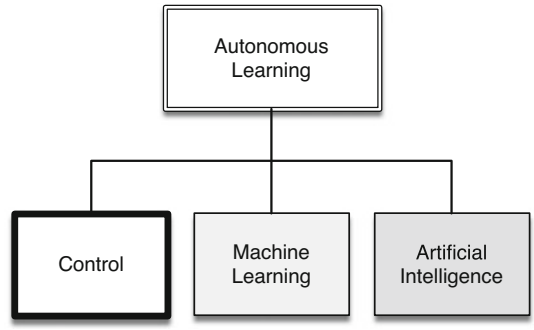
CHAPTER 6

■ ■ ■

Fuzzy Logic

Fuzzy logic [30] is an alternative approach to control system design. Fuzzy logic works within the framework of set theory and is better at dealing with ambiguities. For example, three sets might be defined for a sensor: hard failure, soft failure, and no failure. The three sets might overlap, and at any given time, the sensor may have a degree of membership in each set. In effect, you would be applying a degree of fuzziness. The degree of membership in each set can be used to determine what action to take. An algorithmic approach would have to assign a number to the state of the sensor. This could be problematic and not necessarily represent the actual state of the system.



When you go to a doctor with pain, the doctor will often try and get you to convert a subjective concept, pain, into a number from 0 to 10. As pain is personal and your impression is imprecise, you are giving a fuzzy concept or belief a hard number. As you may have experienced, this is not always productive or useful.

Surveys do the same thing. For example, you will be asked to rate the service in a restaurant from 0 to 5. You then rate a bunch of other things on the same scale. This allows the review to come up with a number for your overall impression of the restaurant. Does the resulting 4.8 mean anything? Netflix abandoned the numerical ratings of movies you have seen for thumbs up and down. It seems that they felt that a binary decision, really two sets, was a better data point than a number.

NASA and the US Department of Defense like to use technology readiness levels (TRLs) that go from 1 to 9 to determine where your work is in terms of readiness. Nine is a technology already operating in a target system. One is just an idea. All the other levels are fuzzy for anything moderately complicated. Even giving a technology a 9 is not informative. The M-16 rifle was deployed to Vietnam. It often jammed. In terms of TRL, it was 9, but a 9 doesn't say how well it is working. Again, the readiness of the rifle, when you read soldiers' and Marines' impressions, was best represented by fuzzy beliefs.

M. Paluszek, S. Thomas, *MATLAB Machine Learning Recipes*,
https://doi.org/10.1007/978-1-4842-9846-6_6

This chapter will show you how to build a simple fuzzy logic engine and implement a fuzzy logic control system for windshield wipers. Unlike the other chapters, we will be working with linguistic concepts, not hard numbers. Of course, when you set your wiper motor speed, you need to pick a number (defuzzify your output), but all the intermediate steps employ fuzzy logic. A second example shows control of an HVAC system in a home. Traditional thermostats must be manually switched from heating to cooling, while modern heat pumps can switch automatically. We will compare a traditional control option with two fuzzy examples.

6.1 Building Fuzzy Logic Systems

6.1.1 Problem

We want to have a tool to build a fuzzy logic controller.

6.1.2 Solution

Build a MATLAB function that takes parameter pairs that define everything needed for the fuzzy controller. This will be stored in a data structure.

6.1.3 How It Works

To create a fuzzy system, you must create inputs, outputs, and rules. You can also choose methods for some parts of the fuzzy inference. The fuzzy inference engine has three steps:

1. Fuzzify the inputs
2. Fire rules
3. Defuzzify the outputs

The fuzzy system data is stored in a MATLAB data structure. This structure has the following fields:

- input (:)
- output (:)
- rules (:)
- implication (@)
- aggregation (@)
- defuzzify (@)

The first three fields are arrays of struct arrays. There are separate structures for fuzzy sets and rules, described as follows. The last three fields are function handles for the implementation of these steps in the fuzzy process.

The fuzzy set structure, which is the same for inputs and outputs of the system, has the following fields:

- name
- range (2) (two-element array with minimum and maximum values)
- comp {:} (cell array of label strings)
- type {:} (cell array of membership function handles)
- params {:} (cell array of parameter vectors)

The fuzzy rule struct has the following fields:

- input (:) (vector of input component numbers)
- output (:) (vector of outputs)
- operator {:} (cell array of operator function handles)

Defuzzification requires three steps: implication, aggregation, and the defuzzification of the aggregate. These will be simply function handles. Implication applies the rule strength to the output membership functions, and aggregation combines this data from all the rules for each output across its range. The final defuzzification step produces a crisp value for each output.

This is a lot of data to organize. We do it with the function `BuildFuzzySystem`. The following code snippet shows how it assigns data to the data structure using parameter pairs. The `'id'` field increments the index used for either the input, output, or rule.

BuildFuzzySystem.m

```
d = struct;
j = 1;

for k = 1:2:length(varargin)
  switch (lower(varargin{k}))
    case 'id'
      j = varargin{k+1};
    case 'input comp'
      d.input(j).comp = varargin{k+1};
    case 'input type'
      d.input(j).type = varargin{k+1};
    case 'input name'
       d.input(j).name = varargin{k+1};
    case 'input params'
      d.input(j).params = varargin{k+1};
    case 'input range'
      d.input(j).range = varargin{k+1};
    case 'output comp'
      d.output(j).comp = varargin{k+1};
```

This code continues with other cases. Since the fuzzy variables are by nature linguistic, a section of code will map any string names of the fuzzy variables in the rule definitions into their numerical indices using `contains`, which will save computation later.

BuildFuzzySystem.m

```
% match rules to sets if cell array
for k = 1:length(d.rules)
  inputs = d.rules(k).input;
  if iscell(inputs)
    nIn = length(inputs);
    input = zeros(1,nIn);
    for j = 1:nIn
      comp = d.input(j).comp;
      val = find(contains(comp,inputs(j)));
      if ~isempty(val)
        input(j) = val;
      end
    end
    d.rules(k).input = input;
  end
  outputs = d.rules(k).output;
  if iscell(outputs)
    nOut = length(outputs);
    output = zeros(1,nOut);
    for j = 1:nOut
      comp = d.output(j).comp;
      val = find(contains(comp,outputs(j)));
      if ~isempty(val)
        output(j) = val;
      end
    end
    d.rules(k).output = output;
  end
end % array of rules
```

The following is a snippet showing how to use `BuildFuzzySystem`, showing just the creation of the first input for the SmartWipers example. This example will be described fully in a later recipe.

```
>> SmartWipers = BuildFuzzySystem(...
              'id',1,...
              'input comp',{'Dry'  'Drizzle'  'Wet'},...
              'input type', {@TrapezoidMF  @TriangleMF  @TrapezoidMF}
                  ,...
              'input params',{[0 0 10 50]  [40 50]  [50 90 101 101]},...
              'input range',[0 100],...
              'input name','Wetness')

SmartWipers =
   struct with fields:
```

```
             input: [1x1 struct]

>> SmartWipers.input(1)

ans =
  struct with fields:

      comp: {'Dry'  'Drizzle'  'Wet'}
      type: {@TrapezoidMF  @TriangleMF  @TrapezoidMF}
    params: {[0 0 10 50]  [40 50]  [50 90 101 101]}
     range: [0 100]
      name: 'Wetness'
```

Fuzzy sets in this context consist of a set of linguistic categories or components defining a variable. For instance, if the variable is "age," the components might be "young," "middle aged," and "old." Each fuzzy set has a range over which it is valid, for instance, a good range for "age" might be 0 to 100. Each component has a membership function that describes the degree to which a value in the set's range belongs to each component. For instance, a person who is 50 would rarely be described as "young," but might be described as "middle aged" or "old," depending on the person asked.

To build a fuzzy set, you must divide the variable into components. The simplest are triangles and trapezoids. The following membership functions are provided with this recipe: triangular, trapezoidal, Gaussian, general bell, and sigmoidal. Membership functions are limited in value to between zero and one. The membership functions are shown in Figure 6.1 and described further as follows:

Triangle: The triangular membership function requires two parameters: the center of the triangle and the half-width of the desired triangle base. Triangular membership functions are limited to symmetric triangles.

Trapezoid: The trapezoid membership function requires four parameters: the leftmost point, the start of the plateau, the end of the plateau, and the rightmost points.

Gaussian: A Gaussian membership function is a continuous function with two parameters: the center of the bell and the width (standard deviation) of the bell. Gaussian membership functions are symmetric.

Bell: A general bell function is also continuous and symmetric, but it has three parameters to allow for a flattened top, making it similar to a smoothed trapezoid. It requires three parameters: the center of the bell, the width of the bell at points y = 0.5, and the slope of the function at points y = 0.5.

Sigmoid: Just as a bell function is similar to a smoothed trapezoid, a sigmoidal membership function is similar to a smoothed step function. It takes two parameters: the point at which y = 0.5 and the slope of the function. As the slope approaches infinity, the sigmoidal function approaches the step function.

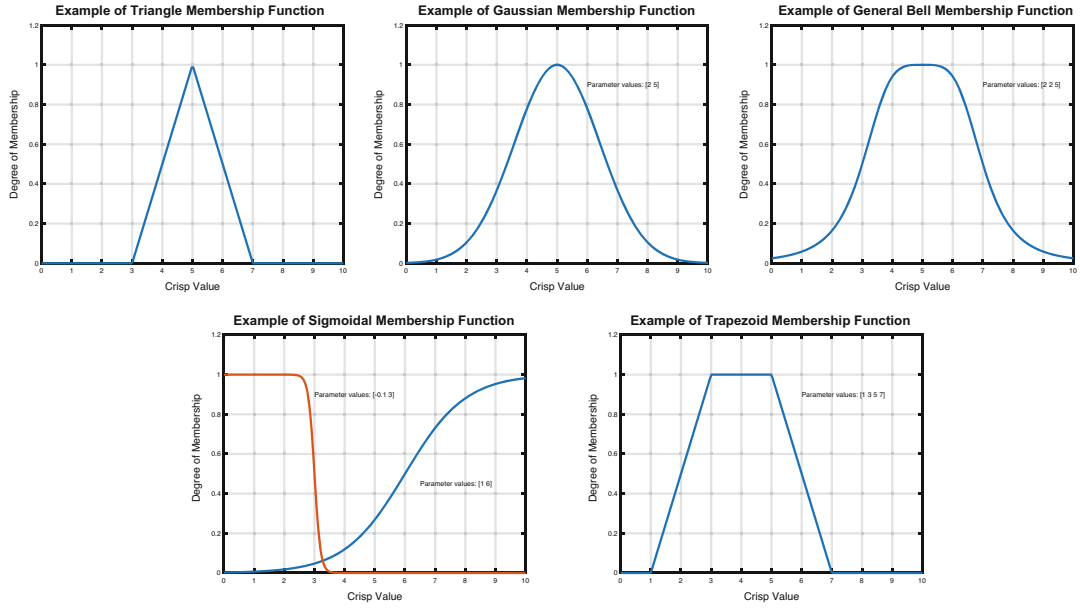


Figure 6.1: *Membership functions*

Fuzzy rules are if-then statements. For example, an air conditioner rule might say IF the room temperature IS high, THEN the blower level IS high. In this case, “room temperature” is the input fuzzy set, “high” is its component for this rule, “blower level” is the output fuzzy set, and “high” is its chosen component. Rules may combine inputs with either an AND or an OR operator. The AND operator is the minimum of the membership values, while the OR operator returns the maximum of the values. In our structure, the rules use numeric indices for the components of each input and output for computational efficiency. An example is

```
>> d.rules(1)

ans =

  struct with fields:

       input: [1 1]
      output: [1 3]
    operator: @FuzzyAND
```

This structure for a fuzzy system is supported by a set of helper functions for the fuzzy operations. This includes membership functions, with an MF suffix; operators, namely AND and OR; implication functions with an IMP suffix; and defuzzification. The following list gives

all the support functions provided with this chapter. This is not an exhaustive list of algorithms, and other commercial or open source tools may provide additional methods:

- Membership functions
  - `TriangleMF.m`, `GaussianMF.m`, `GeneralBellMF.m`, `SigmoidalMF.m`, `TrapezoidMF.m`
- Fuzzy operators, for rules
  - `FuzzyAND.m`, `FuzzyOR.m`
- Implication
  - `ScaleIMP.m`, `ClipIMP.m`
- Aggregation
  - `max`
- Defuzzification
  - `CentroidDF.m`

6.2 Implement Fuzzy Logic

6.2.1 Problem

We want to implement fuzzy logic.

6.2.2 Solution

Build a fuzzy inference engine. This will be a function that calls the steps in fuzzy inference given a fuzzy system as defined in the previous recipe, using function handles to specify options within the algorithm.

6.2.3 How It Works

Let's repeat the three steps in fuzzy inference, adding the substeps within Defuzzify:

1. Fuzzify
2. Fire
3. Defuzzify
   (a) Implication
   (b) Aggregation
   (c) Defuzzify the aggregate

The control flow is in the main function, called `FuzzyInference`. It just calls subfunctions `Fuzzify`, `Fire`, and `Defuzzify` in order. It calls `warndlg` if the inputs are not sensible.

FuzzyInference.m

```
function [y,data] = FuzzyInference( x, system, verbosity )
if length(x) == length( system.input )
  fuzzyX     = Fuzzify( x, system.input );
  strength   = Fire( fuzzyX, system.rules );
  y          = Defuzzify( strength, system, x );
else
  warndlg({'The length of x must be equal to the',...
    'number of input sets in the system.'})
end
```

Since this function is written for educational purposes, we added an informational output struct. This includes the extra step of fuzzifying the outputs after the crisp value is computed from the rules. Therefore, we can examine both `fuzzyX` and `fuzzyY` as well as the strength of the rules firing.

FuzzyInference.m

```
if (nargout>1)
  data.x = x;
  data.fuzzyX = fuzzyX;
  data.strength = strength;
  data.fuzzyY = Fuzzify( y, system.output );
  data.y = y;
end
```

You will notice, in the body of functions, the use of `feval` to evaluate function handles as the input. Earlier versions of this tool used strings for the function names with `eval`, but using handles is now a much faster option than evaluating strings. You pass in the inputs after the handle which can be any expression or variable. For example, for the function

```
function y = MyFun(x)
y = x;
```

You can evaluate it with a number or a variable or an expression, such as

```
>> feval(@MyFun,2)

ans =
     2

>> feval(@MyFun,sin(2))

ans =
       0.9093
```

■ **TIP** Use `feval` instead of `eval` whenever possible.

The `Fuzzify` subfunction code is shown as follows. It evaluates the degree of membership of the inputs in each membership set.

FuzzyInference.m

```
56    function fuzzyX = Fuzzify( x, sets )
57      %% Fuzzify the inputs with the type function
58      % fuzzyX = Fuzzify( x, sets )
65      n = length(sets);
66      fuzzyX = cell(1,n);
67      for i = 1:n
68        nC = length(sets(i).comp);
69        range = sets(i).range(:);
70        if (range(1) <= x(i)) && (x(i) <= range(2))
71          for j = 1:nC
72            fuzzyX{i}(j) = feval(sets(i).type{j},x(i),sets(i).params{j});
73          end
74        else
75          fuzzyX{i}(1:nC) = zeros(1,nC);
76        end
77      end
```

The fuzzy rule logic is shown in the following code. The code applies "Fuzzy AND" or "Fuzzy OR." "Fuzzy AND" is the minimum of a set of membership values. "Fuzzy OR" is the maximum of a set of membership values. Suppose we have a vector `[1 0 1 0]`. The maximum value is 1 and the minimum is 0.

```
>> 1 && 0 &&  1 && 0

ans =

  logical
   0

>> 1 || 0 ||  1 || 0

ans =

  logical
   1
```

This corresponds to the fuzzy logic AND and OR.

The next code snippet shows the `Fire` subfunction in `FuzzyInference`. "Firing" a rule is the process of applying the rule operators to the fuzzified inputs. This determines the numerical strength of each rule using the specific membership values of the inputs.

FuzzyInference.m

```
81    function strength = Fire( FuzzyX, rules )
82      %% Fire a rule using the specified rules.operator function
83      % strength = Fire( FuzzyX, rules )
90      p = length( rules );
91      n = length( FuzzyX );
92
93      strength = zeros(1,p);
94
95      for i = 1:p
96        method = rules(i).operator;
97        dom = zeros(1,n);
98        for j = 1:n
99          comp = rules(i).input(j);
100         if comp ~= 0
101           dom(j) = FuzzyX{j}(comp);
102         else
103           dom(j) = inf;
104         end
105       end
106       strength(i) = feval(method,dom(dom<=1));
107     end
```

Finally, we defuzzify the results. This function first uses the implication function to determine membership. It aggregates the output using the aggregate function which, in this case, is `max`. The final step to computing the crisp values is computing the centroid of the aggregate. For explanatory purposes, this function is annotated with a plot capability of the defuzzification if "verbose" output is requested.

FuzzyInference.m

```
111   function [result,aggregate] = Defuzzify( strength, system, xIn )
112     %% Defuzzify the rule output
113     % result = Defuzzify( strength, system )
120     rules   = system.rules;
121     output  = system.output;
122
123     m       = length( output );
124     p       = length( rules );
125     impfun  = system.implicate;
126     aggfun  = system.aggregate;
127     defuzz  = system.defuzzify;
128
129     nPts    = 200;
130     result  = zeros(1,m);
131
132     if verbose
133       figure('name','Fuzzy Inference')
134       subplot(m,1,1); hold on;
135       xstr = num2str(xIn);
```

```
136         title(sprintf('Fuzzy output for [%s]',xstr))
137       end
138
139       for i = 1:m
140         if verbose
141           subplot(m,1,i); hold on; grid on;
142         end
143         range = output(i).range(:);
144         xO = linspace( range(1),range(2),nPts );
145         mem = zeros(p,nPts);
146         % precompute membership for the output set
147         ls = [];
148         label = {};
149         nC = length(output(i).type);
150         ymf = zeros(nC,nPts);
151         for k = 1:nC
152           mfun   = output(i).type{k};
153           params = output(i).params{k};
154           ymf(k,:) = feval(mfun,xO,params);
155           if verbose
156             plot(xO,ymf(k,:),'-.','linewidth',1);
157           end
158         end
159         % compute the membership for each fired rule
160         for j = 1:p
161           comp = rules(j).output(i);
162           if( comp ~= 0 ) && strength(j)>0
163             mem(j,:)  = feval(impfun, ymf(comp,:),strength(j));
164             if verbose
165               ls(end+1) = plot(xO,mem(j,:),'linewidth',1);
166               label{end+1} = [num2str(j) ' (' num2str(strength(j),3) ')'
                      ];
167             end
168           else
169             mem(j,:)  = zeros(size(xO));
170           end
171         end % rules
172         aggregate = feval(aggfun,mem);
173         result(i) = feval(defuzz,aggregate,xO);
174         if verbose
175           plot(xO,aggregate,'k--','linewidth',2);
176           yy = axis;
177           plot(result(i)*[1 1],yy(3:4),'r','linewidth',3)
178           text(result(i),yy(3) + 0.75*(yy(4)-yy(3)),sprintf('  %g',result
                  (i)))
179           xlabel(output(i).name)
180           if i == 1
181             ll = legend(ls,label,'location','best');
182             ll.Title.String = 'Rules';
183           end
184         end
```

```
185         end % outputs
```

The plots in Figure 6.2 show the total defuzzification process. First, the membership sets of each variable are drawn in dash-dot lines in the background of the plot. Each rule designates a fuzzy output. The implication function combines the strength of the rule with the membership function of that fuzzy output. Clip implication takes the minimum at each point, so the strength limits the membership value. Scale implication uses the product of the strength and the membership. Rules with nonzero strength are plotted as shown with the solid lines, and those rules with nonzero firing strength are shown in the legend. Aggregation then combines the output from each rule into a single vector of membership for the output across its range. The final step is defuzzification of this array, in our case with centroiding via `CentroidDF`. The final crisp value is designated by the thick red line and labeled with the crisp value.

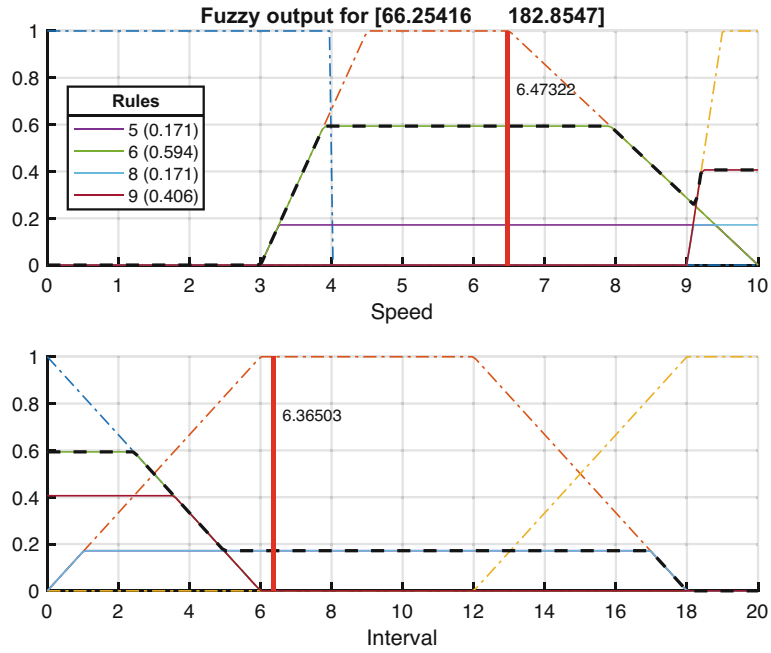


Figure 6.2: *Fuzzy rule plot for smart wipers*

6.3 Window Wiper Fuzzy Controller

6.3.1 Problem

We want a control system to select window wiper speed and interval based on rainfall. This is an implementation of the SmartWipers automatic windshield wiper control system from Cheok [8]. The inputs to the control system are the rain wetness and intensity, and the outputs are the wiper speed and interval.

6.3.2 Solution

Build a fuzzy logic control system using the tools we've developed. First, we will write a function to create the fuzzy system data structure, then a demo script to use it.

6.3.3 How It Works

To call a fuzzy system, use the function `y = FuzzyInference( x, system ).`

The script `SmartWipersDemo` implements the rainfall demo. The demo loads the fuzzy system from the function `SmartWipersSystem`, which uses `BuildFuzzySystem` from Recipe 6.1. The following code performs the fuzzy inference on a full range of the two inputs.

SmartWipersDemo.m

```
% Generate regularly space arrays in the 2 inputs
n = 30; % Number of samples
x = linspace(SmartWipers.input(1).range(1),SmartWipers.input(1).range
    (2),n);
y = linspace(SmartWipers.input(2).range(1),SmartWipers.input(2).range
    (2),n);

% Perform fuzzy inference over the input range
z1 = zeros(n,n);
z2 = zeros(n,n);
for k = 1:n
  for j = 1:n
    temp = FuzzyInference([x(k),y(j)], SmartWipers);
    z1(k,j) = temp(1);
    z2(k,j) = temp(2);
  end
end
```

First, the demo will plot the input and output fuzzy variables using `FuzzyPlot`. Fuzzy inference is performed on each set of crisp inputs plotted. Figure 6.3 shows the inputs to the fuzzy logic system. Figure 6.4 shows the outputs. The rule base is displayed using `PrintFuzzyRules` and plotted using `surf`.

The inputs that are tested in the fuzzy logic system demo are given in Figure 6.5. This is just the full range of each input.

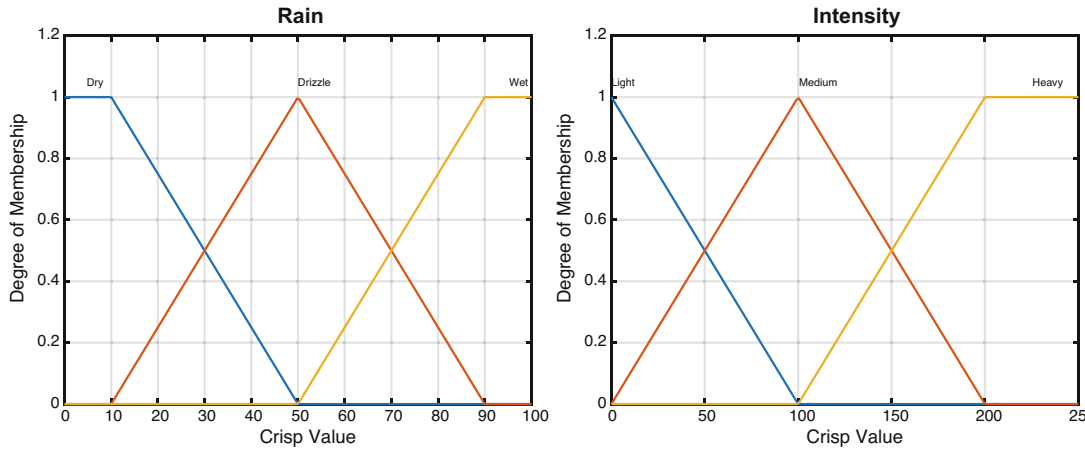


Figure 6.3: *Rain wetness and intensity are the inputs for the smart wiper control system*

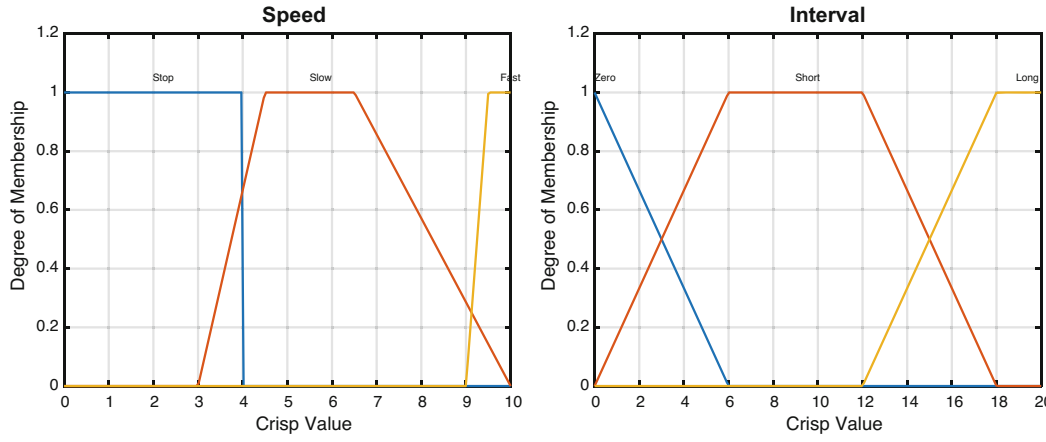


Figure 6.4: *Wiper speed and interval are the outputs for the smart wiper control system*

The printed rules are shown as follows:

```
>> SmartWipersDemo

1. if Wetness is Dry FuzzyAND Intensity is Light then   Speed is Stop
    Interval is Long
2. if Wetness is Dry FuzzyAND Intensity is Medium then  Speed is Slow
    Interval is Long
3. if Wetness is Dry FuzzyAND Intensity is Heavy then   Speed is Slow
    Interval is Short
4. if Wetness is Drizzle FuzzyAND Intensity is Light then       Speed is
    Slow   Interval is Long
5. if Wetness is Drizzle FuzzyAND Intensity is Medium then      Speed is
    Slow   Interval is Short
```

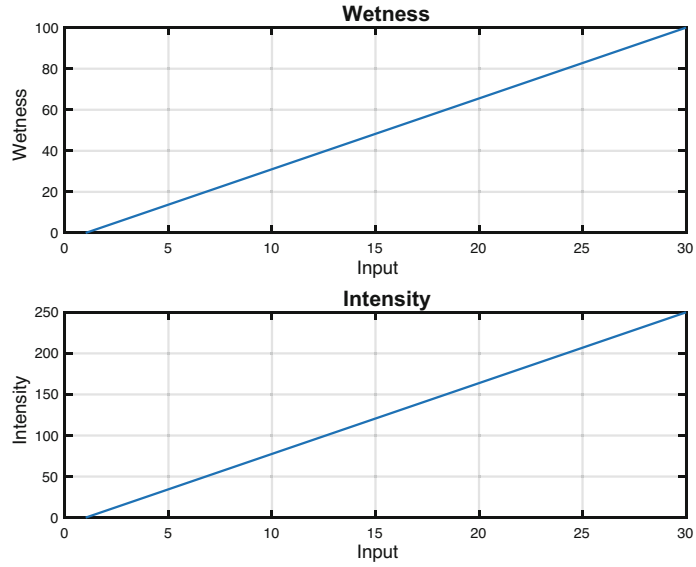


Figure 6.5: *Rain wetness and intensity input numbers*

```
6. if Wetness is Drizzle FuzzyAND Intensity is Heavy then       Speed is
    Slow   Interval is Zero
7. if Wetness is Wet FuzzyAND Intensity is Light then   Speed is Slow
    Interval is Short
8. if Wetness is Wet FuzzyAND Intensity is Medium then  Speed is Fast
    Interval is Short
9. if Wetness is Wet FuzzyAND Intensity is Heavy then   Speed is Fast
    Interval is Zero
```

Figure 6.6 gives surface plots to show how the outputs relate to the inputs via the rules. The surface plots are generated by the following code. We add a `colorbar` to make the plot more readable. The color is related to z value. We use `view` in the second plot to make it easier to read the figure. You can use `rotate3d on` to allow you to rotate the figure with the mouse.

SmartWipersDemo.m

```
41 % Plot the outputs as surfaces
42 NewFigure('Wiper Speed from Fuzzy Logic');
43 surf(x,y,z1)
44 xlabel('Raindrop Wetness')
45 ylabel('Droplet Frequency')
46 zlabel('Wiper Speed')
47 colorbar
48
49 NewFigure('Wiper Interval from Fuzzy Logic');
50 surf(x,y,z2)
51 xlabel('Raindrop Wetness')
52 ylabel('Droplet Frequency')
```

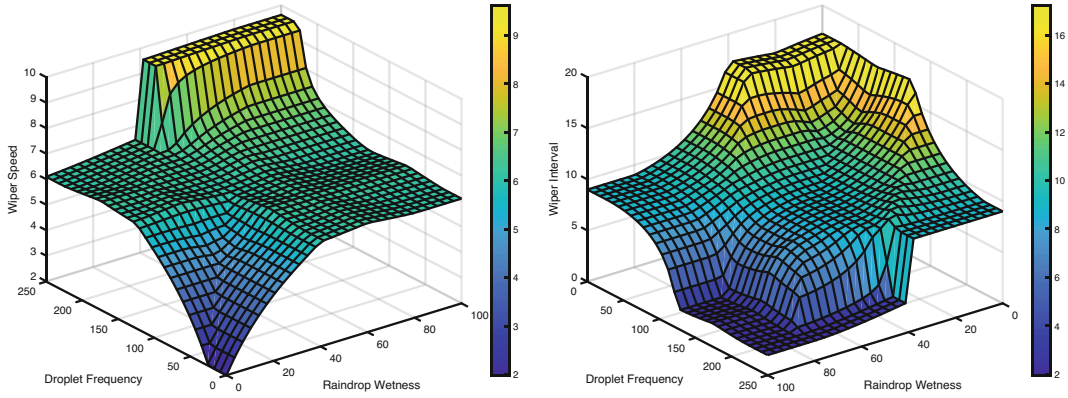


Figure 6.6: *Wiper speed and interval vs. droplet frequency and wetness*

```
53  zlabel('Wiper Interval')
54  view([142.5 30])
55  colorbar
```

■ **TIP** Use `rotate3d on` to rotate a figure with the mouse.

The `SmartWipersTest` script tests the fuzzy inference using random inputs generated over the input range. This is done using the `FuzzyRand` function as follows.

FuzzyRand.m

```
1  %% FUZZYRAND Compute random inputs within range of the fuzzy input sets
2  %% Inputs
3  % system (.) Fuzzy system from BuildFuzzySystem
4  %% Outputs
5  % y      (n) Random crisp values of the inputs
6
7  function y = FuzzyRand(system)
8
9  if nargin==0
10   system = SmartWipersSystem;
11   y = FuzzyRand(system)
12   return;
13 end
14
15 nIn = length(system.input);
16 y   = ones(1,nIn);
17
18 for k = 1:nIn
19   range = system.input(k).range;
20   y(k)  = range(1) + (range(2)-range(1))*rand(1);
21 end
```

The demo then prints out the crisp and fuzzy values of the inputs and outputs including the strength of the rules. This can provide useful insight when you are developing a new fuzzy system. In the random inputs captured as follows, the rain wetness is both drizzle and wet, the intensity is evenly split between medium and heavy, and the output is a slow speed with a short interval:

```
>> SmartWipersTest
------
Inputs
------
Wetness
----
Crisp: 64.4673
Range: 0 to 100
         Set          Value
    ____________    _______

    {'Dry'    }           0
    {'Drizzle'}     0.63832
    {'Wet'    }     0.36168

Intensity
----
Crisp: 152.816
Range: 0 to 250
        Set          Value
    ___________    _______

    {'Light' }           0
    {'Medium'}     0.47184
    {'Heavy' }     0.52816

Strength of rule firings:
----
     Input      Output      Fire Strength
    ________    ________    _____________

    {[1 1]}     {[1 3]}              0
    {[1 2]}     {[2 3]}              0
    {[1 3]}     {[2 2]}              0
    {[2 1]}     {[2 3]}              0
    {[2 2]}     {[2 2]}        0.47184
    {[2 3]}     {[2 1]}        0.52816
    {[3 1]}     {[2 2]}              0
    {[3 2]}     {[3 2]}        0.36168
    {[3 3]}     {[3 1]}        0.36168

-------
Outputs
-------
Speed
```

```
----
Crisp: 6.48238
Range: 0 to 10
      Set        Value
    ________     _____

    {'Stop'}       0
    {'Slow'}       1
    {'Fast'}       0

Interval
----
Crisp: 8.14129
Range: 0 to 20
       Set        Value
    _________     _____

    {'Zero' }       0
    {'Short'}       1
    {'Long' }       0
```

6.4 Simple Discrete HVAC Fuzzy Controller

6.4.1 Problem

We want a control system to automatically switch between air conditioning and heating.

6.4.2 Solution

Build a fuzzy logic control system that can turn on the heating system and air conditioning based on the air temperature.

6.4.3 How It Works

Most older heating, ventilation, and air conditioning systems require the user to pick "AC" and "heat" modes. This doesn't work very well when the temperature is varying a lot from day to day such as during the fall or spring of a region, like New England in the United States, where the temperature varies significantly over the year.

The first step is fuzzifying the input. In the simplest implementation of the control system, there are two input variables: the measured internal temperature of the house and the target or setpoint temperature. The fuzzy categories are shown in Figure 6.7. These are overlapping trapezoids with the temperature in Celsius.

A simple fuzzy control matrix using these variables is shown in Table 6.1. This is the set of rules for the fuzzy controller in `HVACSimplestFuzzyController`. The rules are combined based on the degree of membership of the internal and target temperature in the different categories.

The dynamical model we will use to simulate the house temperature as a result of the control system is illustrated in Figure 6.8.

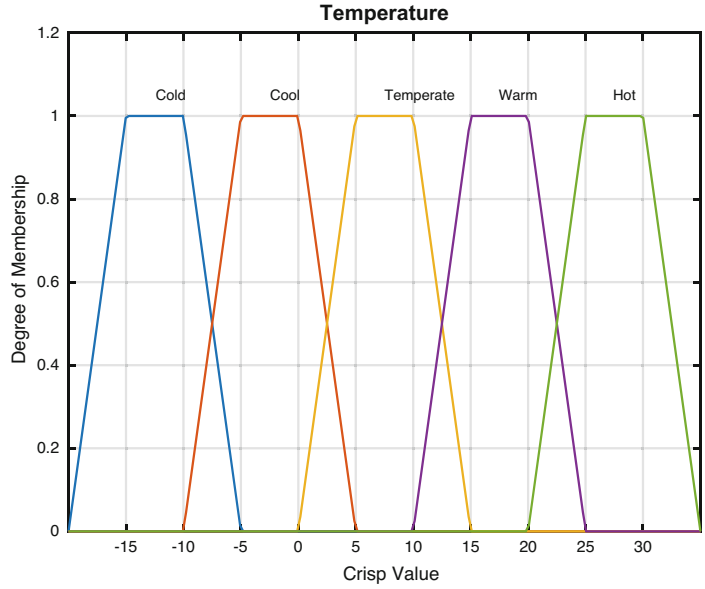


Figure 6.7: *Temperature categories (C) for HVAC*

Table 6.1: *The set of rules for the fuzzy HVAC system. The current value is in the top row; the target is in the first column*

	Cold	**Cool**	**Temperate**	**Warm**	**Hot**
Cold	No change	AC	AC	AC	AC
Cool	Heat	No change	AC	AC	AC
Temperate	Heat	Heat	No change	AC	AC
Warm	Heat	Heat	Heat	No change	AC
Hot	Heat	Heat	Heat	Heat	No change

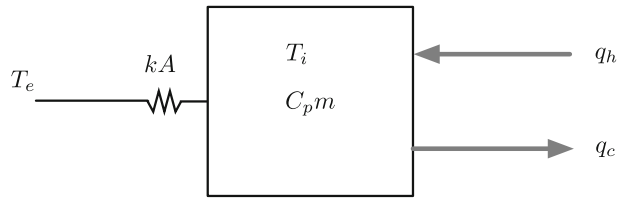


Figure 6.8: *House model*

The dynamical equations are

$$mC_p \frac{dT_i}{dt} = kA(T_e - T_i) + q_h - q_c \tag{6.1}$$

m is the thermal mass of the air in the house. A is the surface area of the house. C_p is the specific heat of air. k is the average thermal conductivity of the walls. q_h is the heater flux, and q_c is the air conditioning flux. Both are positive. The dynamic model is shown in the following code.

HVACSim.m

```
function [dT,qL] = RHS(~,tI,d)
%% Simulation right hand side (dynamics)
% Model the changing internal temperature of the house given the HVAC
% output and the external temperature.
%
% [dT,qL] = RHS(~,tI,d)
%
% dT: change in internal temperature
% qL: heat load on the house from outside

qL = d.k*d.A*(d.tE-tI);
dT = (qL + d.qH - d.qC)/(d.cP*d.m);
```

The simulation has two sets of initial conditions at the top: one in which the AC mode will be triggered, that is, a warm day, and one in which the heat will be triggered, a cold day.

HVACSim.m

```
% A/C example
%%{
tSet = 297; % Set point temperature (deg-K)
tI   = tSet+3; % Initial internal temperature (deg-K)
delT = 10; % Celsius
tE   = [ones(1,iS)*(tI + delT) ones(1,n-iS)*(tI - delT) ];
%}

% Heat example
%%{
tSet = 294; % Set point temperature (deg-K)
tI   = tSet-10; % Initial internal temperature (deg-K)
tE   = [ones(1,iS)*(tI - 20) ones(1,n-iS)*(tI - 10) ];
%}
```

A standard bang-bang controller has a deadband and hysteresis. The following code shows the controller. The controller makes its decision to switch from heating to cooling based on the previous heating/cooling command and the demand. Note that it continues heating/cooling through 90% of the deadband. This prevents limit cycling.

HVACSim.m

```
function q = Controller(t,tSet,tDB,q,qMax)
%% Non-fuzzy Controller with hysteresis
% Typical crisp controller with a deadband and hysteresis
```

```
%
% q = Controller(t,tSet,tDB,q,qMax)

if( q < 0 )
  if( t < tSet - 0.9*tDB)
    q = 0;
  end
elseif( q > 0 )
  if( t > tSet + 0.9*tDB)
    q = 0;
  end
else
  if( abs(t - tSet) > tDB )
    if( t > tSet)
      q = -qMax;
    elseif ( t < tSet )
      q = qMax;
    end
  end
end
```

The performance is shown in Figure 6.9. In this case, the external temperature drops 15 degrees Celsius in the middle of the simulation, and the heating system switches from heating to cooling. Hysteresis keeps the HVAC from shifting between heat and cool when the temperature crosses the setpoint.

The fuzzy controller has two modes, initialize and update. The initialize mode creates the fuzzy controller data structure. The following code shows the initialization through the first two rules.

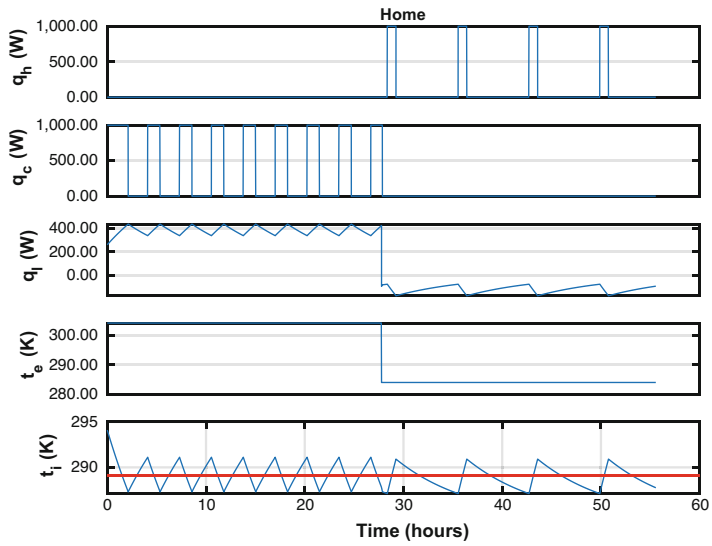


Figure 6.9: *Non-fuzzy hysteresis controller performance*

HVACSimplestFuzzyController.m

```
9  function [q,cat] = HVACSimplestFuzzyController(mode,tI,tSet,d)
21   case 'initialize'
22     if nargin<2
23       qMax = 1000;
24     else
25       qMax = tI;
26     end
27
28     % External and set point temps
29     bT = [0 4 6 10]; % 4 vertices of each input trapezoid
30     oT = [10 15 20 25 30];
31     iP = cell(1,5);
32     for k  = 1:5
33       iP{k} =  bT + oT(k);
34     end
35
36     % Define an arbitrary output range, 0 to 6 for the mode
37     oP = {[0 0 1.5 2.5] [1.5 2 4 4.5] [3.5 4.5 6 6]};
38
39     d = BuildFuzzySystem(...
40       'id',1,...
41       'input comp',{'Cold'  'Cool' 'Temperate' 'Warm'  'Hot'} ,...
42       'input type', {@TrapezoidMF  @TrapezoidMF @TrapezoidMF
            @TrapezoidMF @TrapezoidMF} ,...
43       'input params',iP,...
44       'input range',[bT(1) + oT(1) + eps bT(4)+ oT(5) - eps],...
45       'input name','Temperature',...
46       'id',2,...
47       'input comp',{'Cold'  'Cool' 'Temperate' 'Warm'  'Hot'} ,...
48       'input type', {@TrapezoidMF  @TrapezoidMF @TrapezoidMF
            @TrapezoidMF @TrapezoidMF} ,...
49       'input params',iP,...
50       'input range',[bT(1) + oT(1) bT(4)+ oT(5)],...
51       'input name','Target',...
52       'id',1,...
53       'output comp',{'AC'  'None'  'Heat'},...
54       'output type',{@TrapezoidMF  @TrapezoidMF  @TrapezoidMF},...
55       'output params',oP,...
56       'output name','Setting',...
57       'output range',[0 6],...
58       'implicate',@ClipIMP,...
59       'aggregate',@max,...
60       'defuzzify',@CentroidDF,...
61       'id',1,...
62       'rule input',[1 1],...
63       'rule output',2,...
64       'rule operator',@FuzzyAND,...
65       'id',2,...
66       'rule input',[2 2],...
67       'rule output',2,...
```

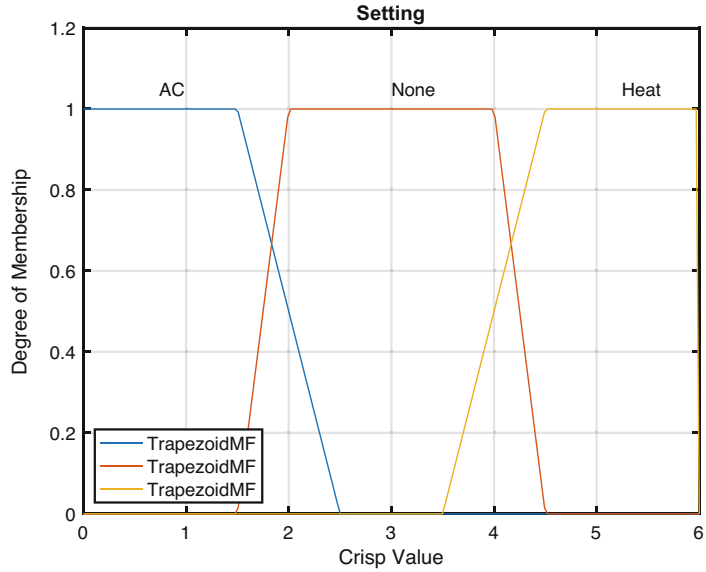


Figure 6.10: *Fuzzy controller output set (arbitrary mode setting)*

```
68          'rule operator',@FuzzyAND,...
```

The logic uses fuzzy AND only.

The following script plots the inputs to the simple fuzzy HVAC controller, which are the current temperature and the desired temperature, and the outputs. The output categories are shown in Figure 6.10.

HVACFuzzyPlot.m

```
%% Plot the HVAC fuzzy controller

h = waitbar(0,'HVAC Demo: plotting the rule base');

dFuzzy = HVACSimplestFuzzyController('initialize');
n = 30; % Number of samples

x = linspace(dFuzzy.input(1).range(1),dFuzzy.input(1).range(2),n);
y = linspace(dFuzzy.input(2).range(1),dFuzzy.input(2).range(2),n+2);

z = zeros(n,n+2);
for k = 1:n
  for j = 1:n+2
    z(k,j) = FuzzyInference([x(k),y(j)], dFuzzy);
  end
  waitbar(k/n)
end
close(h);
```

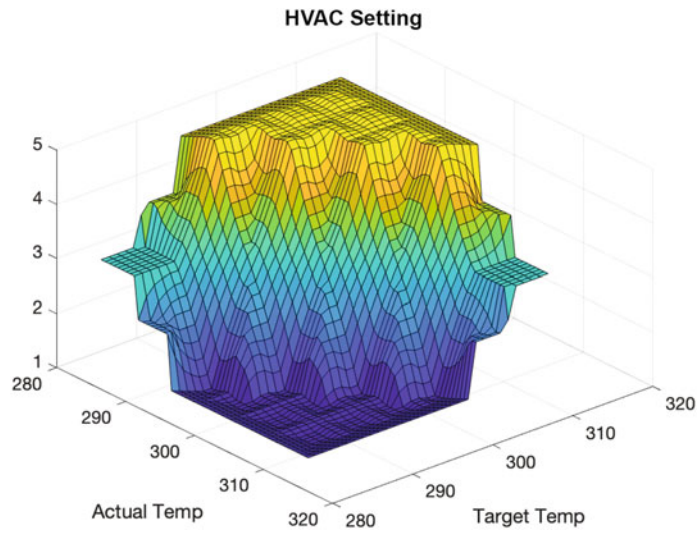


Figure 6.11: *Fuzzy controller inputs and outputs*

```
19
20 NewFigure('State from Fuzzy Logic');
21 surf(x,y,z');
22 xlabel(dFuzzy.input(1).name)
23 ylabel(dFuzzy.input(2).name)
24 zlabel('State')
25 colorbar
```

The results of the rule base are shown in Figure 6.11.

The simulation run with the fuzzy controller is shown in Figure 6.12. This is achieved by setting both `useFuzzy` and `useSimple` flags at the top of `HVACSim` to true. The simulation is much slower than the one using hysteresis. Note the deadband issue with the output to the HVAC; the temperature of the house is held constant in the face of the large external load q_l, but the system is constantly switching on and off to do so.

6.5 Variable HVAC Fuzzy Controller

6.5.1 Problem

The discrete fuzzy controller has a deadband issue. Modern HVAC, such as heat pumps, may have a variable setting, which will produce a smoother result.

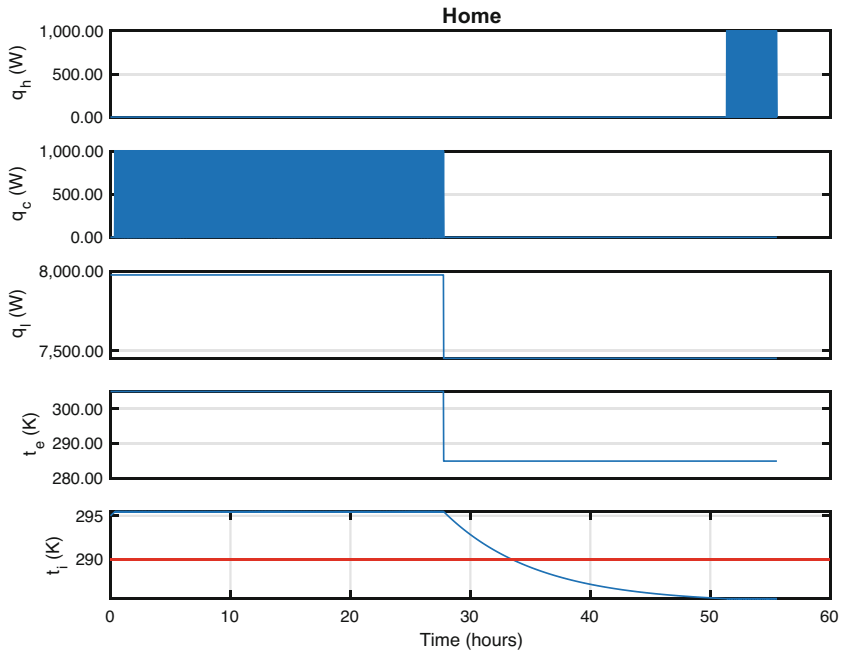


Figure 6.12: *Fuzzy simulation results*

6.5.2 Solution

In this version of the controller, we will have different inputs, rules, and outputs. The inputs will be the external temperature and the delta temperature from the setpoint. There will be a mode output and a value output which can be anywhere in the range, either negative from AC or positive for heat. There are less rules: if the delta is large, set the system on high; if it's smaller, set the system on low; and if it's close to the setpoint, keep the system off.

6.5.3 How It Works

As before, we build the system in an initialize section of the function. We then run the inference and compute the control output in an update section. A demo function will run if the function is called with no inputs, which plots the rule base. In this case, we used the string values of the fuzzy inputs and outputs to define the rules, which will be converted to indices by `BuildFuzzySystem`.

HVACFuzzyController.m

```
1  %% Fuzzy logic control system for HVAC
7  %% Form:
8  %   d     = HVACFuzzyController('initialize',qMax)
9  %   [d,q] = HVACFuzzyController('update',tE,t,tSet,d)
10 %
25 function [q,cat] = HVACFuzzyController(mode,tE,tI,tSet,d)
```

```
26
27 if( nargin < 1)
28   Demo;
29   return
30 end
31
32 %% Initialize
33 % Create the system
34 switch mode
35   case 'initialize'
36     if nargin<2
37       qMax = 1000;
38     else
39       qMax = tE;
40     end
41     oP = {qMax*[0 0 0.2 0.2] qMax*[0.2 0.3 0.5 0.6] qMax*[0.5 0.7 1.01
         1.01]};
42
43     d = BuildFuzzySystem(...
44               'id',1,...
45               'input comp',{'Cold'  'Temperate' 'Hot'} ,...
46               'input type', {@TrapezoidMF  @TrapezoidMF @TrapezoidMF}
                    ,...
47               'input params',{[-11 -11 50 70] [60 70 75 80] [70 80 111
                    111]},...
48               'input range',[-10 110],...
49               'input name','Ext Temp (F)',...
50               'id',2,...
51               'input comp',{'Chilly'  'OK'  'Warm'} ,...
52               'input type', {@TrapezoidMF  @TrapezoidMF @TrapezoidMF}
                    ,...
53               'input params',{[-21 -21 -8 0] [-5 -1 1 5] [0 8 21
                    21]},...
54               'input range',[-20 20],...
55               'input name','Delta-Temp (F)',...
56               'id',1,...
57               'output comp',{'AC' 'Off' 'Heat'},...
58               'output type',{@TrapezoidMF  @TrapezoidMF @TrapezoidMF
                    },...
59               'output params',{[-1.1 -1.1 -0.5 0] [-0.5 0 0 0.5] [0 0.5
                     1.1 1.1]},...
60               'output name','Mode',...
61               'output range',[-1 1],...
62               'id',2,...
63               'output comp',{'Zero'  'Low'  'High'},...
64               'output type',{@TrapezoidMF  @TrapezoidMF  @TrapezoidMF
                    },...
65               'output params',oP,...
66               'output name','Output',...
67               'output range',[0 qMax],...
68               'id',1,... % Cold and Too cold, Heat/high
```

```
                'rule input',{'Cold','Chilly'},...
                'rule output',{'Heat','High'},...
                'rule operator',@FuzzyAND,...
                'id',2,... % temperate and too cold, Heat/low
                'rule input',{'Temperate','Chilly'},...
                'rule output',{'Heat','Low'},...
                'rule operator',@FuzzyAND,...
                'id',3,... % Hot and too cold, AC/off
                'rule input',{'Hot','Chilly'},...
                'rule output',{'Off','Zero'},...
                'rule operator',@FuzzyAND,...
                'id',4,... % Cold and OK, Heat/zero
                'rule input',{'Cold','OK'},...
                'rule output',{'Off','Zero'},...
                'rule operator',@FuzzyAND,...
                'id',5,... % temperate and OK, off/off
                'rule input',{'Temperate','OK'},...
                'rule output',{'Off','Zero'},...
                'rule operator',@FuzzyAND,...
                'id',6,... % Hot and OK, AC/off
                'rule input',{'Hot','OK'},...
                'rule output',{'Off','Zero'},...
                'rule operator',@FuzzyAND,...
                'id',7,... % Cold and too hot, Heat/off
                'rule input',[1 3],...
                'rule output',[2 1],...
                'rule operator',@FuzzyAND,...
                'id',8,... % temperate and too hot, AC/low
                'rule input',[2 3],...
                'rule output',[1 2],...
                'rule operator',@FuzzyAND,...
                'id',9,... % Hot and too hot, AC/high
                'rule input',[3 3],...
                'rule output',[1 3],...
                'rule operator',@FuzzyAND,...
                'implicate',@ScaleIMP,...
                'aggregate','sum',...
                'defuzzify',@CentroidDF);
    q = d;
    cat = [];
  case 'update'
    kToC  = 273;
    delta = (tI - tSet)*9/5;    % in K
    tF    = (tE-kToC)*9/5 + 32; % in F
    % expect internal temperature to be within specified range
    if tF>d.input(1).range(2)
      tF = d.input(1).range(2) - eps;
    elseif tF<d.input(1).range(1)
      tF = d.input(1).range(1) + eps;
    end
    % limit delta temperature range
```

```
    if delta>d.input(2).range(2)
      delta = d.input(2).range(2) - eps;
    elseif tF<d.input(2).range(1)
      delta = d.input(2).range(1) + eps;
    end
    [cat,data] = FuzzyInference([tF;delta], d);

    mode = sign(cat(1));
    if abs(cat(1))<0.01
      mode = 0;
    end
    q = mode*cat(2);

end
```

In the update case, we check the inputs against the range and limit them if needed. This helps avoid numerical issues after the conversion from Celsius to Kelvin and allows us to have a smaller range for the delta variable without being concerned with large excursions in internal temperature. The mode output is computed using the `sign` function on the mode variable. If the mode value is very small, less than 0.01, we set the mode to 0. The final output setting requested of the HVAC, the `q`, is the product of the two variables.

The plots which follow are produced by the demo. Figure 6.13 and Figure 6.14 show the system in- puts and outputs. Since there are only two inputs, we can again produce surface plots of the outputs in Figure 6.15, Figure 6.16, and Figure 6.17.

HVACFuzzyController.m

```
%% Demonstrate the controller
function Demo

d = HVACFuzzyController('initialize');

% Plot the fuzzy variables
FuzzyPlot( d.input(1) );
FuzzyPlot( d.input(2) );
FuzzyPlot( d.output(1) );
FuzzyPlot( d.output(2) );

PrintFuzzyRules( d )

% differentiate btwn internal and external temp
t = linspace(d.input(1).range(1)+1e-12,d.input(1).range(2)-1e-12,51);
t_K = 5/9*(t-32)+273;  % convert input from C to K
tSet = 297;  % example setpoint (K)
delta = linspace(d.input(2).range(1)+1e-12,d.input(2).range(2)-1e
    -12,31);
q = zeros(length(delta),length(t_K));
mode = zeros(length(delta),length(t_K));
val = zeros(length(delta),length(t_K));
for k = 1:length(t)
```

```
157   for j = 1:length(delta)
158     [q(j,k),cat] = HVACFuzzyController('update',t_K(k),tSet+5/9*delta(j
          ),tSet,d);
159     mode(j,k) = cat(1);
160     val(j,k) = cat(2);
161   end
162 end
163
164 NewFigure('HVAC Output from Fuzzy Logic');
165 surf(t,delta,q)
166 xlabel('Outside Temperature (F)')
167 ylabel('Delta (F)')
168 zlabel('HVAC Output (W)')
169 colorbar
170 set(gca,'ydir','reverse')
171
172 NewFigure('HVAC Mode from Fuzzy Logic');
173 surf(t,delta,mode)
174 xlabel('Temperature (F)')
175 ylabel('Delta (F)')
176 zlabel('HVAC Mode')
177 colorbar
178 set(gca,'ydir','reverse')
179
180 NewFigure('HVAC Value from Fuzzy Logic');
181 surf(t,delta,val)
182 xlabel('Temperature (F)')
183 ylabel('Delta (F)')
184 zlabel('HVAC Value (W)')
185 colorbar
186 set(gca,'ydir','reverse')
187
188 tSet = 297;
189 tI   = tSet+5;
190 Q = zeros(size(t));
191 for k = 1:length(t)
192   Q(k) = HVACFuzzyController('update',t_K(k),tSet+5,tSet,d);
193 end
194
195 PlotSet(t_K,Q,'x label','T_e (K)','y label','Q (W)','plot title','Fuzzy
      HVAC');
196 y = get(gca,'ylim');
197 line(tSet*[1;1],y,'color','r')
198 line(tI*[1;1],y,'color','g')
199
200 tE = 280;
201 tSet = 294;
202 Q = zeros(size(delta));
203 for k = 1:length(delta)
204   Q(k) = HVACFuzzyController('update',tE,tSet+delta(k)*5/9,tSet,d);
205 end
```

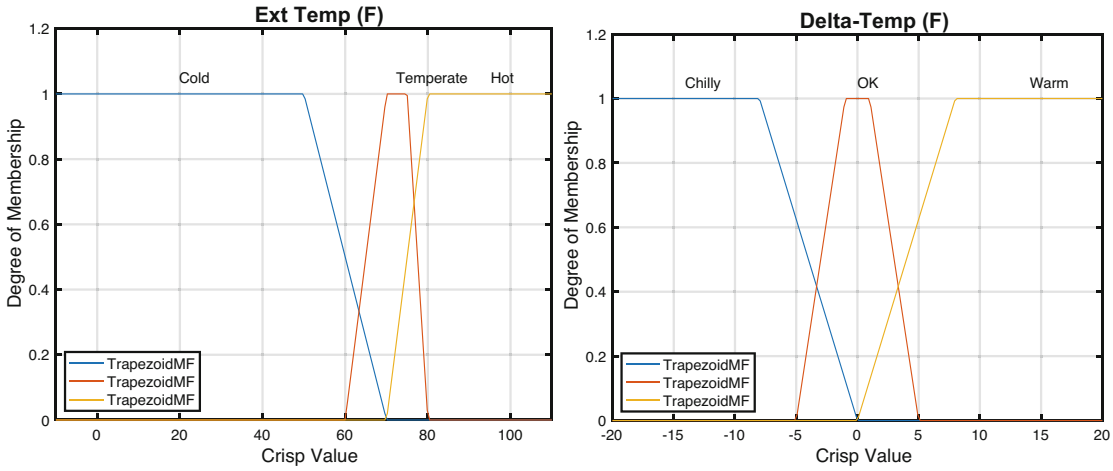


Figure 6.13: *The fuzzy inputs of the variable controller*

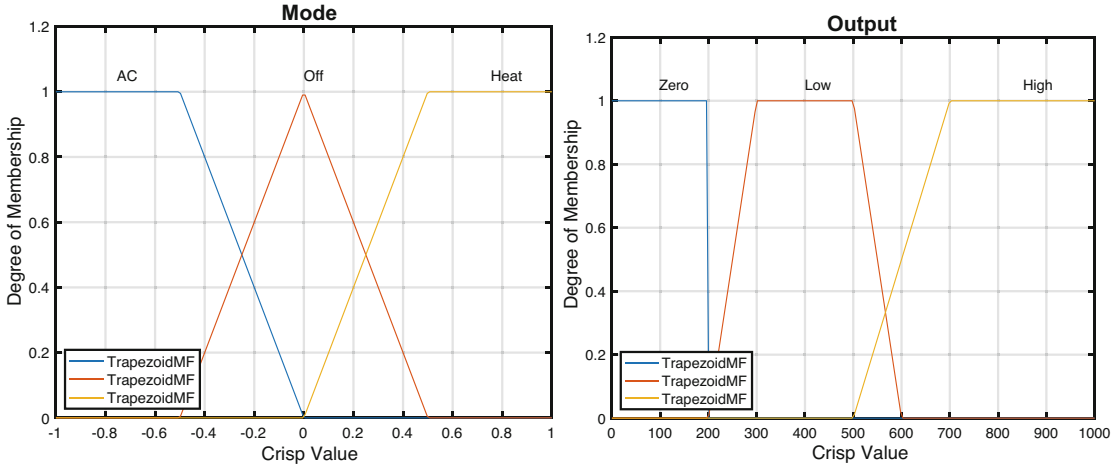


Figure 6.14: *The fuzzy outputs of the variable controller*

```
206  PlotSet(delta,Q,'x label','\Delta T (F)','y label','Q (W)',...
207    'figure title','Fuzzy HVAC','plot title',sprintf('Te = %g F',(tE-273)
           *9/5+32));
```

Here is the rule base:

```
1. if Ext Temp (F) is Cold FuzzyAND Delta-Temp (F) is Chilly then       Mode is Heat    Output is High
2. if Ext Temp (F) is Temperate FuzzyAND Delta-Temp (F) is Chilly then  Mode is Heat    Output is Low
3. if Ext Temp (F) is Hot FuzzyAND Delta-Temp (F) is Chilly then        Mode is Off     Output is Zero
4. if Ext Temp (F) is Cold FuzzyAND Delta-Temp (F) is OK then   Mode is Off     Output is Zero
5. if Ext Temp (F) is Temperate FuzzyAND Delta-Temp (F) is OK then      Mode is Off     Output is Zero
6. if Ext Temp (F) is Hot FuzzyAND Delta-Temp (F) is OK then    Mode is Off     Output is Zero
```

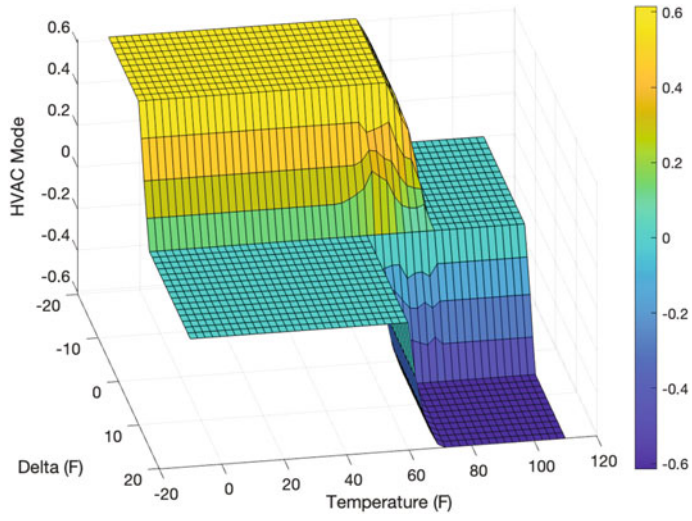


Figure 6.15: *The mode output of the variable controller following the reults*

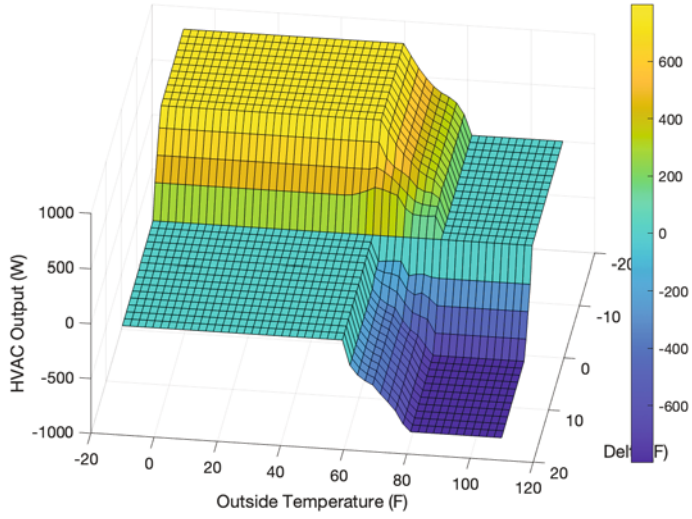


Figure 6.16: *The HVAC output of the variable controller*

```
7. if Ext Temp (F) is Cold FuzzyAND Delta-Temp (F) is Warm then         Mode is Off     Output is Zero

8. if Ext Temp (F) is Temperate FuzzyAND Delta-Temp (F) is Warm then    Mode is AC      Output is Low
9. if Ext Temp (F) is Hot FuzzyAND Delta-Temp (F) is Warm then  Mode is AC      Output is High
```

Finally, we try this version of the controller in the simulation. The plots in Figures 6.18 and 6.19 show results for both AC and heat. This updated fuzzy controller produces smooth results. Compare this to Figure 6.9 for the non-fuzzy bang-bang controller which produced limit cycling of the internal temperature.

Additional work for this system could include adding another input for the humidity. The system would need to be matched with the capabilities of the actual HVAC system.

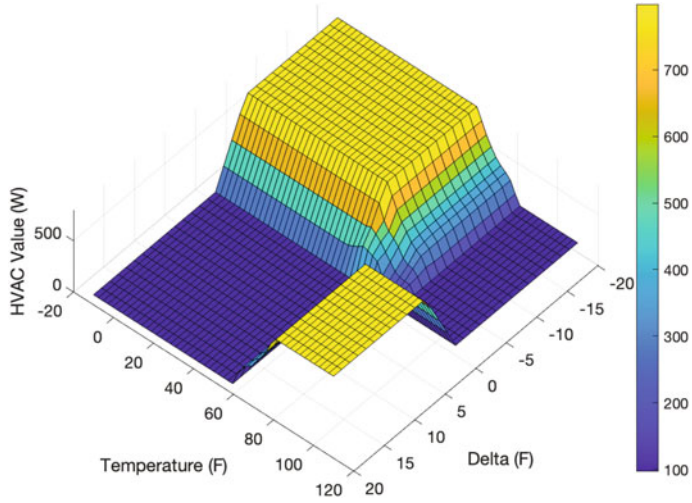


Figure 6.17: *The resulting combined AC or heat setting*

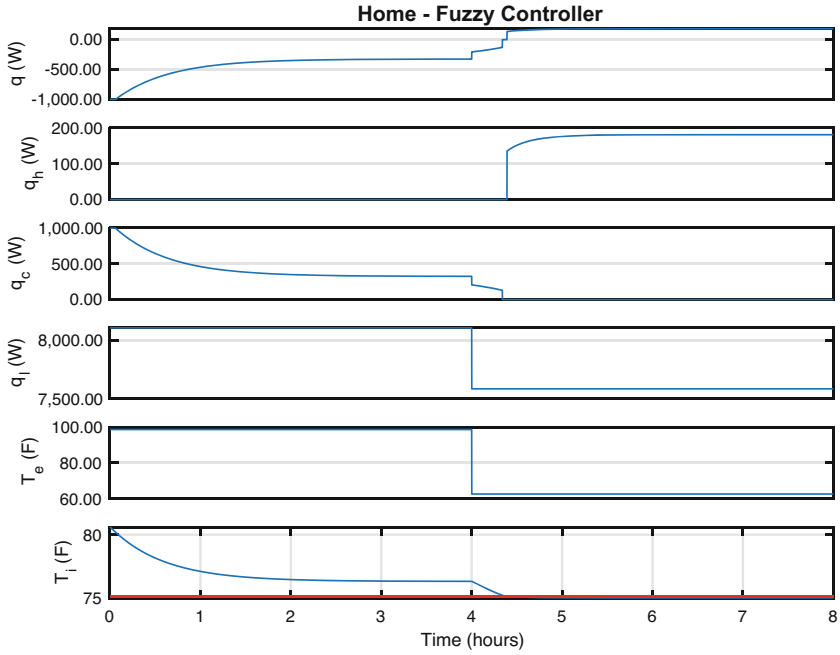


Figure 6.18: *Simulation results for the variable controller for AC*

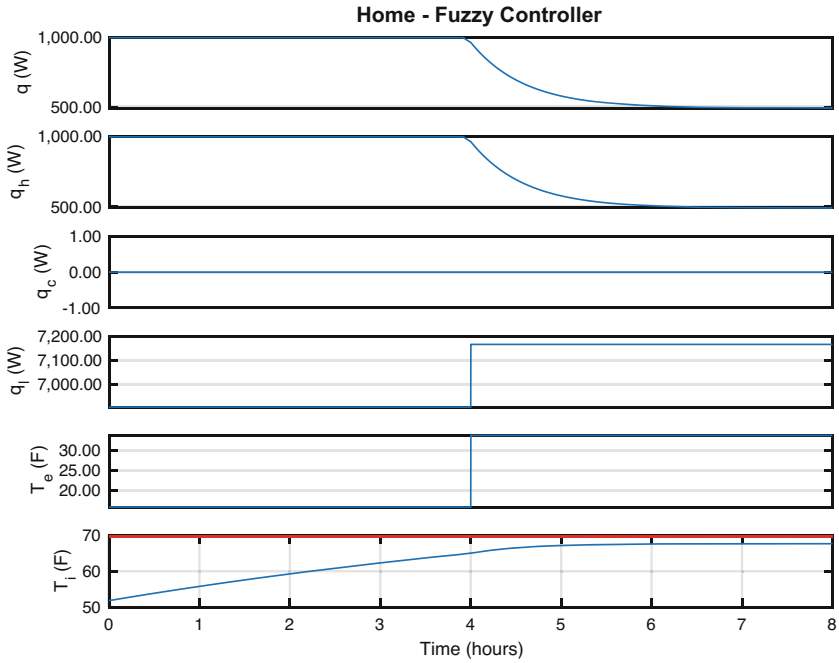


Figure 6.19: *Simulation results for the variable controller for heat*

6.6 Summary

This chapter demonstrated fuzzy logic. A windshield wiper demonstration gives an example of how it is used. The smart wiper system automatically adjusts wiper speed and wiper interval. A second system demonstrates a fuzzy HVAC system. Table 6.2 lists the functions and scripts included in the companion code. Fuzzy helper functions are grouped in Table 6.3.

Table 6.2: *Chapter code listing*

File	Description
`BuildFuzzySystem`	Builds a fuzzy logic system (data structure) using parameter pairs
`SmartWipersSystem`	Creates and returns the smart wipers data structure
`SmartWipersDemo`	Demonstrates a fuzzy logic control system for windshield wipers
`FuzzyPlot`	Plots a fuzzy set
`FuzzyInference`	Performs fuzzy inference given a fuzzy system and crisp data x
`FuzzyRand`	Creates a random set of inputs from a fuzzy system
`HVACSim`	Heating ventilation and air conditioning simulation
`HVACSimplestFuzzyController`	Discrete output simplest rule controller
`HVACFuzzyController`	Multi-input and output system fuzzy logic control system for HVAC
`HVACFuzzyPlot`	Plots the HVAC fuzzy controller rule base
`PrintFuzzyRules`	Prints fuzzy rules in a system struct to the command line

Table 6.3: *Fuzzy helper function listing*

`CentroidDF`	Centroid defuzzification
`GeneralBellMF`	General Bell membership function
`GaussianMF`	Gaussian membership function
`TriangleMF`	Triangle membership function
`TrapezoidMF`	Trapezoid membership function
`SigmoidalMF`	Displays a neural net with multiple layers
`ClipIMP`	Clip implication function
`ScaleIMP`	Scale implication function
`FuzzyOR`	Fuzzy OR (maximum of membership values)
`FuzzAND`	Fuzzy AND (minimum of membership values)