New Feature Documentation

This notebook demonstrates the new features of *Fuzzy Logic* 2.

New Definition of Universal Space

The first thing to note is that originally the universal space for fuzzy sets in *Fuzzy Logic* 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.

```
<< FuzzyLogic`
```

```
SetOptions[FuzzySet, UniversalSpace → {0, 100, 1}];
```

New Membership Functions

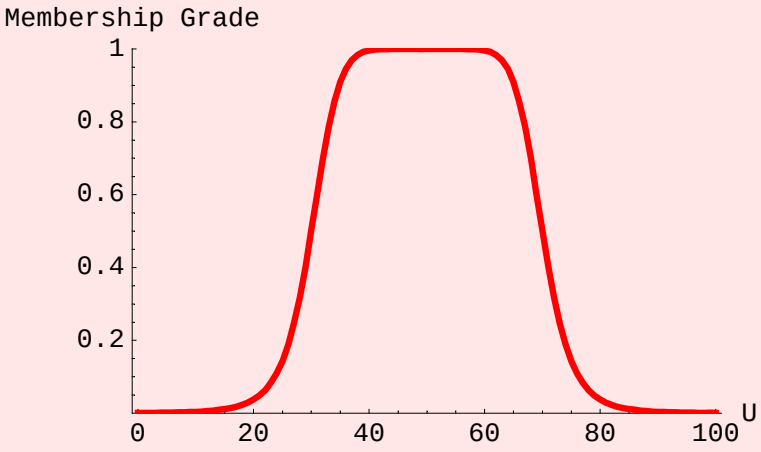
A number of new functions and options were added for creating membership functions. Here is a demonstration of the additions.

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*, and with a slope of *s* / *2w* at the crossover points.

```
FS1 = FuzzyBell[50, 20, 4];
```

```
FuzzyPlot[FS1, PlotJoined → True];
```

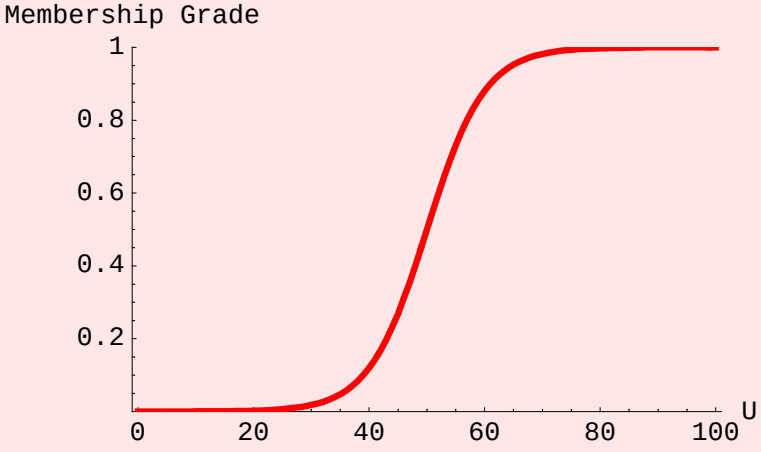


Sigmoidal Fuzzy Sets

`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.

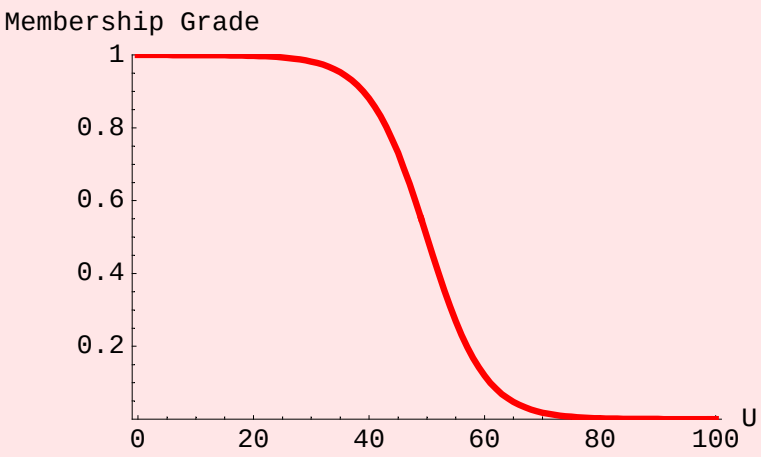
```
FS2 = FuzzySigmoid[50, .2];
```

```
FuzzyPlot[FS2, PlotJoined → True];
```




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

```
FuzzyPlot[FS3, PlotJoined → True];
```

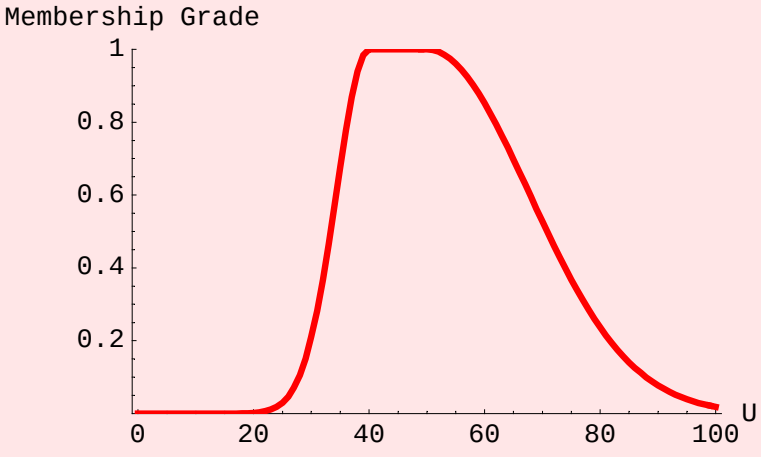


Double-Sided Gaussian Fuzzy Sets

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 means, the fuzzy set has a membership grade of 1.

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

```
FuzzyPlot[FS4, PlotJoined → True];
```

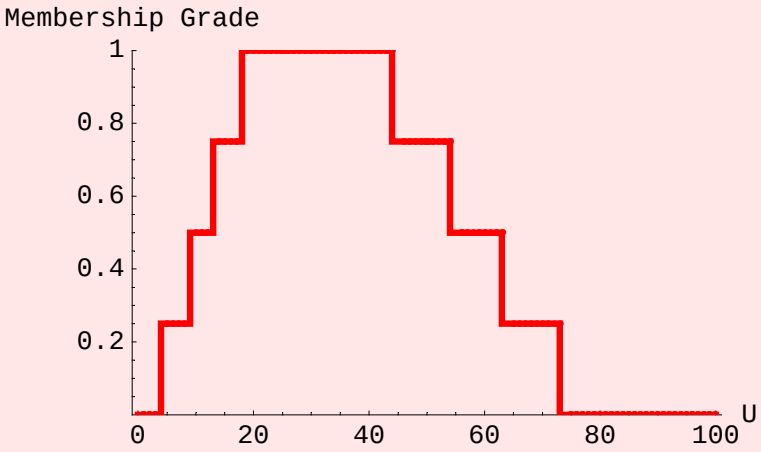


Digital Fuzzy Sets

`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 the default *n* value are 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 an L5 fuzzy set or a fuzzy set with five possible membership grades.

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

```
FuzzyPlot[D1, Crisp -> True];
```



ChopValue Option

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

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

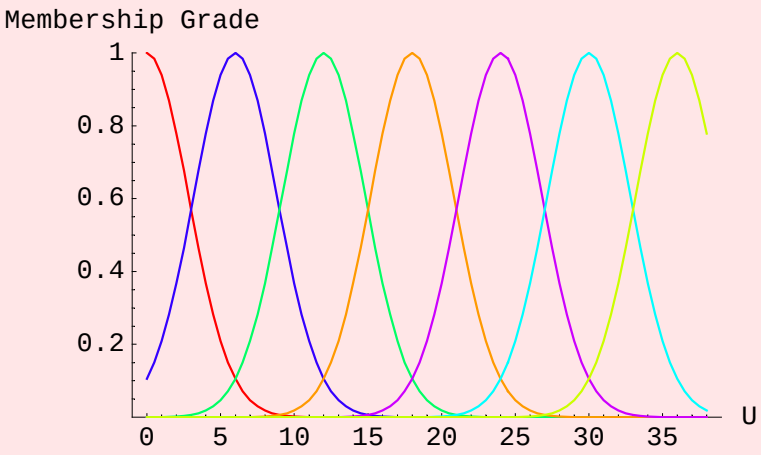
```
FuzzySet[
 {{0, 1.12535×10^-7}, {1, 5.35535×10^-7}, {2, 2.35258×10^-6},
  {3, 9.54016×10^-6}, {4, 0.0000357128}, {5, 0.00012341},
  {6, 0.000393669}, {7, 0.00115923}, {8, 0.00315111},
  {9, 0.00790705}, {10, 0.0183156}, {11, 0.0391639},
  {12, 0.0773047}, {13, 0.140858}, {14, 0.236928},
  {15, 0.367879}, {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}]
```

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

```
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}]
```

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

```
FuzzyPlot [Var2, PlotJoined → True];
```



Fuzzy Graph

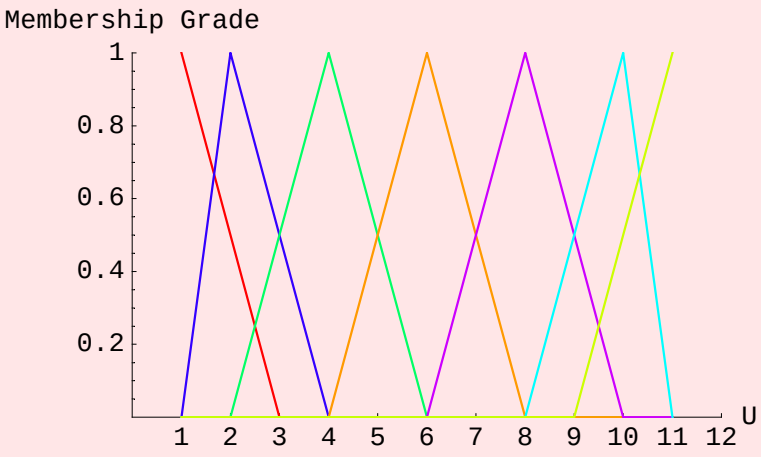
Another visualization function in *Fuzzy Logic* is a fuzzy graph. A fuzzy graph describes a mapping between an input linguistic variable and an output linguistic variable. In essence, a fuzzy graph serves as an approximation to a function, which is described as a collection of fuzzy *if-then* rules.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

Demonstration

```
SetOptions[FuzzySet, UniversalSpace → {1, 11, 1}];
```

```
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]};
```

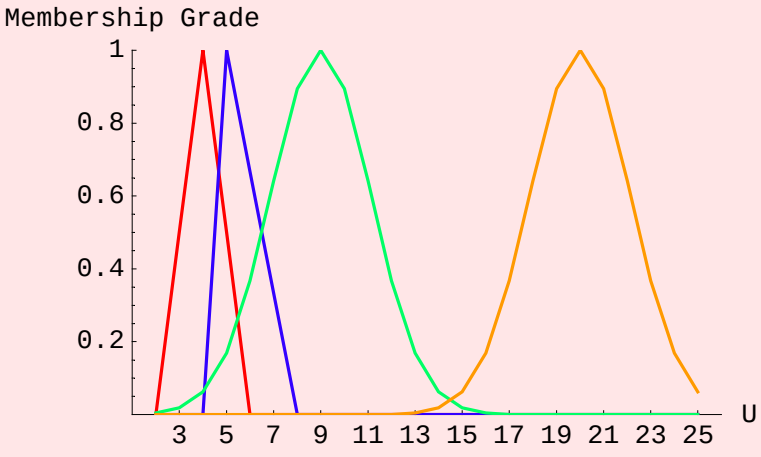
```
FuzzyPlot[Input1, PlotJoined → True];
```




```
SetOptions[FuzzySet, UniversalSpace → {2, 25, 1}];
```

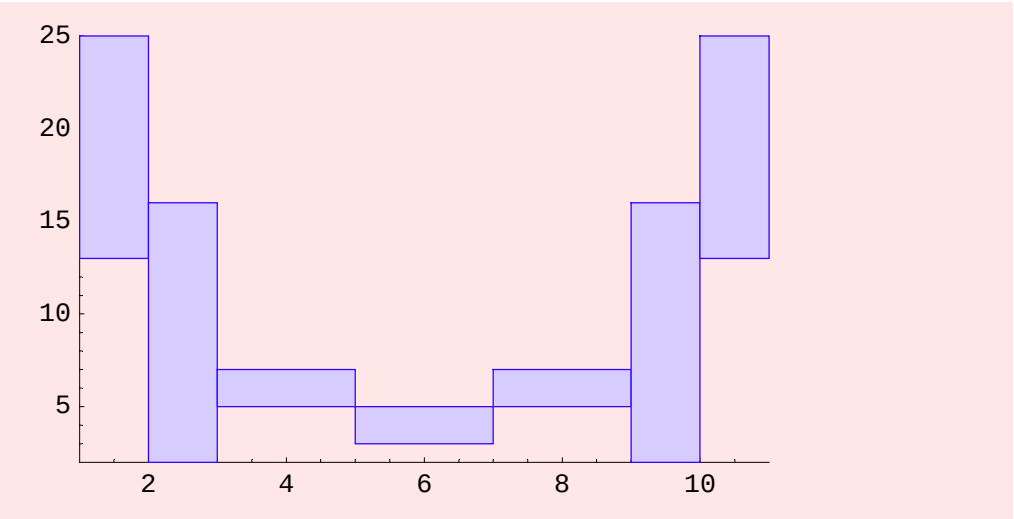
```
Output1 = {VeryLow, Low, Middle, High} =
   {FuzzyTrapezoid[2, 4, 4, 6], FuzzyTrapezoid[4, 5, 5, 8],
    FuzzyGaussian[9, 3, ChopValue → .001],
    FuzzyGaussian[20, 3, ChopValue → .001]};
```

```
FuzzyPlot[Output1, PlotJoined → True];
```




```
Rules1 = {{Tiny, High}, {VerySmall, Middle},
   {Small, Low}, {Medium, VeryLow}, {Big, Low},
   {VeryBig, Middle}, {Huge, High}};
```

```
FuzzyGraph[Rules1];
```



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

Defuzzifications

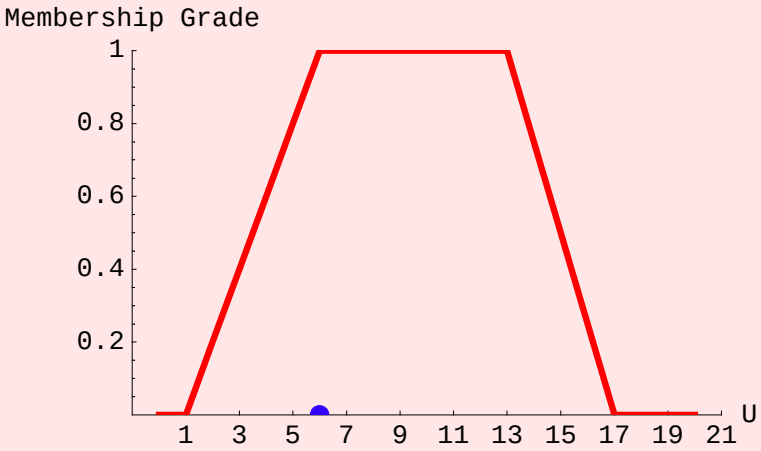
```
MF1 = FuzzyTrapezoid[1, 6, 13, 17, UniversalSpace → {0, 20}];
```

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.

```
SmallestOfMax[MF1, ShowGraph → True, PlotJoined → True];
```

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

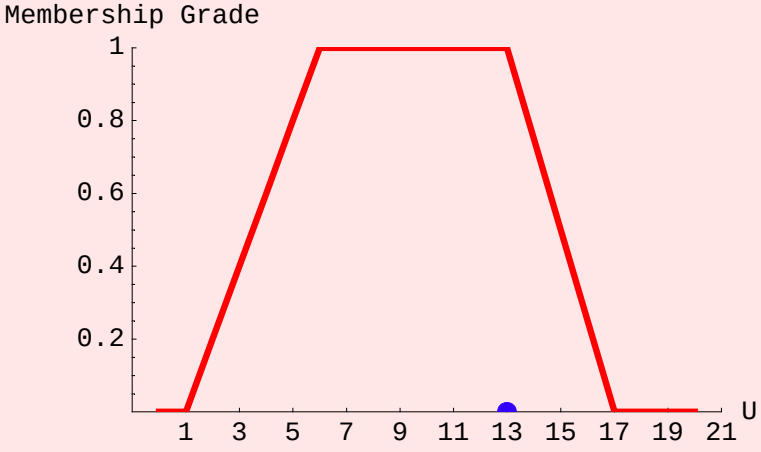


Largest of Max (Defuzzification)

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

```
LargestOfMax[MF1, ShowGraph → True, PlotJoined → True];
```

```
Largest of max is 13.
```

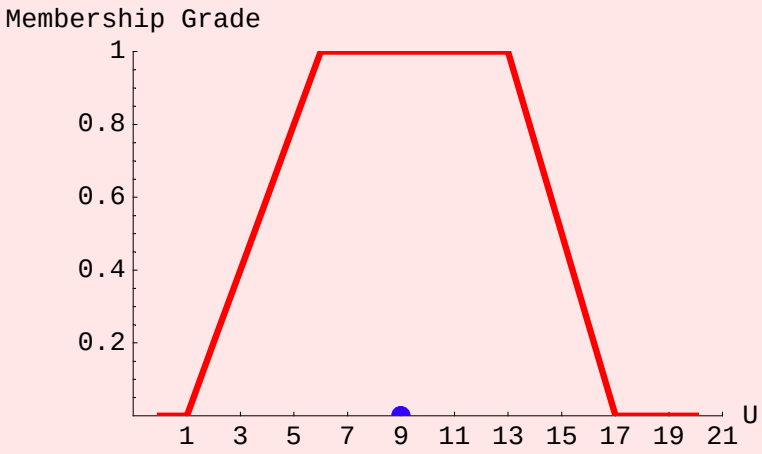


Bisector of Area (Defuzzification)

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

```
BisectorOfArea[MF1, ShowGraph → True, PlotJoined → True];
```

```
Bisector of area is 9.
```



Additional Operators

Fuzzy Cardinality

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

```
FuzzyCardinality[MF1]
```

$$\left\{\{15, \frac{1}{5}\}, \{14, \frac{1}{4}\}, \{13, \frac{2}{5}\}, \{12, \frac{1}{2}\}, \{11, \frac{3}{5}\}, \{10, \frac{3}{4}\}, \{9, \frac{4}{5}\}, \{8, 1\}\right\}$$

Core

The function `Core` was included in the original version of *Fuzzy Logic*, but it was named `Nucleus`.

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

```
Core[FS4]
```

```
{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 is 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.

```
Subsethood[FS1, FS2]
```

```
0.509356
```

```
Subsethood[FS2, FS1]
```

```
0.413921
```

Hamming Distance

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

```
HammingDistance[FS1, FS2]
```

```
49.7321
```

Level Set

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

```
LevelSet[FS1]
```

```
{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.}
```

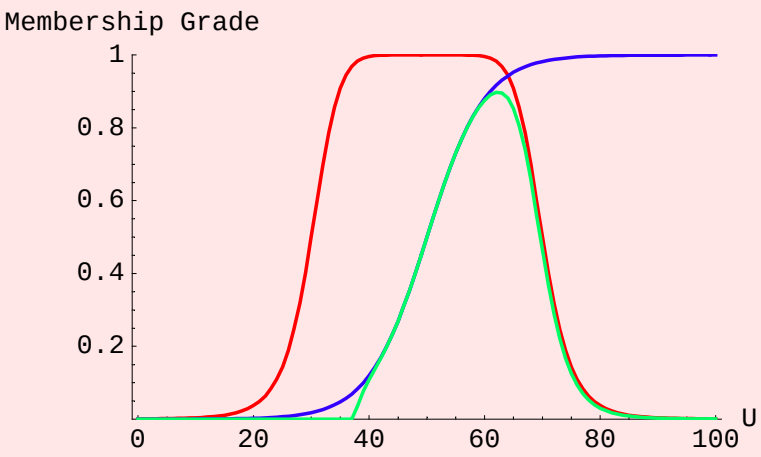
New Intersections and Unions

Yu-Type Union and Intersection

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

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

```
FuzzyPlot[FS1, FS2, YuInt, PlotJoined → True];
```

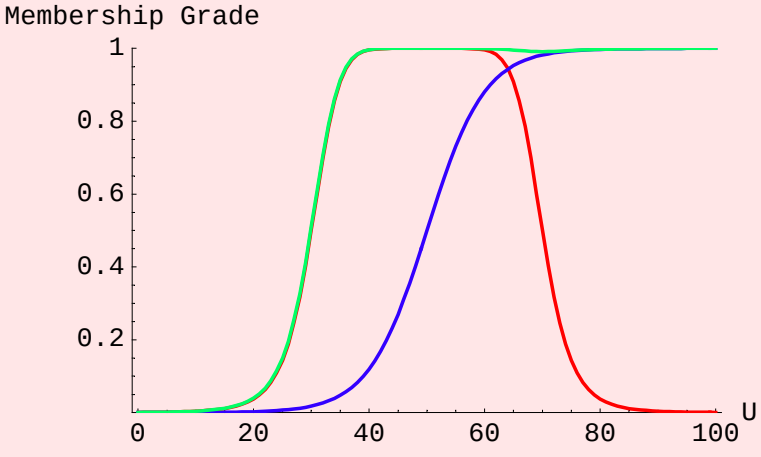


Weber-Type Union and Intersection

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

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

```
FuzzyPlot[FS1, FS2, WeUn, PlotJoined → True];
```



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. An alpha-level set of a fuzzy set/fuzzy relation is the crisp set that contains all the elements of universal space whose membership grades in the set/relation are greater than or equal to the specified value of alpha. A strong alpha-level set of the fuzzy set/fuzzy relation is the crisp set that contains all the elements of universal space whose membership grades in the 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 the set/relation.

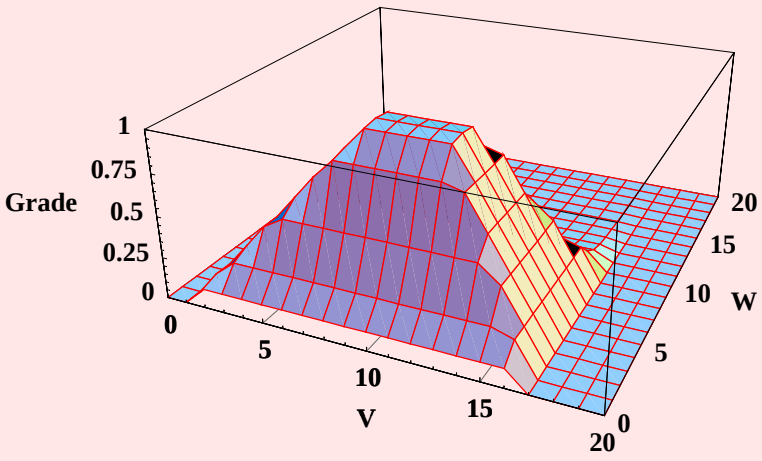
```
SetOptions[FuzzySet, UniversalSpace → {0, 20, 1}];
```

```
FS1 = FuzzyTrapezoid[1, 8, 12, 17];
```

```
FS2 = FuzzyBell[5, 3, 2];
```

```
rel1 = SetsToRelation[Min, FS1, FS2];
```

```
FuzzySurfacePlot[rel1];
```



```
AlphaLevelSet[rel1, .2]
```

```
{{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}}
```

```
StrongAlphaLevelSet[rel1, .2]
```

```
{{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}}
```

```
LevelSet[rel1]
```

```
{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.}
```

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 determine a fuzzy relation `Relat1` given the system's input `A` and output `B`.

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

```
FuzzySet[{{1, 0.2}, {2, 0.8}, {3, 1}},
 UniversalSpace → {1, 3, 1}]
```

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

```
FuzzySet[{{1, 0.5}, {2, 0.8}, {3, 0.6}},
 UniversalSpace → {1, 3, 1}]
```

```
Relat1 = FindFuzzyRelation[A, B]
```

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

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

```
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}}]
```

```
NewA = FindFuzzySet[Rel2, B]
```

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

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.

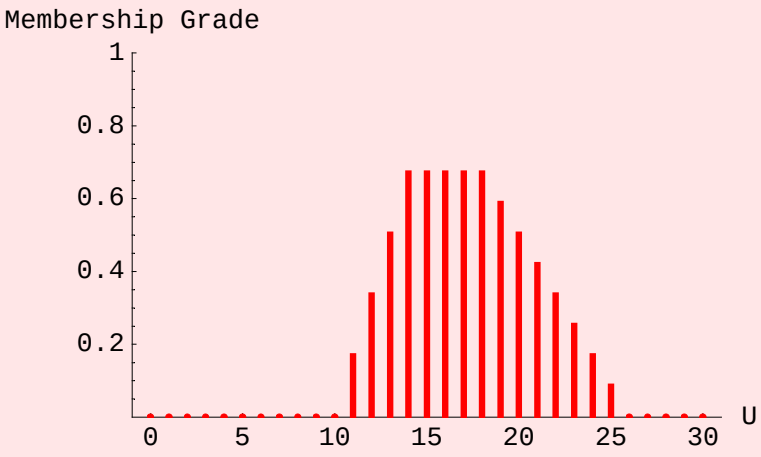
Random Fuzzy Set

`RandomFuzzySet[{a, b}]` creates a random fuzzy set with universal space from *a* to *b*. By default, this function creates 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 function `RandomFuzzySet`, try reevaluating the function a number of times with different parameters.

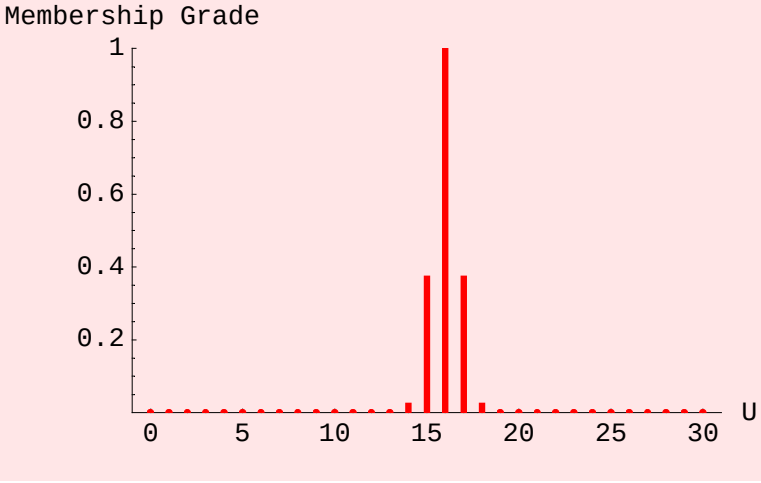
```
FS1 = RandomFuzzySet[{0, 30}];
```

```
FuzzyPlot[FS1];
```



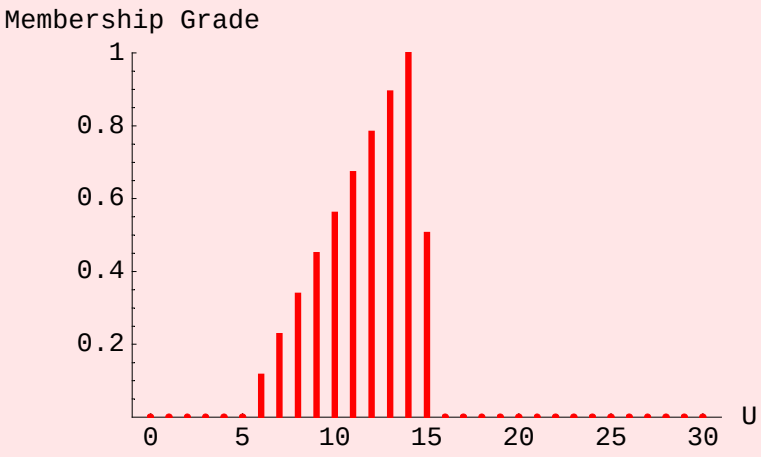

```
FS2 = RandomFuzzySet[{0, 30}, Type → Gaussian];
```

```
FuzzyPlot[FS2];
```



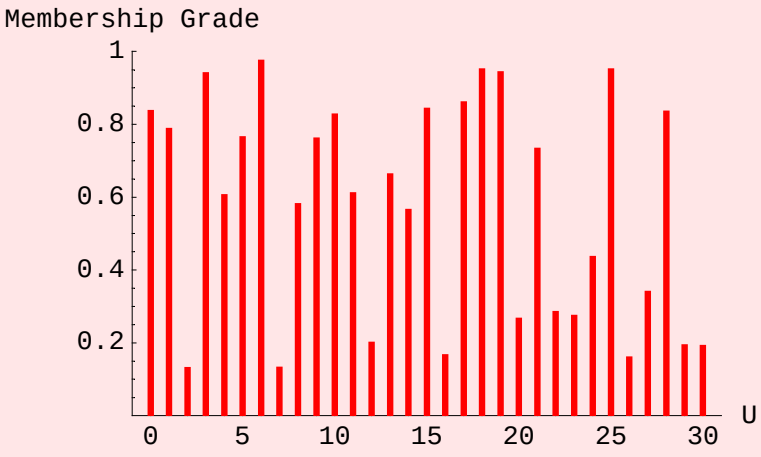

```
FS3 = RandomFuzzySet[{0, 30}, Type → Triangular, Normal → True];
```

```
FuzzyPlot[FS3];
```




```
FS3 = RandomFuzzySet[{0, 30}, Type → Complete];
```

```
FuzzyPlot[FS3];
```

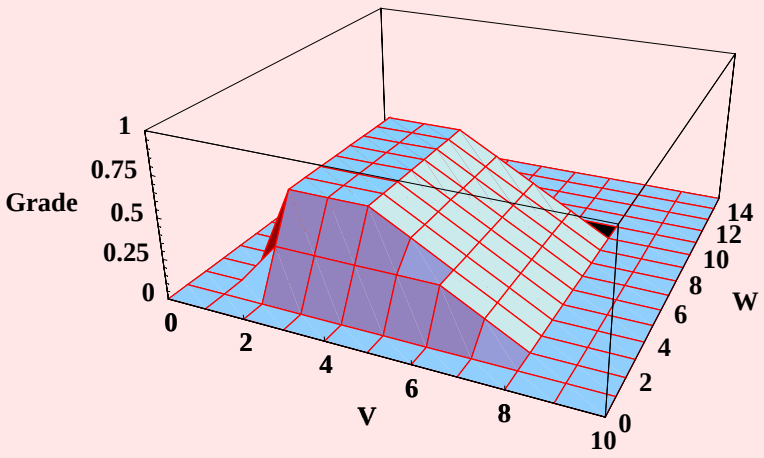


Random Fuzzy Relation

`RandomFuzzySet[{{a1 , b1}, {a2, b2}}]` creates a random fuzzy relation with a universal space of {{*a1*, *b1*}, {*a2*, *b2*}}. By default, this function creates 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`.

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

```
FuzzySurfacePlot[fr1];
```



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 the option `Type` are `Mamdani`, `Model`, and `Scaled`.

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 `FuzzyTrapezoid`, use the function `ReleaseHold`.

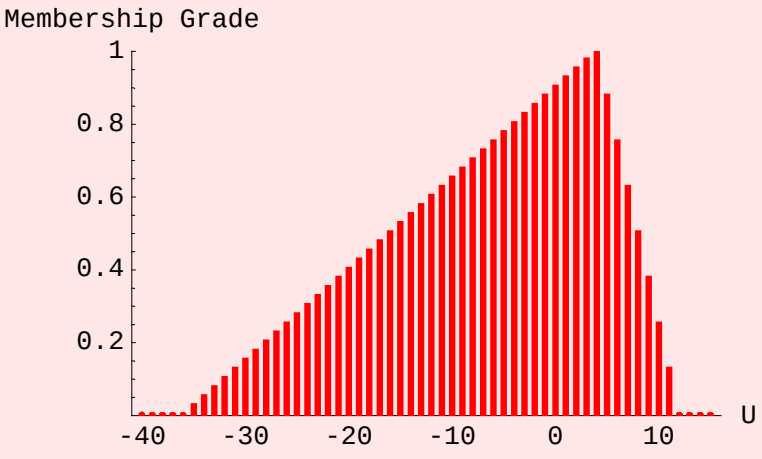
```
SetOptions[FuzzySet, UniversalSpace → {-40, 15, 1}];
```

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

```
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 command `ReleaseHold`.

```
FuzzyPlot[ReleaseHold[Multi1]];
```



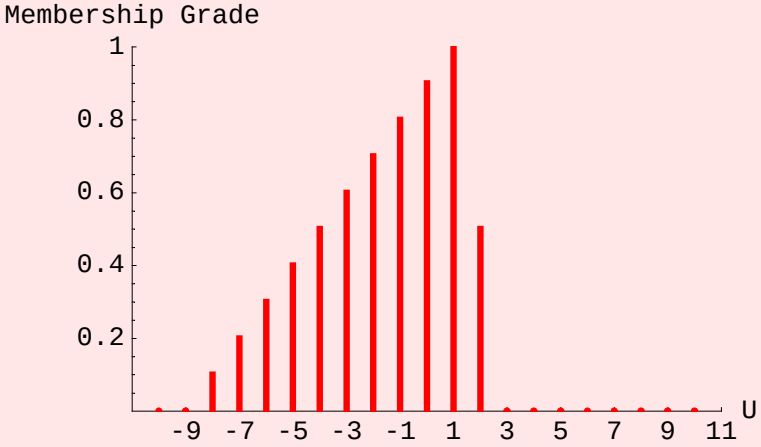
`FuzzyDivide[{a1, b1, c1, d1}, {a2, b2, c2, d2}]` returns the division of the fuzzy numbers represented by the two lists. The fuzzy division is returned as an unevaluated `FuzzyTrapezoid`. To evaluate `FuzzyTrapezoid`, use the function `ReleaseHold`.

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

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

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

```
FuzzyPlot[ReleaseHold[Div1]];
```



Fuzzy Clustering

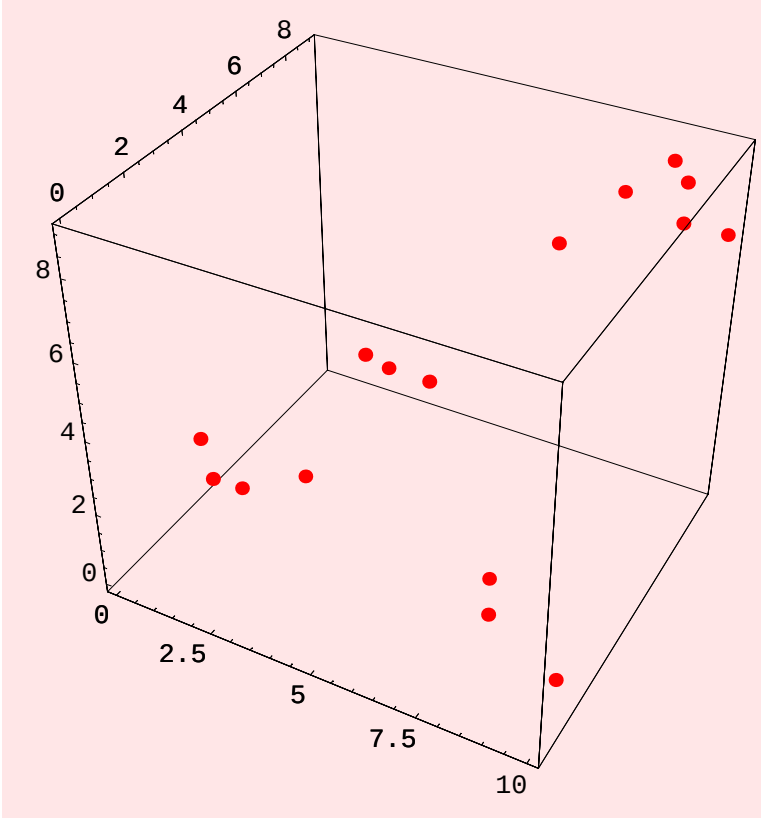
Fuzzy C-Means (FCM) Clustering

`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 that determines when the algorithm terminates (*epsilon*). This function runs recursively until the terminating criteria are 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 belongs 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 that consists of four groups of data.

```
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}};
```

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



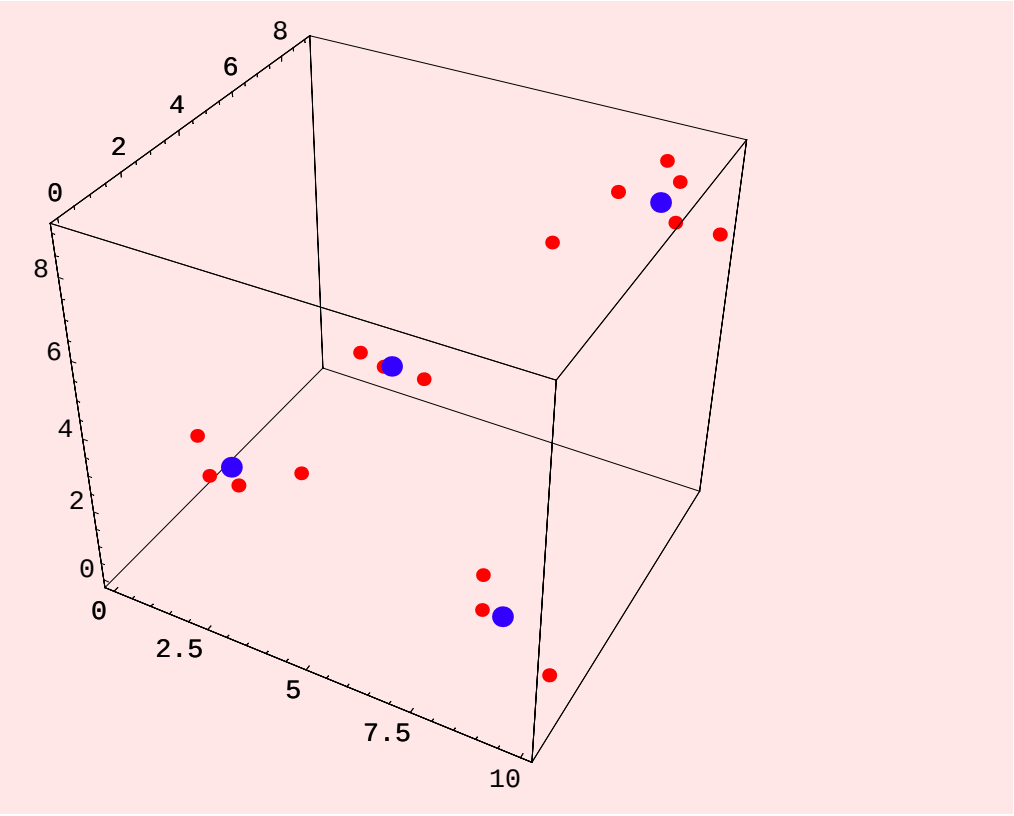
```
Res1a =
 FCMCluster[TrainData3D, InitializeU[TrainData3D, 4], 2, .01]
```

```
0.859596

0.390469

0.580971

0.33459

0.159146

0.0271684

0.00819179
```

```
{{{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 as well as the cluster center progression. Because there are problems showing the cluster centers for data of higher dimensions, the function `ShowCenters` works only for 2D clustering. If you want to see the 3D cluster centers with the original data, you could do the following.

```
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.