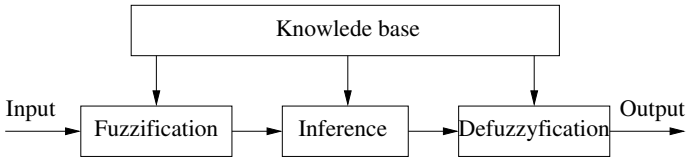
12 Adaptive Neuro-Fuzzy Inference System (ANFIS)

(based on *Fuzzy Logic Toolbox. User's Guide.*)



- The fuzzy inference system that we have considered is a model that maps
  - input characteristics to input membership functions,
  - input membership function to rules,
  - rules to a set of output characteristics,
  - output characteristics to output membership functions, and
  - the output membership function to a single-valued output, or
  - a decision associated with the output.
- We have only considered membership functions that have been fixed, and somewhat arbitrarily chosen.
- Also, we have only applied fuzzy inference to modeling systems whose rule structure is essentially predetermined by the user's interpretation of the characteristics of the variables in the model.
- In general the shape of the membership functions depends on parameters that can be adjusted to change the shape of the membership function.
- The parameters can be automatically adjusted depending on the data that we try to model.

Model Learning and Inference Through ANFIS

- Suppose we already have a collection of input/output data and would like to build a fuzzy inference model/system that approximate the data.
- Such a model would consist of a number of membership functions and rules with adjustable parameters similarly to that of neural networks.
- Rather than choosing the parameters associated with a given membership function arbitrarily, these parameters could be chosen so as to tailor the membership functions to the input/output data in order to account for these types of variations in the data values.
- The neuro-adaptive learning techniques provide a method for the fuzzy modeling procedure to learn information about a data set, in order to compute the membership function parameters that best allow the associated fuzzy inference system to track the given input/output data.
- Using a given input/output data set, the toolbox function **anfis** constructs a fuzzy inference system (FIS) whose membership function parameters are tuned (adjusted) using either a backpropagation algorithm alone, or in combination with a least squares type of method.
- This allows your fuzzy systems to learn from the data they are modeling.

FIS Structure and Parameter Adjustment

- A network-type structure similar to that of a neural network, which maps inputs through input membership functions and associated parameters, and then through output membership functions and associated parameters to outputs, can be used to interpret the input/output map.
- The parameters associated with the membership functions will change through the learning process.
- The computation of these parameters (or their adjustment) is facilitated by a gradient vector, which provides a measure of how well the fuzzy inference system is modeling the input/output data for a given set of parameters.
- Once the gradient vector is obtained, any of several optimization routines could be applied in order to adjust the parameters so as to reduce some error measure (usually defined by the sum of the squared difference between actual and desired outputs).
- **anfis** uses either back propagation or a combination of least squares estimation and backpropagation for membership function parameter estimation.

Familiarity Breeds Validation: Know Your Data

- The modeling approach used by **anfis** is similar to many system identification techniques.
- First, you hypothesize a parameterized model structure
  (relating inputs to membership functions to rules to outputs to membership functions, and so on).
- Next, you collect input/output data in a form that will be usable by **anfis** for training.
- You can then use **anfis** to train the FIS model to emulate the training data presented to it by modifying the membership function parameters according to a chosen error criterion.
- In general, this type of modeling works well if the training data presented to **anfis** for training (estimating) membership function parameters is fully representative of the features of the data that the trained FIS is intended to model.
- This is not always the case, however. In some cases, data is collected using noisy measurements, and the training data cannot be representative of all the features of the data that will be presented to the model.

  This is where model validation comes into play.

Model Validation Using Checking and Testing Data Sets

- Model validation is the process by which the input vectors from input/output data sets on which the FIS was not trained, are presented to the trained FIS model, to see how well the FIS model predicts the corresponding data set output values.
- You use a validation data set to check and control the potential for the model overfitting the data.
- When checking data is presented to **anfis** as well as training data, the FIS model is selected to have parameters associated with the minimum checking data model error.
- One problem with model validation for models constructed using adaptive techniques is selecting a data set that is both representative of the data the trained model is intended to emulate, yet sufficiently distinct from the training data set so as not to render the validation process trivial.
- If you have collected a large amount of data, hopefully this data contains all the necessary representative features, so the process of selecting a data set for checking or testing purposes is made easier.
- However, if you expect to be presenting noisy measurements to your model, it is possible the training data set does not include all of the representative features you want to model.
- The basic idea behind using a checking data set for model validation is that after a certain point in the training, the model begins overfitting the training data set.
- In principle, the model error for the checking data set tends to decrease as the training takes place up to the point that overfitting begins, and then the model error for the checking data suddenly increases.

Two examples

- In the first example in the following section, two similar data sets are used for checking and training, but the checking data set is corrupted by a small amount of noise.
- This example illustrates of the use of the ANFIS Editor GUI with checking data to reduce the effect of model overfitting.
- In the second example, a training data set that is presented to **anfis** is sufficiently different than the applied checking data set.

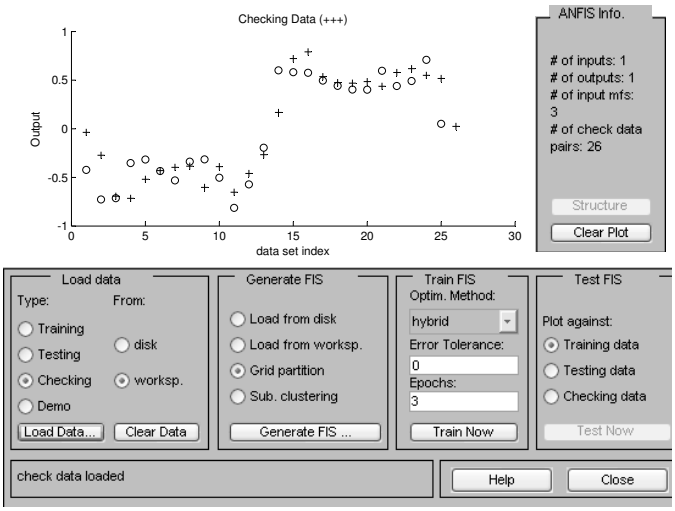
Example 1: Checking Data Helps Model Validation

- By examining the checking error sequence over the training period, it is clear that the checking data set is not good for model validation purposes.
- This example illustrates the use of the ANFIS Editor GUI to compare data sets.
- Load training and checking data sets into the MATLAB workspace from the command line:

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

- Open the ANFIS editor GUI: **anfisedit**

- Load data set `fuzex1trnData` for training from the workspace
  The training data appears in the plot in the center of the GUI as a set of circles.
- Notice the horizontal axis is marked data set index. This index indicates the row from which that input data value was obtained (whether or not the input is a vector or a scalar).
- Load checking data `fuzex1chkData` from the workspace.
  This data appears in the GUI plot as pluses superimposed on the training data.

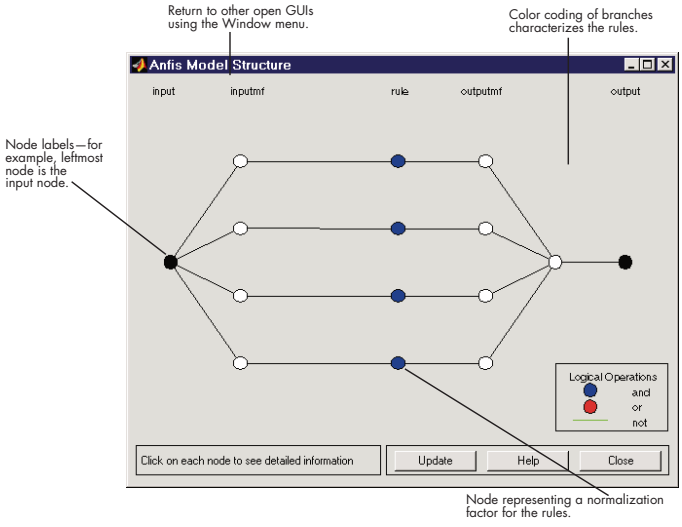


- This data set will be used to train a fuzzy system by adjusting the membership function parameters that best model this data.
- The next step is to specify an initial fuzzy inference system for **anfis** to train.
- You can either initialize the FIS parameters to your own preference, or if you do not have any preference for how you want the initial membership functions to be parameterized, you can let **anfis** do this for you.

Automatic FIS Structure Generation with ANFIS

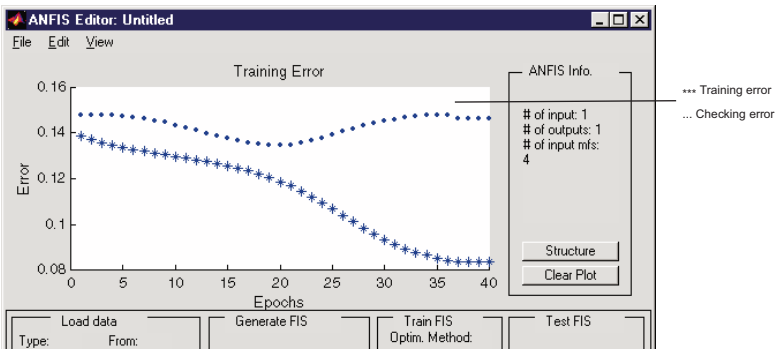
- Select a partition method: grid partitioning or subtractive clustering (described later).
  `Grid partition` is the default partitioning method.
- To Generate FIS choose first the number of membership functions, MFs, then
- the type of input and output membership functions.
  Notice there are only two choices for the output membership function: constant and linear.

We can examine the structure of the the resulting FIS:



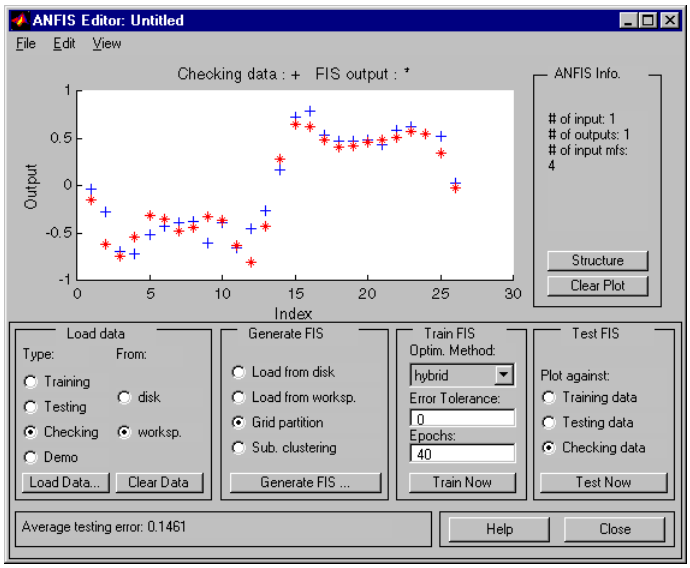
ANFIS Training

- The two **anfis** parameter optimization method options available for FIS training are `hybrid` (the default, mixed least squares and backpropagation) and `backpropa` (backpropagation).
- Error Tolerance is used to create a training stopping criterion, which is related to the error size. The training will stop after the training data error remains within this tolerance. This is best left set to 0 if you don't know how your training error is going to behave.
- Running training for 40 epochs gives the following error results:
- Notice how the checking error decreases up to a certain point in the training and then it increases.
- This increase represents the point of model overfitting.



- **anfis** chooses the model parameters associated with the minimum checking error (just prior to this jump point).
- This is an example for which the checking data option of **anfis** is useful.

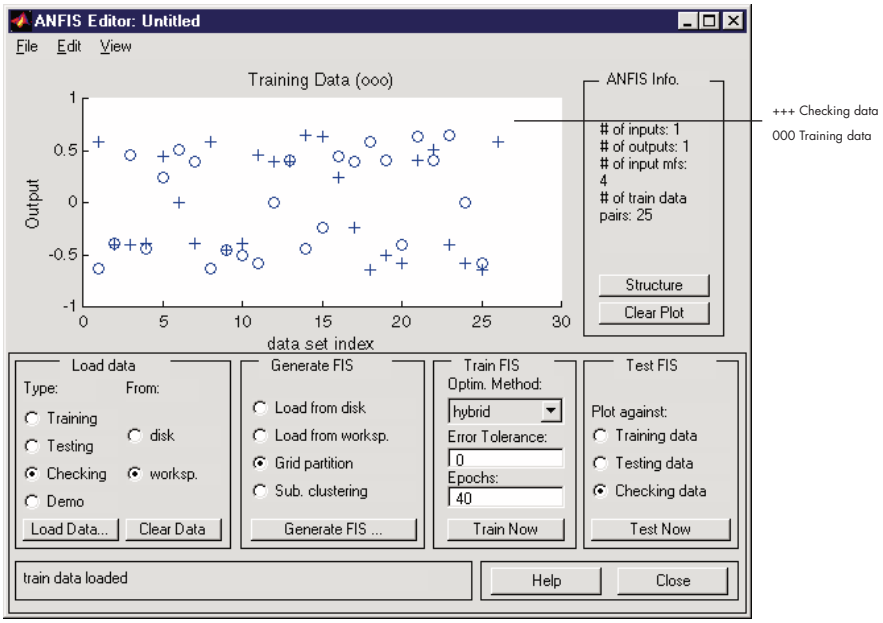
Testing the FIS against the checking data gives the following results.



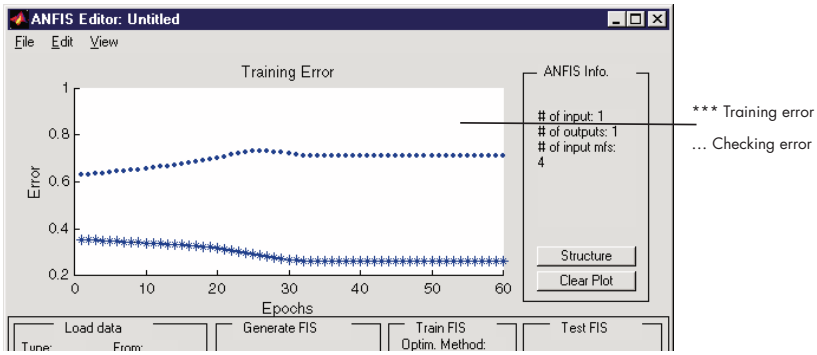
Example 2: Checking Data Does not Validate Model

After `Clear Plot` we load the 2nd set of training and checking data:

You should get something that looks like this.



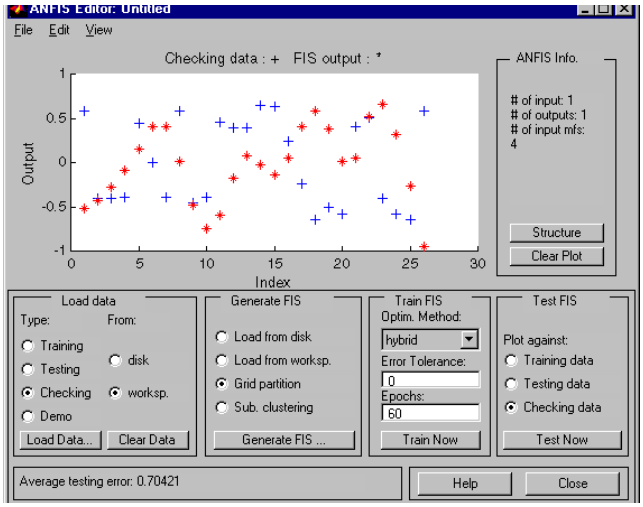
- Training for 60 epochs gives the following results:
- Notice the checking error is quite large.
- It appears that the minimum checking error occurs within the first epoch.



- Recall that using the checking data option with **anfis** automatically sets the FIS parameters to be those associated with the minimum checking error.
- Clearly this set of membership functions would not be the best choice for modeling the training data. Whats wrong here?
- This example illustrates the problem discussed earlier wherein the checking data set presented to anfis for training was sufficiently different from the training data set.
- As a result, the trained FIS did not capture the features of this data set very well.
- This illustrates the importance of knowing the features of your data set well enough when you select your training and checking data.

- When this is not the case, you can analyze the checking error plots to see whether or not the checking data performed sufficiently well with the trained model.
- In this example, the checking error is sufficiently large to indicate that either more data needs to be selected for training, or you may want to modify your membership function choices (both the number of membership functions and the type).
- Otherwise the system can be retrained without the checking data, if you think the training data captures sufficiently the features you are trying to represent.

The result of approximating the data is as follows:



Using anfis for Chaotic Time-Series Prediction

The demo **mgtsdemo** uses **anfis** to predict a time series that is generated by the following Mackey-Glass (MG) time-delay differential equation:

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(x-\tau)} - 0.1x(t)$$

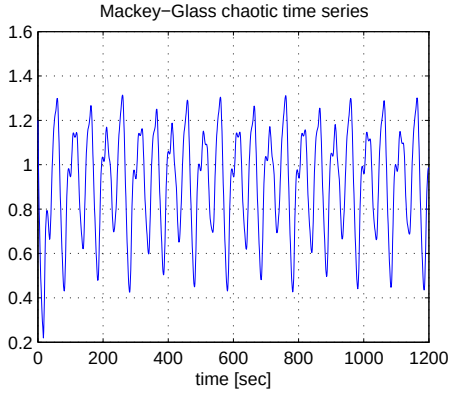
This time series is chaotic, and so there is no clearly defined period. The series will not converge or diverge, and the trajectory is highly sensitive to initial conditions. This is a benchmark problem in the neural network and fuzzy modeling research communities.
To obtain the time series value at integer points, we applied the fourth-order Runge-Kutta method to find the numerical solution to the above MG equation; the result was saved in the file `mgdata.dat`.

Here we assume:

$x(0) = 1.2, \tau = 17$, and $x(t) = 0$ for $t < 0$.

To plot the MG time series, type:

```
load mgdata.dat;
t = mgdata(:,1);
x = mgdata(:,2);
plot(t,x)
```



- In time-series prediction, we want to use known values of the time series up to the point in time, say, t, to predict the value at some point in the future, say, $t + P$.
- The standard method for this type of prediction is to create a mapping from D sample data points, sampled every units in time, $(x(t - (D - 1)), \ldots, x(t-), x(t))$, to a predicted future value $x(t + P)$.
- Following the conventional settings for predicting the MG time series, we set $D = 4$ and $\Delta = P = 6$.
- For each t, the input training data for **anfis** is a four-dimensional vector of the following form.

$$w(t) = [x(t - 18)x(t - 12)x(t - 6)x(t)]$$

- The output training data corresponds to the trajectory prediction:

$$s(t) = x(t + 6)$$

- For each t, ranging in values from 118 to 1117, the training input/output data will be a structure whose first component is the four-dimensional input w, and whose second component is the output s.
- There will be 1000 input/output data values. We use the first 500 data values for the **anfis** training (these become the training data set), while the others are used as checking data for validating the identified fuzzy model.
- This results in two 500-point data structures, `trnData` and `chkData`.

- Here is the code that generates this data.

```
for t=118:1117,
  Data(t-117,:)=[x(t-18) x(t-12) x(t-6) x(t) x(t+6)];
end
trnData=Data(1:500, :);
chkData=Data(501:end, :);
```

- To start the training, we need an FIS structure that specifies the structure and initial parameters of the FIS for learning.
- This is the task of `genfis1`.

```
fismat = genfis1(trnData);
```

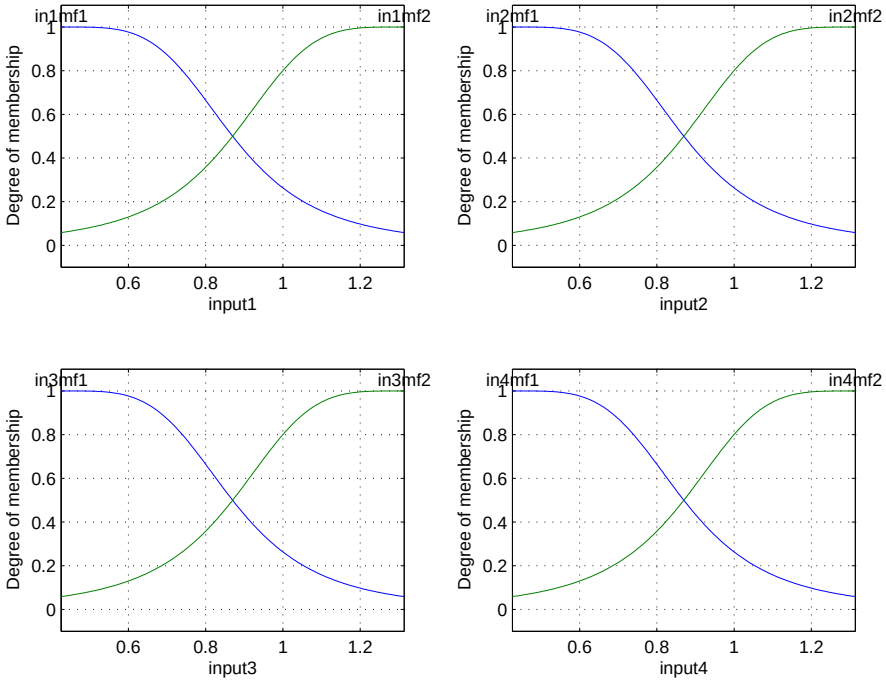
- Since we did not specify numbers and types of membership functions used in the FIS, default values are assumed.
- These defaults provide two generalized bell membership functions on each of the four inputs, eight altogether.
- The generated FIS structure contains 16 fuzzy rules with 104 parameters.
- In order to achieve good generalization capability, it is important to have the number of training data points be several times larger than the number parameters being estimated.
- In this case, the ratio between data and parameters is about five (500/104).

The function `genfis1` generates initial membership functions that are equally spaced and cover the whole input space.

You can plot the input membership functions using the following commands.

```
subplot(2,2,1);
  plotmf(fismat, 'input', 1)
subplot(2,2,2);
  plotmf(fismat, 'input', 2)
subplot(2,2,3);
  plotmf(fismat, 'input', 3)
subplot(2,2,4);
  plotmf(fismat, 'input', 4)
```

These initial membership functions are as shown:



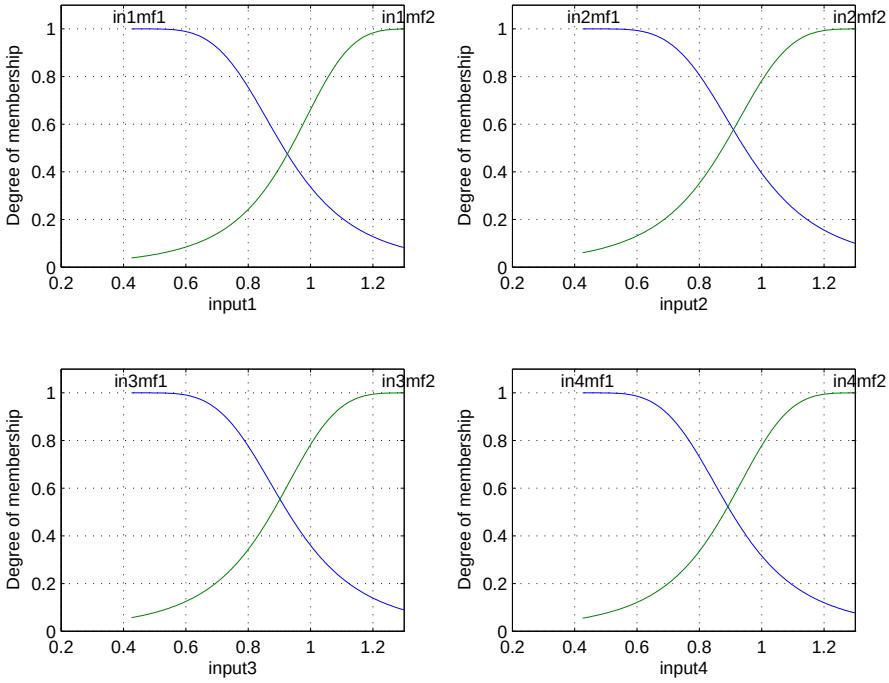
To start the training, type

```
[fismat1,error1,ss,fismat2,error2] = anfis(trnData,fismat,[],[],chkData);
```

Because the checking data option of **anfis** was invoked, the final FIS you choose would ordinarily be the one associated with the minimum checking error.
This is stored in `fismat2`.
The following code will plot these new membership functions.

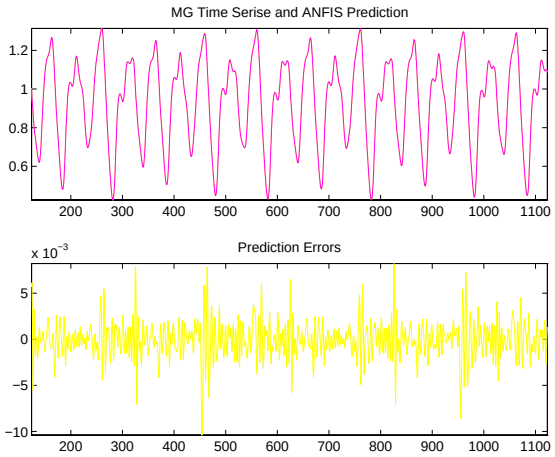
```
subplot(2,2,1);
  plotmf(fismat2, 'input', 1)
subplot(2,2,2);
  plotmf(fismat2, 'input', 2)
subplot(2,2,3);
  plotmf(fismat2, 'input', 3)
subplot(2,2,4);
  plotmf(fismat2, 'input', 4)
```

Here is the result:



- To plot the error signals, type `plot([error1; error2]);`
- Here `error1` and `error2` are the root mean squared error for the training and checking data, respectively.
- In addition to these error plots, you may want to plot the FIS output versus the training or checking data.
- To compare the original MG time series and the fuzzy prediction side by side, try

```
anfis_output = evalfis([trnData;chkData],fismat2);
index = 125:1124;
subplot(211),
  plot(t(index),[x(index) anfis_output]);
subplot(212),
  plot(t(index), x(index)- anfis_output);
```



- Note that the difference between the original MG time series and the **anfis** estimated values is very small.
- This is why you can only see one curve in the first plot. The prediction error is shown in the second plot with a much finer scale.
- Note that we have only trained for 10 epochs. Better performance is expected if we apply more extensive training.