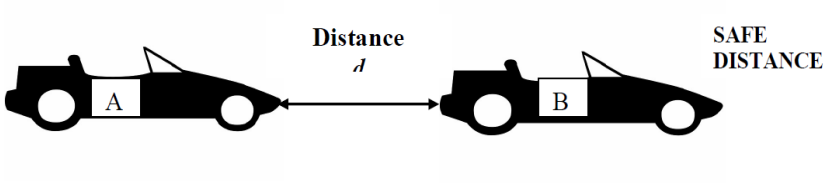
FUZZY LOGIC SYSTEMS LAB CLASS GUIDE

Task 1 – Build a Mamdani FIS: 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.

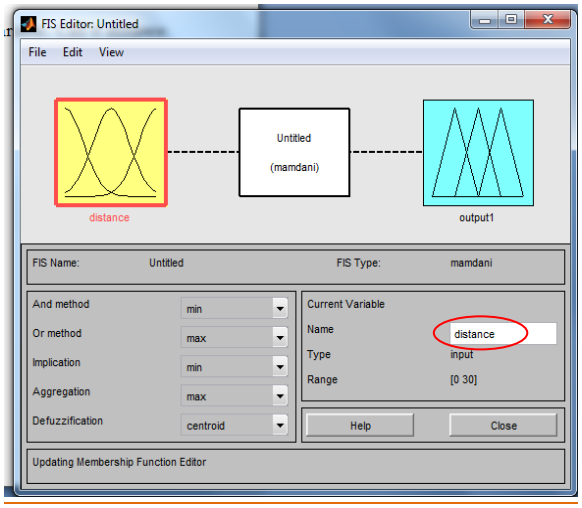
Membership functions for INPUT: Distance *d* (meters): **short, medium, high**.

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

We will use the fuzzy toolbox of Matlab to define the Mamdani fuzzy system.

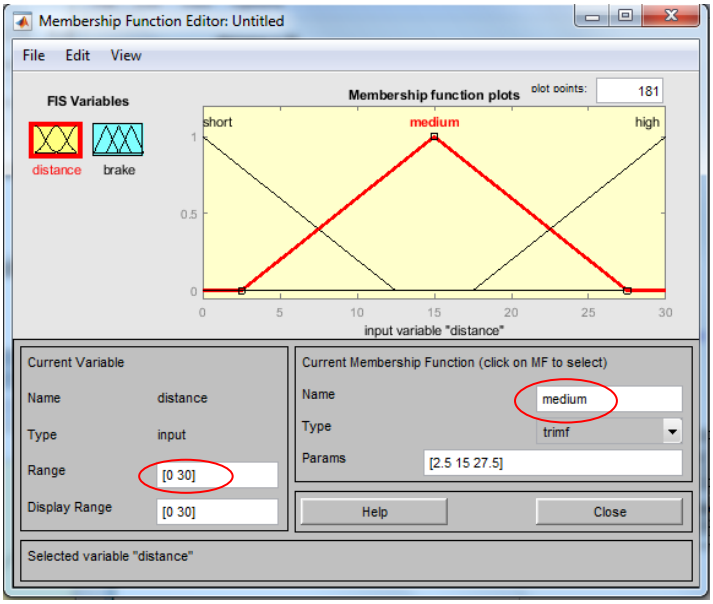
Inside MATLAB type
» fuzzy

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*.



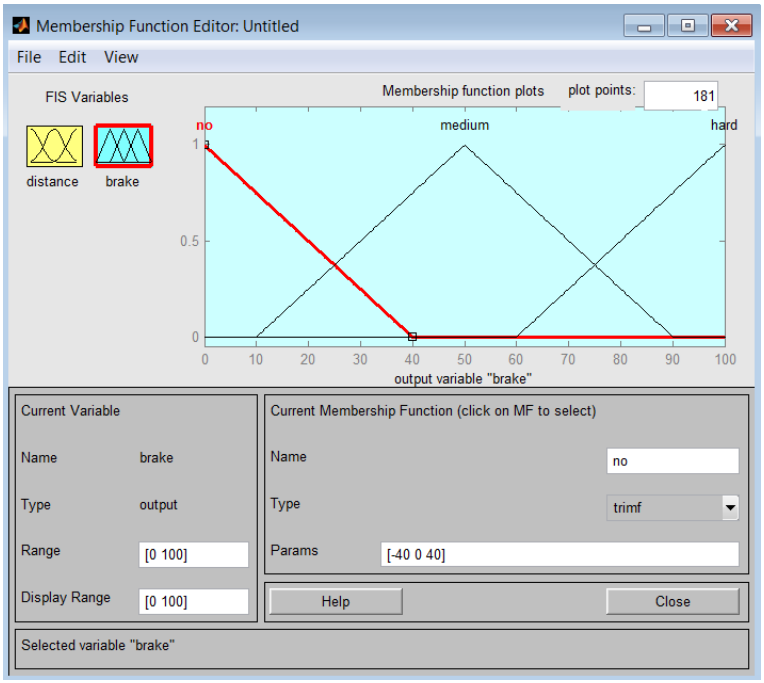
Next, click the input block twice with the mouse to open the membership function window. First define the range, say from 0 to 30 m.

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



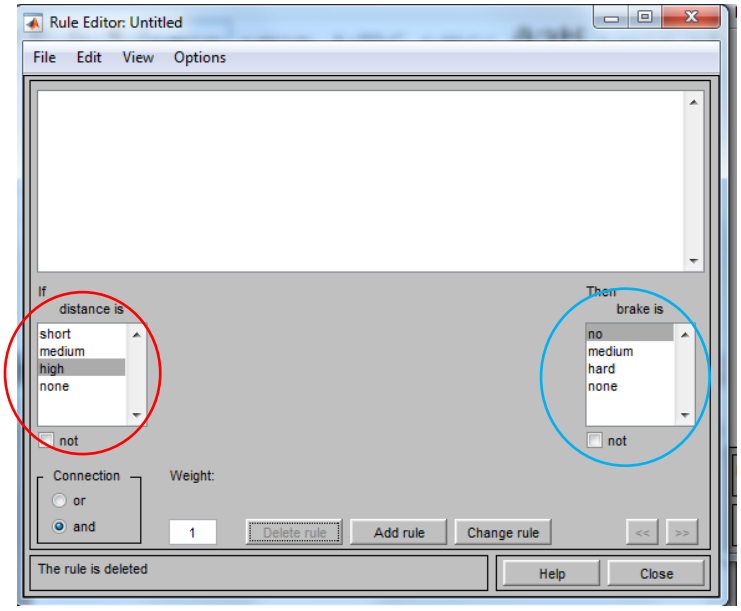
In the previous figure the membership function of the middle has been activated and renamed as *medium*.

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>

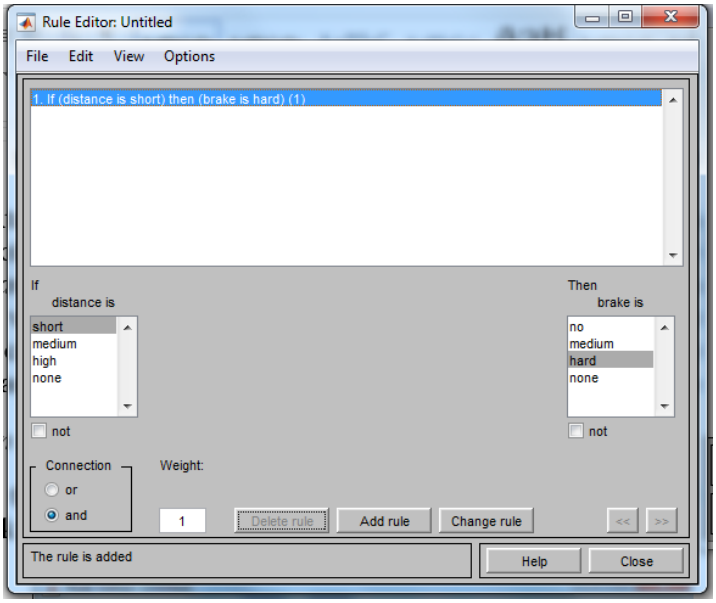
What is missing from the fuzzy system now is the rule base. Open *Edit* menu and click *Rules*. Then the following display opens.



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 short, then brake is hard

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



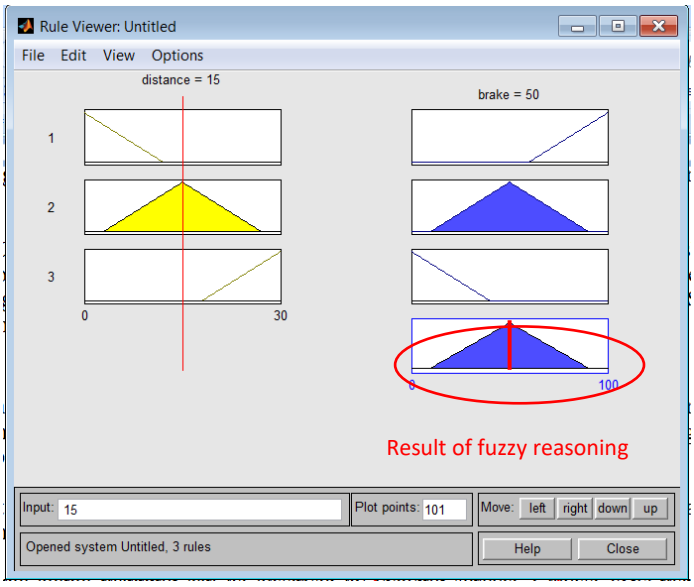
Let us set two other rules, one for *medium* distance and the other for *high* distance.

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 to express the rule base.

Check under *Options* and 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.*



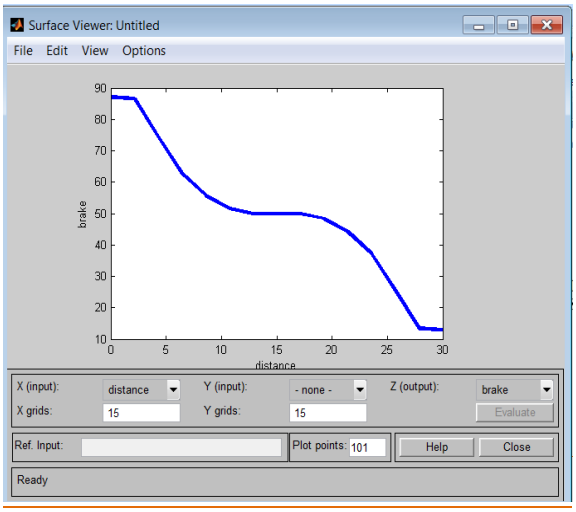
c. <u>Use fuzzy reasoning to check the operability of the rule base</u>

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%.

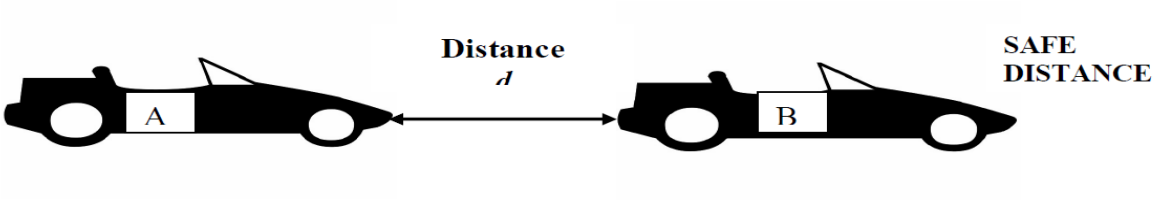
Different input values can be tried by moving the red, vertical line on the left-hand side. Changing the input value results in different output values.

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:



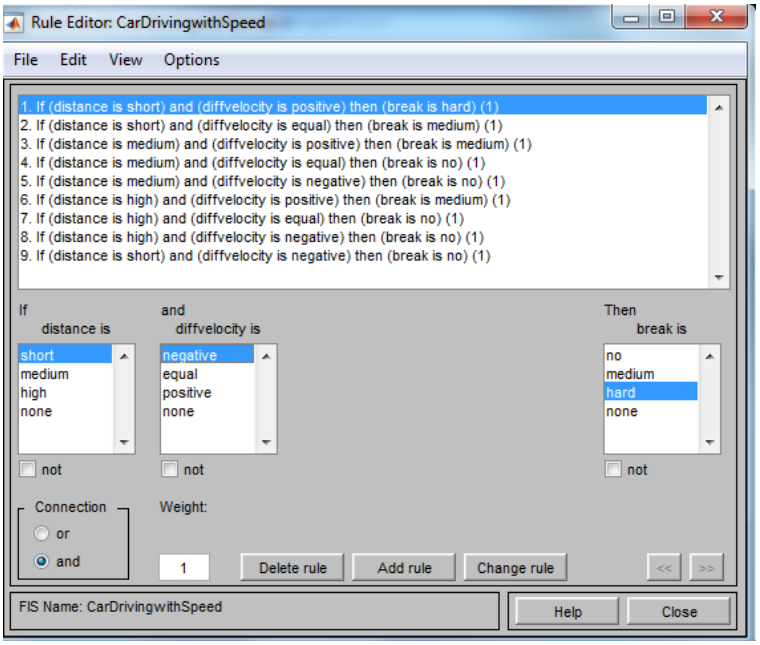
What is missing ?



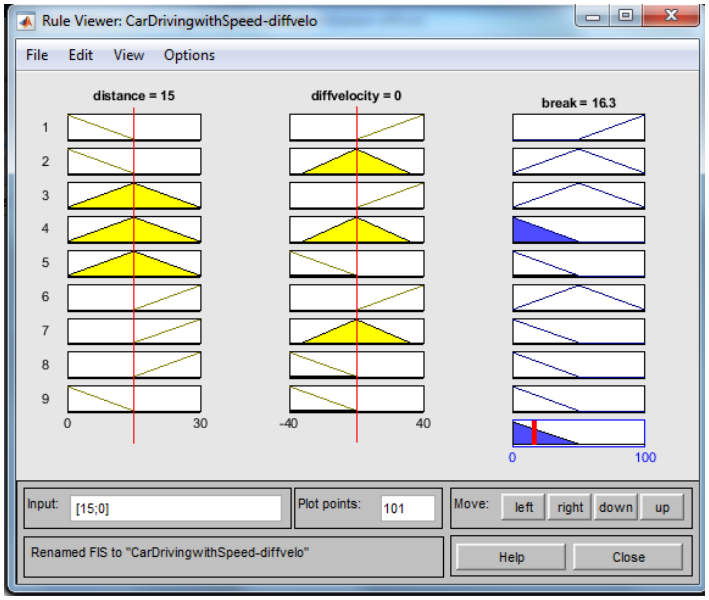
SPEED!
(relative speed between vehicles A and B).

SOLUTION:
Set up a new rule base with two inputs, *distance* and *diffvelocity*, and one output, *braking power*.
The second input can be added by opening *Edit* menu *Add Variable*. The rest of the steps are as before. Remember to give a name for the added input. Call it *diffvelocity*, meaning the difference of velocity to respect the current velocity of the vehicle. Set the range from -40 to 40 km/h and divide the range into three membership functions: *negative*, *equal*, and *positive* speed.

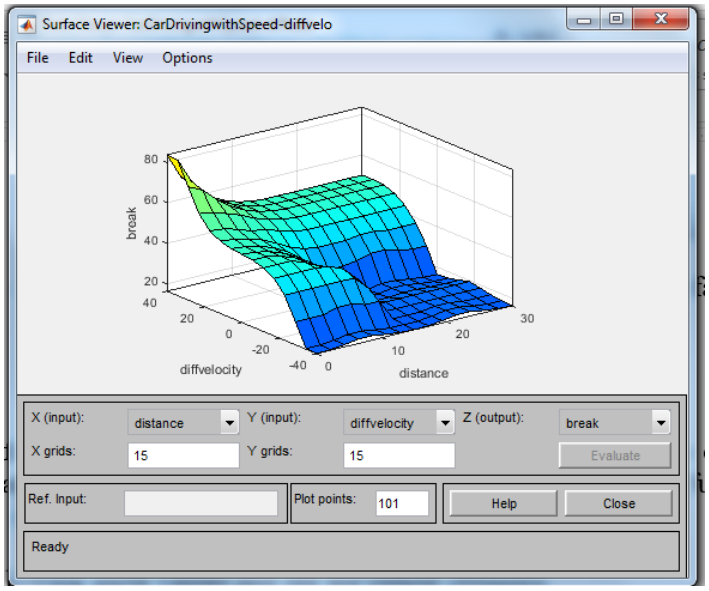
Delete the previous rules and introduce the nine rules shown in next figure, which are easy to understand. Of course, the rules are not unique. Other rules could be used as well. The rule base must be viewed and tested to see its effectiveness and how well the system specifications are satisfied.



Viewing the rules is shown in next figure.



Again, you can test the system by choosing different input values. Finally, the surface view of the rules is shown:

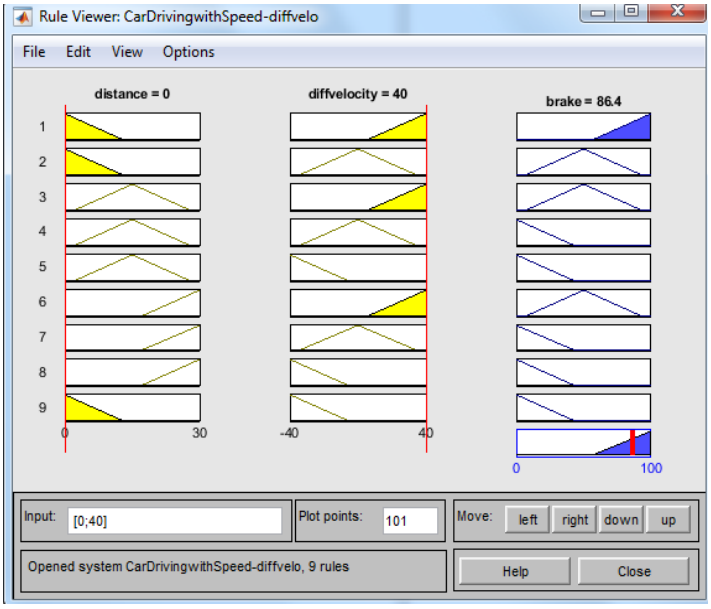


If the result is not satisfactory, then refine the system. Note that the right hand-side corner is flat with value zero over a fairly large area. You can change that by introducing a new membership function **little** for braking power. In that case you should also have to change the rule base.

Let's now chose the following input values and see the output obtained:

a) distance = 30 and diffvelocity = -40
b) distance = 0 and diffvelocity = 40

In both cases, the inferred output (break) value is not the minimum (0), neither the maximum (100), as we would expect for the inputs a) and b), respectively. Discuss with your neighbor why do you think this happens.



Change the defuzzification parameter from *centroid* to *mom* (mean of maximum) and take a look to the results now. So, the defuzzification parameter is also relevant in a fuzzy system!

Task 2 – Build a Sugeno FIS: Function $y = x^2$

Sugeno (with constant outputs):

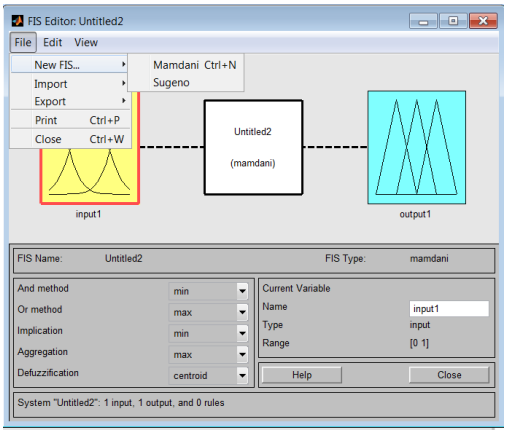
Consider input-output data given in the next table. The data is obtained from the function $y = x^2$. You are going to build a Sugeno model that fits this set of data.

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

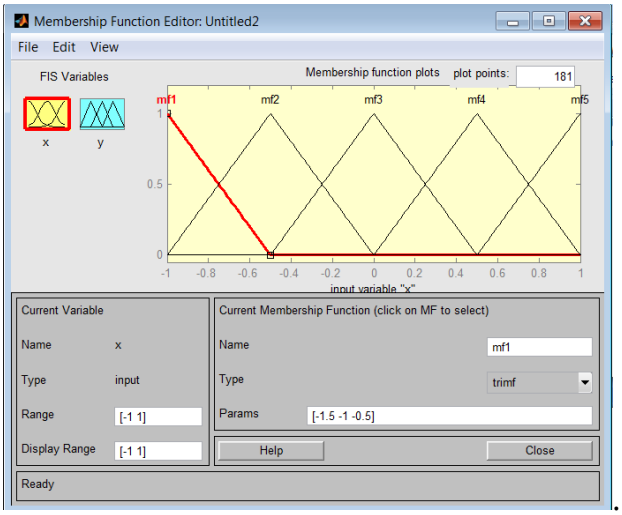
SOLUTION:

The data can be represented using Sugeno reasoning with e.g. triangular membership functions at the input and singletons at the output, positioned at y_i. Remember that in Sugeno reasoning the consequent, the output side, is deterministic: If x is X_i then $y = y_i$

In the FIS editor choose ***File; New FIS; Sugeno***.

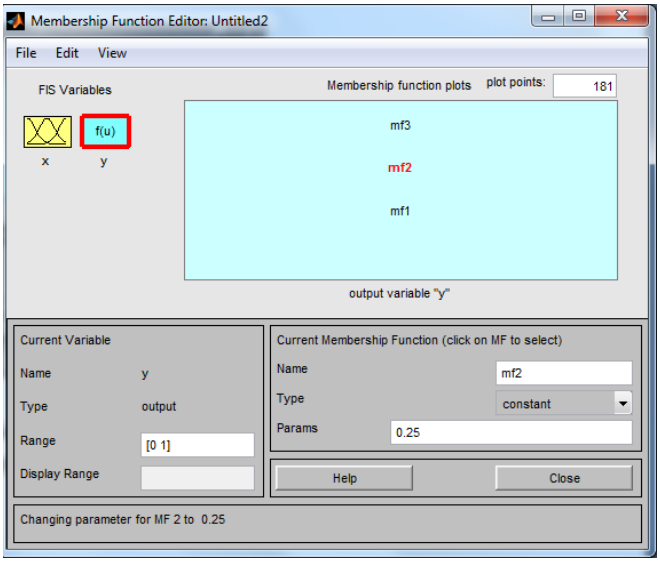


Then, activate the input and name it *x*. Similarly, activate the output and name it *y*. Next, *edit membership functions.* Let the *x* input range be [-1 1]. Determine the input membership functions as before by *Add MF's* (5 triangular). Note: It is easier to delete all 3 predefined memberships that appear as default and create from scratch 5 membership functions using the option "Edit / Add MFs".

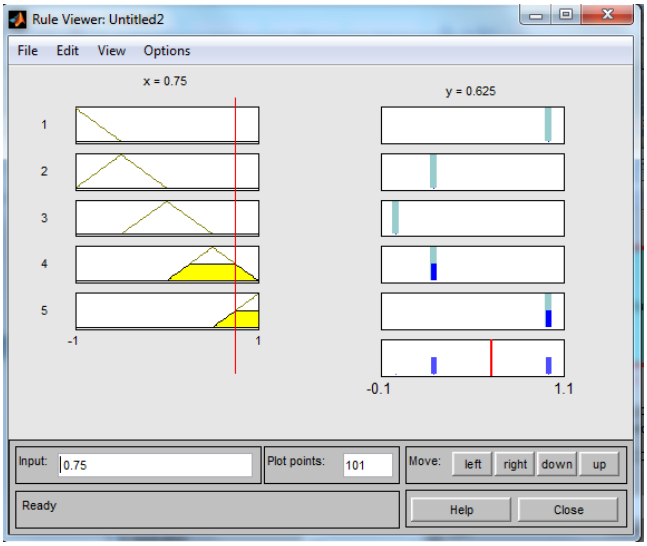


The number of x_i points is five. For each data point generate a triangular membership function, the maximum of which occurs exactly at a given data point, say *mf3* has maximum at $x = 0$, which is one of the data points. This is seen in the above figure.

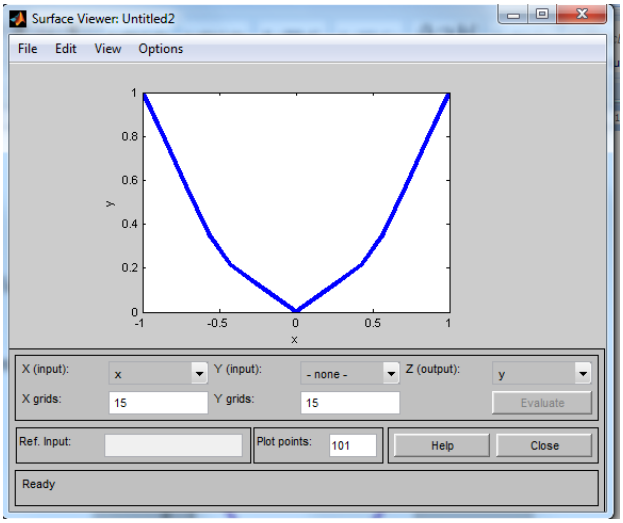
Next, repeat the same for the output. The range is [0 1], which is the default value. Then *Add MF's*. Choose three output membership functions type *constant*. Now you change the value of the constants to 1, 0.25 and 0, and the name of the membership functions if you like.



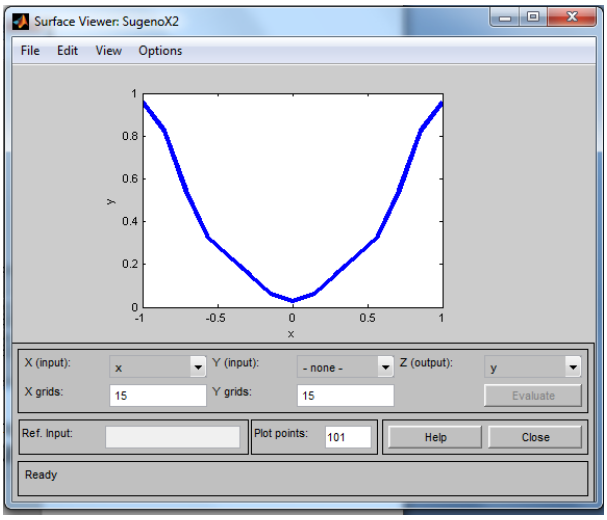
The final task is to form the rulebase. Choose *Edit rules* and write them in as before. The Sugeno FIS system is now complete. Let us view the result. First the overall rules:



Then the Surface View:



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



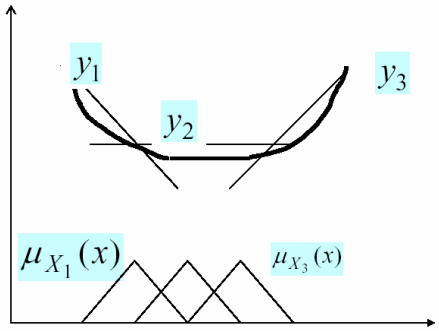
The result is smoother than before, but does not agree as well at point $x = 0.5$, $y = 0.25$.

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 consequent 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 – Play with Anfis: anfis editor with artificial data examples

Matlab console

Load the data in the workspace:

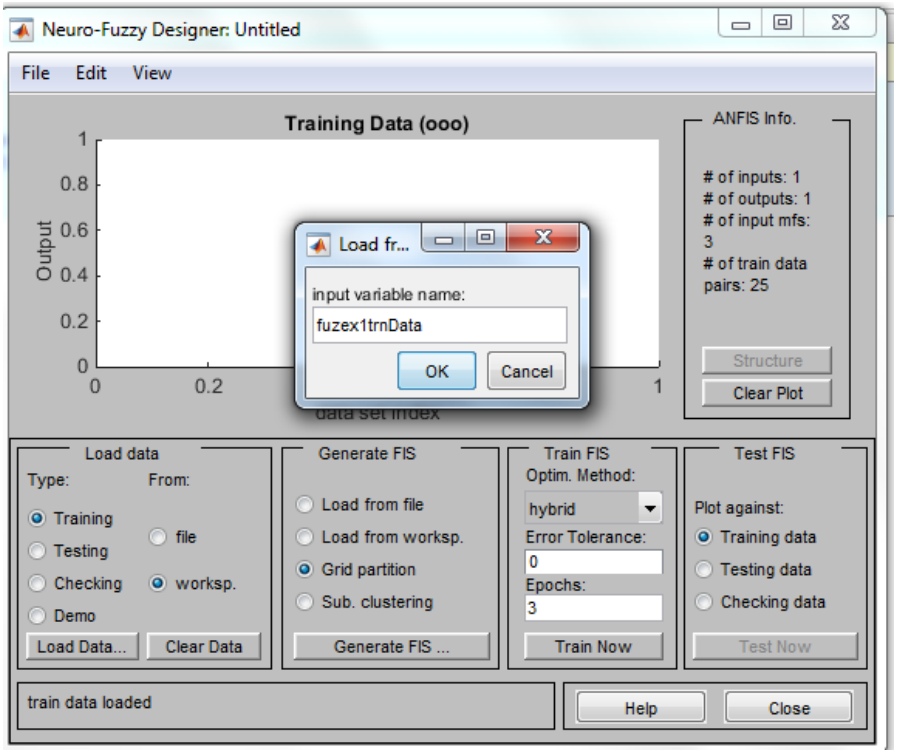
```
load fuzex1trnData.dat
load fuzex1chkData.dat
load fuzex2trnData.dat
load fuzex2chkData.dat
```

Open the Anfis Editor:

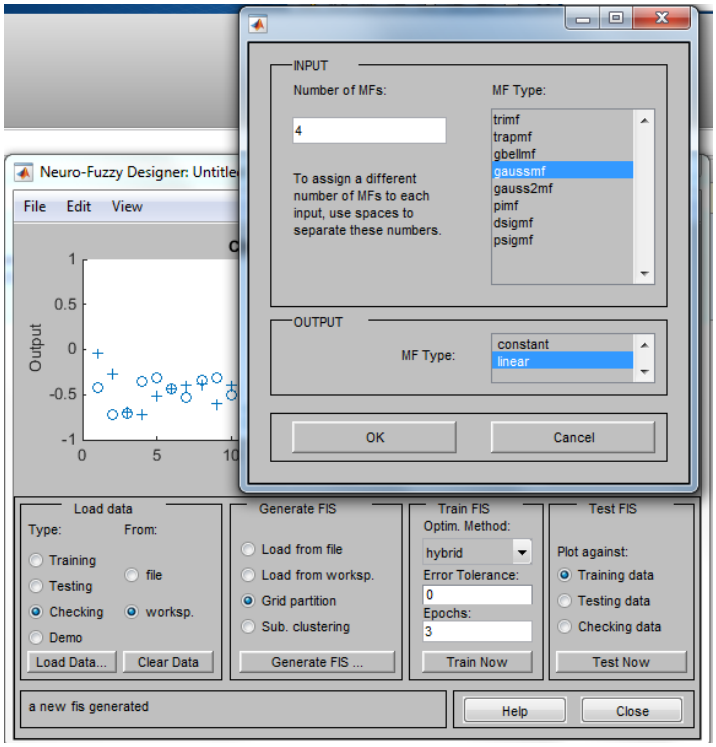
```
anfisedit
```

DATA SET 1

- Load Training 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: overfitting!!!
- 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 – Build an Anfis model: 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 ϵ [-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,:);
```

Take a look to the training output:
```
plot(trndata(:,1),trndata(:,2),'*r')
```

and the cheking output:
```
hold on
plot(chkdata(:,1),chkdata(:,2),'*b')
```

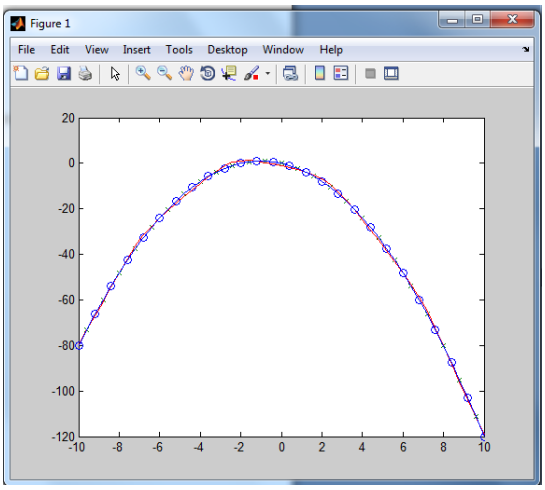
Set the number and type of membership functions:
```
numMFs=5;
mfType='gbellmf';
```

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 + gradient descent) algorithm to identify the parameters of Sugeno-type FIS.

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

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

```
anfis_y=evalfis(fismat1,x(:,1));
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 – Develop a Simulink + Anfis Control System: $\ddot{x} = -0.6\dot{x} + f(x)$

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

where • (one dot) stands for the first derivative and •• (two dots) stands for the second derivative

$f(x)$ 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.**

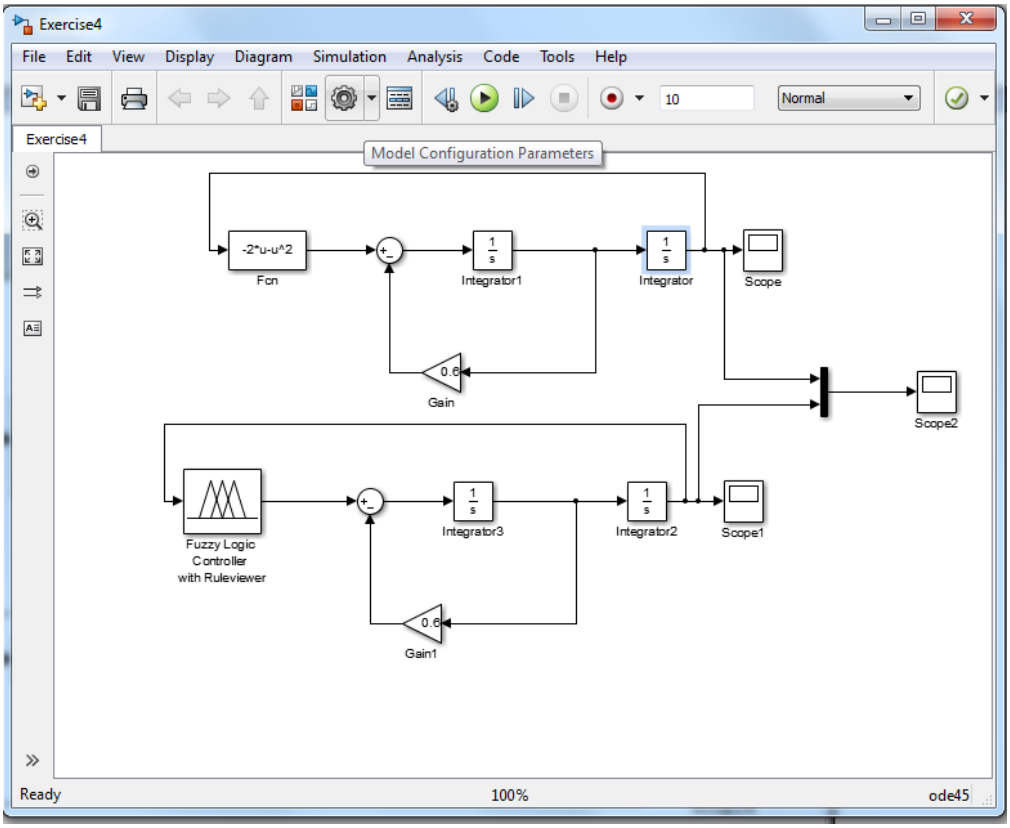
SOLUTION:
Open a Simulink model: *New / Simulink / Blank Model*

Then open the library browser: *Simulation / 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 "Simulink / User-Defined Functions" module.



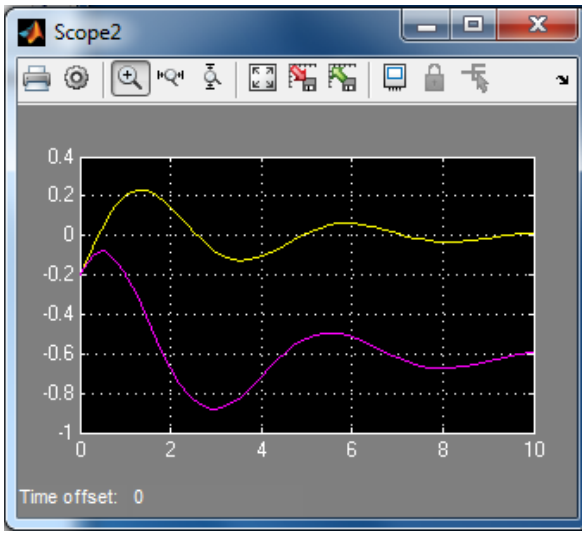
If you do not have fismat1.fis in the workspace you can load it:

```
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 off in simulink you should do the following: from the Simulink screen go to *Simulation / Model Settings* then search "*implement logic signals as Boolean*" and then deactivate it.

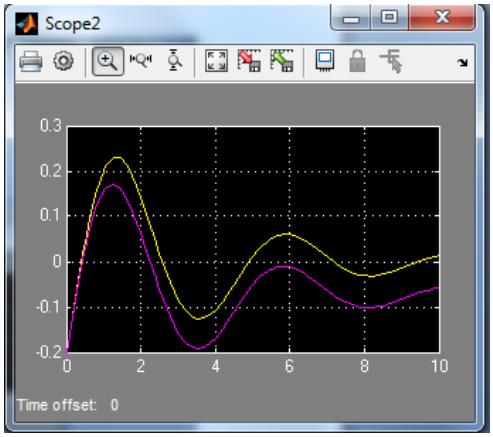
Remember to set the values $x(0) = -0.2$ and $\dot{x}(0) = 0.5$.
After that, it is possible to compare the results in Scope2 (or take the results in Matlab workspace). The result with 5 membership functions (yellow: function, pink: fuzzy) are shown in the next figure.



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 system with exact function and fuzzy approximation is smaller:



When we increase the number of membership functions the result might be a little 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=5;
mfType='trimf';
fismat=(genfis1(trndata,numMFs,mfType))
numEpochs=40;
[fismat1,trnErr,ss,fismat2,chkErr]=anfis(trndata,fismat,numEpochs,NaN,chk
data);
```