2.5. Sugeno-style fuzzy inference

The result of Sugeno reasoning is an exact number.

Consider **input-output** **data** given in the table.

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

The data is from the function $y = x^2$. 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.

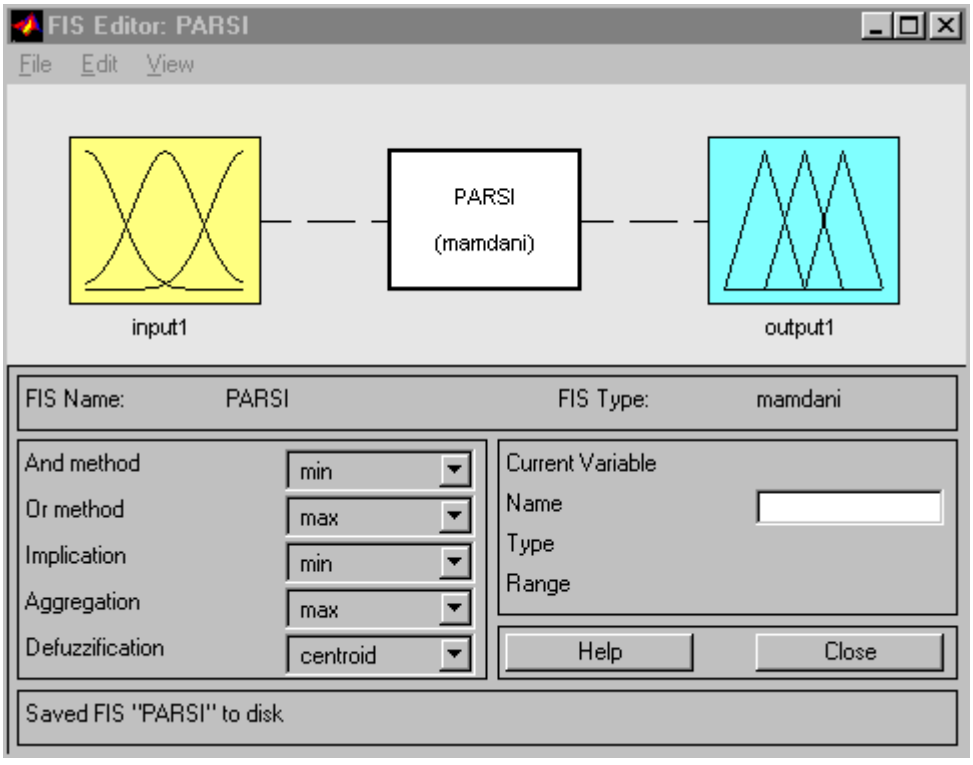

Fig. 2.30. Default display of FIS Editor with Mamdani reasoning.

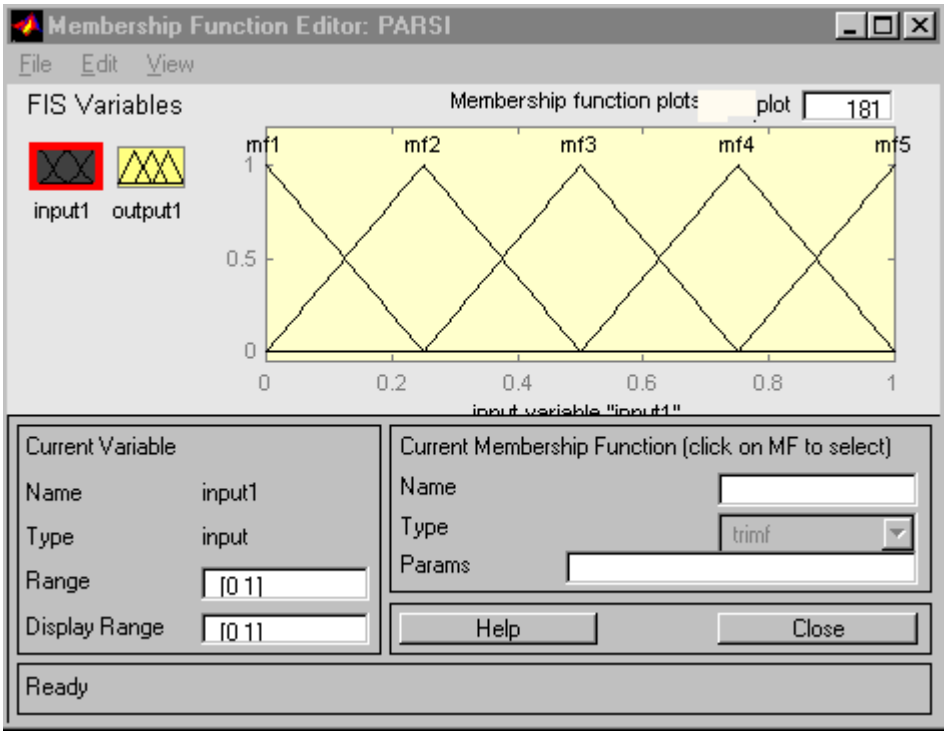

Fig. 2.31. The number of x_i points is five. Choose as many triangular membership functions for input variable.

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.

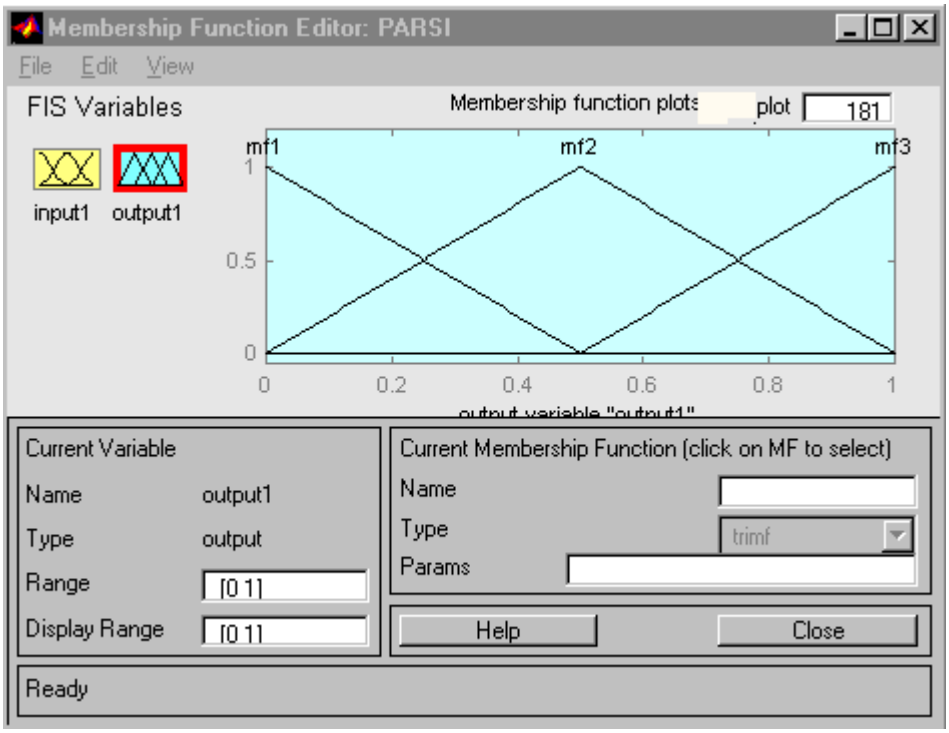

Fig. 2.32. The number of y_i points three. Choose as many triangular membership functions for output variable.

Then make the base of the triangles narrow, so that they resemble singletons.

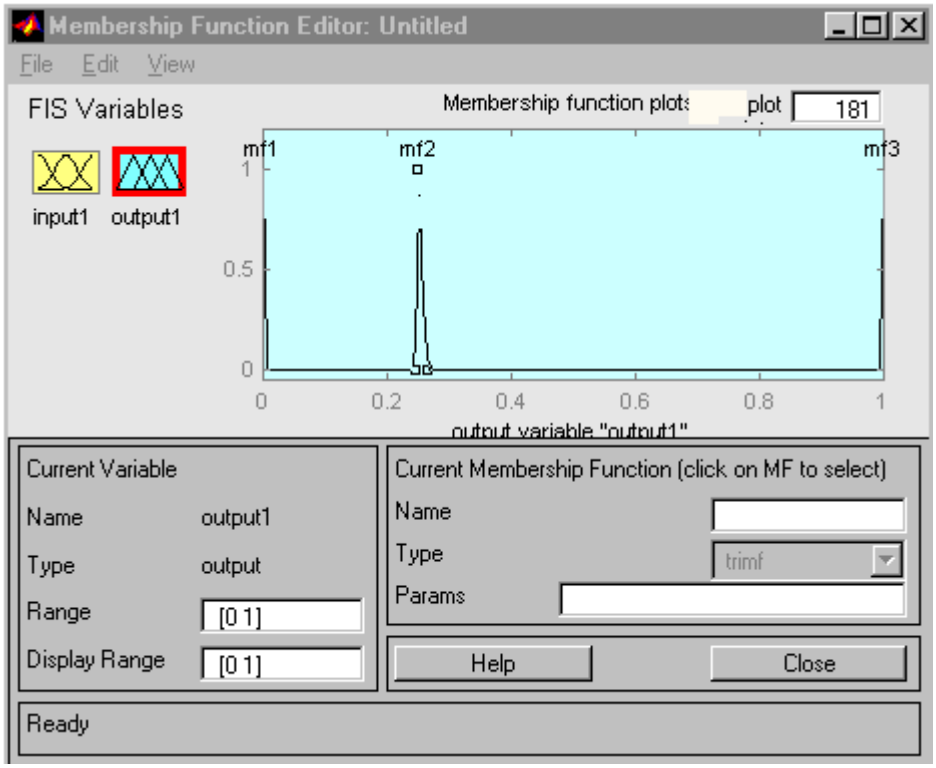

Fig. 2.32. Make the basis of output triangular membership functions as narrow as possible to make them resemble singletons.

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. Why?

Next set the rules to be

If x is mf_i then y is omf_i.

Complete rule base for the example is shown below.

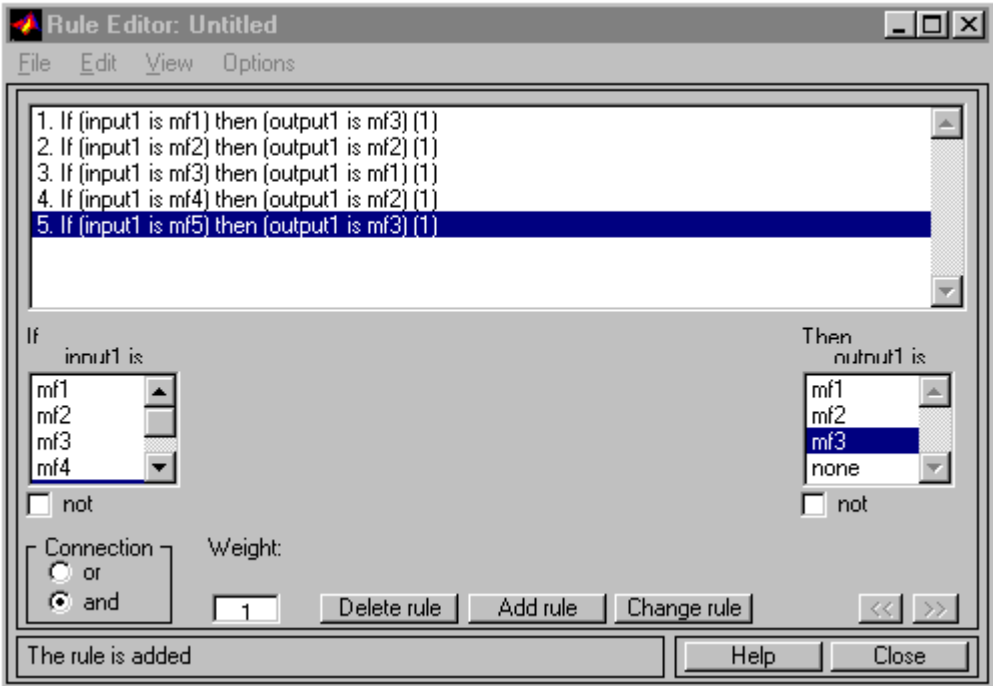

Fig. 2.32. The rule base corresponding to the input-output data shown in the Table.

We can view the rules by *View rules*

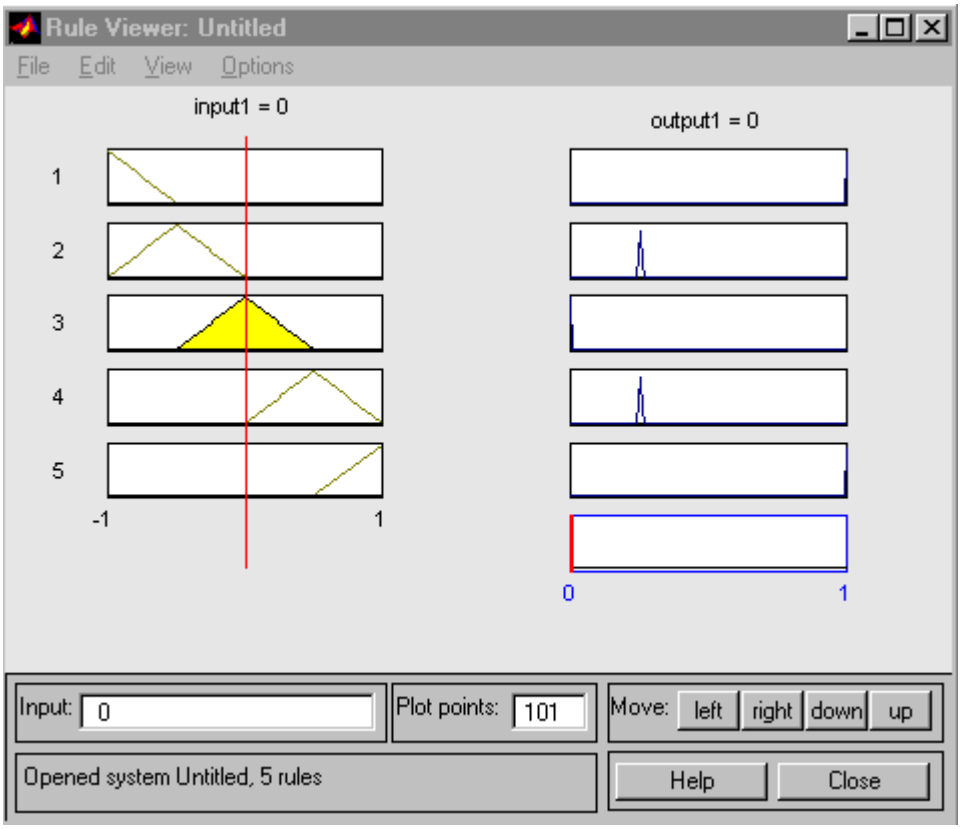

Fig. 2.33. The complete rule base of the example.

The result of our data fitting can be shown below

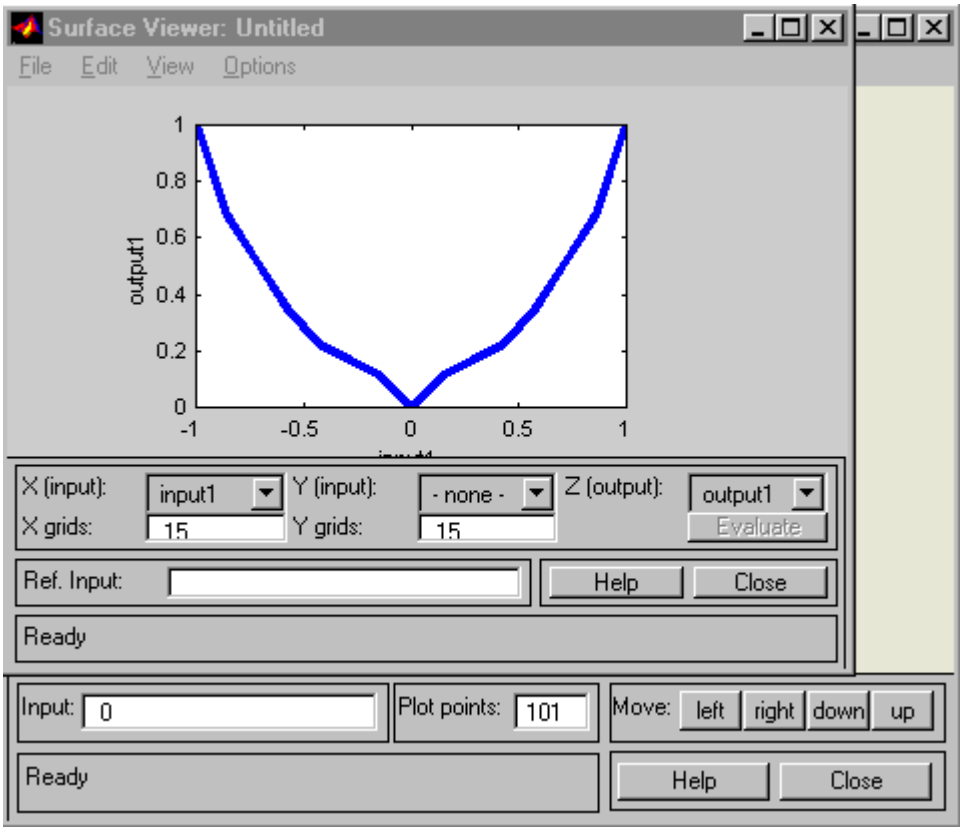

Fig. 2.34. The resulting fuzzy system approximating the given data.

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.

EXERCISE: Keep the range the same. Add four more data points. Repeat the procedure and compare the results.

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 $y = y_i$.

In Mamdani reasoning the determinism is produced with singletons.

Let us repeat the above example with Sugeno reasoning.
In the FIS editor choose ***New Sugeno FIS***.

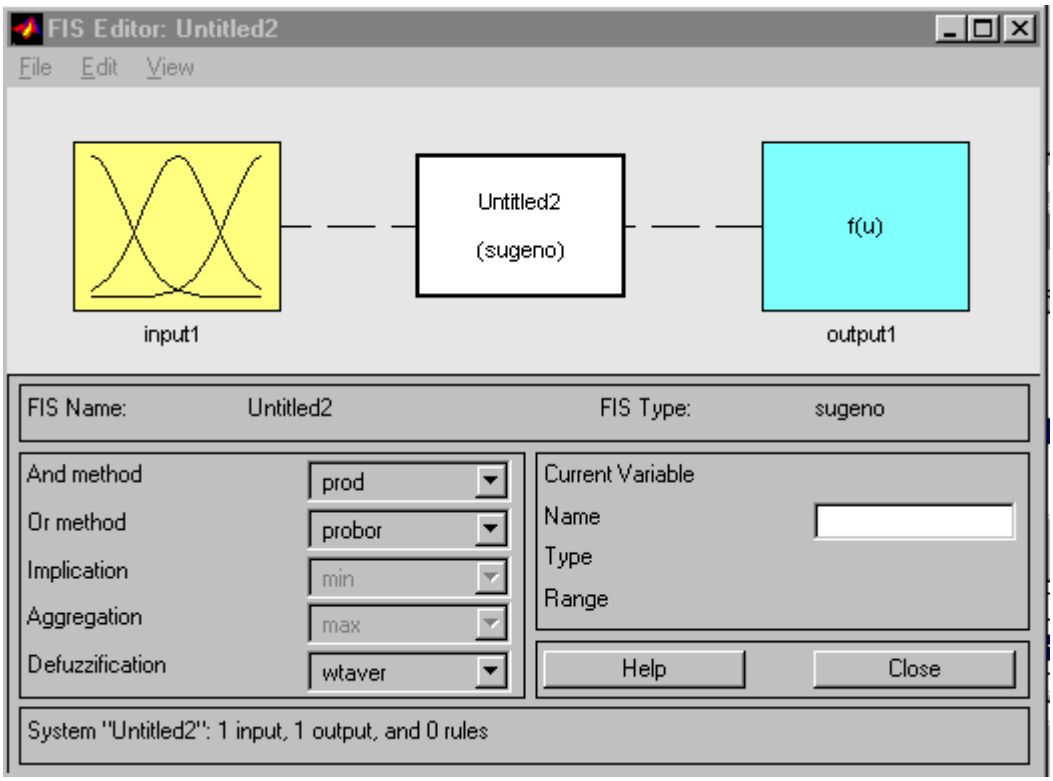

Fig. 2.35. FIS Editor for Sugeno reasoning.

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]. Click *Ready* so that your choice is recorded. 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*. At this point the following appears.

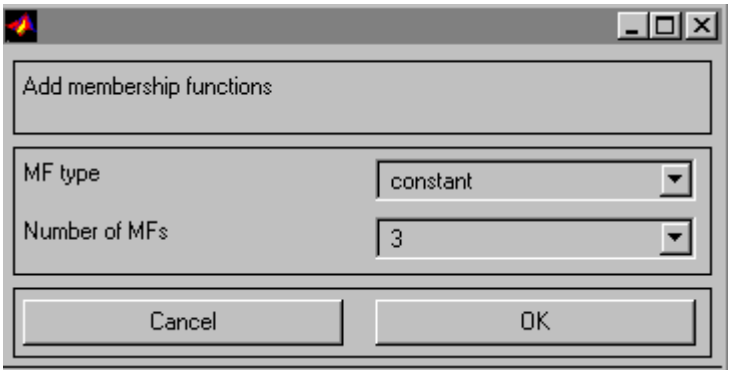

Fig. 2.36. Choose three, constant output membership functions.

MF type *constant* corresponds to Mamdani singletons. There are three different values of output, so choose 3 as *Number of MFs*. Clicking OK results in

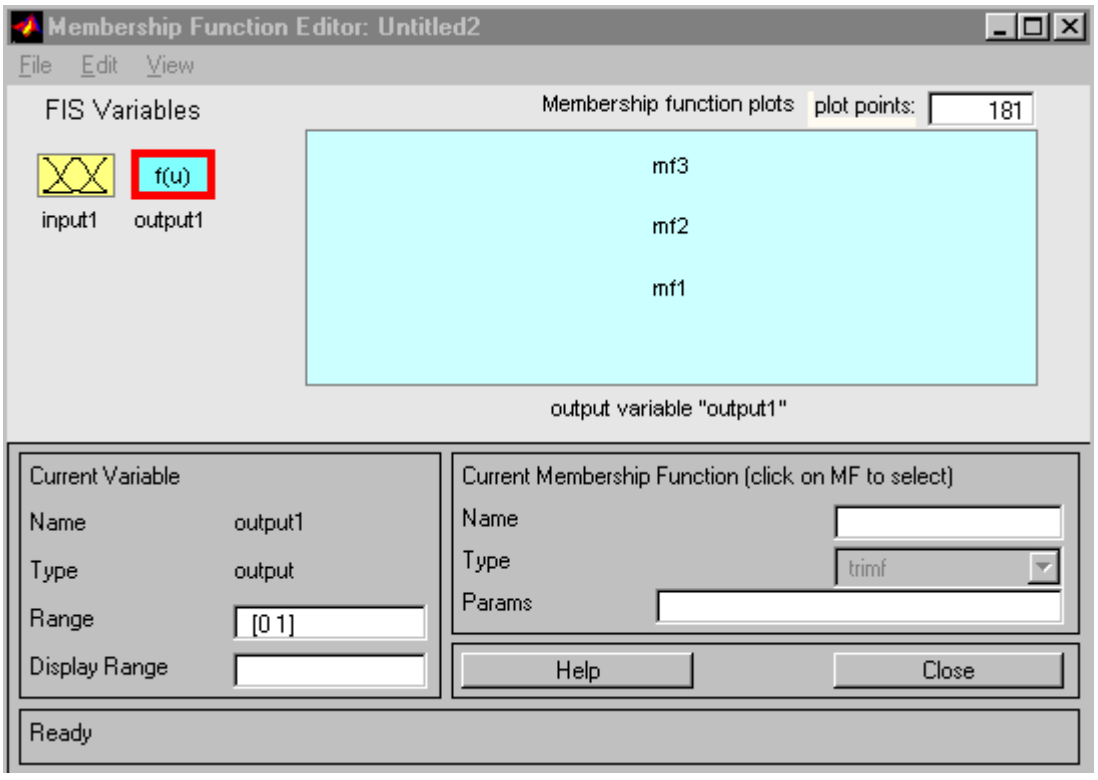

Fig. 2.37. Membership Function Editor in Sugeno type of fuzzy system.

The values of the constant membership functions are determined by activating them by clicking with the mouse. Now you change the name and value of the constant.

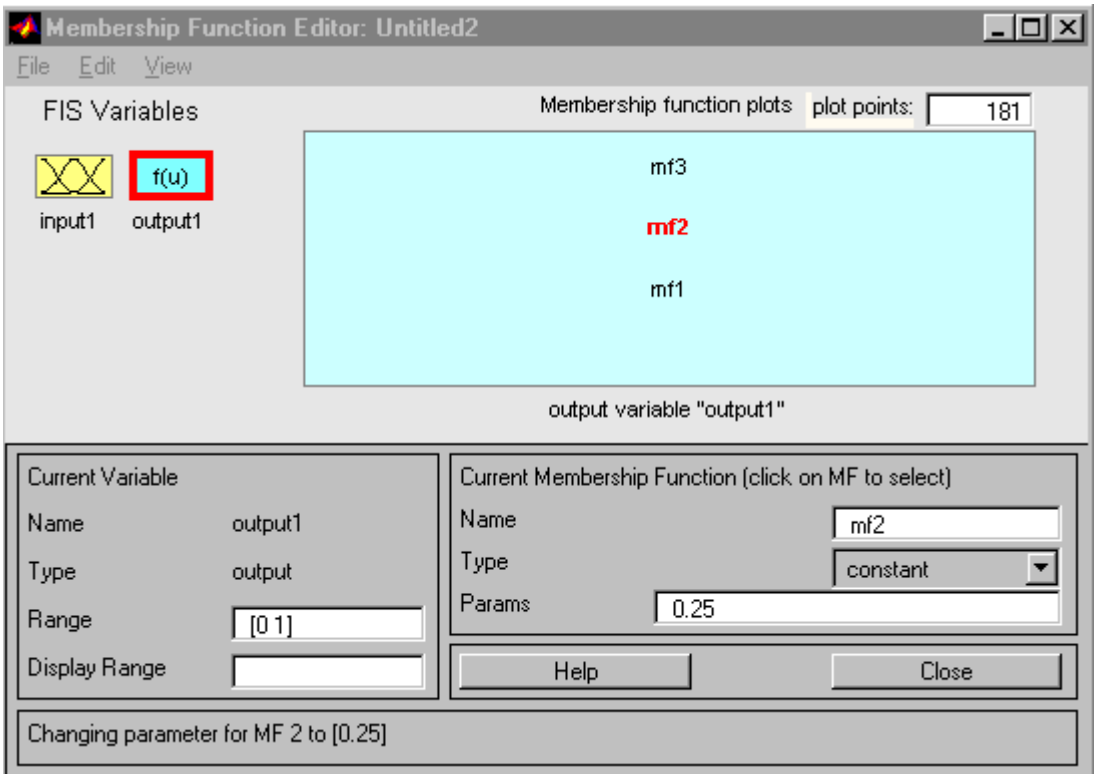

Fig. 2.38. Activate membership function *mf2*.

The final task is to form the rulebase. Choose *Edit rules* and write them in as before.

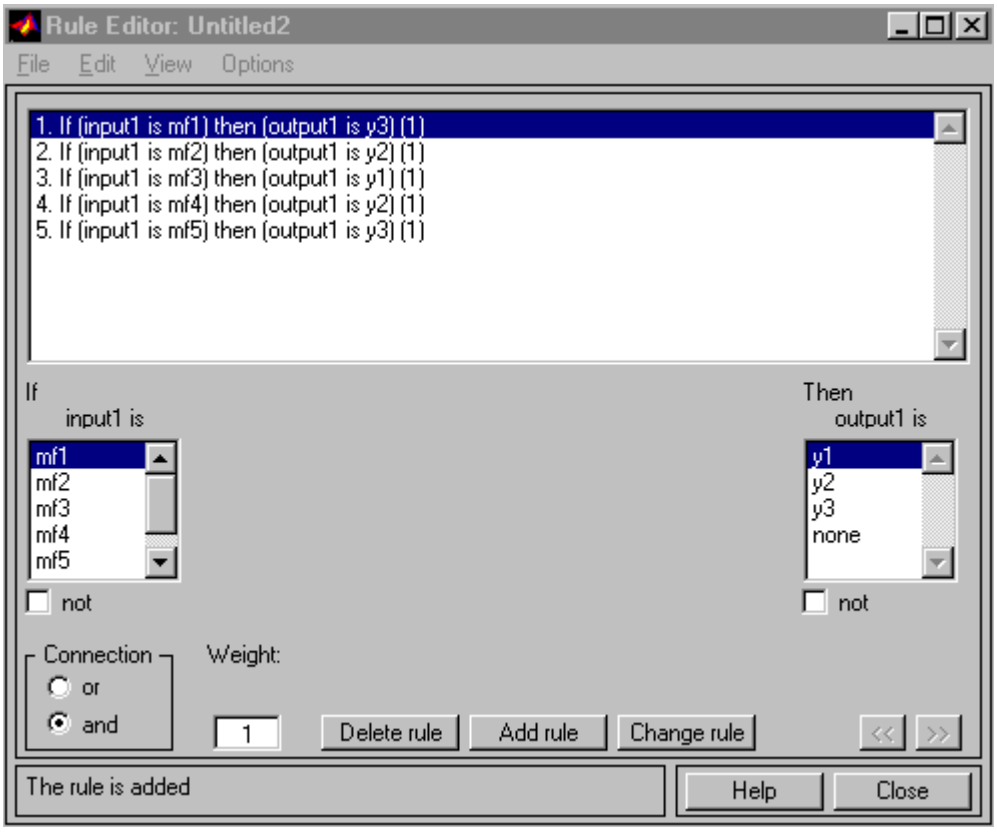

Fig. 2.39. The rule base for the example in Sugeno type of fuzzy system.

The Sugeno FIS system is now complete. Let us view the result. First the overall rules

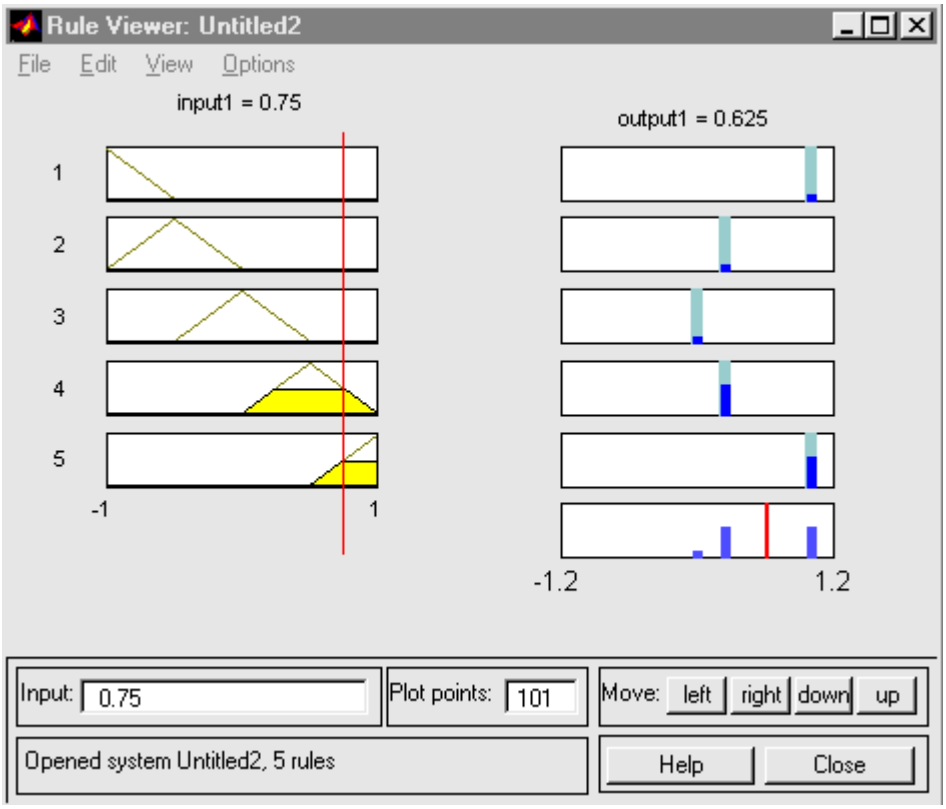

Fig. 2.40. The complete rule base of the example in the case of Sugeno type of fuzzy system.

Then the Surface View.

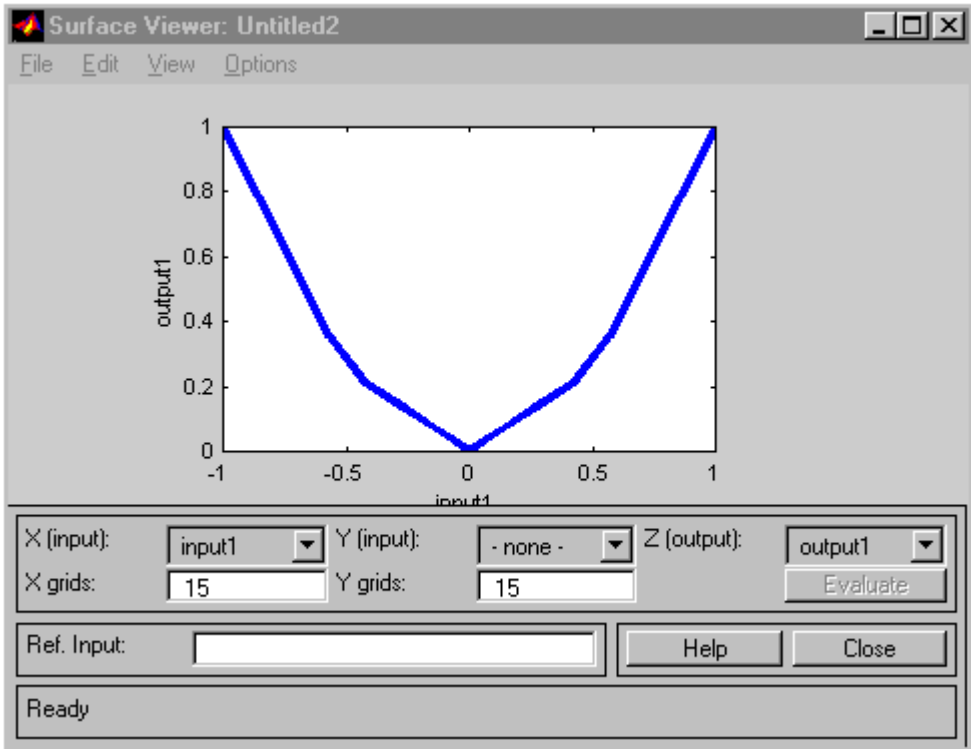

Fig. 2.41. The resulting fit of Sugeno type of fuzzy system.

The result is the roughly the same as we obtained before.

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

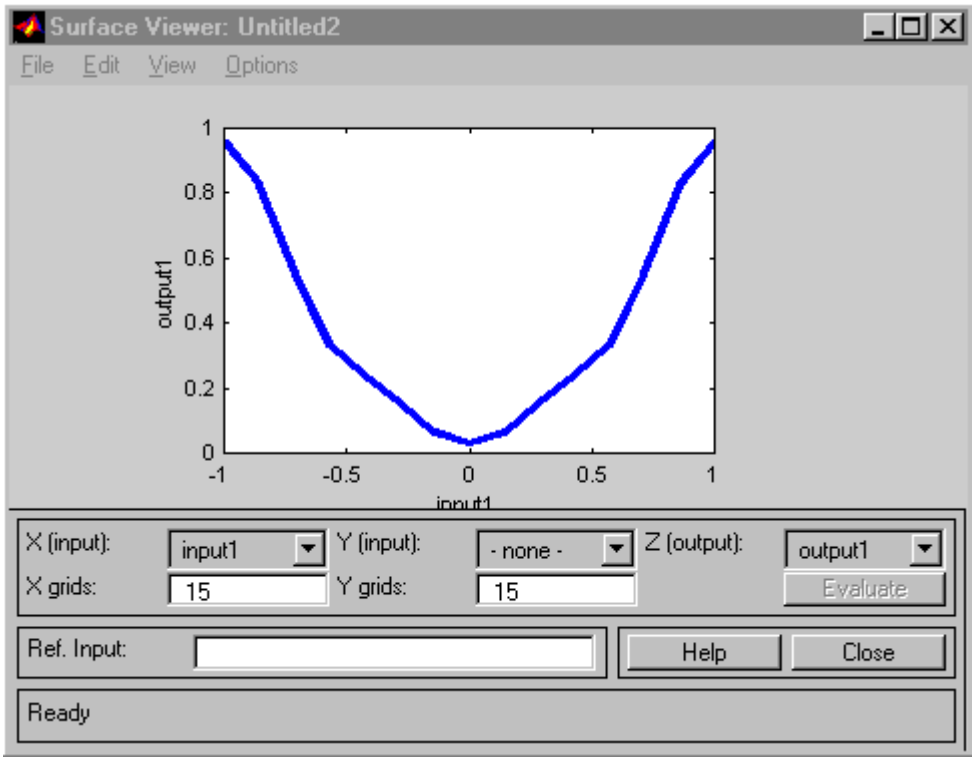

Fig. 2.42. Fuzzy Sugeno system with gaussian type of membership functions.

More generally the above procedure would be the following. The input-output data is given by

x	y
x_0	y_0
x_1	y_1
$\vdots$	$\vdots$
x_n	y_n

The rules would be of the form

If x is X_i then $y = y_i$.

Then Sugeno type of reasoning with weighted average leads to

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

y_i = discretization points of membership functions

m = number of rules

REMARK: Weighted average requires that the rule base is complete and input fuzzy membership functions cover the input space. Otherwise there is a danger to divide with zero. Note that *product* is used for *then* implication.

Sugeno reasoning allows us to use also functions of input x, not only constants, on the right hand side. The rules would look like

$$\text{If } x \text{ is } X_i \text{ then } y = f_i(x).$$

Function f_i can be a different nonlinear mapping for each rule.

x may be a vector and more complicated rule structures can also appear.

Simplest examples of functions f_i are straight lines. Let $i = 2$. Then

$$y = p_1 x + r_1$$

$$y = p_2 x + r_2$$

This is supported by *Fuzzy Toolbox*. More general functions can also be used, but these have to be set up by yourself.

EXAMPLE: 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

$$1: \quad y = -x$$

$$2: \quad y = 0$$

$$3: \quad y = x$$

Define them by *Add MF's*. Choose *linear* type.

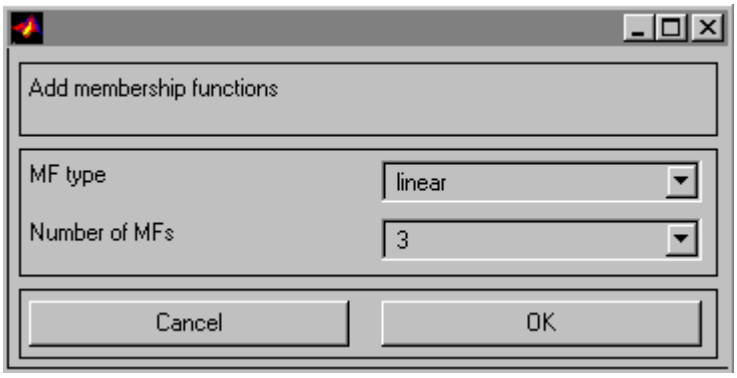

Fig. 2.43. Choose linear form of membership functions instead of constant.

The two parameters for each straight line can be chosen in *Params* box. The slope is first and then the constant.

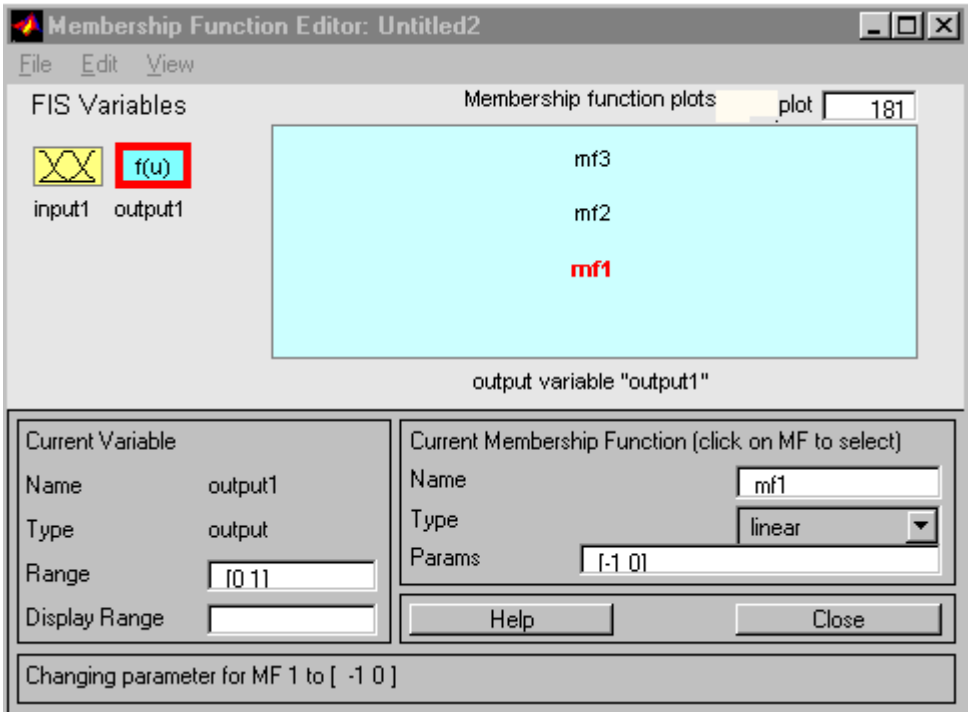

Fig. 2.44.

The rule base could be e.g.

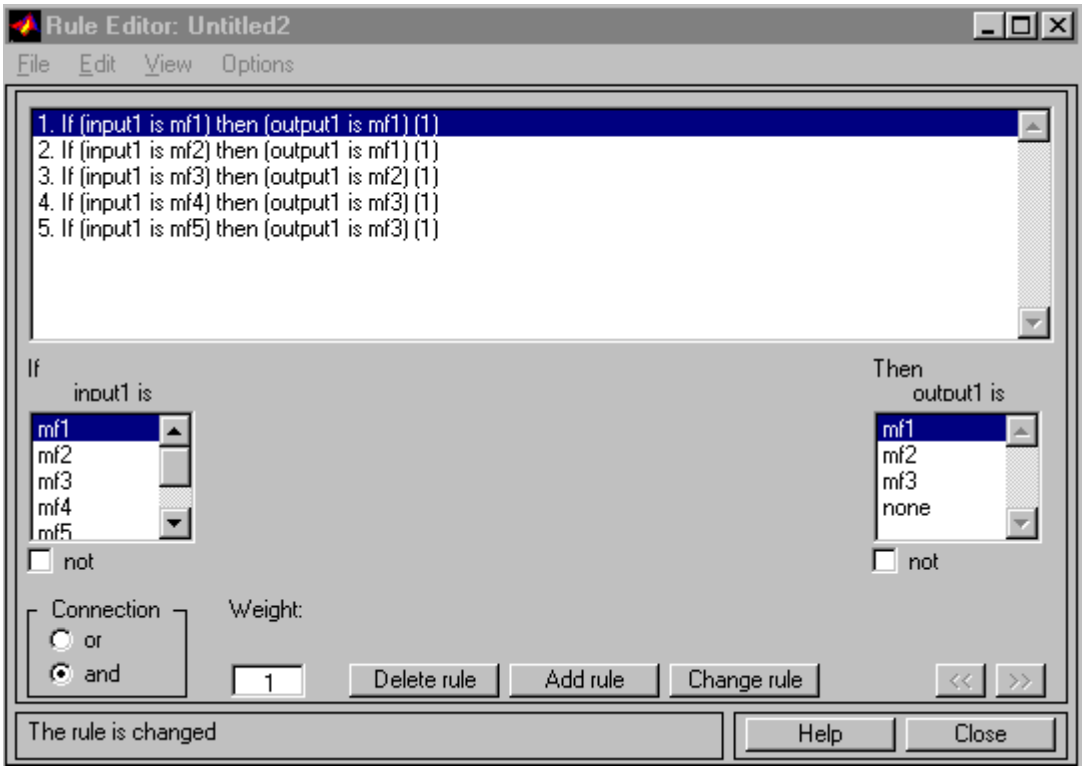

Fig. 2.45.

The overall view of the rules is shown below.

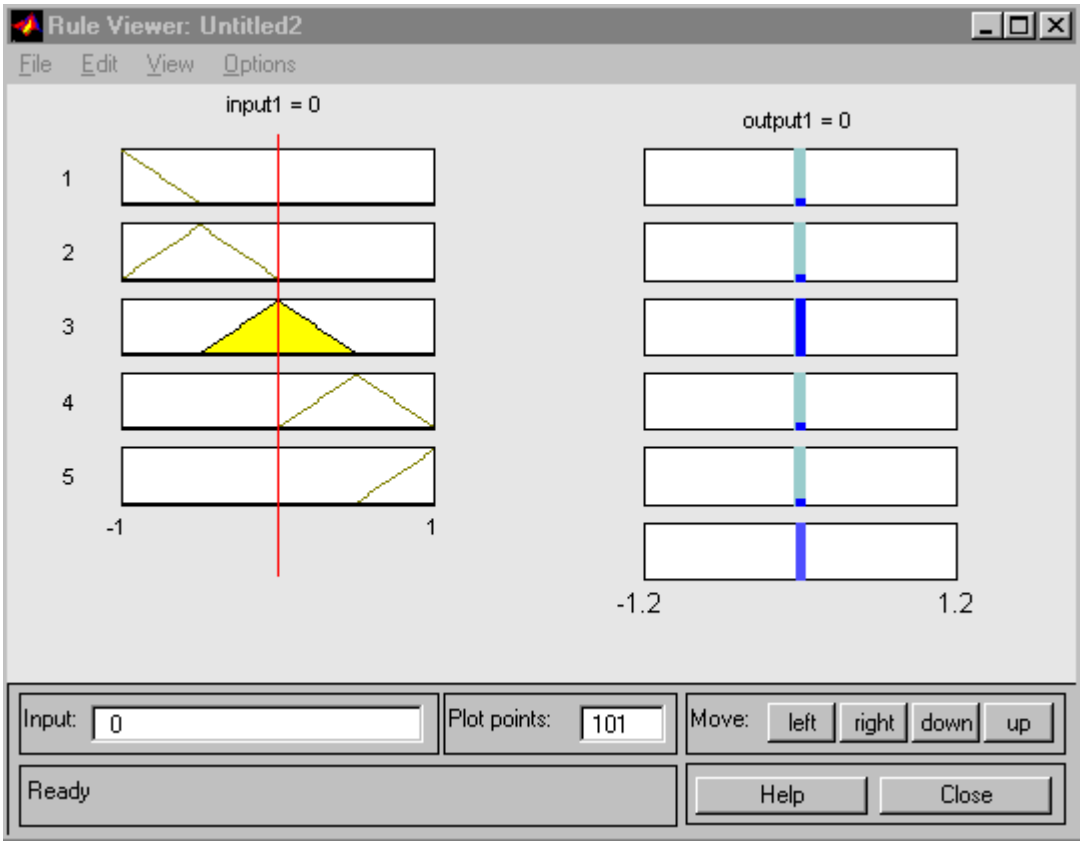

Fig. 2.46.

The Surface View becomes:

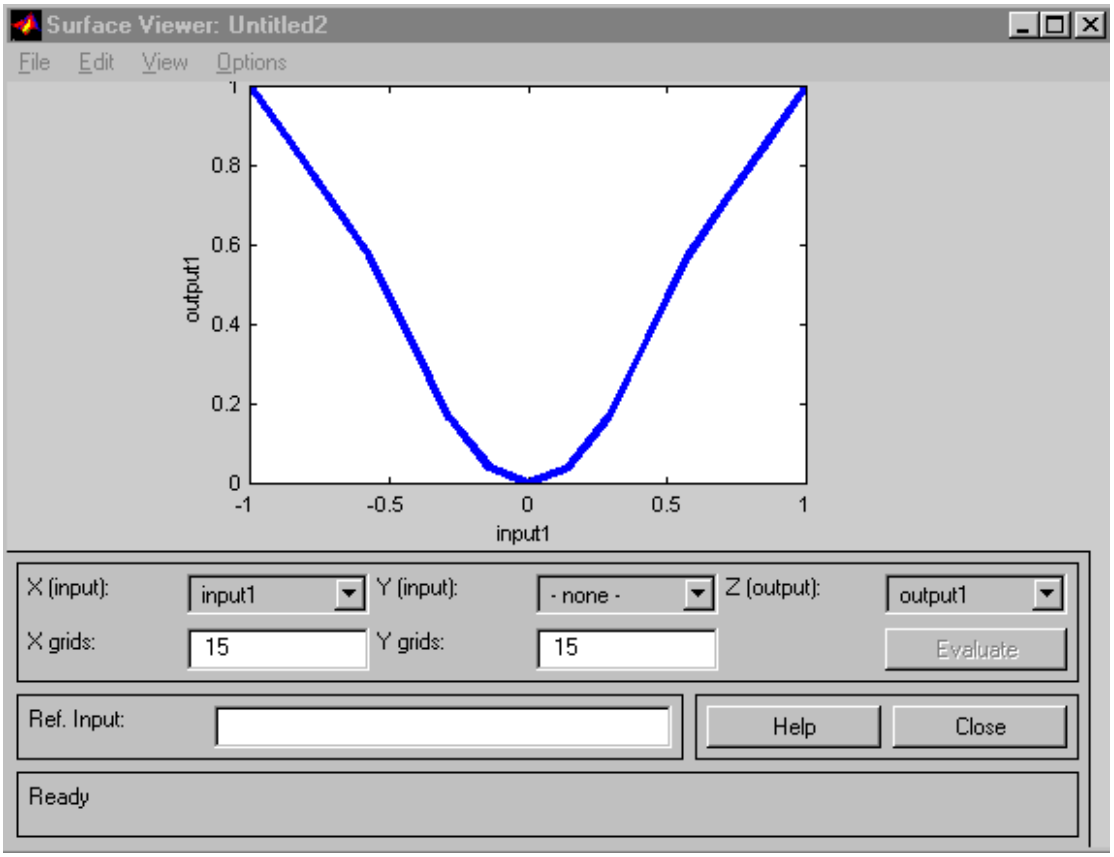

Fig. 2.47.

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.

Suppose 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.

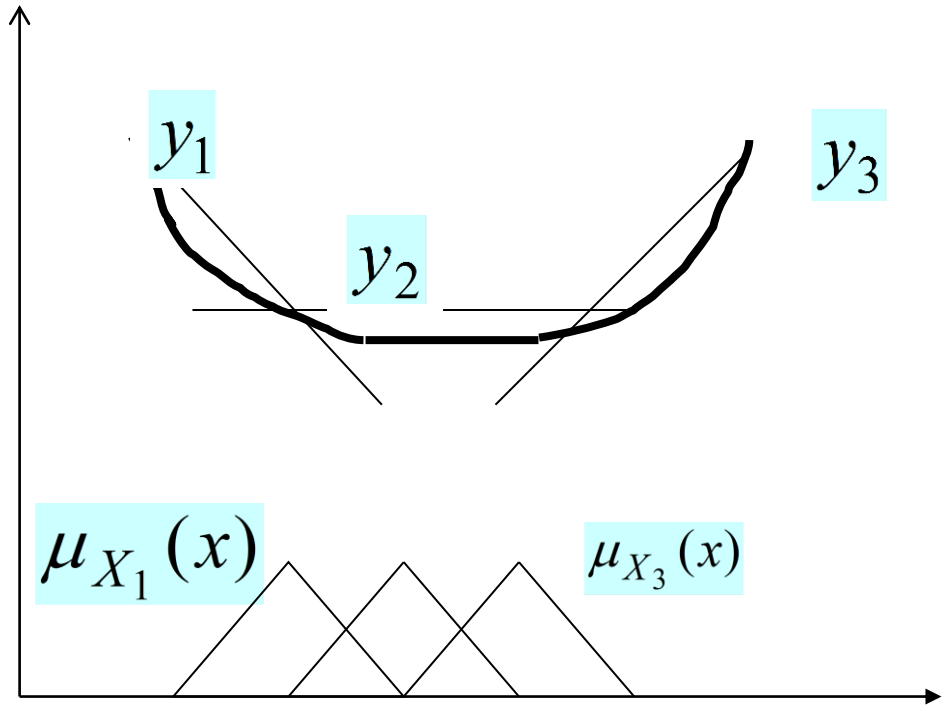

Fig. 2.48. A function defined by Sugeno model. The straight lines y_i, $i = 1,2,3$, represent the local models. The bold line is the final result.

The rules can be more general like

If x_1 *is* A_1^i and … and x_n is A_n^i
then $y^i = f^i(x_1,\ldots,x_n)$

where the consequences of the fuzzy rules are functions of the input vector $\mathbf{x} = [x_1,\cdots,x_n]$. A general linear function of Sugeno type at the output is

$$y^j = c_0^j + \sum_{k=1}^{n} c_k^j x_k ,$$

where c_k^j are real-valued parameters and specific to each rule. Since each rule has a crisp output, the aggregate result is obtained by weighted average.

$$y(\mathbf{x}) = \frac{\sum_{i=1}^{n} w_i(\mathbf{x}) y^i(\mathbf{x})}{\sum_{i=1}^{n} w_i(\mathbf{x})} ,$$

where $\mathbf{x} = [x_1,\cdots,x_n]$ and w_i depends on the applied *T*-norm for logical *and*. If product is chosen and the number of rule is *m*, then

$$w_i = \prod_{i=1}^{m} \mu_{A_i}(x), \quad i = 1,\cdots,n .$$

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 the best parameters to fit data instead of trying to do it heuristically. Depending on the choice of T-norm and defuzzification, the expression above may change, but will always have similar structure.

Basis Functions and curve fitting

Mendel-Wang's Mamdani type, fuzzy logic system is given by

$$y(\mathbf{x}) = \frac{\sum_{j=1}^{n} w^j \left(\prod_{i=1}^{m} \mu_{X_i^j}(x_i) \right)}{\sum_{j=1}^{n} \left(\prod_{i=1}^{m} \mu_{X_i^j}(x_i) \right)},$$

where w^j is the place of the output singleton. The membership function $\mu_{X_i^j}(x_i)$ corresponds to the input x_i of the rule j. The *and*-connective in the premise are realized with *product* and defuzzification with Center of Gravity method. The rules are of Mamdani type. Comparing the equation with the Sugeno expression, it is easy to see that choosing y^j to be constant and $w_j = y^j$, the result is the same.

It is useful to define a *fuzzy basis function*

$$b_j(\mathbf{x}) = \frac{\prod_{i=1}^{m} \mu_{X_i^j}(x_i)}{\sum_{j=1}^{n} \left(\prod_{i=1}^{m} \mu_{X_i^j}(x_i) \right)}$$

where the denominator normalizes the product of membership functions so that the sum of basis functions yields one at each point. Functions b_j are called basis functions, although they are not always orthogonal.

The fuzzy system can now be written in a simple way

$$y(\mathbf{x}) = \sum_{j=1}^{n} w_i b_i(\mathbf{x}).$$

Although the above does not look like a fuzzy system, it can be interpreted as such, because it is formed using membership functions, T-norms and T-conorms. This form further implies the use of optimization techniques in determining the parameters in a fuzzy system in the best way.

Curve fitting

Suppose input-output data

x	y
-1.0	−1.2
-0.5	−0.3
0	0.2
0.5	0.4
1.0	1.2

is given. Assume a linear model $y = ax + b$. Find the best a and b to fit the data.

Least squares fit:

Form a cost function

$$J(a,b) = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$

This represents the cumulative error. Minimize $J(a,b)$ with respect to (a,b).

Plot the data

```
xi=[-1 -.5 0 0.5 1]; yi=[-1.2 -0.3 0.2 0.4 1.2];
plot(x,y); grid
```

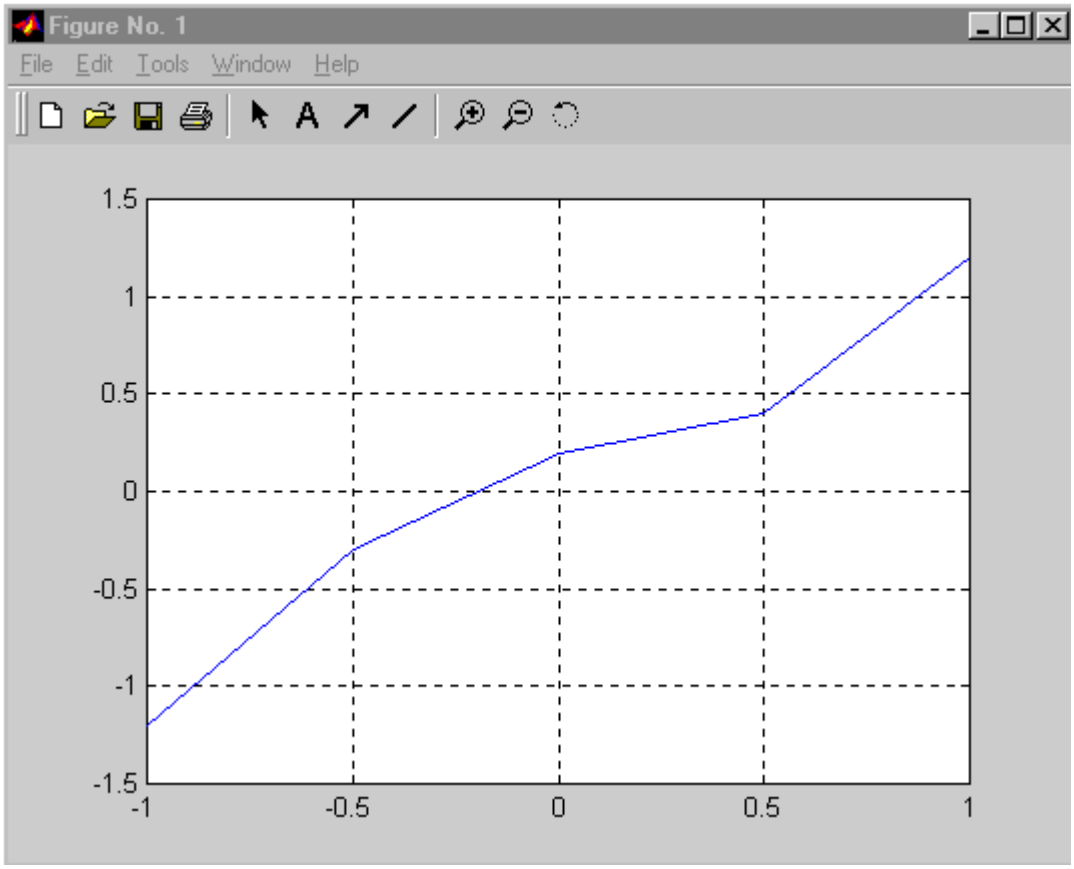

Fig. 2.49. Plotting data.

Many numerical optimization schemes can be applied to solve the minimization problem. These include gradient (steepest descent), Gauss-Newton, Levenberg-Marquardt. These are included in the **MATLAB Optimization Toolbox**. They can be studied by typing *optdemo*.
One special one is ANFIS which is included in the Fuzzy Toolbox. ANFIS is used later on fuzzy systems.

Apply here LSQNONLIN in MATLAB

help LSQNONLIN
LSQNONLIN Solves non-linear least squares problems.
LSQNONLIN solves problems of the form:
min sum {FUN(X).^2} where X and the values returned by FUN can be x vectors or matrices.

X=LSQNONLIN(FUN,X0) starts at the matrix X0 and finds a minimum X to the sum of squares of the functions in FUN. FUN accepts input X and returns a vector (or matrix) of function values F evaluated at X. NOTE: FUN should return FUN(X) and not the sum-of-squares
sum(FUN(X).^2)). (FUN(X) is summed and squared implicitly in the algorithm.)

Examples
FUN can be specified using @:
x = lsqnonlin(@myfun,[2 3 4])
where MYFUN is a MATLAB function such as:
function F = myfun(x)
F = sin(x);

FUN can also be an inline object:
fun = inline('sin(3*x)')
x = lsqnonlin(fun,[1 4]);.

Apply to the current problem

lsqnonlin('([-1.2 -.3 .2 .4 1.2] -x(1)[-1 -.5 0 .5 1]-x(2)*ones(1,5))',[0 0])*

ans = 1.1000 0.0600

or

*lsqnonlin(@(x) (yi -x(1)*xi-x(2)*ones(1,5)),[0 0])*

which produces the same result.

Plotting both the data and the fitted straight line the figure below is obtained.

```
yy=1.1*x+0.06
 plot(x,yy,'or')
hold  % Current plot held
 plot(x,y)
grid
```

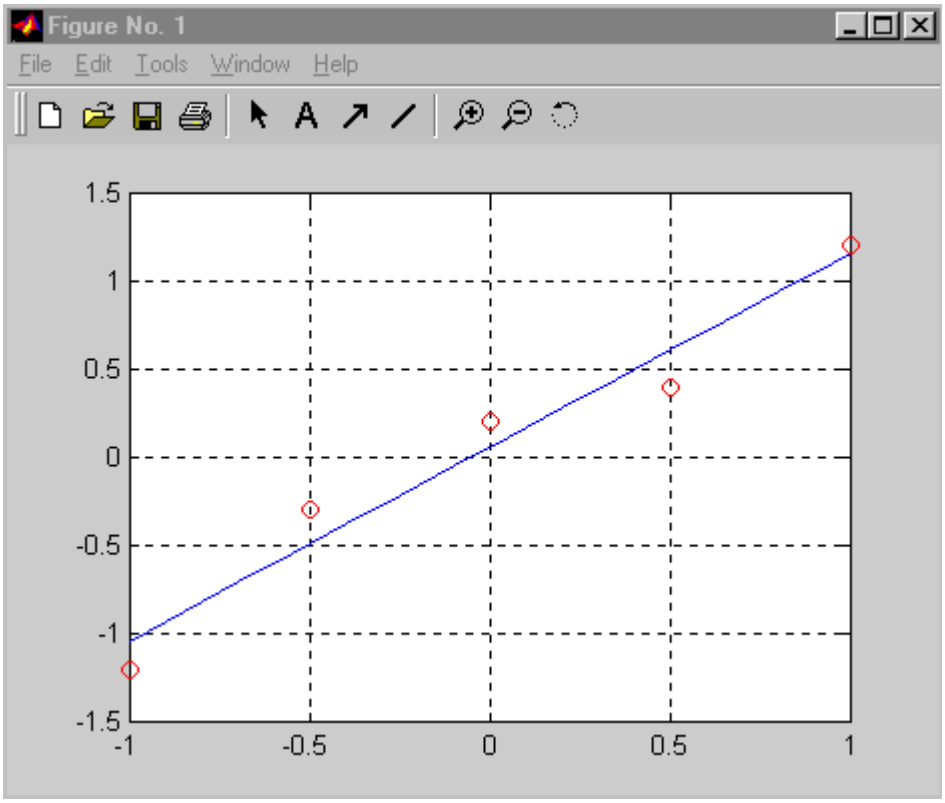

Fig. 2.50. Result of linear curve fitting.

One could use also polynomial fit. Then the command is *polyfit*. This command fits a polynomial of given degree to the data:

> *POLYFIT Fit polynomial to data.*
> *POLYFIT(X,Y,N) finds the coefficients of a polynomial P(X) of degree N that fits the data, P(X(I))~=Y(I), in a least-squares sense.*
> *[P,S] = POLYFIT(X,Y,N) returns the polynomial coefficients P and a structure S for use with POLYVAL to obtain error estimates on predictions. If the errors in the data, Y, are independent normal with constant variance, POLYVAL will produce error bounds which contain at least 50% of the predictions.*
> *The structure S contains the Cholesky factor of the Vandermonde matrix (R), the degrees of freedom (df), and the norm of the residuals (normr) as fields.*

If a third order polynomial is used, the command works as follows:

```
[p,s]=polyfit(xi,yi,3)
p =  0.6667   -0.1429    0.5333    0.1314
s =  R: [4x4 double]    df: 1      normr: 0.0956
```

Here *p* gives the coefficients of the polynomial in descending order (highest power first). To compare the answer with the least squares fit, the polynomial must be evaluated at the points *x*. For that there is a convenient command *polyval(p,x)*. Apply that

```
y1=polyval(p,xi);
```

and plot all the data and the least square straight line and third order polynomial to the same figure.

```
xx=-1:0.01:1;
y1=polyval(p,xx);
yy=1.1*xx+0.06;
plot(x,y);hold; plot(xx,y1,'xr',xx,yy,'obl'); grid
```

This yields

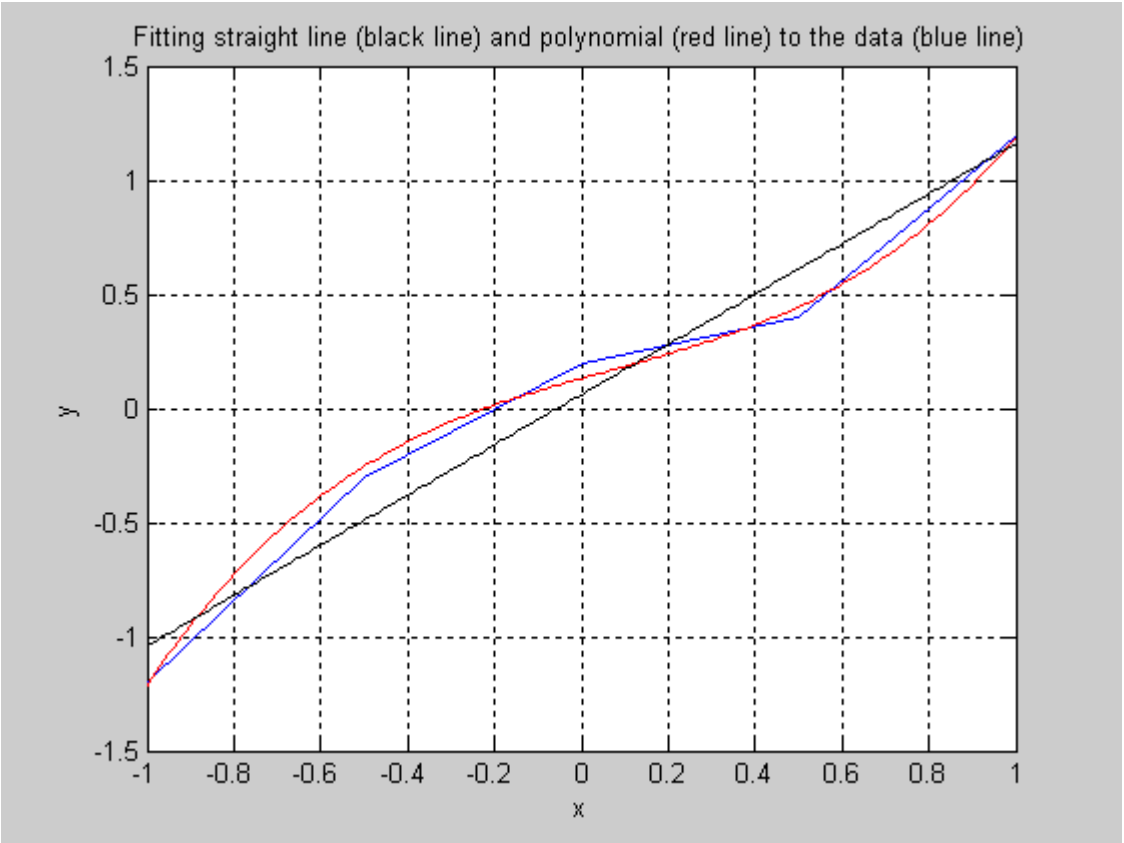

Fig. 2.51. Curve fitting on given data (blue), straight line (black) and third order polynomial fit (red).

The joy of MATLAB commands in curve fitting is not complete, if spline functions and their fit are not mentioned. Splines offer many times the best fits. Study MATLAB command *spline*.