The FIS Editor handles the high-level issues for the system: How many input and output variables? What are their names? The Fuzzy Logic Toolbox doesn't limit the number of inputs. However, the number of inputs may be limited by the available memory of your machine. If the number of inputs is too large, or the number of membership functions is too big, then it may also be difficult to analyze the FIS using the other GUI tools.

The Membership Function Editor is used to define the shapes of all the membership functions associated with each variable.

The Rule Editor is for editing the list of rules that defines the behavior of the system.

The Rule Viewer and the Surface Viewer are used for looking at, as opposed to editing, the FIS. They are strictly read-only tools. The Rule Viewer is a MATLAB based display of the fuzzy inference diagram shown at the end of the last section. Used as a diagnostic, it can show (for example) which rules are active, or how individual membership function shapes are influencing the results. The Surface Viewer is used to display the dependency of one of the outputs on any one or two of the inputs — that is, it generates and plots an output surface map for the system.
The Basic Tipping Problem

Given a number between 0 and 10 that represents the quality of service at a restaurant (where 10 is excellent), and another number between 0 and 10 that represents the quality of the food at that restaurant (again, 10 is excellent), what should the tip be?

The starting point is to write down the three golden rules of tipping, based on years of personal experience in restaurants.

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

We’ll assume that an average tip is 15%, a generous tip is 25%, and a cheap tip is 5%. It is also useful to have a vague idea of what the tipping function should look like this.

Obviously the numbers and the shape of the curve are subject to local traditions, cultural bias, and so on, but the three rules are pretty universal.

Now we know the rules, and we have an idea of what the output should look like. Let’s begin working with the GUI tools to construct a fuzzy inference system for this decision process.
The FIS Editor

Double-click on an input variable icon to open the Membership Function Editor.

These menu items allow you to save, open, or edit a fuzzy system using any of the five basic GUI tools.

Double-click on the system diagram to open the Rule Editor.

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

The name of the system is displayed here. It can be changed using one of the Save as... menu options.

These pop-up menus are used to adjust the fuzzy inference functions, such as the defuzzification method.

This status line describes the most recent operation.

This edit field is used to name and edit the names of the input and output variables.

The following discussion tells you how to build a new fuzzy inference system from scratch. If you want to save time and follow along quickly, you can load the pre-built system by typing

fuzzy tipper
This loads the FIS associated with the file \texttt{tipper}.fis (the .fis is implied) and launches the FIS Editor. However, if you load the prebuilt system, you will not be building rules and constructing membership functions.

The FIS Editor displays general information about a fuzzy inference system. There is a simple diagram at the top that shows the names of each input variable on the left, and those of each output variable on the right. The sample membership functions shown in the boxes are just icons and do not depict the actual shapes of the membership functions.

Below the diagram is the name of the system and the type of inference used. The default, Mamdani-type inference, is what we've been describing so far and what we'll continue to use for this example. Another slightly different type of inference, called Sugeno-type inference, is also available. This method is explained in “Sugeno-Type Fuzzy Inference” on page 2-78. Below the name of the fuzzy inference system, on the left side of the figure, are the pop-up menus that allow you to modify the various pieces of the inference process. On the right side at the bottom of the figure is the area that displays the name of either an input or output variable, its associated membership function type, and its range. The latter two fields are specified only after the membership functions have been. Below that region are the \texttt{Help} and \texttt{Close} buttons that call up online help and close the window, respectively. At the bottom is a status line that relays information about the system.

To start this system from scratch, type

\begin{verbatim}
fuzzy
\end{verbatim}

at the MATLAB prompt. The generic untitled FIS Editor opens, with one input, labeled \texttt{input1}, and one output, labeled \texttt{output1}. For this example, we will construct a two-input, one output system, so go to the \texttt{Edit} menu and select \texttt{Add input}. A second yellow box labeled \texttt{input2} will appear. The two inputs we will have in our example are \texttt{service} and \texttt{food}. Our one output is \texttt{tip}. We'd like to change the variable names to reflect that:

1. Click once on the box (yellow) on the left marked \texttt{input1} (the box will be highlighted in red).

2. In the white edit field on the right, change \texttt{input1} to \texttt{service} and press \texttt{Return}. 

2-35
3 Click once on the box (yellow) marked input2 (the box will be highlighted in red).

4 In the white edit field on the right, change input2 to food and press Return.

5 Click once on the box (blue) on the right marked output1.

6 In the white edit field on the right, change output1 to tip.

7 From the File menu, select Export and then To Workspace...

8 Enter the variable name tipper and click on OK.

You will see the diagram updated to reflect the new names of the input and output variables. There is now a new variable in the workspace called tipper that contains all the information about this system. By saving to the workspace with a new name, you also rename the entire system. Your window will look something like this.
Leave the inference options in the lower left in their default positions for now. You’ve entered all the information you need for this particular GUI. Next define the membership functions associated with each of the variables. To do this, open the Membership Function Editor. You can open the Membership Function Editor in one of three ways:

- Pull down the **View** menu item and select **Edit Membership Functions**.
- Double-click on the icon for the output variable, **tip**.
- Type `mfedit` at the command line.
The Membership Function Editor

This is the “Variable Palette” area. Click on a variable here to make it current and edit its membership functions.

This graph field displays all the membership functions of the current variable.

These menu items allow you to save, open, or edit a fuzzy system using any of the five basic GUI tools.

These text fields display the name and type of the current variable.

This edit field lets you set the range of the current variable.

This edit field lets you set the display range of the current plot.

This status line describes the most recent operation.

This edit field lets you change the numerical parameters for the current membership function.

This pop-up menu lets you change the type of the current membership function.

This edit field lets you change the name of the current membership function.

Click on a line to select it and you can change any of its attributes, including name, type and numerical parameters. Drag your mouse to move or change the shape of a selected membership function.

The Membership Function Editor shares some features with the FIS Editor. In fact, all of the five basic GUI tools have similar menu options, status lines, and Help and Close buttons. The Membership Function Editor is the tool that lets you display and edit all of the membership functions associated with all of the input and output variables for the entire fuzzy inference system.
When you open the Membership Function Editor to work on a fuzzy inference system that does not already exist in the workspace, there are not yet any membership functions associated with the variables that you have just defined with the FIS Editor.

On the upper left side of the graph area in the Membership Function Editor is a “Variable Palette” that lets you set the membership functions for a given variable. To set up your membership functions associated with an input or an output variable for the FIS, select an FIS variable in this region by clicking on it.

Next select the Edit pull-down menu, and choose Add MFS .... A new window will appear, which allows you to select both the membership function type and the number of membership functions associated with the selected variable. In the lower right corner of the window are the controls that let you change the name, type, and parameters (shape), of the membership function, once it has been selected.

The membership functions from the current variable are displayed in the main graph. These membership functions can be manipulated in two ways. You can first use the mouse to select a particular membership function associated with a given variable quality, (such as poor, for the variable, service), and then drag the membership function from side to side. This will affect the mathematical description of the quality associated with that membership function for a given variable. The selected membership function can also be tagged for dilation or contraction by clicking on the small square drag points on the membership function, and then dragging the function with the mouse toward the outside, for dilation, or toward the inside, for contraction. This will change the parameters associated with that membership function.

Below the Variable Palette is some information about the type and name of the current variable. There is a text field in this region that lets you change the limits of the current variable’s range (universe of discourse) and another that lets you set the limits of the current plot (which has no real effect on the system).
The process of specifying the input membership functions for this two input tipper problem is as follows:

1. Select the **input** variable, service, by double-clicking on it. Set both the **Range** and the **Display Range** to the vector [0 10].

2. Select **Add MFs...** from the **Edit** menu. The window below opens.

3. Use the tab to choose **gaussmf** for **MF Type** and 3 for **Number of MFs**. This adds three Gaussian curves to the input variable service.

4. Click once on the curve with the leftmost *hump*. Change the name of the curve to **poor**. To adjust the shape of the membership function, either use the mouse, as described above, or type in a desired parameter change, and then click on the **membership function**. The default parameter listing for this curve is [1.5 0].

5. Name the curve with the middle hump, **good**, and the curve with the rightmost hump, **excellent**. Reset the associated parameters if desired.

6. Select the **input** variable, **food**, by clicking on it. Set both the **Range** and the **Display Range** to the vector [0 10].

7. Select **Add MFs...** from the **Edit** menu and add two **trapmf** curves to the input variable **food**.

8. Click once directly on the curve with the leftmost trapezoid. Change the name of the curve to **rancid**. To adjust the shape of the membership function, either use the mouse, as described above, or type in a desired parameter change, and then click on the **membership function**. The default parameter listing for this curve is [0 0 1 3].

9. Name the curve with the rightmost trapezoid, **delicious**, and reset the associated parameters if desired.
Next you need to create the membership functions for the output variable, \textit{tip}. To create the output variable membership functions, use the Variable Palette on the left, selecting the output variable, \textit{tip}. The inputs ranged from 0 to 10, but the output scale is going to be a tip between 5 and 25 percent.

Use triangular membership function types for the output. First, set the \textbf{Range} (and the \textbf{Display Range}) to \([0, 30]\), to cover the output range. Initially, the \textit{cheap} membership function will have the parameters \([0, 5, 10]\), the \textit{average} membership function will be \([10, 15, 20]\), and the \textit{generous} membership function will be \([20, 25, 30]\). Your system should look something like this.

Now that the variables have been named, and the membership functions have appropriate shapes and names, you’re ready to write down the rules. To call up the Rule Editor, go to the \textbf{View} menu and select \textit{Edit Rules...}, or type \texttt{ruleedit} at the command line.
Constructing rules using the graphical Rule Editor interface is fairly self evident. Based on the descriptions of the input and output variables defined with the FIS Editor, the Rule Editor allows you to construct the rule statements automatically, by clicking on and selecting one item in each input variable box, one item in each output box, and one connection item. Choosing none as one of the variable qualities will exclude that variable from a given rule. Choosing not under any variable name will negate the
associated quality. Rules may be changed, deleted, or added, by clicking on the appropriate button.

The Rule Editor also has some familiar landmarks, similar to those in the FIS Editor and the Membership Function Editor, including the menu bar and the status line. The Format pop-up menu is available from the Options pull-down menu from the top menu bar—this is used to set the format for the display. Similarly, Language can be set from under Options as well. The Help button will bring up a MATLAB Help window.

To insert the first rule in the Rule Editor, select the following:

- poor under the variable service
- rancid under the variable food
- The or radio button, in the Connection block
- cheap, under the output variable, tip.

The resulting rule is

1. If (service is poor) or (food is rancid) then (tip is cheap) (1)

The numbers in the parentheses represent weights that can be applied to each rule if desired. You can specify the weights by typing in a desired number between zero and one under the Weight setting. If you do not specify them, the weights are assumed to be unity (1).

Follow a similar procedure to insert the second and third rules in the Rule Editor to get

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

To change a rule, first click on the rule to be changed. Next make the desired changes to that rule, and then click Change rule. For example, to change the first rule to

1. If (service not poor) or (food not rancid) then (tip is not cheap) (1)
Select the **not** check box under each variable, and then click **Change rule**.

The **Format** pop-up menu from the **Options** menu indicates that you’re looking at the verbose form of the rules. Try changing it to **symbolic**. You will see

1. 
   
   \[
   (service==poor) \Rightarrow (tip=cheap) \quad (1)
   \]

2. 
   
   \[
   (service==good) \Rightarrow (tip=average) \quad (1)
   \]

3. 
   
   \[
   (service==excellent) \Rightarrow (tip=generous) \quad (1)
   \]

There is not much difference in the display really, but it is slightly more language neutral, since it doesn’t depend on terms like “if” and “then.” If you change the format to indexed, you’ll see an extremely compressed version of the rules that has squeezed all the language out.

1. 1 (1) : 1
2. 2 (1) : 1
3. 3 (1) : 1

This is the version that the machine deals with. The first column in this structure corresponds to the input variable, the second column corresponds to the output variable, the third column displays the weight applied to each rule, and the fourth column is shorthand that indicates whether this is an OR (2) rule or an AND (1) rule. The numbers in the first two columns refer to the index number of the membership function. A literal interpretation of rule 1 is “If input 1 is MF1 (the first membership function associated with input 1) then output 1 should be MF1 (the first membership function associated with output 1) with the weight 1. Since there is only one input for this system, the AND connective implied by the 1 in the last column is of no consequence.

The symbolic format doesn’t bother with the terms, *if, then*, and so on. The indexed format doesn’t even bother with the names of your variables. Obviously the functionality of your system doesn’t depend on how well you have named your variables and membership functions. The whole point of naming variables descriptively is, as always, making the system easier for you to interpret. Thus, unless you have some special purpose in mind, it will probably be easier for you to stick with the **verbose** format.

At this point, the fuzzy inference system has been completely defined, in that the variables, membership functions, and the rules necessary to calculate tips
are in place. Now look at the fuzzy inference diagram presented at the end of the previous section and verify that everything is behaving the way you think it should. This is exactly the purpose of the Rule Viewer, the next of the GUI tools we’ll look at. From the View menu, select View rules....

The Rule Viewer

The Rule Viewer displays a roadmap of the whole fuzzy inference process. It is based on the fuzzy inference diagram described in the previous section. You see a single figure window with 10 small plots nested in it. The three small...
plots across the top of the figure represent the antecedent and consequent of the first rule. Each rule is a row of plots, and each column is a variable. The first two columns of plots (the six yellow plots) show the membership functions referenced by the antecedent, or the if-part of each rule. The third column of plots (the three blue plots) shows the membership functions referenced by the consequent, or the then-part of each rule. If you click once on a rule number, the corresponding rule will be displayed at the bottom of the figure. Notice that under food, there is a plot which is blank. This corresponds to the characterization of none for the variable food in the second rule. The fourth plot in the third column of plots represents the aggregate weighted decision for the given inference system. This decision will depend on the input values for the system.

There are also the now familiar items like the status line and the menu bar. In the lower right there is a text field into which you can enter specific input values. For the two-input system, you will enter an input vector, [9 8], for example, and then click on Input. You can also adjust these input values by clicking anywhere on any of the three plots for each input. This will move the red index line horizontally, to the point where you have clicked. You can also just click and drag this line in order to change the input values. When you release the line, (or after manually specifying the input), a new calculation is performed, and you can see the whole fuzzy inference process take place. Where the index line representing service crosses the membership function line “service is poor” in the upper left plot will determine the degree to which rule one is activated. A yellow patch of color under the actual membership function curve is used to make the fuzzy membership value visually apparent. Each of the characterizations of each of the variables is specified with respect to the input index line in this manner. If we follow rule 1 across the top of the diagram, we can see the consequent “tip is cheap” has been truncated to exactly the same degree as the (composite) antecedent—this is the implication process in action. The aggregation occurs down the third column, and the resultant aggregate plot is shown in the single plot to be found in the lower right corner of the plot field. The defuzzified output value is shown by the thick line passing through the aggregate fuzzy set.

The Rule Viewer allows you to interpret the entire fuzzy inference process at once. The Rule Viewer also shows how the shape of certain membership functions influences the overall result. Since it plots every part of every rule, it can become unwieldy for particularly large systems, but, for a relatively small number of inputs and outputs, it performs well (depending on how
much screen space you devote to it) with up to 30 rules and as many as 6 or 7 variables.

The Rule Viewer shows one calculation at a time and in great detail. In this sense, it presents a sort of micro view of the fuzzy inference system. If you want to see the entire output surface of your system, that is, the entire span of the output set based on the entire span of the input set, you need to open up the Surface Viewer. This is the last of our five basic GUI tools in the Fuzzy Logic Toolbox, and you open it by selecting **View surface...** from the View menu.
Upon opening the Surface Viewer, we are presented with a two-dimensional curve that represents the mapping from service quality to tip amount. Since this is a one-input one-output case, we can see the entire mapping in one plot. Two-input one-output systems also work well, as they generate three-dimensional plots that MATLAB can adeptly manage. When we move...
beyond three dimensions overall, we start to encounter trouble displaying the results. Accordingly, the Surface Viewer is equipped with pop-up menus that let you select any two inputs and any one output for plotting. Just below the pop-up menus are two text input fields that let you determine how many x-axis and y-axis grid lines you want to include. This allows you to keep the calculation time reasonable for complex problems. Clicking the Evaluate button initiates the calculation, and the plot comes up soon after the calculation is complete. To change the x-axis or y-axis grid after the surface is in view, simply change the appropriate text field, and click either X-grids or Y-grids, according to which text field you changed, to redraw the plot.

The Surface Viewer has a special capability that is very helpful in cases with two (or more) inputs and one output: you can actually grab the axes and reposition them to get a different three-dimensional view on the data. The Ref. Input field is used in situations when there are more inputs required by the system than the surface is mapping. Suppose you have a four-input one-output system and would like to see the output surface. The Surface Viewer can generate a three-dimensional output surface where any two of the inputs vary, but two of the inputs must be held constant since computer monitors cannot display a five-dimensional shape. In such a case the input would be a four-dimensional vector with NaNs holding the place of the varying inputs while numerical values would indicate those values that remain fixed. An NaN is the IEEE symbol for not a number.

This concludes the quick walk-through of each of the main GUI tools. Notice that for the tipping problem, the output of the fuzzy system matches our original idea of the shape of the fuzzy mapping from service to tip fairly well. In hindsight, you might say, “Why bother? I could have just drawn a quick lookup table and been done an hour ago!” However, if you are interested in solving an entire class of similar decision-making problems, fuzzy logic may provide an appropriate tool for the solution, given its ease with which a system can be quickly modified.

**Importing and Exporting from the GUI Tools**

When you save a fuzzy system to disk, you’re saving an ASCII text FIS file representation of that system with the file suffix .fis. This text file can be edited and modified and is simple to understand. When you save your fuzzy system to the MATLAB workspace, you’re creating a variable (whose name
you choose) that will act as a MATLAB structure for the FIS system. FIS files and FIS structures represent the same system.

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