Part 5

Learning outcomes

In this session you will learn how to use the R Fuzzy Toolbox.

Download the 'T1 Fuzzy' toolbox from the module web pages, rename it as ‘fuzzy.r’ (it may have a version number stamp like 'fuzzy-v0_7.r'), and then load this (it is just a series of functions) into the environment by entering:

```
source('fuzzy.r')
```

Now let's draw some membership functions. R has a number of functions available :

dsigmf - the difference between 2 sigmoidal membership functions (a ‘hat’ shape)
gauss2mf – gaussian combination membership function
gaussmf – gaussian curve membership function (this is a popular membership function)
gbellmf – generalised bell-shaped function
psigmf – the product of two sigmoid functions
smf – s shaped membership function
trapmf – trapezoidal shaped membership function (this is a popular membership function)
trimf – triangular shaped membership function (popular)
zmf – z shaped

All of these membership functions take various parameters. Try them out to discover what they each do (or check out the MATLAB documentation linked via the module web pages). Type in the following code:

```
x = seq(0,10,0.1)
y = trimf(x,c(3, 6, 8))
plot(x, y, type='l')
title('An example triangular membership function')
```

The three parameters (3,6,8) determine the points at which the triangle joins the x axis (3,8) and the peak (6). Experiment with some others – especially **gaussmf** and **trapmf**, e.g.:

```
y = gaussmf(x,c(1,6))
plot(x, y, type='l')
```

Try different membership functions, with different parameters, to understand their effects.

Another potentially useful command is **evalmf** which evaluates any membership function.
Use the help system to find out how this works

```
y= evalmf(x,c(3,6,8),'trimf')
plot(x, y, type='l')
y= evalmf(x,c(3,6,8),'gaussmf')
plot(x, y, type='l')
```

When you build a fuzzy system in R consisting of rules, membership functions and so on you develop a fuzzy inference system – known as an FIS in R. To be able to consider membership functions properly, we place everything inside an FIS. The following code creates a linguistic variable representing 'age' with 3 associated linguistic terms or labels. We see in this code that to do this we create a FIS first. Note the use of # to allow for comments. Enter this code into a function file and see the results. Where necessary, refer to the equivalent functions in MATLAB help to fully understand what is going on.

```
# This is a simple function to create a linguistic variable
# for age with 3 linguistic labels - young, middleaged and old

# the first statement creates a new FIS with string name test
# and R variable a. The FIS takes certain defaults which you can
# ignore for the moment.
a= newfis('test');


# This adds a variable called 'linguistic age' to the FIS which is
# of type input and lies between 0 and 70
a= addvar(a,'input', 'age',c(0, 70));


# We now add three membership functions - read the help to understand
# the parameters
a= addmf(a,'input',1,'young','gaussmf',c(10, 0));
a= addmf(a,'input',1,'middleaged','gaussmf',c(10, 40));
a= addmf(a,'input',1,'old','gaussmf',c(10, 70));


# This plots the membership functions
plotmf(a,'input',1)
```

Now add a new variable into your r-file called 'male weight' which describes someone as small, medium or large over a domain ranging from 65Kg to 120Kg. Use triangular membership functions. Plot the result.

Finally add a third variable to the same r-file 'male feet size' choosing five sensible labels and domain. Use membership functions that are not Gaussian or triangular.

Pictures are very important when developing fuzzy systems. There is a command to show all three variables in together on one single plot window. First set the window to accept 3-subplots in a column:

```
par(mfrow=c(3,1))
```

Then issue the **plotmf** commands for each of the three variables. You should be able to adjust the window size with the mouse until the plots look nice. Since you will also need to be able to embed these sort of pictures in documents take this figure and place it in a file using the **pdf()** or **png()** command, or through the word processing package you use on a regular basis.

Part 6

Learning outcomes

In this session you will be able to use if-then rules in a FIS and develop a small FIS.

Having had some exposure to membership functions, we can now start to consider how to put these together in rules in a FIS in R. To do this we are going to develop a small set of rules. We have an example system that gives advice on whether a bank should give a loan based on two things: period of employment and salary[1]. We have conducted a knowledge acquisition exercise with an expert from which we have gleaned the following information:

1. The salaries under consideration range from £0 to £100,000.
2. The period of employment is their number of year they have been in their current job, ranging from 0 to 40 years.
3. The linguistic salary ('salary') has three labels, *low*, *medium* and *high* where *medium* is best represented by a trapezoidal membership function and *low* and *high* are 'shoulders'.
4. The linguistic period of employment ('period') is covered by five labels (*very short*, *short*, *medium*, *long* and *very long*) and are represented by triangular membership functions.
5. The decision takes three linguistic terms *no*, *maybe* and *yes*. *No* and *yes* are triangular whilst *maybe* is trapezoidal. The decision range is 0 to 100.

There are two rules initially:

1. If *salary* is *high* and *period* is *very long* and then *decision* is *yes*.
2. If *salary* is *low* and *period* is *very short* then *decision* is *no*.

To add these rules to a FIS we use **addrule**, as in the following:

```
rulelist = rbind(c(3,5,3,1,1), c(1,1,1,1,1))
fis= addrule(fis, rulelist)
```

where *fis* is the FIS name and *rulelist* is a matrix. See the MATLAB help to understand more how **addrule** works. Once you have created the required variables, terms and associated rules, you can use **showrule** to print out the rules in a textual format.

Your tasks are to:
1. create the variables and plot them on the same figure
2. add the rules to your system and show the rules;

A 'complete ruleset' would consist of every combination of the terms of salary with all the terms of period; in this case this would be a set of 15 rules. In a system such as this, it is not necessary to have a complete ruleset, but you need to ensure that every combination of inputs (salary 0 to 100000, and period 0 to 40) produces sensible output. If you have spare time put in more rules to try to create a realistic system that fulfills this criteria.

[1] This is obviously a very simple example!

Part 7

Learning outcomes

You will now create the full restaurant tipping system in R, and run the fuzzy inference system to produce output.

Enter the complete tipper FIS as given in the MATLAB tutorial example[2]. Make sure you have entered it correctly by plotting the membership functions (all three on the same plot window), and using **showfis** and **showrule** to print our the FIS and its rules.

```
plotmf(fis,'input',1)
plotmf(fis,'input',2)
plotmf(fis,'ouput',1)
showfis(fis)
showrule(fis)
```

To actually run the FIS on a set of inputs to produce an output we use the evalfis function. In simplest form, we can use it to produce a single output for one set of inputs (i.e. one value of service and food). For example:

```
evalfis(c(1,2),fis)
```

Several inputs can be evaluated at once by putting each set of inputs into a separate row of an input matrix, using any standard R method, for example:

```
inputs= rbind(c(3,7),c(2,7))
evalfis(inputs,fis)
```

The return value of **evalfis** is a single matrix of output values, so this may be put into a variable:

```
outputs= evalfis(inputs,fis)
```

R provides a useful function to visualise the output surface plot for a two-input-one-output FIS:

```
gensurf(fis)
```

Try different values of membership functions, and different rules sets to alter the output surface of the tipper example.

Finally, you may notice that the built-in FIS functions only allow you to **addmf**, not to (for example) **delmf** or **setmf**. This means that you have to keep creating a new FIS each time you want to change even one parameter of one membership function. To avoid this, you can use standard R commands to alter the FIS structures directly, such as:

```
fis$input[[1]]$mf[[1]]$type= 'trimf'
fis$input[[1]]$mf[[1]]$params= c(0,10,10)
plotmf(fis,'input',1)
```

[2] Note that you can find sample code in 'fuzzy.r'!