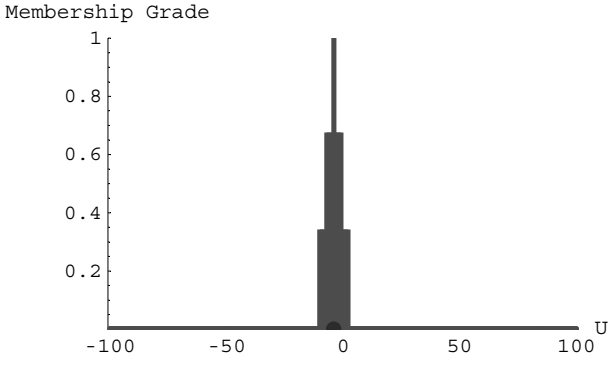
Example 2

In this example, we perform the fuzzy equivalent of the operation: -(5 + 3 - 4).

```
In[13]:= Tot2 = FI[FM[FP[FIVE, THREE], FOUR]];
```

After defuzzification, we receive the expected result of -4.

```
In[14]:= MeanOfMax[Tot2, ShowGraph → True];

          Mean of max is -4.
```



References

A. Kaufmann and M. M. Gupta, *Introduction to Fuzzy Arithmetic Theory and Application,* Van Nostrand Reinhold, New York, 1991.

Z. Wang and G. J. Klir, *Fuzzy Measure Theory*, Plenum Press, New York, 1992.

7 Digital Fuzzy Sets and Multivalued Logic

7.1 Introduction

This notebook deals with digital fuzzy sets and Łukasiewicz multivalued logic arithmetic operations. We'll be using the *Fuzzy Logic* package, along with standard *Mathematica* functions.

This loads the package.

```
In[1]:= << FuzzyLogic`
```

7.2 Creating Digital Fuzzy Sets

One of the ways by which fuzzy membership functions and fuzzy systems may be categorized is according to whether the membership functions are continuous or discrete.

Definition 1

A continuous-universal space membership function has a value defined for each point in universal space and a continuous-universal space system operates on and produces continuous-universal space membership functions.

If a continuous-universal space membership function A(u) can take on any value (the grade of membership) in the continuous interval [0, 1], then the continuous-universal space membership function A(u) is called an analog fuzzy sets.

$$A : U \to [\,0, 1]$$

Numerical processing using digital computers requires finite data with finite precision. In engineering practice, we are often forced to make compromises. No matter how elegant the theory, system complexity

often demands a numerical approach to analysis.

Definition 2

A discrete-universal space membership function has a value (the grade of membership) only at discrete points in universal space and a discrete-universal space system operates on and produces discrete-universal space membership functions.

$$A : \{u_1, u_2, u_3, \ldots, u_s\} \rightarrow [0, 1]$$

If a discrete-universal membership function can take only a finite number, n≥2 of distinct values, then we call this fuzzy set a digital fuzzy set.

$$A : \{u_1, u_2, u_3, \ldots, u_s\} \rightarrow \{0/n - 1, 1/n - 1, 2/n - 1, \ldots, n - 2/n-1, n - 1/n - 1\}$$

For digital implementation, an analog fuzzy set's membership function is discretized along both the universal space and membership-value dimensions.

From the standpoint of practical applications, these approximations will not be troublesome. In most applications, especially when using computational techniques, a finite number of the elements of set U and the grade of membership A(u) is taken into consideration.

An assumption of denumerability of the universal space U permits a very simple graphic interpretation of the fuzzy sets defined in U (Stachowicz and Kochanska, 1985). The elements of the collection of objects of space U, due to it denumerability, can be arranged in a sequence. One can plot for each element a segment of the length corresponding to the value of the membership function of the given element in the fuzzy set under consideration.

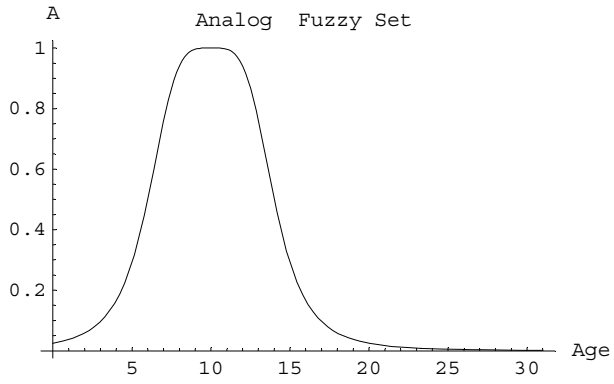
Assume that universal space is quantized into 32 discrete values with increment equal 1 (`Universal Space → {0, 31, 1}`), while membership-values can take n = 2^3 distinct values (that is, they are encoded in three bits). So we dedicated 32 x 3 = 96 bits for each linguistic value.

Example 1

Fuzzy set with continuous U. Let U = $\mathbb{R}^+$ be the set of possible ages for teens. Then the fuzzy set A = " about 10 years old" may be expressed as

```
In[2]:= AAnalog[u_]=1/(1+((u-10)/4)^4);
```

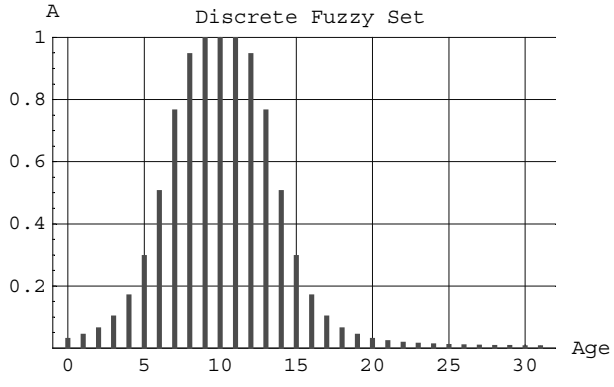
```
In[3]:= Plot[ AAnalog[u],{u,0,31},
        AxesLabel->{"Age","A"},PlotLabel->" Analog  Fuzzy Set"];
```



Now, let `UniversalSpace -> {0, 31, 1}` then the discrete fuzzy set A has a form :

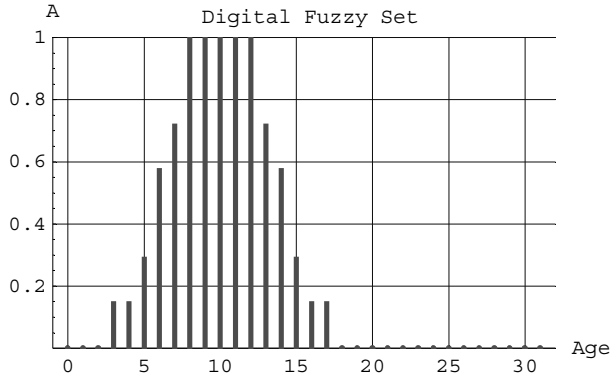
```
In[4]:= ADiscrete=CreateFuzzySet[AAnalog,UniversalSpace->{0,31,1}];
```

```
In[5]:= FuzzyPlot[ADiscrete,
        AxesLabel->{"Age","A"},PlotLabel->" Discrete Fuzzy Set",
        GridLines->Automatic];
```



Finally, a digital fuzzy set A (Łukasiewicz Set) with discrete universal space and n = 8 levels has the following form:

```
In[6]:= ADigital=ToDigital[ADiscrete,8]

Out[6]= FuzzySet[
          {{3, 1/7}, {4, 1/7}, {5, 2/7}, {6, 4/7}, {7, 5/7}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1},
           {13, 5/7}, {14, 4/7}, {15, 2/7}, {16, 1/7}, {17, 1/7}}, UniversalSpace → {0, 31, 1}]

In[7]:= FuzzyPlot[ADigital,
        AxesLabel->{"Age","A"},PlotLabel->" Digital Fuzzy Set",
        GridLines->Automatic];
```



Let's compare the level sets for discrete and digital form of the fuzzy set A.

```
In[8]:= Map[LevelSet,{ADiscrete,ADigital}]

Out[8]= {{0.00131459, 0.00159744, 0.00196053, 0.00243272,
          0.00305573, 0.00389105, 0.00503135, 0.00661978, 0.00888365, 0.0121951,
          0.0171847, 0.024961, 0.0375532, 0.0588235, 0.0963493, 0.164948,
          0.290579, 0.5, 0.759644, 0.941176, 0.996109, 1.}, {1/7, 2/7, 4/7, 5/7, 1}}
```

Let's compare the discrete and digital form of the fuzzy set A using the concept of the Hamming distance.

```
In[9]:= HammingDistance[ADiscrete,ADigital]

Out[9]= 0.812772
```

7.3 Łukasiewicz Multivalued logic

For any given n, the truth values in a multivalued logic are labeled by rational numbers in the unit [0, 1]. These values are obtained by evenly dividing the interval between 0 and 1, exclusive. These values can also be interpreted as degrees of truth.

The first series of n-valued logic was proposed by the great Polish logician Jan Łukasiewicz in the early 1930s as a generalization of his three-valued logic. He defines logical connectives by the following equations:

$p' = 1 - q$

$p \wedge q = \min (p, q)$,

$p \vee q = \max (p, q)$,

$p \Rightarrow q = \min (1, 1 - p + q)$,

$p \Leftrightarrow q = 1 - | p - q |$.

Łukasiewicz, in fact, used negation and implication as primitives and defined the other logic operations using these two primitives as follows:

$p \vee q = (p \Rightarrow q) \Rightarrow q$,

$p \wedge q = (p' \vee q')'$,

$p \Leftrightarrow q = (p \Rightarrow q) \wedge (q \Rightarrow p)$.

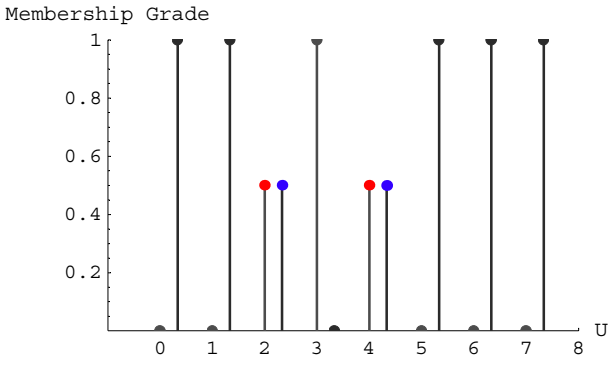
The sequence (L_2, L_3, ... , L_{inf}) of these logics contains two extreme cases L_2 and L_{inf}. Logic L_2 is the classical two-valued logic. Logic L_{inf} is an infinite-valued logic and is isomorphic to fuzzy set theory based on the standard fuzzy operators. Under this correspondence, operations of negation, conjunction, and disjunction on fuzzy propositions are defined in the same way as the operations of complementation, intersection, and union on fuzzy sets.

Example 2

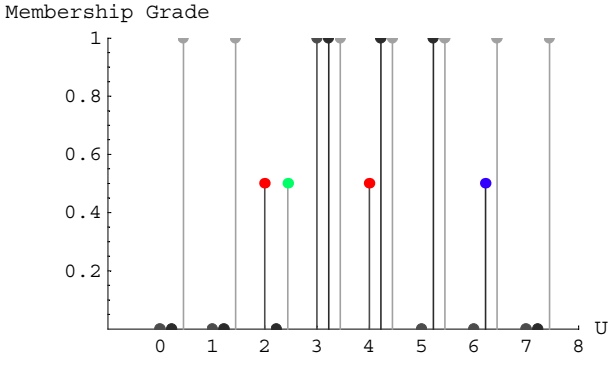
The introduction of new intermediate truth value in three-valued logic naturally affects the truth-table definitions of the five connectives of classical logic. Jan Łukasiewicz used only negation and implication as primitives and defined the other logic operations in terms of these two primitives.

```
In[10]:= SetOptions[FuzzySet,UniversalSpace->{0,7,1}];
         SetOptions[FuzzyPlot,ShowDots->True];

In[12]:= Luk1=DigitalSet[1,3,3,5];

In[13]:= Luk2=DigitalSet[3,3,5,7];
```

```
In[14]:= Complement[Luk1];
         FuzzyPlot[Luk1,Complement[Luk1]];
```




```
In[16]:= Impl= Implication[Luk1,Luk2];

In[17]:= FuzzyPlot[Luk1,Luk2,Impl];
```



Let's perform the standard min intersection:

```
In[18]:= IntA=Intersection[Luk1, Luk2];
```

This operation can also be performed using the `Implication` and `Complement` primitives.

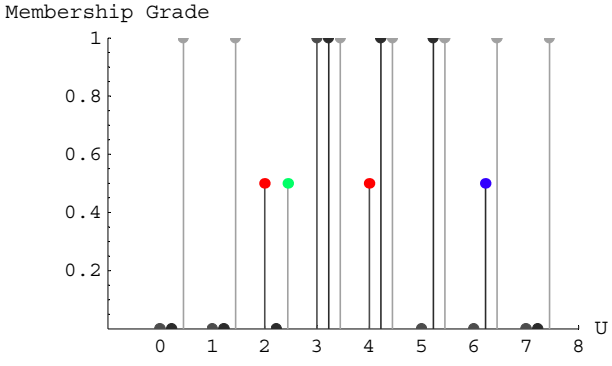
```
In[19]:= IntB=Complement[Implication[Implication[Complement[Luk1],Complement[Luk2]],Comple
         ment[Luk2]]];
```

```
In[21]:= Equality[IntA,IntB]

Out[21]= True
```

Modus ponens, for example, may be expressed as the following proposition:

$$[(p \Rightarrow q) \wedge p] \Rightarrow q$$

```
In[22]:= Modusponens=Implication[Intersection[Implication[Luk1,Luk2],Luk1],Luk2];

In[23]:= FuzzyPlot[Luk1,Luk2,Modusponens];
```



References

G. J. Klir, and T. A. Folger, *Fuzzy Sets, Uncertainty, and Information*, Prentice Hall, Englewood Cliffs, NJ, 1988.

G. J. Klir, and Bo Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice Hall, Upper Saddle River, NJ, 1995.

G. J. Klir, Ute H. St. Clair, and Bo Yuan , *Fuzzy Set Theory*, Prentice Hall, Upper Saddle River, NJ, 1997.

M. S. Stachowicz and M. E. Kochanska, Graphic interpretation of fuzzy sets and fuzzy relations, *Mathematics at the Service of Man,* Edited by A. Ballester, D. Cardus, and E. Trillas, based on materials of Second World Conf., Universidad Politecnica Las Palmas, Spain, 1982.

H. J. Zimmermann, *Fuzzy Set Theory and Its Applications,* 3rd ed., Kluwer Academic Publishers, Boston, MA, 1996.

8 Additional Examples

8.1 Introduction

This notebook contains a number of simple problems, which can be dealt with using fuzzy sets. We provide a solution for each problem, but encourage you to try to come up with your own solutions to some of the problems.

If the *Fuzzy Logic* package isn't already loaded, it should be loaded before beginning the problems in this notebook. This loads the package.

```
In[1]:= << FuzzyLogic`
```

8.2 Example 1: Classifying Houses

Problem. A realtor wants to classify the houses he offers to his clients. One indicator of comfort of these houses is the number of bedrooms in it. Let the available types of houses be represented by the following set.

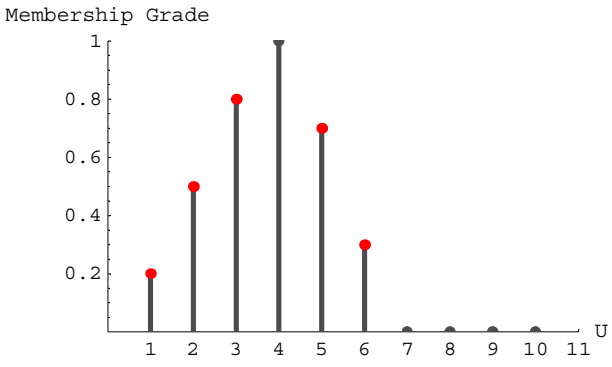
$$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

The houses in this set are described by u number of bedrooms in a house. The realtor wants to describe a 'comfortable house for a 4-person family,' using a fuzzy set.

Solution. The fuzzy set "comfortable type of house for a 4-person family " may be described using a fuzzy set in the following manner.

```
In[2]:= HouseForFour = FuzzySet[
          {{1, .2}, {2, .5}, {3, .8}, {4, 1}, {5, .7}, {6, .3}}, UniversalSpace → {1, 10}];
```

```
In[3]:= FuzzyPlot[HouseForFour, ShowDots → True];
```



8.3 Example 2: Representing Age

Problem 2-1. Fuzzy sets can be used to represent fuzzy concepts. Let U be a reasonable age interval of human beings.

$$U = \{0, 1, 2, 3, \ldots, 100\}$$

Solution 2-1. This interval can be interpreted with fuzzy sets by setting the universal space for age to range from 0 to 100.

```
In[4]:= SetOptions[FuzzySet, UniversalSpace → {0, 100}];
```

Problem 2-2. Assume that the concept of "young" is represented by a fuzzy set `Young`, whose membership function is given by the following fuzzy set.

```
In[5]:= Young = FuzzyTrapezoid[0, 0, 25, 40];
```

The concept of "old' can also be represented by a fuzzy set, `Old`, whose membership function could be defined in the following way.

```
In[6]:= Old = FuzzyTrapezoid[50, 65, 100, 100];
```

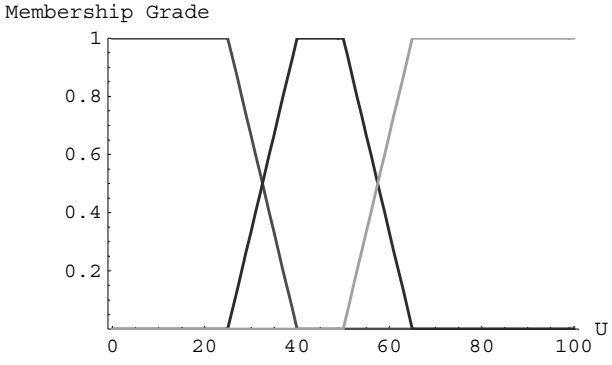
We define the concept of middle-aged to be neither young nor old. We do this by using fuzzy operators from the *Fuzzy Logic* package.

Solution 2-2. We can find a fuzzy set to represent the concept of middle-aged by taking the intersection of the complements of our `Young` and `Old` fuzzy sets.

```
In[7]:= MiddleAged = Intersection[Complement[Young], Complement[Old]];
```

We can now see a graphical interpretation of our age descriptors by using the FuzzyPlot command.

```
In[8]:= FuzzyPlot[Young, MiddleAged, Old, PlotJoined → True];
```



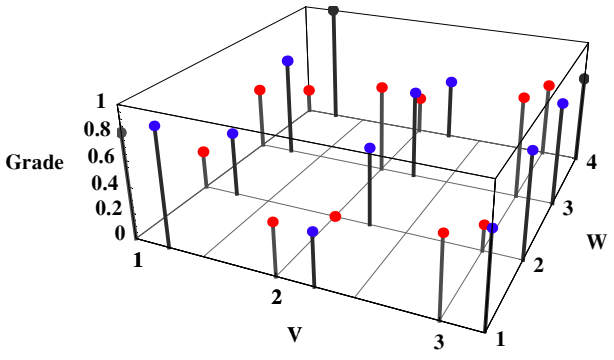
From the graph, you can see that the intersection of "not young" and "not old" gives a reasonable definition for the concept of "middle-aged."

8.4 Example 3: Finding the Disjunctive Sum

Problem. Find the disjunctive sum of the two fuzzy relations defined here.

```
In[9]:= RMat = {{.8, .3, .5, .2}, {.4, 0, .7, .3}, {.6, .2, .8, .6}};

In[10]:= Smat = {{.9, .5, .8, 1}, {.4, .6, .7, .5}, {.7, .8, .8, .7}};

In[11]:= R = FromMembershipMatrix[RMat];

In[12]:= S = FromMembershipMatrix[Smat];
```

```
In[13]:= FuzzyPlot3D[R, S, ShowDots → True];
```



The disjunctive sum of fuzzy relations R and S in the universal space VxW, can be found with the following formula.

$$\text{DisSum} = (R \cap S') \cup (R' \cap S)$$

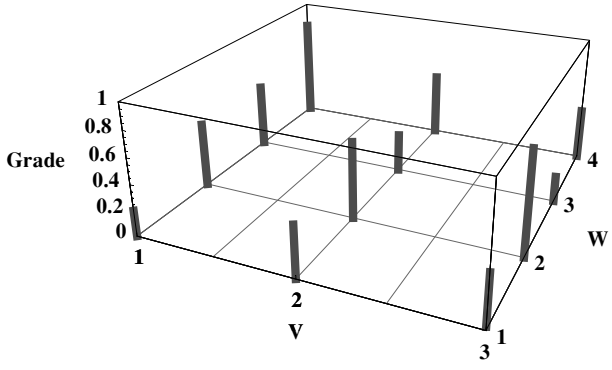
The disjunctive sum is thus a relation in VxW that has the following property:

For all (v, w) in VxW, DisSum(v, w) =
Max(Min(R(v, w), 1 - S(v, w)), Min(1 - R(v, w), S(v, w)))

Solution. We can find disjunctive sum of the two fuzzy relations R and S by using the formula derived earlier and some of the functions from the *Fuzzy Logic* package.

```
In[14]:= DisSum = Union[Intersection[R, Complement[S]], Intersection[Complement[R], S]];
```

```
In[15]:= FuzzyPlot3D[DisSum];
```




```
In[16]:= ToMembershipMatrix[DisSum] // MatrixForm
```

```
Out[16]//MatrixForm=
( 0.2  0.5  0.5  0.8 )
( 0.4  0.6  0.3  0.5 )
( 0.4  0.8  0.2  0.4 )
```

8.5 Example 4: Natural Numbers

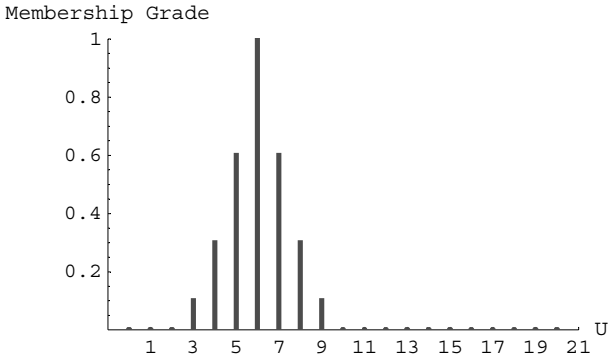
Problem. Suppose you are asked to define the set of natural numbers close to 6. There are a number of different ways in which you could accomplish this using fuzzy sets.

Solution 1. One solution would be to manually create a fuzzy set describing the numbers near 6. This can be done as follows:

```
In[17]:= SetOptions[FuzzySet, UniversalSpace → {0, 20}];
```

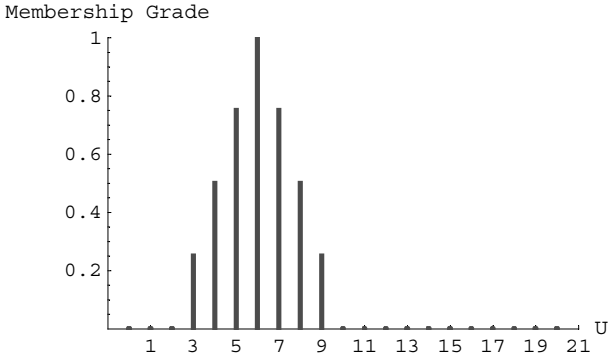
```
In[18]:= Six1 = FuzzySet[{{3, .1}, {4, .3}, {5, .6}, {6, 1.0}, {7, .6}, {8, .3}, {9, .1}}];
```

```
In[19]:= FuzzyPlot[Six1];
```



Solution 2. A second solution would be to use the `FuzzyTrapezoid` function to create the fuzzy set. For a case such as this, a triangular fuzzy set would probably be better than a trapezoid, so we set the middle two parameters of the `FuzzyTrapezoid` function to 6.

```
In[20]:= Six2 = FuzzyTrapezoid[2, 6, 6, 10];

In[21]:= FuzzyPlot[Six2];
```



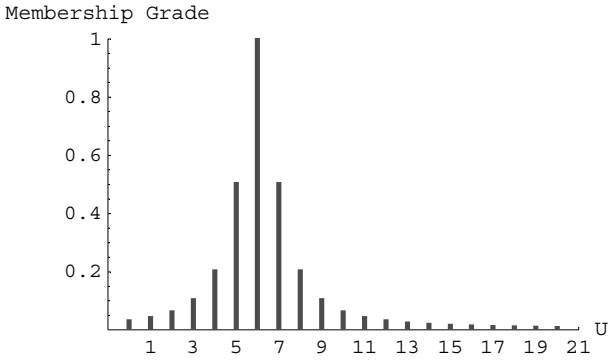
Solution 3. Another solution would be to use a function to create a fuzzy set representing numbers near 6.

$$\text{In[22]:= } \texttt{CloseTo[x\_] :=} \frac{1}{1 + (\#1 - x)^2} \texttt{\&}$$

We can use this function to create a fuzzy set for numbers near 6.

```
In[23]:= Six3 = CreateFuzzySet[CloseTo[6]];
```

```
In[24]:= FuzzyPlot[Six3];
```



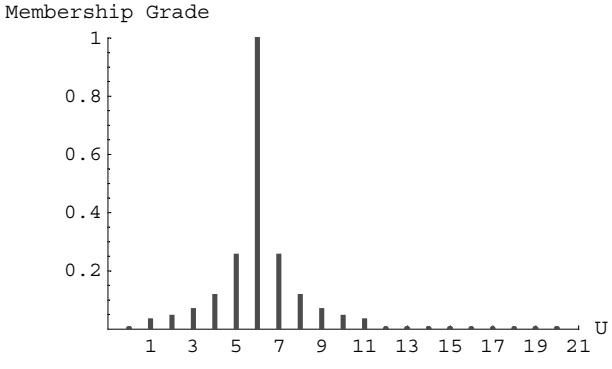
Note that this is a convenient method because the function `CloseTo` can be called with any integer argument to produce a fuzzy set close to that number.

Solution 4. Still another solution is to use a piecewise function to describe the fuzzy set.

```
In[25]:= NearSix[x_] := Which[x == 6, 1, x > 6 && x < 12, 1/(x - 5)^2, x < 6 && x > 0, 1/(7 - x)^2, True, 0]
```

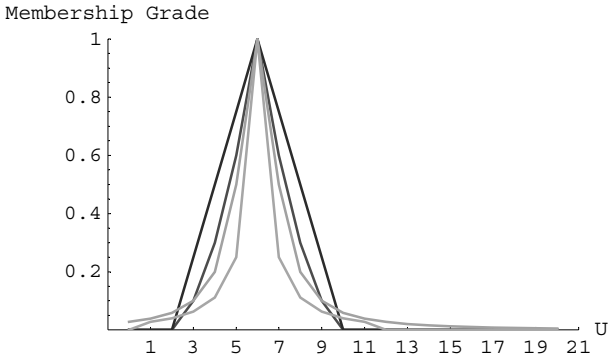
```
In[26]:= Six4 = CreateFuzzySet[NearSix];
```

```
In[27]:= FuzzyPlot[Six4];
```



Now, we can view all four of our fuzzy representations of the number six to see how they compare. We do this by plotting them all on the same graph with the `FuzzyPlot` function.

```
In[28]:= FuzzyPlot[Six1, Six2, Six3, Six4, PlotJoined → True];
```

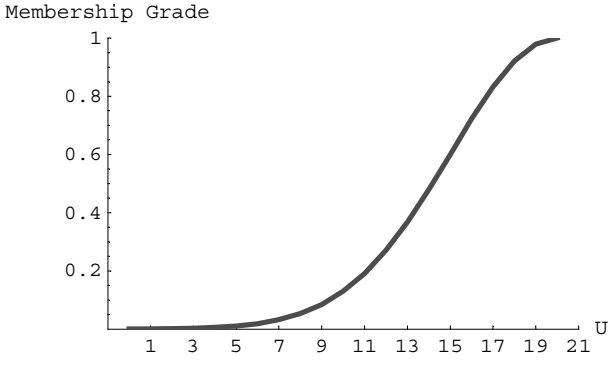


8.6 Example 5: Fuzzy Hedges

Problem. Suppose you had already defined a fuzzy set to describe a hot temperature.

```
In[29]:= Hot = FuzzyGaussian[20, 7, UniversalSpace → {0, 20}];
```

```
In[30]:= FuzzyPlot[Hot, PlotJoined → True];
```



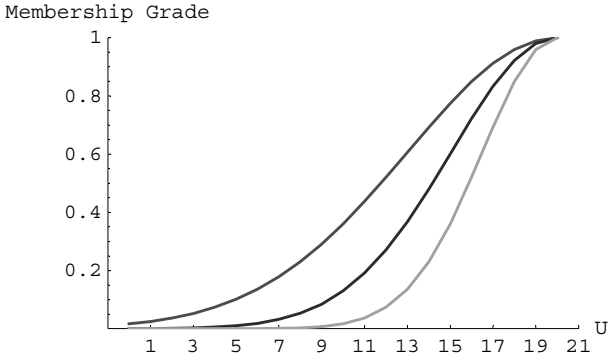
Now, suppose we want to talk about the degree to which something is hot. We need some sort of fuzzy modifier or a hedge to change our fuzzy set. Look at how we can accomplish this.

Solution. We can start by defining how a fuzzy set should be modified to represent the hedges "Very" and "Fairly." Two functions in the *Fuzzy Logic* package, `Concentrate` and `Dilate`, can be used to define our two hedges.

```
In[31]:= Very := Concentrate
         Fairly := Dilate
```

Now we can look at a graph of the fuzzy sets `FairlyHot`, `Hot`, and `VeryHot`.

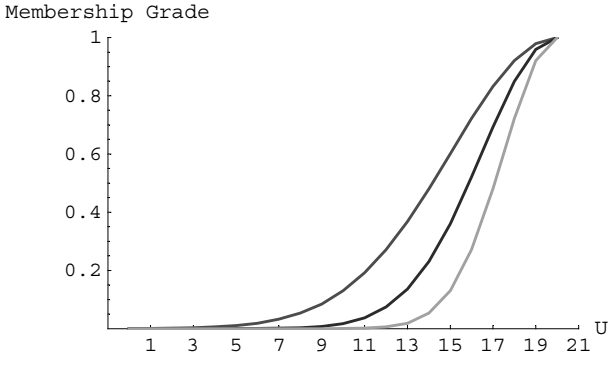
```
In[33]:= FuzzyPlot[Fairly[Hot], Hot, Very[Hot], PlotJoined → True];
```



Note that the `FairlyHot` membership function is a more general, spread-out fuzzy set. The `VeryHot` fuzzy set is a more focused, concentrated fuzzy set.

We can also apply more than one modifier to a fuzzy set. For instance, let us compare `Hot`, `VeryHot`, and `VeryVeryHot`.

```
In[34]:= FuzzyPlot[Hot, Very[Hot], Very[Very[Hot]], PlotJoined → True];
```



As we might expect, the `VeryVeryHot` fuzzy set is even more concentrated than the `VeryHot` fuzzy set.

8.7 Example 6: Distance Relation

Problem. Let R be a fuzzy relation between the sets, X = {NYC, Paris} and Y = {Beijing, NYC, London}, that represents the idea of "very far." In list notation, the relation could be represented as follows [Klir & Folger, 1988].

$$R(X,Y) = 1.0/NYC, Beijing + \ 0/NYC, NYC + 0.6/NYC, London$$
$$+ 0.9/Paris, Beijing + 0.7/Paris, NYC + 0.3/Paris, London$$

Solution. We can represent this fuzzy relation in *Mathematica* in the following way. We can start by creating the membership matrix to represent the relation.

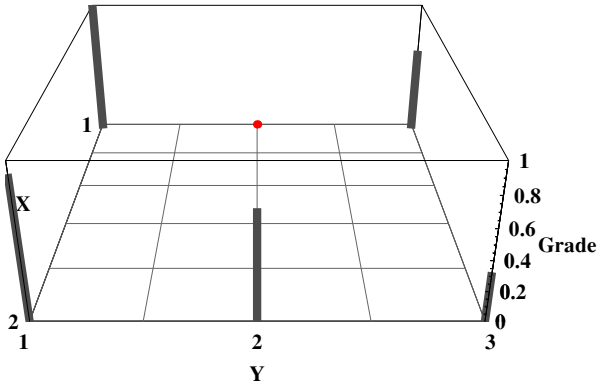
```
In[35]:= DistMat = {{1, 0, 0.6}, {0.9, 0.7, 0.3}}

Out[35]= {{1, 0, 0.6}, {0.9, 0.7, 0.3}}
```

We need to represent the cities in each set with numbers. For set X, let NYC be 1, and Paris be 2; for set Y, let Beijing be 1, NYC be 2, and London be 3. Now we can create the relation using the `FromMembershipMatrix` function.

```
In[36]:= DistRel = FromMembershipMatrix[DistMat, {{1, 2}, {1, 3}}]

Out[36]= FuzzyRelation[
          {{{1, 1}, 1}, {{1, 2}, 0}, {{1, 3}, 0.6}, {{2, 1}, 0.9}, {{2, 2}, 0.7}, {{2, 3}, 0.3}},
          UniversalSpace → {{1, 2, 1}, {1, 3, 1}}]
```

We can plot this relation using the `FuzzyPlot3D` function. We will use some of *Mathematica*'s `Plot3D` options to put the graph in a form that lines up with the membership matrix so that you can see the correlation.

```
In[37]:= FuzzyPlot3D[DistRel, AxesLabel → {" X", "Y", "Grade  "},
           ViewPoint → {2, 0, 1}, AxesEdge → {{-1, -1}, {1, -1}, {1, 1}}];
```




```
In[38]:= ToMembershipMatrix[DistRel] // MatrixForm
```

```
Out[38]//MatrixForm=
( 1    0    0.6 )
( 0.9  0.7  0.3 )
```

By customizing the graph, you can get it to match the membership matrix, which makes understanding the fuzzy relation easier.

8.8 Example 7: Choosing a Job

Problem. Fuzzy sets can be used to aid in decision making or management. We illustrate this with an example from Klir and Folger [Klir and Folger, 1988]. Given four jobs (Jobs 1, 2, 3, and 4), our task is to choose the job that will give us the highest salary, given the constraints that the job should be interesting and close to our home.

Solution. The first constraint of job interest can be represented with the following fuzzy set.

```
In[39]:= Interest = FuzzySet[{{1, .4}, {2, .6}, {3, .8}, {4, .6}}, UniversalSpace → {1, 4}]

Out[39]= FuzzySet[{{1, 0.4}, {2, 0.6}, {3, 0.8}, {4, 0.6}}, UniversalSpace → {1, 4, 1}]
```

We can see that Job 3 has the highest membership grade, meaning that Job 3 is the most interesting of the four jobs. Job 1 on the other hand is the least interesting, since it has a membership grade of only 0.4.

We can form a fuzzy set for our second constraint in a similar manner. Here is a fuzzy set used to represent the driving distance to the four jobs.

```
In[40]:= Drive = FuzzySet[{{1, .1}, {2, .9}, {3, .7}, {4, 1}}, UniversalSpace → {1, 4}]

Out[40]= FuzzySet[{{1, 0.1}, {2, 0.9}, {3, 0.7}, {4, 1}}, UniversalSpace → {1, 4, 1}]
```

In the fuzzy set above, the membership grades indicate the length of the drive to work. A high membership grade indicates that it is a short drive to work - a good thing. A small membership grade indicates an undesirable, long drive to work. From the fuzzy set above, we can see that Job 4 is located near our home, while Job 1 is a long way from our home.

Finally, we need to figure in the goal of a good salary. There is no real difference between a constraint and a goal in this problem, so we figure in the worth of the salary the same way we did for the previous constraints. We could use a formula to convert a salary into a membership grade for each job [Klir & Folger, 1988], but to stay with the tradition of our previous constraints, we arbitrarily assign a membership grade to each job based on salary.

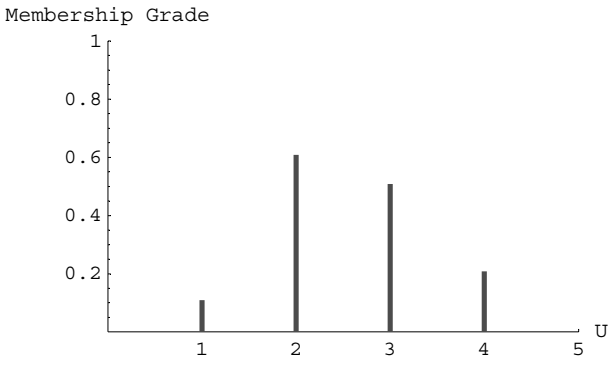
```
In[41]:= Salary = FuzzySet[{{1, .875}, {2, .7}, {3, .5}, {4, .2}}, UniversalSpace → {1, 4}]

Out[41]= FuzzySet[{{1, 0.875}, {2, 0.7}, {3, 0.5}, {4, 0.2}}, UniversalSpace → {1, 4, 1}]
```

From this fuzzy set, we see that Job 1 pays the highest salary, and Job 4 pays the lowest. Now that all of our criteria is represented as fuzzy sets, we need to decide on a function to make the decision. We will use the standard `Intersection` to make the fuzzy decision. Applying the `Intersection` operation can be thought of as adding the constraints and goals to come up with the best overall decision.

```
In[42]:= Decision = Intersection[Interest, Drive, Salary]

Out[42]= FuzzySet[{{1, 0.1}, {2, 0.6}, {3, 0.5}, {4, 0.2}}, UniversalSpace → {1, 4, 1}]
```

We can plot the decision fuzzy set to see the results graphically.

```
In[43]:= FuzzyPlot[Decision];
```



At last, we can look for the maximum membership grade to decide which job best satisfies our goals and constraints. In this example, we see that Job 2 appears to be the best job for us.

There are a number of different ways that the decision in this example could have been made. For example, we could have used a different operator, maybe a product operator, to make our decision; we could have weighted different constraints more heavily than others; or we could have used different functions to arrive at the membership grades. As an exercise, try using a different method and see which job your method selects as the best.

8.9 Example 8: Digital Fuzzy Sets

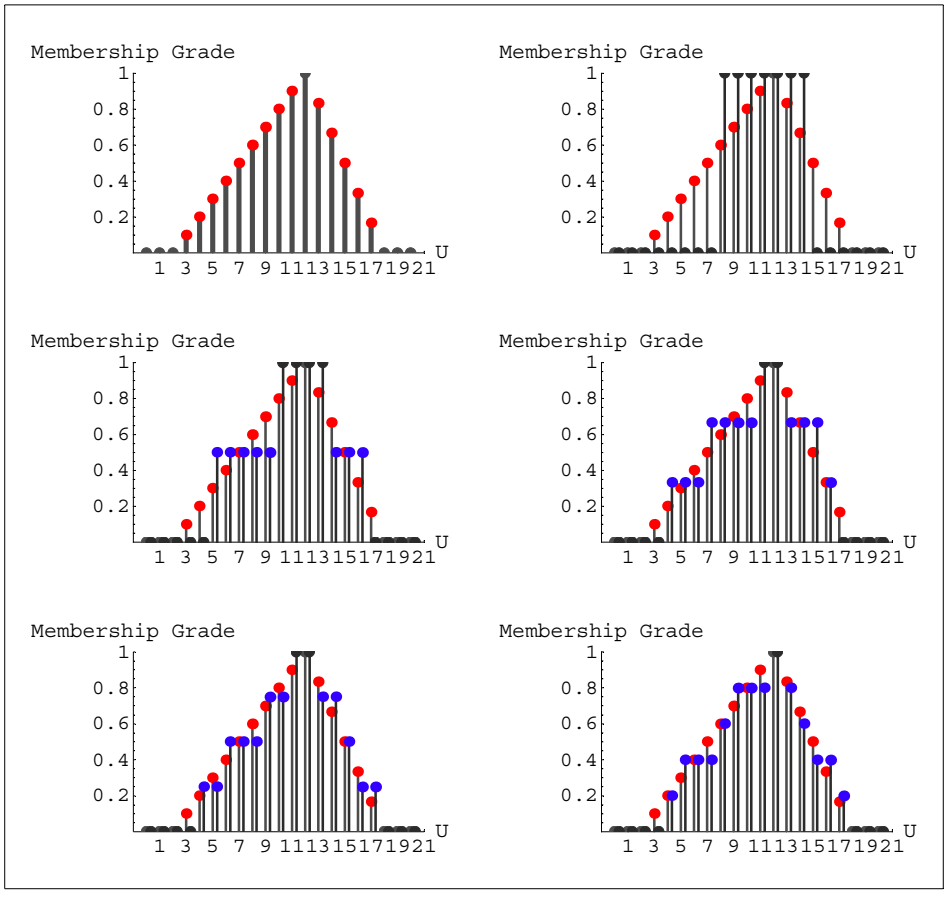
Problem. Suppose you are asked to compare fuzzy sets to digital fuzzy sets.

Solution. Below is an example of a set with 6 digital membership functions defined over the range from 0 to 20.

```
In[44]:= SetOptions[FuzzySet, UniversalSpace → {0, 20, 1}];
         SetOptions[FuzzyPlot, ShowDots → True];

In[46]:= Fset1 = FuzzyTrapezoid[2, 12, 12, 18];

In[47]:= DSet2 = ToDigital[Fset1, 2];
         DSet3 = ToDigital[Fset1, 3];
         DSet4 = ToDigital[Fset1, 4];
         DSet5 = ToDigital[Fset1, 5];
         DSet6 = ToDigital[Fset1, 6];
```

Łukasiewicz sets can be viewed and manipulated in the same manner as infinite valued fuzzy sets. For a graphic representation of the above set, execute the following fuzzy plot command.

```
In[52]:= Block[{$DisplayFunction = Identity},
          graphic = FuzzyPlot[Fset1]; graphic2 = FuzzyPlot[Fset1, DSet2];
          graphic3 = FuzzyPlot[Fset1, DSet3]; graphic4 = FuzzyPlot[Fset1, DSet4];
          graphic5 = FuzzyPlot[Fset1, DSet5]; graphic6 = FuzzyPlot[Fset1, DSet6]];

In[53]:= Show[GraphicsArray[
            {{graphic, graphic2}, {graphic3, graphic4}, {graphic5, graphic6}}], Frame → True];
```



Let us compare the discrete and digital form of our fuzzy sets using the concept of the Hamming distance.

```
In[54]:= {HammingDistance[Fset1, DSet2],
           HammingDistance[Fset1, DSet3], HammingDistance[Fset1, DSet4],
           HammingDistance[Fset1, DSet5], HammingDistance[Fset1, DSet6]} // N

Out[54]= {4., 1.86667, 1.33333, 0.933333, 0.8}
```

As you might expect, as n goes to infinity, Łukasiewicz sets become fuzzy set.

8.10 Example 9: Image Processing

Problem. The table shows the gray levels values for 8-bit image associated with an array of 25 pixels. Use the enhancement function on this image to recognize the pattern in the image.

```
In[55]:= image = {{0.55, 0.54, 0.55, 0.43, 0.44},
            {0.46, 0.53, 0.41, 0.52, 0.41}, {0.45, 0.57, 0.41, 0.44, 0.55},
            {0.43, 0.59, 0.45, 0.53, 0.42}, {0.58, 0.57, 0.6, 0.41, 0.43}};
```

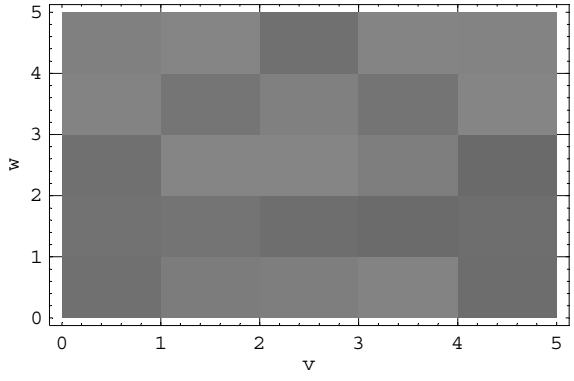
Solution. An image R of m × n dimensions can be considered as an array of fuzzy singletons, each with a value of membership denoting the gray level in the image. The object of contrast enhancement is to process a given image so that the result is more suitable than the original for a specific application in pattern recognition We will demonstrate enhancement of the image shown in next figure. First, let's add a function to the *Fuzzy Logic* package.

```
In[56]:= Needs["Graphics`Colors`"];

In[57]:= Options[DensePlot] = {Hue → .75};

In[58]:= DensePlot[FuzzyRelation[elems_, opts___], opts1___?OptionQ] := Module[{space, col},
           {space} = {UniversalSpace} /. Flatten[{opts}] /. Options[FuzzyRelation];
           {col} = {Hue} /. Flatten[{opts1}] /. Options[DensePlot]; Show[Graphics[RasterArray[
              Transpose[Map[(Hue[col, #1, .75] &) /@ #1 &, GradeMatrix[elems, space], 2]]],
             Frame → True, FrameLabel -> {v, w}, GridLines -> Automatic]]]
```

Now, we can plot the image.

```
In[59]:= fuzzyimage := FromMembershipMatrix[image, {{1, 5.0}, {1, 5}}]
```

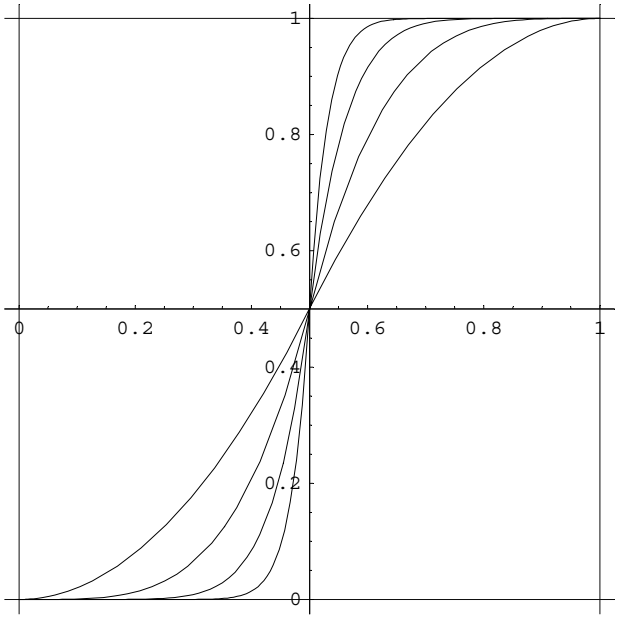
```
In[60]:= DensePlot[fuzzyimage];
```



In general, each membership value in fuzzyimage will be modified to a new membership value to enhance the image by transformation function `Contrast`. The `IntensifyContrast` operator applies a `Contrast` function and intensifies the contrast between the gray levels in the image. As the number of successive applications of the `IntensityContrast` increases, the slop of the curve gets steeper.

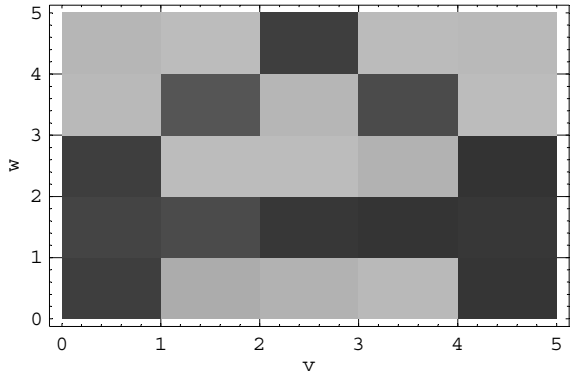
```
In[61]:= Contrast[t_] = Which[t ≤ 0.5, 2 * t^2, t > 0.5 && t ≤ 1, 1 - 2 (1 - t)^2];
```

```
In[62]:= Plot[{Contrast[t], Contrast[Contrast[t]], Contrast[Contrast[Contrast[t]]],
            Contrast[Contrast[Contrast[Contrast[t]]]]}, {t, 0, 1}, AspectRatio → 1,
          AxesOrigin → {0.5, 0.5}, GridLines → {{0, 0.5, 1}, {0, 0.5, 1}}];
```



The graphical effect of this recursive transformation for typical membership function is shown in next figure where the number of successive applications of the `IntensifyContrast` function increases from one to three.

```
In[63]:= image1 := IntensifyContrast[fuzzyimage]

In[64]:= image2 := IntensifyContrast[image1]

In[65]:= image3 := IntensifyContrast[image2]

In[66]:= image4 := IntensifyContrast[image3]

In[67]:= DensePlot[image4];
```



After the original image is processed by the `IntensityContrast` operator several times, it is undoubtedly more suitable for the subsequent pattern recognition and classification. We hope you recognize the pattern in the image. This is a letter **A.**

References

A. Kaufmann and M. M. Gupta, *Introduction to Fuzzy Arithmetic Theory and Application,* Van Nostrand Reinhold, New York, 1991.

G. J. Klir, and T. A. Folger*, Fuzzy Sets, Uncertainty, and Information*, Prentice Hall, Englewood Cliffs, NJ, 1988.

G. J. Klir, and Bo Yuan*, Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice Hall, Upper Saddle River, NJ, 1995.

G. J. Klir, Ute H. St. Clair, and Bo Yuan *, Fuzzy Set Theory*, Prentice Hall, Upper Saddle River, NJ, 1997.

T. J. Ross, *FuzzyLogic with Engineering Applications*, McGraw-Hill, Hightstown, NJ, 1995.

M. S. Stachowicz and M. E. Kochanska, *Graphic interpretation of fuzzy sets and fuzzy relations, Mathematics at the Service of Man,* Edited by A. Ballester, D. Cardus, and E. Trillas, based on materials of Second World Conf., Universidad Politecnica Las Palmas, Spain, 1982.

H. J. Zimmermann, *Fuzzy Set Theory and Its Applications,* 3rd ed., Kluwer Academic Publishers, Boston, MA, 1996.

Reference Guide

■ AbsoluteDifference

`AbsoluteDifference[A, B]` returns the fuzzy set/relation representing the absolute difference between fuzzy sets/relations *A* and *B*.

■ AlphaLevelSet

`AlphaLevelSet[A, b]` returns a list of the elements of fuzzy set `A` that have membership grades greater than or equal to *b*.

■ BisectorOfArea

`BisectorOfArea[A]` returns the defuzzified value of fuzzy set *A* using the bisector of area defuzzification strategy.

■ BuildModel

`BuildModel [{{A1, B1}, ... . {An, Bn}}]` creates a model fuzzy relation using the pairs of fuzzy sets *{Ax, Bx}*.

■ Cardinality

`Cardinality[A]` returns the sum of all of the membership grades in fuzzy set *A*.

■ CartesianProduct

`CartesianProduct [space]` returns the cartesian product of the sets represented by *space*.

■ CenterOfArea

CenterOfArea[*A*] returns the defuzzified value of fuzzy set *A* using the center of area (COA) defuzzification strategy.

■ Complement

Complement[*A*, *opts*] returns the complement of fuzzy set/relation *A*. This function has a Type option that can be set to Standard (default), Sugeno[*a*], or Yager[*w*].

■ Complete

Complete is an option for RandomFuzzyNumber and RandomFuzzyRelation.

■ CompositionBasedInference

CompositionBasedInference[*A*, *B*, *opts*] returns a crisp value that is the result of performing a maxmin composition between fuzzy set *A* and fuzzy relation *B* and defuzzifying the result."

■ Concentrate

Concentrate[*A*] returns a concentrated version of fuzzy set/relation *A*.

■ Core

Core[*A*] returns a list of all elements of fuzzy set *A* which have a membership grade of 1.

■ CreateFuzzyRelation

`CreateFuzzyRelation[`*func*`,`*({{a, b}, {c, d}})*`,` *opts*`]` returns a fuzzy relation with membership grades that are the result of applying *func* to all of the elements within the specified range (rows *a* to *b*, columns *c* to *d*). If no range is specified all elements in the universal space are used. The universal space can be set with an option.

■ CreateFuzzySet

`CreateFuzzySet[`*func*`,` *({a, b})*`,` *opts*`]` returns a fuzzy set with membership grades that are the result of applying *func* to all of the elements in the specified range *a* to *b*. If no range is specified, the function is applied to all elements in the universal space, which can be set as an option.

■ CreateFuzzySets

`CreateFuzzySets[`*num*`,` *opts*`]` returns a list with *num* evenly spaced fuzzy sets. The type of membership functions can be specified as an option and may be either `Triangular` or `Gaussian[`*sigma*`]`.

■ Crisp

`Crisp` is an option for `FuzzyPlot`.

■ Defuzzify

`Defuzzify` is an option for `CompositionBasedInference`.

■ Difference

`Difference[`*A*`,`*B*`]` return the fuzzy set/relation representing the difference between fuzzy sets/relations *A* and *B*, which is defined as the intersection between *A* and the complement of *B*.

■ DigitalSet

DigitalSet[*a*, *b*, *c*, *d*, *(h)*, *opts*] returns a Lukasiewicz set with membership grades which linearly increase from zero to *h* in the range *a* to *b*, is *h* in the range *b* to *c*, and linearly decrease from h to zero in the range *c* to *d*. Arguments *a*, *b*, *c*, and *d* must be integers in increasing order, *n* must be greater than or equal to 2, and *h* must be a value between 0 and 1. If *h* isn't specified a value of 1 is assumed. *h* and all other membership values will be assigned the closest possible membership value for the *n*-valued logic. A universal space and the *n* value may be specified as an option.

■ Dilate

Dilate[*A*] returns a dilated version of fuzzy set/relation *A*.

■ DiscreteFuzzyImage

DiscreteFuzzyImage[*A*] returns a fuzzy set that is the image of fuzzy set *A*.

■ DiscreteFuzzyMinus

DiscreteFuzzyMinus[*A*, *B*] returns a fuzzy set that is the result of applying fuzzy subtraction to fuzzy set *A* and *B*.

■ DiscreteFuzzyMultiply

DiscreteFuzzyMultiply[*A*, *B*] returns a fuzzy set that is the result of applying fuzzy multiplication to fuzzy sets *A* and *B*.

■ DiscreteFuzzyPlus

DiscreteFuzzyPlus[*A*, *B*] returns a fuzzy set that is the result of applying fuzzy addition to fuzzy sets *A* and *B*.

■ Dombi

Dombi[*a*] is an option choice for the union and intersection operations. The parameter *a* can range from 0 to infinity.

■ DuboisPrade

DuboisPrade[*a*] is an option choice for the union and intersection operations. The parameter a can range from 0 to 1.

■ Equality

Equality[*A*, *B*] returns true if two fuzzy sets/relations are equal and false otherwise.

■ EquilibriumSet

EquilibriumSet[*A*] returns a list of the elements of fuzzy set *A* for which the membership grade is equal to (1 - membership grade).

■ Extend

Extend[*A*, *B*] returns a fuzzy relation that is the cylindrical extension of fuzzy set *A* over the range *B*.

■ FCMCluster

FCMCluster[*data*, *partmat*, *mu*, *epsilon*] returns a list of cluster centers, a partition matrix indicating the degree to which each data point belongs to a particular cluster center, and a list containing the progression of cluster centers found during the run. The arguments to the function are the data set (*data*), a partition matrix (*partmat*), a value determining the degree of fuzziness of the clustering (*mu*), and a value which determines when the algorithm will terminate (*epsilon*).

■ FindFuzzyRelation

`FindFuzzyRelation[`*A*`, `*B*`]` find the relation which gives fuzzy set *B* as a result of performing a composition based inference with input fuzzy set *A*.

■ FindFuzzySet

`FindFuzzySet[`*A*`, `*B*`]` finds the fuzzy set, which when composed with fuzzy relation *A* gives fuzzy set *B*.

■ FirstProjection

`FirstProjection[`*A*`]` returns a fuzzy set which is the first projection of fuzzy relation *A*.

■ Frank

`Frank[`*s*`]` is an option choice for the union and intersection operations. The parameter *s* can range from 0 to infinity.

■ FromMembershipMatrix

`FromMembershipMatrix[`*mat*`, `*({{va, vb}, {wa, wb}})*`]` returns a fuzzy relation using the matrix of membership grades for the elements in the range of rows from *va* to *vb* and columns from *wa* to *wb*. If no range is specified, it uses the elements from 1 through the dimension of the matrix.

■ FuzzyArithmeticMean

`FuzzyArithmeticMean[`*A1*`, `*A2*`, ... , `*An*`]` returns a fuzzy set/relation that is the arithmetic mean of fuzzy sets/relations *A1, ... , An*.

■ FuzzyBell

FuzzyBell[*c*, *w*, *s*] returns a bell-shaped fuzzy set centered at *c* with crossover points at *c* - *w* and *c* + *w* with a slope of *s* / 2*w* at the crossover points. The universal space may be specified as an option.

■ FuzzyCardinality

FuzzyCardinality[*A*] returns the fuzzy cardinality of fuzzy set *A*.

■ FuzzyConstantTimes

FuzzyConstantTimes[*{a, b, c, d}*, *k*] returns an unevaluated FuzzyTrapezoid representing the fuzzy number which is the result of multiplying the fuzzy number *{a, b, c, d}* by *k*.

■ FuzzyDivide

FuzzyDivide[*{a1, b1, c1, d1}*, *{a2, b2, c2, d2}*] returns an unevaluated FuzzyTrapezoid representing the division of the fuzzy numbers represented by the two lists.

■ FuzzyGaussian

FuzzyGaussian[*mu*, *sigma*] returns a gaussian fuzzy set with center mu and width sigma. The universal space may be specified as an option.

■ FuzzyGeometricMean

FuzzyGeometricMean[*A1*, *A2*, ... , *An*] returns a fuzzy set/relation that is the geometric mean of fuzzy sets/relations *A1*, ... , *An*.

■ FuzzyGraph

FuzzyGraph[*A*] returns a fuzzy graph for a set of control rules *A*.

■ FuzzyHarmonicMean

FuzzyHarmonicMean[*A1*, *A2*, ... , *An*] returns a fuzzy set/relation which is the harmonic mean of fuzzy sets/relations *A1*, ... , *An*.

■ FuzzyImage

FuzzyImage[*{a, b, c, d}*] returns an unevaluated FuzzyTrapezoid representing the image of the fuzzy number represented by the list *{a, b, c, d}*. To evaluate the FuzzyTrapezoid use ReleaseHold.

■ FuzzyMinus

FuzzyMinus[*{a1, b1, c1, d1}*, *{a2, b2, c2, d2}*] returns an unevaluated FuzzyTrapezoid representing fuzzy difference between the two fuzzy numbers represented by the two lists. To evaluate the FuzzyTrapezoid use ReleaseHold.

■ FuzzyModify

FuzzyModify[*func*, *A*] returns a fuzzy set that is the result of applying *func* to all the membership grades of fuzzy set *A*.

■ FuzzyMultiply

FuzzyMultiply[*{a1, b1, c1, d1}*, *{a2, b2, c2, d2}*] returns an unevaluated FuzzyTrapezoid representing the product of the fuzzy numbers represented by the two lists.

■ FuzzyPlot

`FuzzyPlot[`*A1*`, `*A2*`, ... , `*An*`]` will plot a fuzzy set(s) as a series of vertical lines that correspond to membership grade. Options include `PlotJoined` for producing a linear plot, `Crisp` for plotting crisp sets, and `ShowDots` for plotting a dot on each vertical line. Standard plot options will still work with this function.

■ FuzzyPlot3D

`FuzzyPlot3D[A1, A2,... ,An]` will plot a fuzzy relation(s) as a collection of vertical lines corresponding to membership grades. Options include `ShowDots` for plotting a dot on top of each vertical line. Standard plot option will work with this function.

■ FuzzyPlus

`FuzzyPlus[{a1,b1,c1,d1},{a2,b2,c2,d2}]` returns an unevaluated `FuzzyTrapezoid` that represents the sum of the fuzzy numbers represented by the two lists. To evaluate the `FuzzyTrapezoid` use `ReleaseHold`.

■ FuzzyRelation

`FuzzyRelation` is an object of form `FuzzyRelation[`*{{{x1, y1}, z1}, {{x2, y2}, z2}, ...}*`,UniversalSpace->`*{{xmin, xmax}, {ymin, ymax}}*`]`.

■ FuzzyRelationQ

`FuzzyRelationQ[`*A*`]` tests if object *A* is a valid fuzzy relation.

■ FuzzySet

FuzzySet is an object of form FuzzySet[*{{x1 ,y1}, {x2, y2},...,{xn, yn}}*, UniversalSpace->*{min, max}*].

■ FuzzySetQ

FuzzySetQ[*A*] tests if object *A* is a valid fuzzy set.

■ FuzzySigmoid

FuzzySigmoid[*c*, *s*, *opts*] returns a sigmoidal fuzzy set where *s* controls the slope at the crossover point *c*. The universal space may be specified as an option.

■ FuzzySurfacePlot

FuzzySurfacePlot[*A1*, *A2*, ... , *An*] will display a surface plot of fuzzy relation *A*. Options for this function include HideSurfaces for plotting a wire mesh without the polygon surfaces filled in. Standard plotting options will also apply.

■ FuzzyTrapezoid

FuzzyTrapezoid[*a*, *b*, *c*, *d*, *(h)*, *opts*] returns a trapezoidal fuzzy set of height h, which is must be between 0 and 1. The points *a*, *b*, *c*, and *d* are the vertices of the trapezoid in increasing order. The universal space can optionally be set with this function, and the default height is 1.

FuzzyTrapezoid[*{ax, bx, cx, dx}*, *{ay, by, cy, dy}*, *(h)*, *opts*] returns a trapezoidal fuzzy relation with membership grades which linearly increases from zero to *h* in both the x and y directions from *a* to *b*, is *h* from *b* to *c*, and linearly decreases from *h* to zero from *c* to *d*. If *h* isn't explicitly given the function uses $h = 1$.

■ FuzzyTwoGaussian

FuzzyTwoGaussian[*mu1*, *sigma1*, *mu2*, *sigma2*] returns a two-sided gaussian fuzzy set with centers at *mu1* and *mu2* and widths of *sigma1* and *sigma2*. The universal space may be specified as an option.

■ Gaussian

Gaussian[*sigma*] is an option choice for CreateFuzzySets that specifies that Gaussian fuzzy sets with width *sigma* should be created

■ GeneralAggregator

GeneralAggregator[*func*, *A*, *B*] returns a new fuzzy set/relation that is the result of applying the user defined *func* to the membership grades of the fuzzy sets/relations *A* and *B*.

■ GeneralizedMean

GeneralizedMean[*A1*, *A2*, ... , *An*, *alpha*] returns a fuzzy set/relation that is the generalized mean of fuzzy sets/relations *A1, A2, ... , An* with parameter *alpha* in the range (0, Infinity).

■ GlobalProjection

GlobalProjection[*A*] returns the max membership grade of fuzzy relation *A*.

■ Goedel

Goedel is a Type option for the RuleBasedInference function.

■ GradeMatrix

`GradeMatrix` returns the membership grades of a relation in matrix form.

■ Hamacher

`Hamacher[`*v*`]` is an option choice for the union and intersection operations. The parameter *v* can range from 0 to infinity.

■ HammingDistance

`HammingDistance[`*A*`, `*B*`]` returns the Hamming distance between fuzzy sets *A* and *B*.

■ Height

`Height[`*A*`]` returns the largest membership grade among the elements in fuzzy set/relation *A*.

■ HideSurfaces

`HideSurfaces` is an option for `FuzzySurfacePlot`.

■ Identify

`Identify[`*object*`, `*terms*`, `*hedges*`, `*rules*`]` returns the fuzzy term and hedge which is closest to the fuzzy object.

■ Implication

`Implication[`*A*`, `*B*`]` returns the implication of two fuzzy sets *A* and *B*.

■ Included

`Included[A, B]` returns true if fuzzy set/relation *A* is included in fuzzy set/relation *B* and false if not.

■ InitializeU

`InitializeU[`*data*`, `*n*`]` returns a random initial partition matrix for use with the `FCMCluster` function, where *n* is the number of cluster centers desired.

■ IntensifyContrast

`IntensifyContrast[A]` modifies fuzzy set/relation *A* so that there is more of a contrast between membership grades.

■ Intersection

`Intersection[`*A1*`, `*A2*`, ... , `*An*`, `*opts*`]` returns a fuzzy set/relation that is the intersection of fuzzy sets/relations *A1*, *A2*, ... , *An*. This function has a `Type` option, which can be `Standard` (default), `Hamacher[`*v*`]`, `Yager[`*w*`]`, `Frank[`*s*`]`, `Dombi[`*alpha*`]`, or `DuboisPrade[`*alpha*`]`.

■ LargestOfMax

`LargestOfMax[A]` returns the defuzzified value of fuzzy set *A* using the largest of maximum defuzzification strategy.

■ Levels

`Levels` is an option for `DigitalSet` which specifies the number of different levels or membership values to use.

■ LevelSet

`LevelSet[`*A*`]` returns the level set of fuzzy set or fuzzy relation *A*.

■ Mamdani

`Mamdani` is a `Type` option for the `RuleBasedInference` function.

■ MAX

`MAX[`*A*`, `*B*`]` returns the maximum of the fuzzy numbers *A* and *B*.

■ MaxMin

`MaxMin` is a `Type` option for the `Composition` function.

■ MaxProduct

`MaxProduct` is a `Type` option for the `Composition` function.

■ MaxStar

`MaxStar[`*fun*`]` is a `Type` option for the `Composition` function, where *fun* is a user defined function to be used with the composition.

■ MeanOfMax

`MeanOfMax[`*A*`]` returns the defuzzified value of fuzzy set *A* using the mean of maximum (MOM) defuzzification strategy.

■ MIN

`MIN[`A`, `B`]` returns the minimum of the fuzzy numbers A and B.

■ NewElements

`NewElements` performs the functions necessary to take the composition of relations.

■ Normal

`Normal` is an option for `RandomFuzzySet` and `RandomFuzzyRelation`, which when set to `True`, returns a normal fuzzy set or fuzzy relation.

■ Normalize

`Normalize[`A`]` returns fuzzy set/relation A with its height normalized to 1.

■ Opposite

`Opposite[`A`]` returns a fuzzy set which is the antonym of fuzzy set A.

■ PrimitiveMatrix

`PrimitiveMatrix[`n`, `*operation*`]` returns a logic table which shows the results of the *operation* for n valued logic sets.

■ RandomFuzzyRelation

RandomFuzzyRelation[{{a, b}, {c, d}}] returns a random fuzzy relation with a universal space of {{a, b}, {c, d}}. This function has options which allow the user to create Trapezoid or Complete random fuzzy relation. There is also a Normal option, which when set to True returns a normal random fuzzy relation.

■ RandomFuzzySet

RandomFuzzySet[{a, b}] returns a random fuzzy set with a universal space from a to b. This function has options that allow the user to create Trapezoid, Gaussian, Triangular, or Complete random fuzzy sets. There is also a Normal option which when set to True returns a normal random fuzzy set.

■ RelativeCardinality

RelativeCardinality[A] returns the value of the cardinality of fuzzy set A divided by the number of elements in the universal space.

■ RuleBasedInference

RuleBasedInference[{A1, ... , An}, {B1, ... , Bn}, {C1, ... , Cn}, {{Ax, Bx, Cx}, ...}, a, b] returns a crisp value that is the result of performing rule based inference where {A1, ... , An} and {Bn, ... , Bn} represent linguistic input variables, {C1, ... , Cn} is the linguistic output variable, rules are given in a list of triples like {Ax, Bx, Cx}, and the crisp values for the inputs are *a* and *b*.

■ Scaled

Scaled is a Type option for the RuleBasedInference function.

■ SecondProjection

`SecondProjection[`*A*`]` returns a fuzzy set that is the second projection of fuzzy relation *A*.

■ SetsToRelation

`SetsToRelation[`*func*`,` *A*`,` *B*`]` returns a fuzzy relation with elements representing the Cartesian product of the universal spaces of fuzzy sets *A* and *B* and membership grades which are the result of applying *func* to the corresponding membership grades of *A* and *B*.

■ ShowCenters

`ShowCenters[`*graph*`,` *res*`]` displays a 2-D plot showing a graph of a set of data points along with large dots indicating the cluster centers found by the `FCMCluster` function. The variable graph is a plot of the data points and *res* is the results from the `FCMCluster` function.

■ ShowCentersProgression

`ShowCentersProgression[`*graph*`,` *res*`]` displays a 2-D plot showing a graph of a set of data points along with a plot of how the cluster centers migrated during the application of the `FCMCluster` function.

■ ShowDots

`ShowDots` is an option for the fuzzy plotting functions.

■ ShowGraph

`ShowGraph` is an option for `CenterOfArea` and `MeanOfMax` that causes the original fuzzy set and its defuzzification to be shown in graphical form.

■ Similarity

`Similarity[`*A*`, `*B*`]` returns the similarity between fuzzy sets *A* and *B*.

■ SmallestOfMax

`SmallestOfMax[`*A*`]` returns the defuzzified value of fuzzy set *A* using the smallest of maximum defuzzification strategy.

■ Standard

`Standard` is an option choice for `Complement`, `Union`, and `Intersection`.

■ StrongAlphaLevelSet

`StrongAlphaLevelSet[`*A*`, `*b*`]` returns a list of the elements of fuzzy set *A* that have membership grades greater than *b*.

■ Subsethood

`Subsethood[`*A*`, `*B*`]` returns the degree of subsethood of fuzzy set *A* in fuzzy set *B*.

■ Sugeno

`Sugeno[`*a*`]` is an option choice for the complement operator. The parameter *a* can range from -1 to infinity.

■ Support

`Support[`*A*`]` returns a list of the elements of fuzzy set *A* with nonzero membership grades.

■ SymmetricDifference

SymmetricDifference[*A*, *B*] returns the fuzzy set/relation representing the symmetric difference between fuzzy sets/relations *A* and *B*.

■ ToDigital

ToDigital[*A*, *n*] takes a fuzzy set and an integer as input, and it returns a Lukasiewicz set using *n*-valued logic.

■ ToMembershipMatrix

ToMembershipMatrix[*A*] displays the membership matrix of fuzzy relation *A*.

■ Triangular

Triangular is an option choice for CreateFuzzySets that specifies that triangular fuzzy sets should be created.

■ Type

Type is an option used with Complement, Union, Intersection, CreateFuzzySets, and Composition.

■ Union

Union[*A1*, *A2*, ... , *An*, *opts*] returns a fuzzy set/relation that is the union of fuzzy sets/relations *A1*, *A2*, ... , *An*. This function has a Type option, which may be set to Standard (default), Hamacher[*v*], Yager[*w*], Frank[*s*], Dombi[*alpha*], or DuboisPrade[*alpha*].

■ UniversalSpace

`UniversalSpace` is an option for `FuzzySet` and `FuzzyRelation`. It describes the set of values over which a fuzzy set/relation is valid.

■ Weber

`Weber[`*l*`]` is and options for the union and intersection operations. The parameter *l* must be greater than -1.

■ Yager

`Yager[`*w*`]` ia an option choice for complement, union, and intersection operations, where *w* is a parameter ranging from 0 to infinity.

■ Yu

`Yu[`*l*`]` is an option for the union and intersection operations. The parameter *l* must be greater than -1.

■ ZeroPadSet

`ZeroPadSet` pads a set with zeros.

■ ZeroPadSetMembers

`ZeroPadSetMembers[`*elems*`,` *space*`]` gets the full set of membership grades with zeros padded.

Index