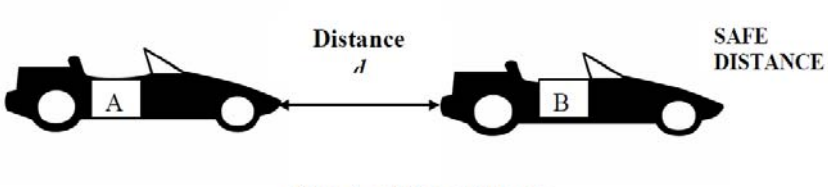
FUZZY LOGIC SYSTEMS LAB CLASS GUIDE

Task 1 – Mamdani: Car Driving



Cars A and B on a highway

You are driving car A on a highway. You want to keep a **safe distance** to car B in front of you. Design a (simplified) fuzzy-logic system, which satisfies the requirements.

Proceed as follows:

a. Determine the required fuzzy variables (input/output) and their ranges.
b. Form the rule base.
c. Use fuzzy reasoning to check the operability of the rule base.

SOLUTION:

a. **Determine the required fuzzy variables**: Start with a simple case

INPUT: Distance
OUTPUT: Braking power
Three (3) membership functions are chosen for both input and output

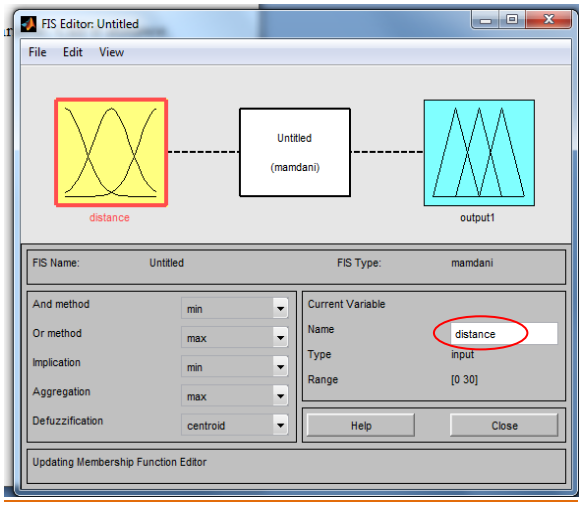
Membership functions for INPUT:
Distance *d* (meters): **low, medium, high**

Membership functions for OUTPUT:
Braking power *b* (%): **no, medium, hard**

We will use the fuzzy toolbox to define the fuzzy system by giving numerical values for the variables indicated

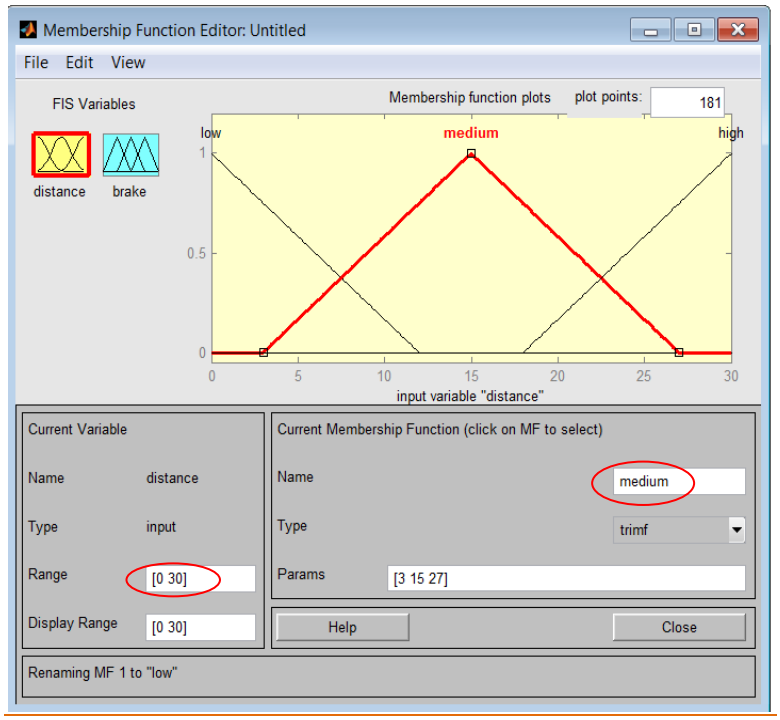
In MATLAB type
» fuzzy

This opens the GUI. Activate the input1 window by clicking with the left button of the mouse and give a name to the fuzzy input variable. Call it *distance*.



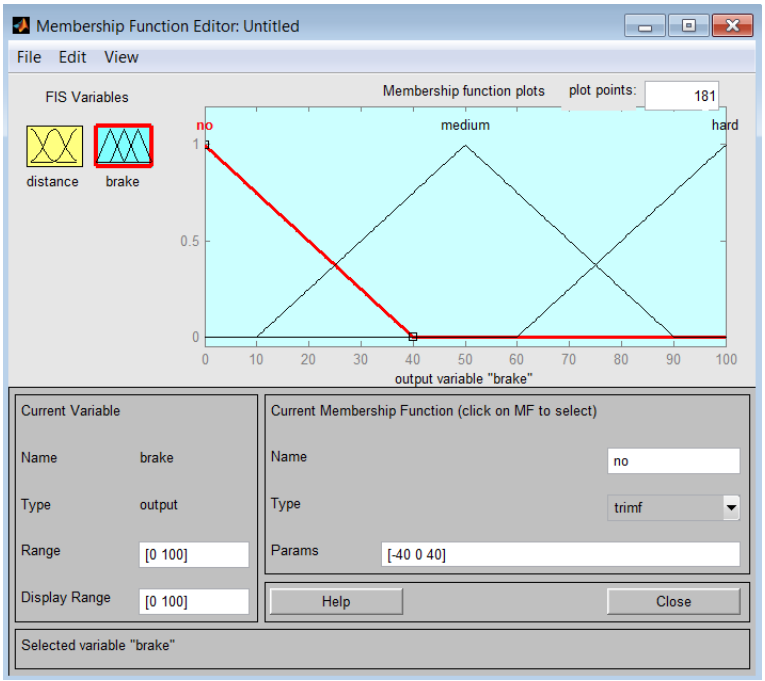
Next click the input block twice with mouse to open the membership function window. First define the range, say from 0 to 30 m/s.

Give a name to each membership function by selecting each one directly with the mouse: Call them *low*, *medium* and *high*. When you are finished, click *Close*.



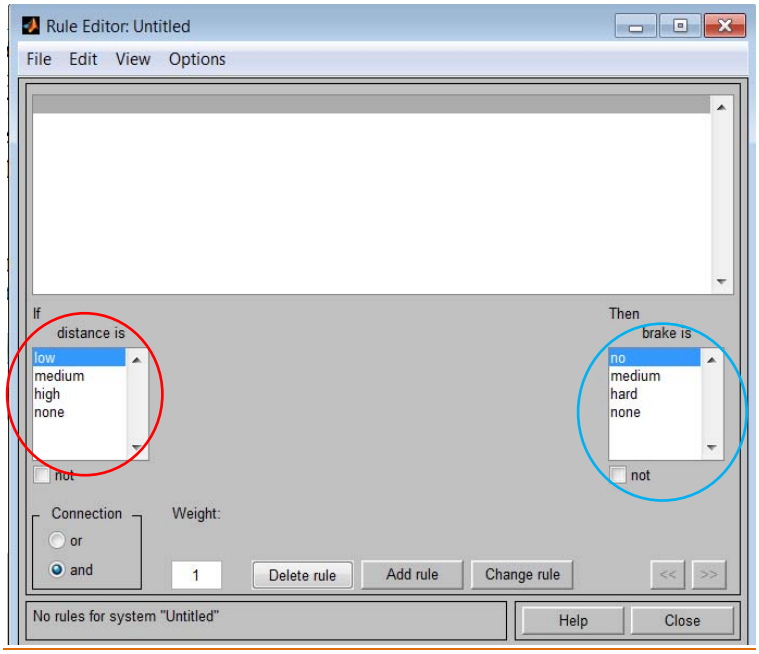
In the previous figure the membership function of the middle has been activated and renamed as *medium*.

Repeat the same procedure with the output braking power. Define the name of the output, brake, and its range, e.g. [0 100]. Use three membership functions: *no*, *medium* and *hard*.



b. <u>Form the rule base</u>

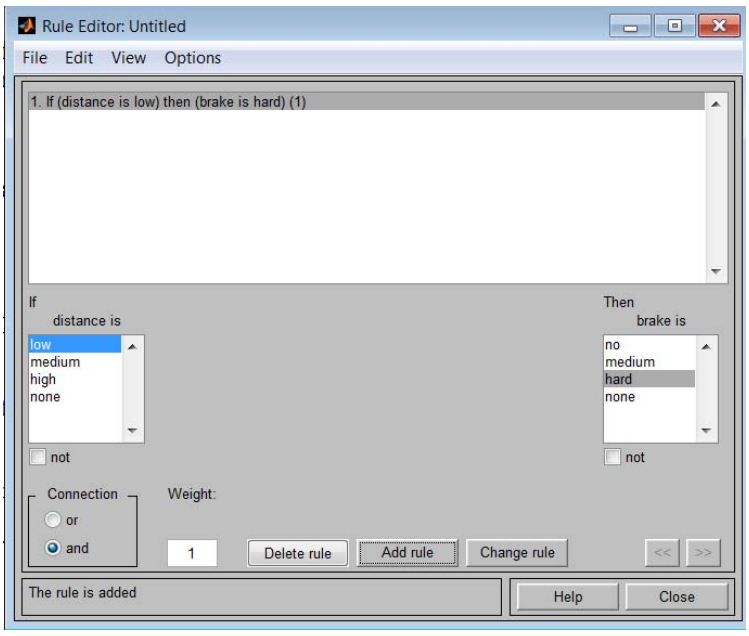
What is missing from the fuzzy system now is the rule base. Open *Edit* menu and click *Rules*. Then the following display opens.



The left-hand side contains the membership functions of the input, *distance*. The right-hand side has the membership functions of the output, *brake*. It can happen that the input side has several variables, which can be connected either by *and* or *or*, the *Connection* block is in the lower left-hand corner. In the current example we have only one input variable, so the connective is not used. The weight factor (default value = 1), indicates the importance of a specific rule. The construction of the rule base is the hardest part of the design task. Here a simple-minded rule base is constructed based on driving experience. A typical rule is:

If distance is low, then brake is hard

With the mouse choose the membership function *low* for the distance and *hard* for brake. This is done in the figure above. Then click ***Add rule***. The result is seen below.



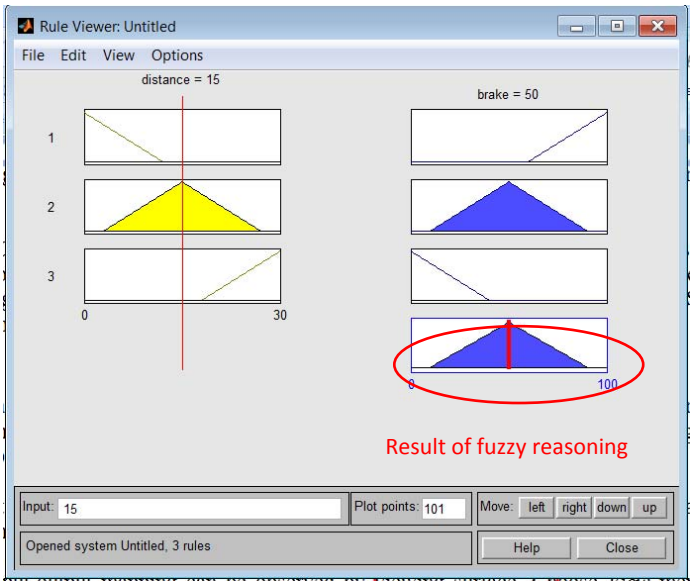
Let us set two other rules, one for *medium* distance and the other for *high* distance.

If distance is medium, then brake is medium
If distance is high, then brake is no

Our simple, rule base is now complete. Note the weighting parameter (1) at the end of each rule, i.e. all rules have the same weighting.

Now the design of the fuzzy system is complete. The Toolbox provides two more interesting ways expressing the rule base.

Check under *Options* and there choose *Format*. You can see that the rules as shown are given *verbose*. The other forms are *symbolic* and *indexed*. Check in what form the rule base is given in each case. Viewing rules gives you the overall picture of the developed fuzzy system. From the main FIS editor, choose from *View* menu *Rules.*



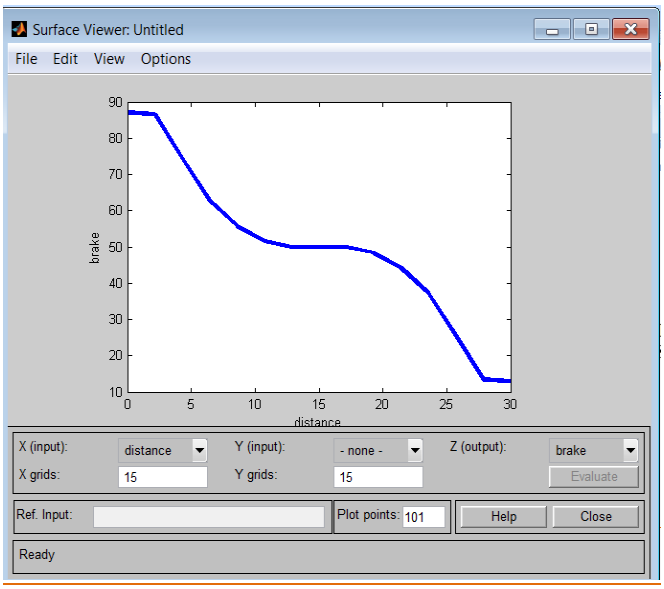
c. <u>Use fuzzy reasoning to check the operability of the rule base</u>

The red line on the left indicates the value of the input, 15 m. In the right-hand side lower corner is the result of fuzzy reasoning. It is a fuzzy set. Applying defuzzification method, in the figure center of gravity has been chosen, a crisp value is obtained. Result of fuzzy reasoning is brake = 50%.

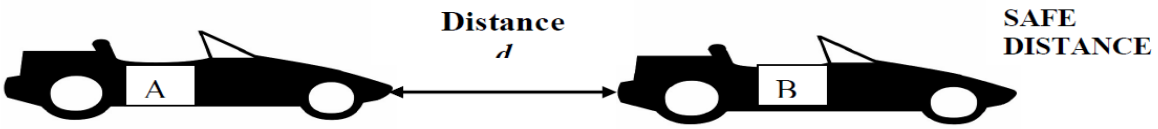
Different input values can be tried by moving the red, vertical line on the left-hand side. Changing the input value results in different output values.

Finally, the input-output mapping can be observed by viewing the surface. Choose *View* menu and under it *Surface.* It is clear that our map is nonlinear. This is where the power of fuzzy systems is strong.

The fuzzy system viewed as input-output mapping:



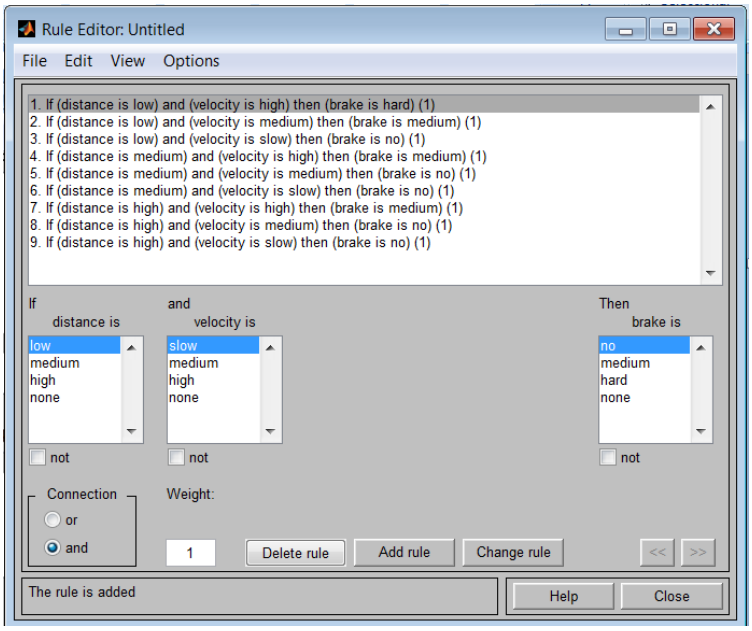
What is missing ?



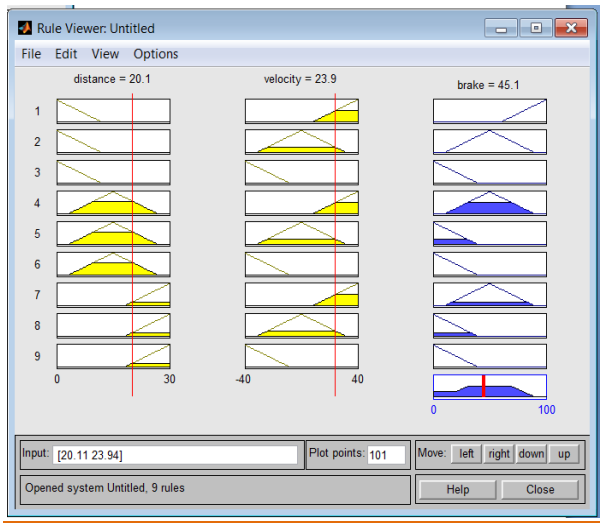
SPEED!
(relative speed between vehicles A and B).

SOLUTION:
Set up a new rule base with two inputs *distance* and *speed*, one output, *braking power*.
The second input can be added by opening *Edit* menu *Add Variable*. Otherwise the steps are as before. Remember to give a name for the added input. Call it *velocity*. Set the speed range from -40 to 40 km/h and divide the range into three membership functions: *slow*, *medium*, and *high* speed.

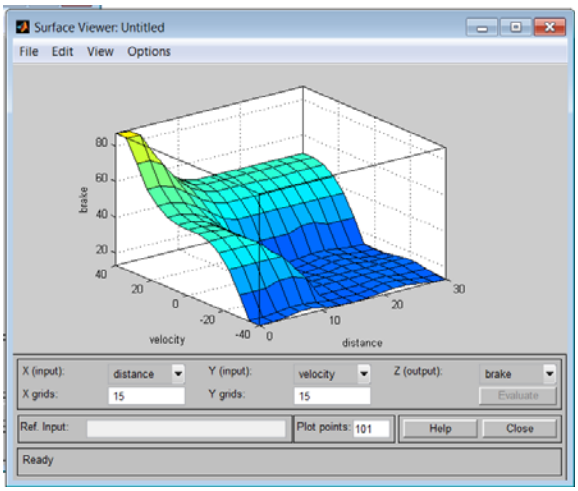
Delete the previous rules and introduce the nine rules shown in next figure, which are easy to understand. Of course, the rules are not unique. Other rules could be used as well. The rule base must be viewed and tested to see its effectiveness and how well the system specifications are satisfied.



Viewing the rules is shown in next figure.



Again you can test the system by choosing different values of inputs. If the result is not satisfactory, then refine the system, e.g. it seems that the area of short distance and high speed does not give strong enough braking response. Finally, the surface view of the rules is shown:



Note that the right hand-side corner is flat with value zero over a fairly large area. You can change that by introducing a new membership function **little** for braking power. You also have to change the rule base.

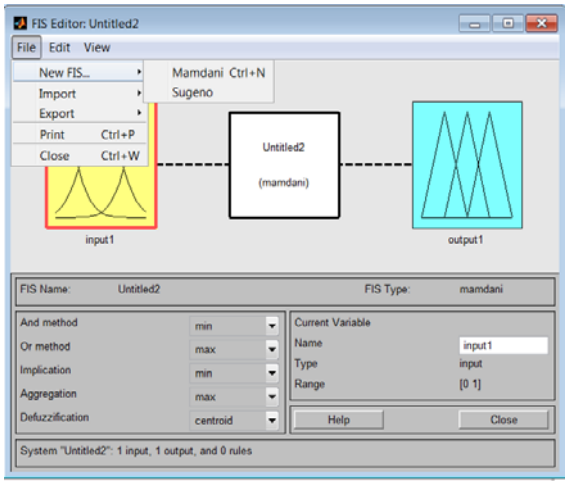
Task 2 – Mamdani and Sugeno: Function $y = x^2$

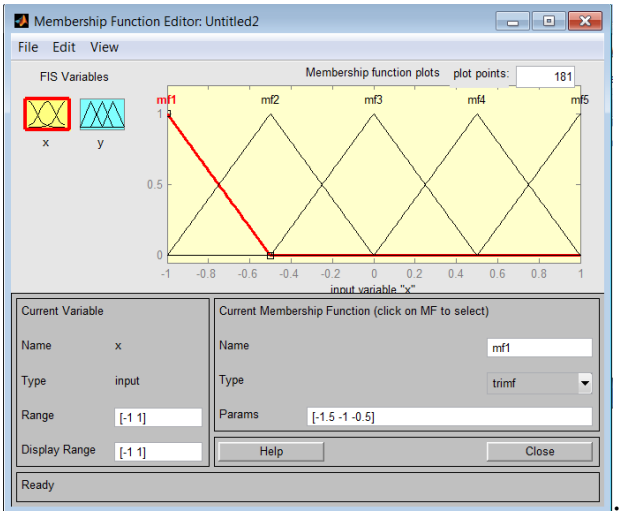
Mamdani:

Consider input-output data given in the table. The data is from the function $y = x^2$.

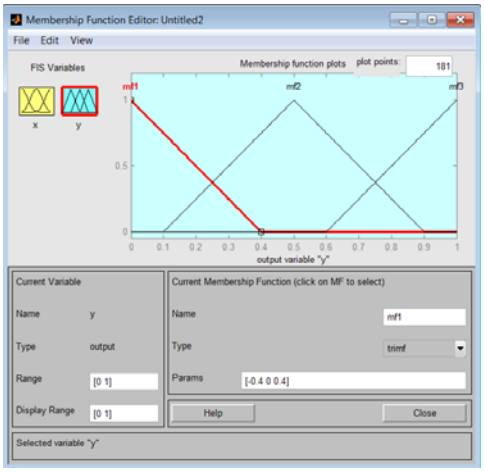
x	y
-1.0	1.0
-0.5	0.25
0	0
0.5	0.25
1.0	1.0

SOLUTION:
The data can be represented using Mamdani reasoning with e.g. triangular membership functions at the input and singletons at the output positioned at y_i.

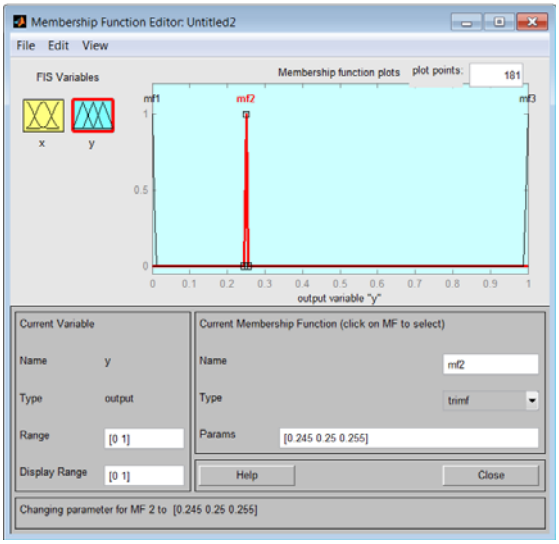




The number of x_i points is five. Choose as many triangular membership functions for input variable. For each data point generate a triangular membership function, the maximum of which occurs exactly at a given data point, say *mf3* has maximum at $x = 0$, which is one of the data points. This is seen in the above figure. Mamdani reasoning does not support singletons, so let's first choose triangular membership functions at the output.

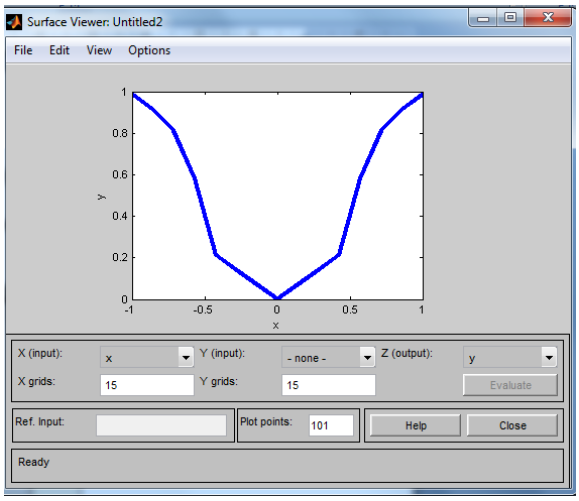


The number of y_i points is three. Choose as many triangular membership functions for output variable. Then make the base of the triangles narrow, so that they resemble singletons.



Make the basis of output triangular membership functions as narrow as possible to make them resemble singletons.

Observe again the position of singletons. They have nonzero value exactly at the output data point. Only three membership functions are needed at the output.
Next set the rules. Define the rule base that corresponds to the input-output data shown in previous Table. The result of our data fitting can be shown below.

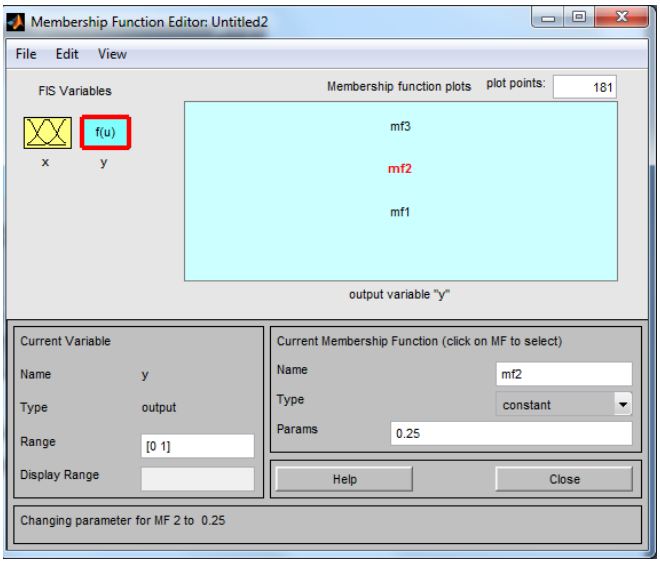


The fit is **exact** at the given data points. Overall shape is that of parabola, but to have a better fit, more data points would be needed.

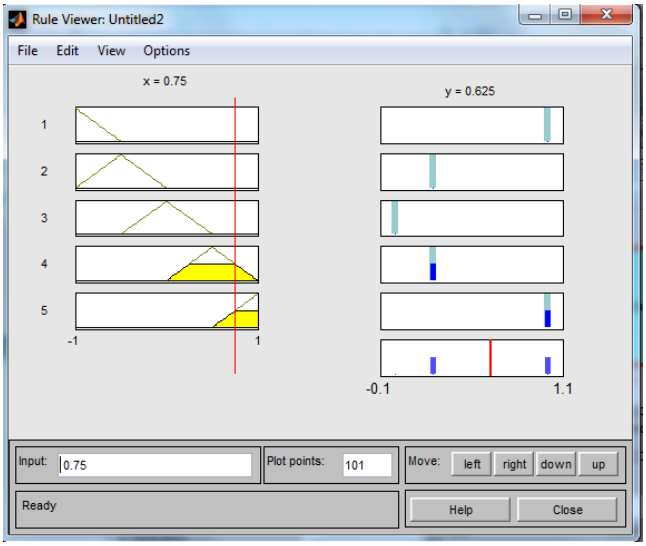
Sugeno (with constant outputs):

The same procedure as above can be produced with **Sugeno** reasoning. In Sugeno reasoning the consequence, the output side, is deterministic: If x is X_i then $y = y_i$

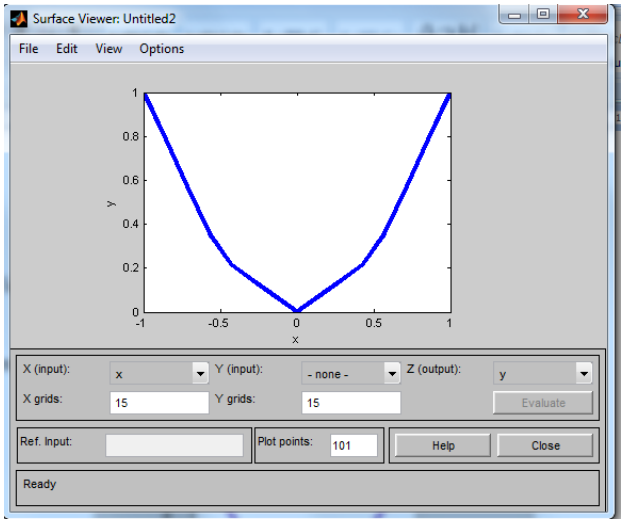
SOLUTION:
In the FIS editor choose ***File; New FIS; Sugeno***.
Then activate the input and name it *x*. Similarly activate output and name it *y*. Next *edit membership functions*. Let the input range be [-1 1]. Determine the input membership functions as before by *Add MF's* (5 triangular). Next repeat the same for output. The range is [0 1], which is default value. Then *Add MF's*. Choose three output membership functions type *constant*. Now you change the name and value of the constant.



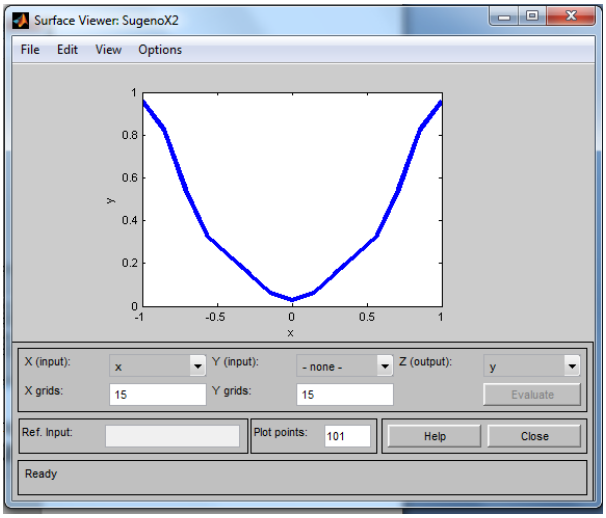
The final task is to form the rulebase. Choose *Edit rules* and write them in as before. The Sugeno FIS system is now complete. Let us view the result. First the overall rules:



Then the Surface View:



Trying *gaussmf* membership functions results in the following approximating system:

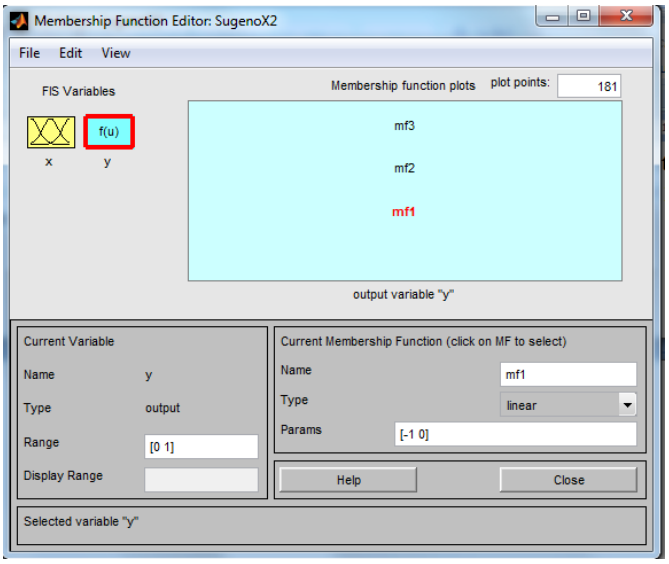


Sugeno (with linear outputs):

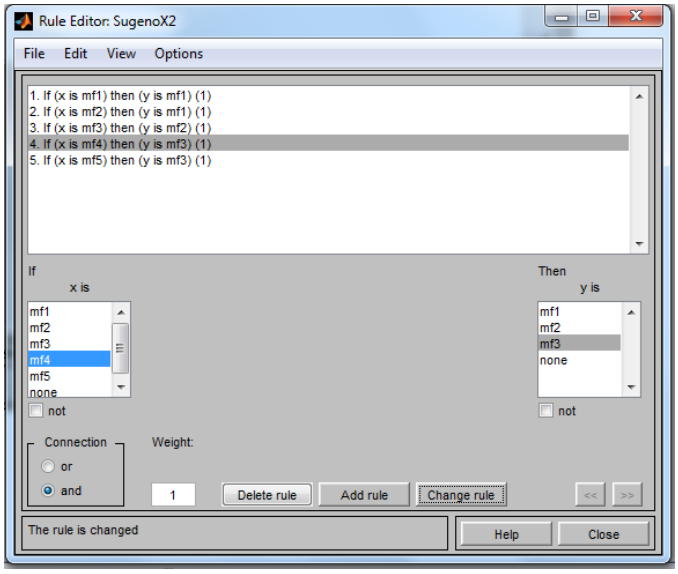
Consider again the example above. Set up a Sugeno system with five membership functions as before. Use three straight lines at the output: One with a negative slope, one constant, and one with a positive slope. Start with very simple ones:

1: $y = -x$
2 : $y = 0$
3 : $y = x$

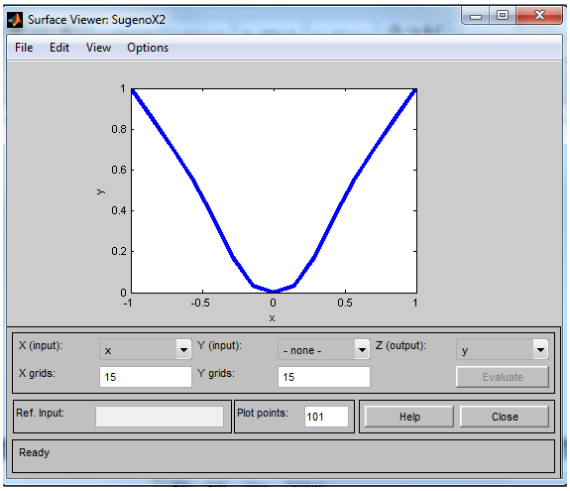
SOLUTION:
Define them by *Add MF's*. Choose *linear* type. The two parameters for each straight line can be chosen in *Params* box. The slope is first and then the constant.



The rule base could be e.g.:



The Surface View becomes:



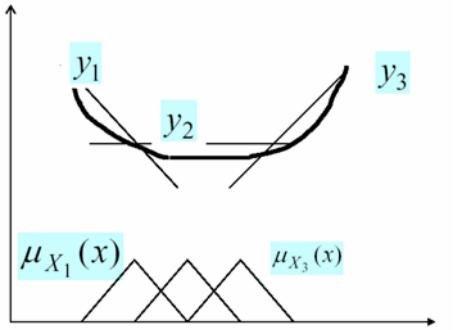
The result is smoother than before, but does not agree as well at point $x = 0.5$, $y = 0.25$. This would require further fine-tuning of parameters.

Remember that the rules are of the form: If x is X_i then $y = f_i(x)$. Combining results of all the rules leads to a weighted average:

$$y(x) = \frac{\sum_{i=1}^{m} \mu_{X_i}(x) f_i(x)}{\sum_{i=1}^{m} \mu_{X_i}(x)}$$

where m = number of rules.

The interpretation is that for a given value of x, the membership functions smooth (interpolate) the output function f_i . This is illustrated below for the case of straight lines y_1 , y_2 and y_3.



The straight lines y_1, y_2 y_3 represent the local models. The bold line is the final result.

REMARK: Sugeno systems using constants or linear functions in the consequence are clearly parameterized maps. Therefore it is possible to use optimization techniques to find best parameters to fit data instead of trying to do it heuristically.

Task 3 – Anfis: anfis editor with artificial data examples

Matlab console

Load the data in the workspace:

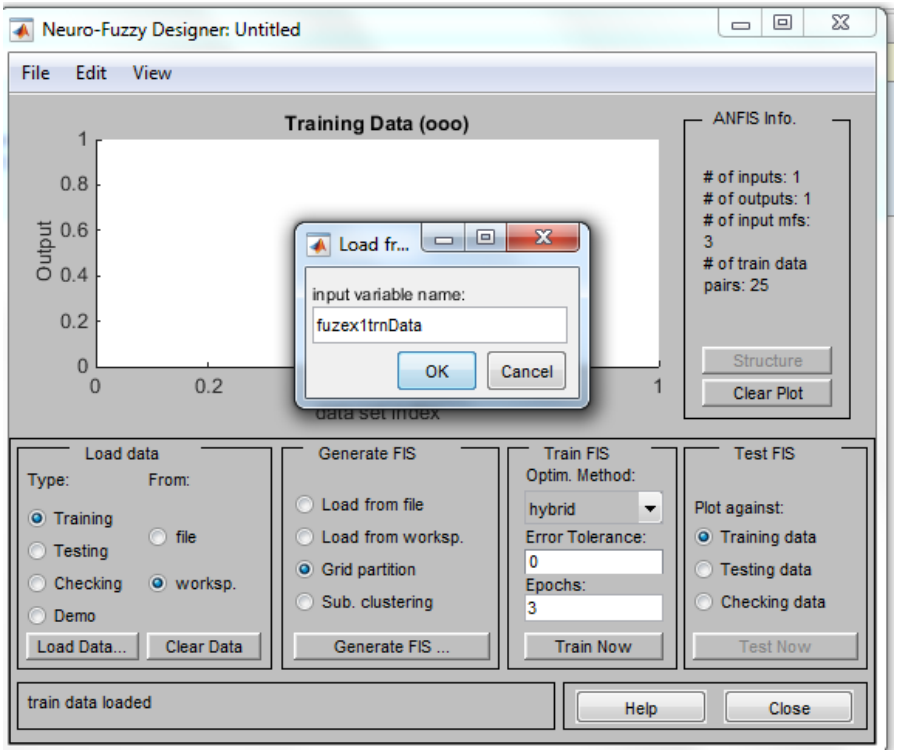
```
load fuzex1trnData.dat
load fuzex1chkData.dat
load fuzex2trnData.dat
load fuzex2chkData.dat
```

Open the Anfis Editor:

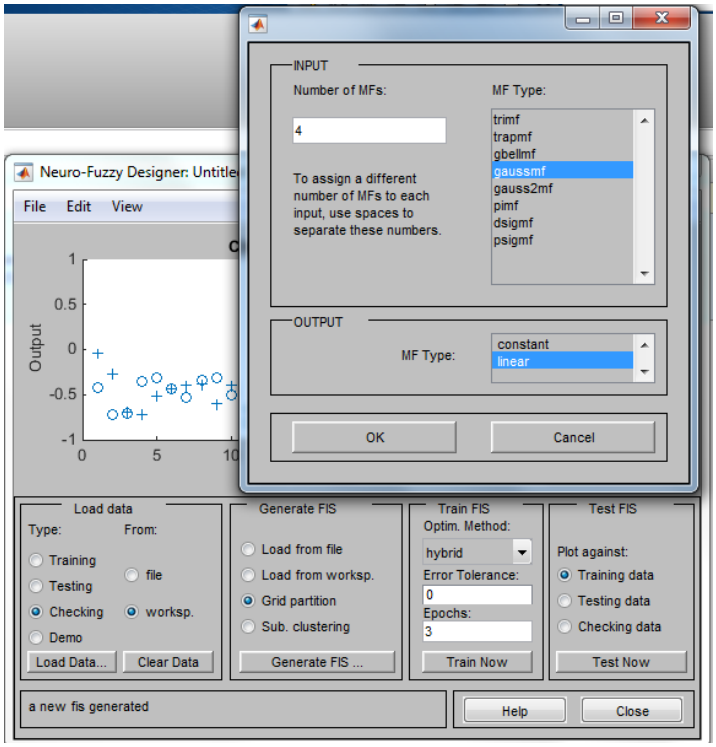
```
anfisedit
```

DATA SET 1

- Load Trainig and Checking data of data set 1, by selecting the option *worksp.* at the left hand side of the panel and writing the name of the data matrix. For the first data set *fuzex1trnData*, chose *Training* button as shown in the next figure and for the second data set *fuzex1chkData*, chose *Checking*.



- Generate FIS with a Grid partition of 4 Gaussian MF and linear output:



- See the rules before training (*view* menu bar, and then *Rules*) and see that the output is set to zero for all rules. *Edit* the *FIS properties*, and go to the output. See that the parameters are set to 0.
- Train using *hybrid* optimization method during 40 *epochs*, pushing the *Train Now* button. Anfis chooses the model parameters associated with the minimum checking error.
- See Training and Checking plots.
- *View* (menu bar) *Rules* and *Surface*.
- Test the inference results over training and checking data, pushing the *Test Now* button (right hand side).
- From *Edit* you can change membership functions, rules, etc.

DATA SET 2

- Load Trainig and Checking data of data set 2: *fuzex2trnData* and *fuzex2chkData*, respectively.
- Generate FIS grid partition, train FIS and test FIS, like before. Do you get a good model that characterize correctly the data?
- Increase the number of membership functions and see that the training is well characterized by the rules → almost one rule per data!!!
- The Checking data is still bad characterized.

Conclusion: the data is poor, and the checking is not represented in the training → more data is needed or at least more training data is needed

Task 4 – Anfis: Function $y = -2x-x^2$

Make fuzzy approximation of function $y = f(x) = -2x - x^2$, when $x \in [-10, 10]$. Use matlab console to invocate ANFIS functions. Compare fuzzy approximation and original function.

SOLUTION:
Generate 51 input-output pairs between x ∈ [-10, 10], and choose training and checking data sets:

```
numPts=51;
x=linspace(-10,10,numPts)';
y=-2*x-x.^2;
data=[x y];
trndata=data(1:2:numPts,:);
chkdata=data(2:2:numPts,:);
```

Set the number and type of membership functions:

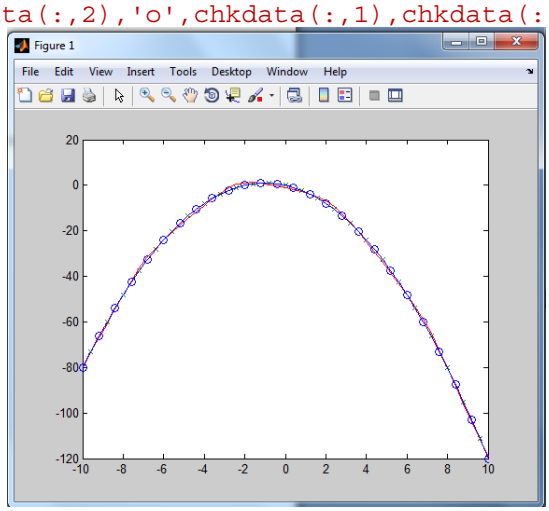
```
numMFs=5;
mfType='gbellmf';
```

Generate the FIS-matrix and execute the ANFIS-training by 40 rounds. *genfis1* generates a Sugeno-type FIS (Fuzzy Inference System) structure from data using grid partition and uses it as initial condition for anfis training. *anfis* is the training routine for Sugeno-type FIS. It uses a hybrid learning (least-squares + backpropagation gradient descent) algorithm to identify the parameters of Sugeno-type FIS.

```
fismat=(genfis1(trndata,numMFs,mfType))
numEpochs=40;
[fismat1,trnErr,ss,fismat2,chkErr]=anfis(trndata,fismat,numEpochs,NaN,chkdata);
```

Compare training and checking data to the fuzzy approximation. *evalfis* performs fuzzy inference calculations.

```
anfis_y=evalfis(x(:,1),fismat1);
plot(trndata(:,1),trndata(:,2),'o',chkdata(:,1),chkdata(:,2),'x',x,anfis_y,'-')
```



Draw also original function to the same picture so it is possible to compare function and fuzzy approximation:

```
hold on;
plot(x,y)
```

Save the Fuzzy Inference System developed:

```
writefis(fismat1,'fismat1')
```

Task 5 – Simulink + Anfis (Control System): $\ddot{x} = -0.6\dot{x} + f(x)$

Simulate the behavior of system: $\ddot{x} = -0.6\dot{x} + f(x)$

$f(x)$ is the same than in the previous exercise, i.e. $f(x) = -2x - x^2$.
Start by values: $x(0) = -0.2$ *and* $\dot{x}(0) = 0.5$

Replace $f(x)$ by the fuzzy approximation defined in task 4 and compare the results. Study what happens if the number of membership functions in the fuzzy system is increased. Test for example 5, 15 and 60 membership functions. See the risk in increasing the number of membership functions.

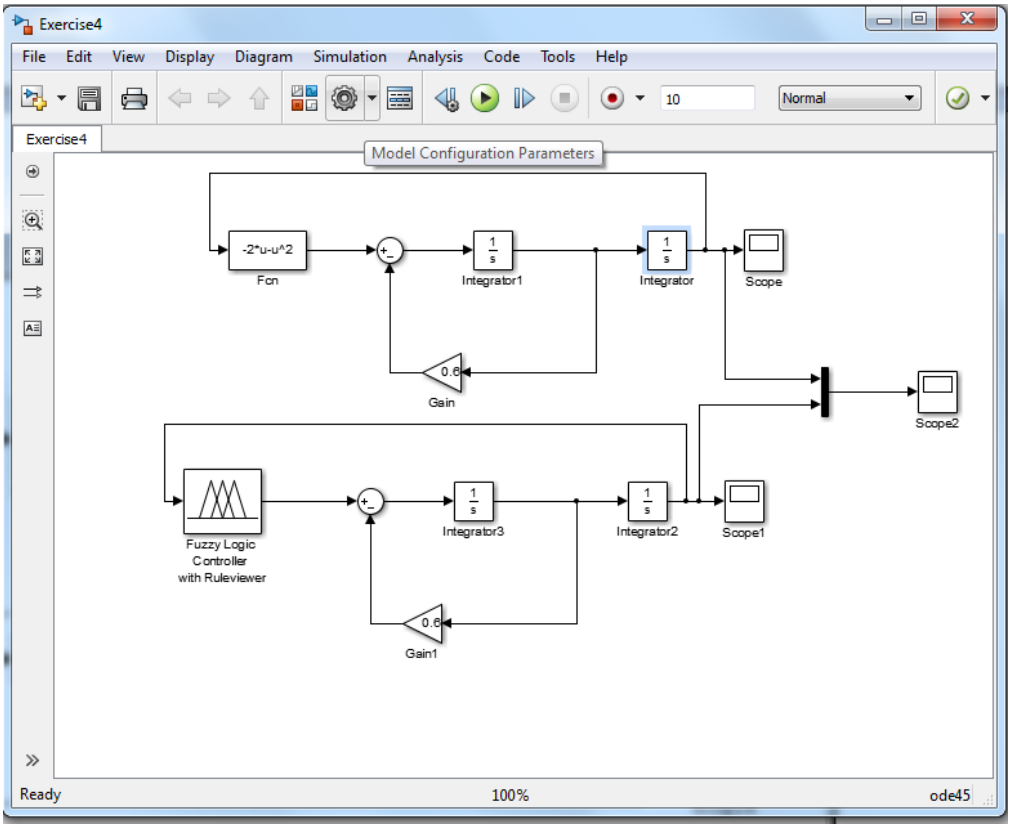
SOLUTION:
Open a Simulink model: New / Simulink Model

Then open the library browser: View / Library Browser

Develop the model described in next Figure. You can find integrators, gains and scopes in "Commonly Used Blocks". In this module you can also find the Sum. Once you take the sum and put it in the Simulink model window you have to click on it and change the second + for an – in order to obtain the sum/subtraction block.

To initialize the integrators you should click the corresponding figure and actualize with the desired value the Initial Condition. Idem for the Gains.

You can find the Fuzzy Logic Controller in the "Fuzzy Logic Toolbox" module. Chose the Fuzzy Logic Controller with Ruleviewer. The Function (Fcn) block is located at "User-Defined Functions" module.



If you do not have fismat1.fis in the workspace you can load it:

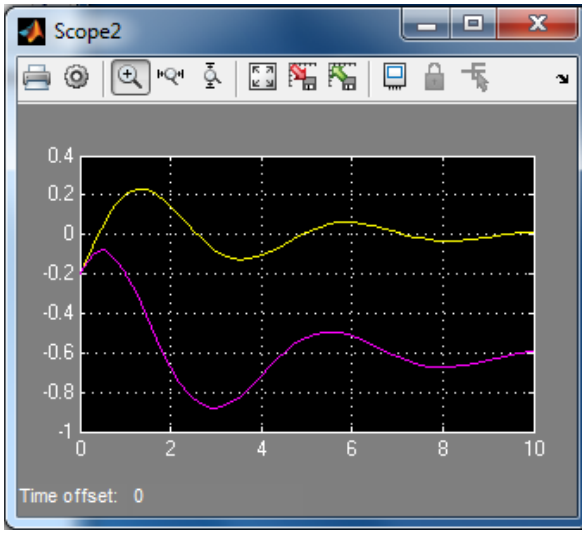
```
fismat1=readfis('fismat1')
```

Double click the Fuzzy Logic Controller block and write the name of the FIS matrix, i.e. fismat1.

Note that you have to choose Boolean logic off in SIMULINK, otherwise the fuzzy block doesn't work.
In order to choose Boolean logic of in simulink you should deactivate: From the Simulink screen/ Simulation / Model Configuration parameters / Optimization / Implement logic signals as Boolean data.

You need also to set the *Simulation/Model Configuration Parameters/Diagnostics/Connectivity/Mux blocks used to create bus signals* to *error*.

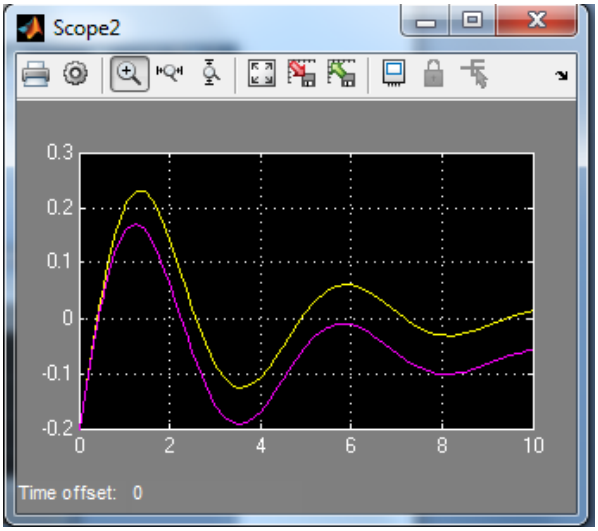
Remember to set the values $x(0) = -0.2$ and $\dot{x}(0) = 0.5$.
After that, it is possible to compare the results in Scope2 (or take the results in Matlab workspace). Result with 5 membership functions (yellow: function, pink: fuzzy):



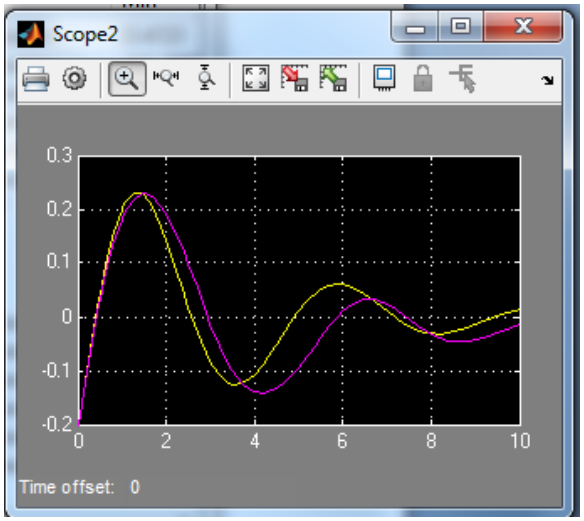
Increase the number of membership functions from 5 to 15. When changing the number, you have to do also new ANFIS-training:

```
numMFs=15;
mftype='gbellmf';
fismat=(genfis1(trndata,numMFs,mfType))
numEpochs=40;
[fismat1,trnErr,ss,fismat2,chkErr]=anfis(trndata,fismat,numEpochs,NaN,chk
data);
```

After that the difference between systems with exact function and fuzzy approximation is smaller:



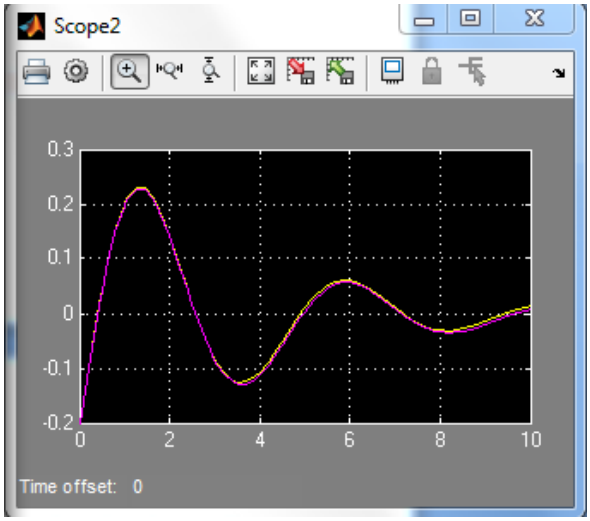
When we increase the number of membership functions from 15 to 60 the result might be better…



…but there is a danger that if too many membership functions are used, the system became overfitted!

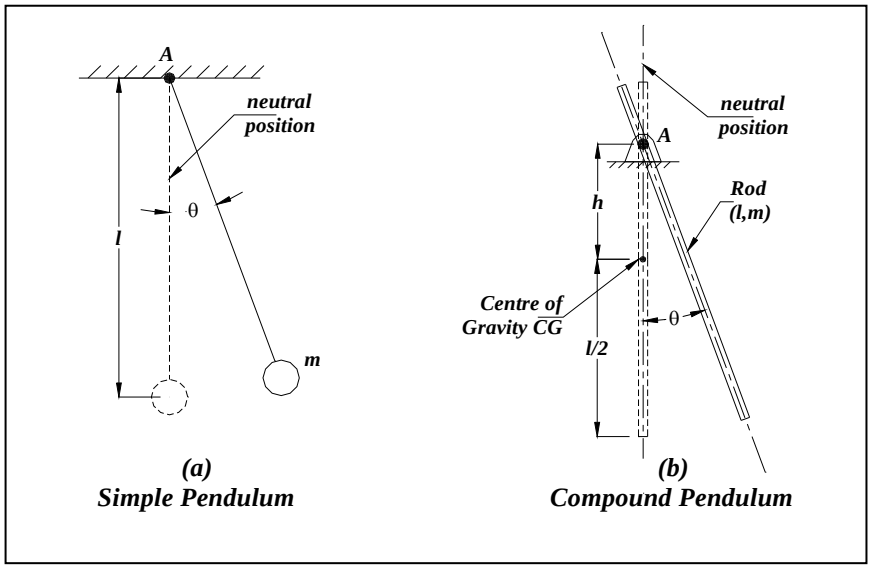
If a triangular membership function is now used, the fuzzy model is more precise and the final results are much better.

```
numMFs=10;
mfType='trimf';
fismat=(genfis1(trndata,numMFs,mfType))
numEpochs=40;
[fismat1,trnErr,ss,fismat2,chkErr]=anfis(trndata,fismat,numEpochs,NaN,chk
data);
```

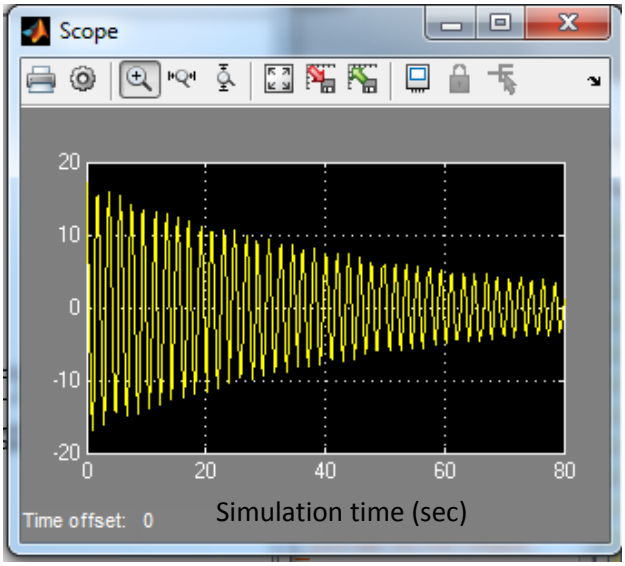


- Task 6 – Simulink + Mamdani (Control System): Compound Pendulum Angle

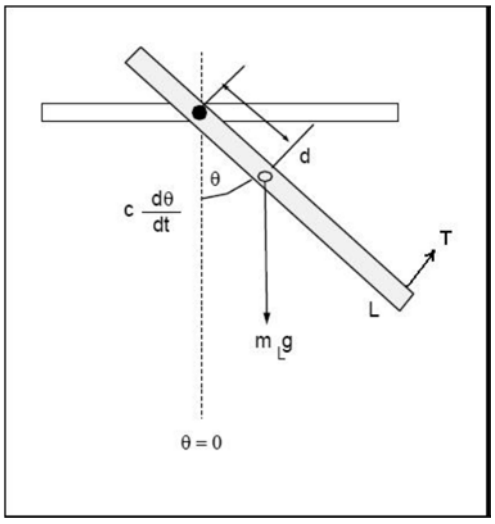
The compound pendulum is schematically shown in next figure, and it consists of a uniform slender bar of total mass m and length l, which may be suspended at various points A along the bar with the aid of a sliding pivot situated at any distance h from the centre of gravity of the pendulum. The centre of mass is at the middle of the rod.



Such a system has a natural oscillatory response. This is corroborated by the open-loop response to a 2 volt step input as seen in the figure below. The pendulum has a motor with a propeller.



You can see that even after 80 seconds have gone by, the pendulum is still oscillating more than +/- 2 degrees. This underscores the need for control methodologies to be implemented on the thrust tester to improve the system's transient response. Towards this, an encoder is mounted to the pivot point shaft to measure the angle subtended by the pendulum. Feeding this information back into the system allows for many different approaches to quickly stabilize the compound pendulum at a desired angle.



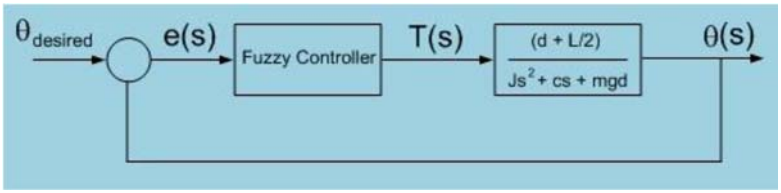
From the free body diagram above, it can be seen that evaluating Newton's second law about the pivot point of the pendulum yields the following equation:

$$J\ddot{\theta} + c\dot{\theta} + mgd \sin\theta = \left(d + \frac{L}{2}\right) T$$

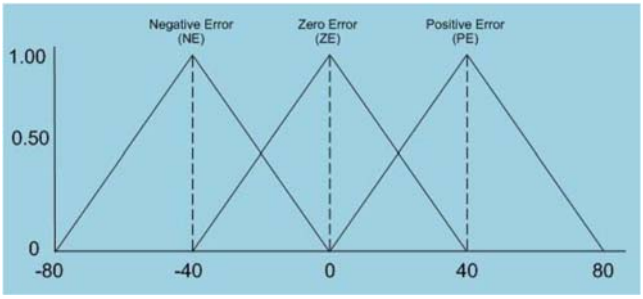
The open-loop transfer function of the plant can be found by taking the Laplace Transform of the second-order differential equation (and linearizing the sin(theta) term => sin(theta)=theta):

$$\frac{\theta(s)}{T(s)} = \frac{d + \frac{L}{2}}{Js^2 + cs + mgd}$$

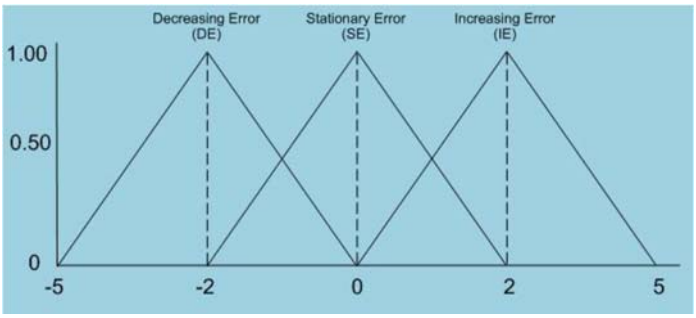
Therefore, the system's control block diagram is:



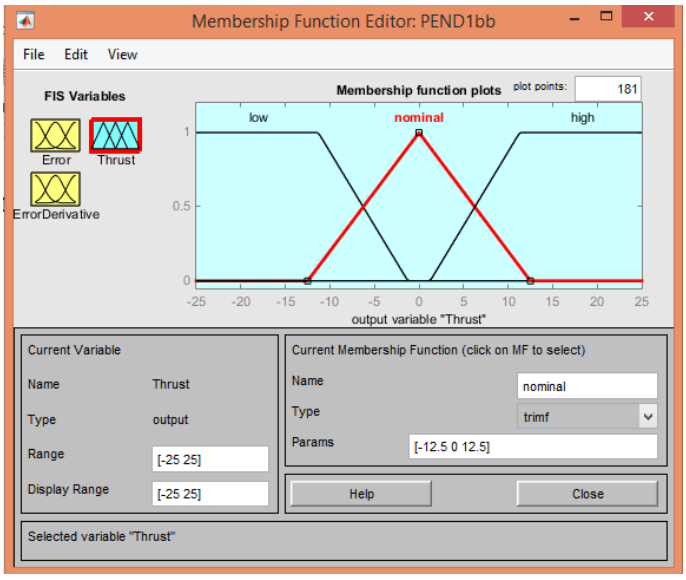
In order to design a Mamdani FIS you can use the information that follows:
For example, if the input for our thrust tester system is the subtended angle (measured by an encoder) and you are trying to stabilize this at 20 degrees, then a plot of the membership functions corresponding to the error would look like.



As an example, if the angle was measured at 40 degrees, then the error would be -20 degrees. This would make the membership values of negative error (NE) and zero error (ZE) both equal to 0.5. The positive error (PE) variable would be assigned a value of 0. To further increase the effectiveness of the controller, the derivative of the error should also be evaluated. Therefore, the system will be able to determine if the pendulum is moving further from or towards the desired angle at any instant (i.e. if the error is increasing, decreasing or remaining constant). Since the sampling time is on the order of milliseconds, the change in error will never be greater than a few degrees. As such, the bounds on the membership function is +/- 5 and is shown below.



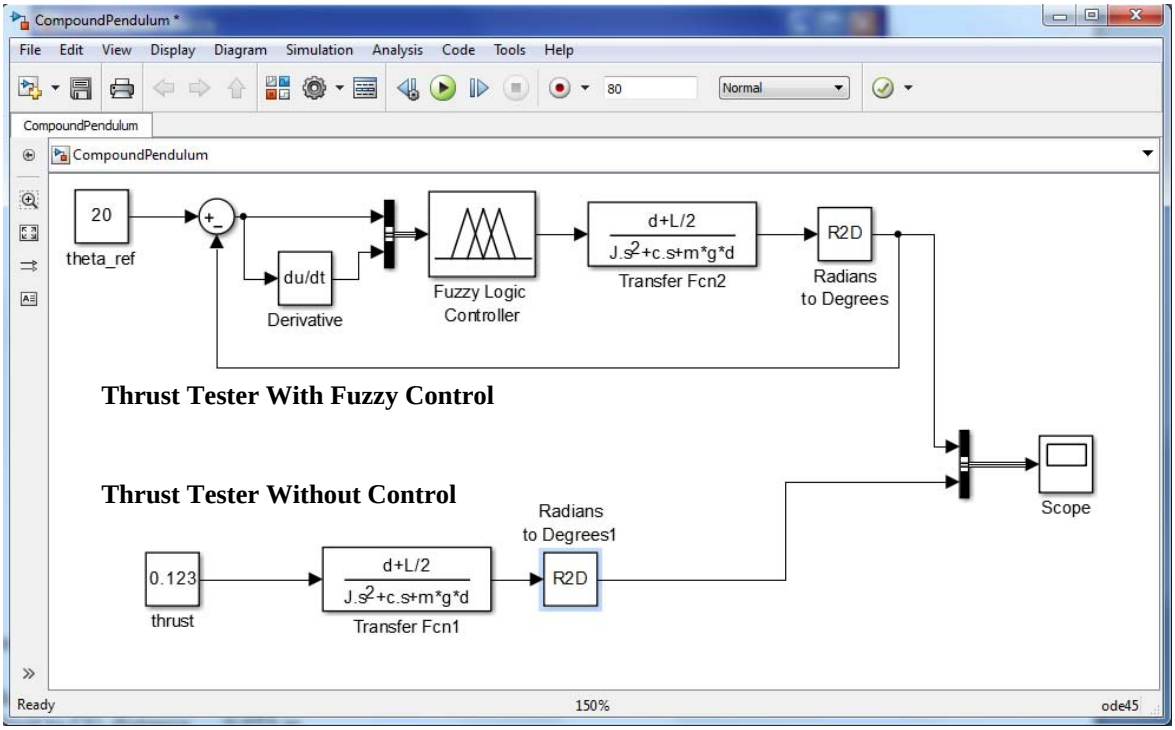
Similar membership functions can be defined for the output variable, thrust, using bounds between -25 and 25.



Below is the rule base used for the fuzzy controller:

1. IF (error is negative) AND (errorDer is decreasing) THEN (thrust is low)
2. IF (error is negative) AND (errorDer is stationary) THEN (thrust is low)
3. IF (error is negative) AND (errorDer is increasing) THEN (thrust is nominal)
4. IF (error is zero) AND (errorDer is decreasing) THEN (thrust is low)
5. IF (error is zero) AND (errorDer is stationary) THEN (thrust is nominal)
6. IF (error is zero) AND (errorDer is increasing) THEN (thrust is high)
7. IF (error is positive) AND (errorDer is decreasing) THEN (thrust is nominal)
8. IF (error is positive) AND (errorDer is stationary) THEN (thrust is high)
9. IF (error is positive) AND (errorDer is increasing) THEN (thrust is high)

Once you have available the Mamdani FIS, you can create the whole model using Simulink. The derivative and the transfer function blocks are in the "Continuous" module. Theta_ref and Thrust are constant blocks defined in the "Commonly Used Blocks". You can find R2D (Radians to Degrees) block in "SimulinkExtras/Transformations" module.



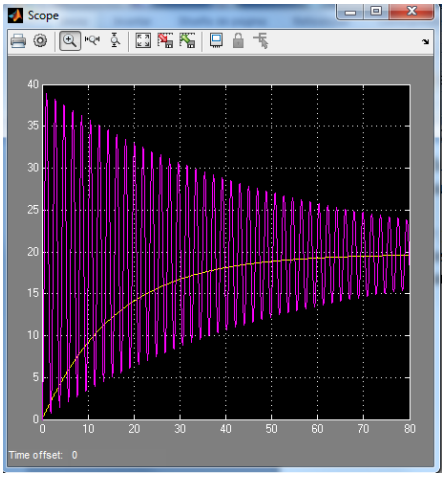
You can set, initially, the parameters to the following values:

L	Bar length	0.495 *m*
d	Pivot to CG distance	0.023 *m*
m	Mass of pendulum	0.43 *kg*
J	Moment of Inertia	0.0090 *kgm*2
c	Viscous damping	0.00035 *Nms* / *rad*
g	Earth's Gravity 9.80665 m/s^2	

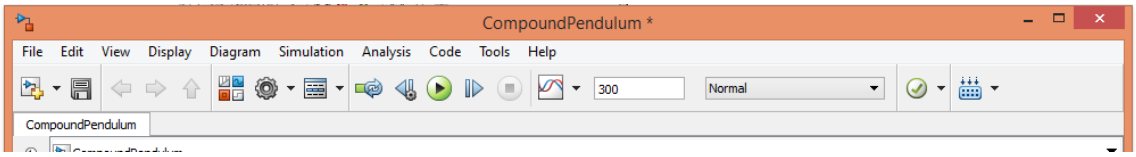
To do so, you just have to define them in the workspace.

In the *simulation/configuration parameters/Solver* set the *Zero-crossing control* to *disable all* and in the *simulation/configuration parameters/diagnostics* set the *Automatic solver parameter selection* to *none*.

The plot shows the 2 Volt step response, that correspond to a torque value of 0.123, of the open loop system (pink) and the response under fuzzy control (yellow). In both cases, the system stabilizes at around 20 degrees. However, the fuzzy controller has a much quicker settling time with no overshoot.



Select a simulation stop time of 300 and run it again.



You will see that around a time of 250 the open loop signal has no more overshoot.