TOOLBOX FOR NEURO-FUZZY IDENTIFICATION AND DATA ANALYSIS
For use with Matlab©

Gianluca Bontempi
Mauro Birattari

Iridia - CP 194/6
Université Libre de Bruxelles
50, av. Franklin Roosevelt
1050 Bruxelles - Belgium
email: {gbonte, mbiro }@ulb.ac.be
http://iridia.ulb.ac.be

Apr 19, 1999

Chapter 1

Release notes

This document is a preliminary user guide of the *Toolbox for neuro-fuzzy identification and data analysis* for use with Matlab.

The software makes part of a larger IRIDIA project, whose goal is the implementation of set of local modeling approaches for data analysis and regression.

The toolbox has been implemented and tested in Matlab language 4.2 but it work under Matlab 5 as well.

1.1 Conditions/Disclaimer

By using the toolbox the user agrees to all of the following:

- If one is going to publish any work where this toolbox has been used, please remember it was obtained free of charge and include a reference to [3, 5].
- The toolbox is copyrighted freeware by Gianluca Bontempi, Mauro Birattari, Iridia, Universite' Libre de Bruxelles ULB. It is not permitted to utilize any part of this software in commercial and/or military applications.
- The toolbox is provided "as-is" without warranty of any kind, either express or implied, including but not limited to the implied warranties or conditions of merchantability or fitness for a particular purpose. In no event shall Gianluca Bontempi, Mauro Birattari and/or the IRIDIA-ULB laboratory be liable for any special, incidental, indirect, or consequential damages of any kind, or damages whatsoever resulting from loss of use, data, or profits, whether or not the authors have been advised of the possibility of such damages, and/or on any theory of liability arising out of or in connection with the use or performance of this software.

Chapter 2

The fuzzy inference system

The toolbox performs the identification of a Takagi-Sugeno fuzzy architecture starting from a set of N input-output samples. The Takagi-Sugeno fuzzy model (also known as the TS fuzzy model) was proposed by Takagi, Sugeno and Kang [9, 8] in an effort to develop a systematic approach to generating fuzzy rules from a given input-output data set. A TS fuzzy inference system is a set of r rules

$$\begin{cases} \text{If } x_1 \text{ is } A_1^1 \text{ and } x_2 \text{ is } A_2^1 \ldots \text{ and } x_n \text{ is } A_n^1 \text{ then } y^1 = f^1(x_1, x_2, ..., x_n) \\ \ldots \\ \text{If } x_1 \text{ is } A_1^r \text{ and } x_2 \text{ is } A_2^r \ldots \text{ and } x_n \text{ is } A_n^r \text{ then } y^r = f^r(x_1, x_2, ..., x_n) \end{cases} \tag{2.1}$$

The first part (antecedent) of each rule is defined as a fuzzy AND proposition where A_j^i is a fuzzy set on the jth premise variable defined by the membership function $\mu_j^i : \Re^n \to [0,1]$. The second part (consequent) is a crisp function f^i $i = 1, \ldots, r$ of the input vector $[x_1, x_2, \ldots, x_n]$.

By means of the fuzzy sets A_j^i the input domain of the function f is softly partitioned in smaller regions where the mapping is locally approximated by the models f^i. The TS inference system uses the weighted mean criterion to recombine all the local representations in a global approximator:

$$y = \frac{\sum_{i=1}^r \mu^i y^i}{\sum \mu^i} \tag{2.2}$$

where μ^i is the degree of fulfillment of the ith rule.

An interesting special case is provided by the linear TS fuzzy inference system where the consequents are linear models $f^i = \sum_{j=1}^n a_j^i x_j + b^i$ [8]. In this case the TS system can be used to return a local linear approximation about a generic point of the input domain. Consider for example an input $\hat{x} = [\hat{x_1}, \hat{x_2}, \ldots, \hat{x_m}]$. The TS rule combination (Eq. 2.2) returns a linear approximation $f_{lin}(\cdot)$ to the function $f(\cdot)$ about $\hat{x}$:

$$f_{lin}(\hat{x}) = \frac{\sum_{i=1}^r \mu^i (\sum_{j=1}^n p_{ij} \hat{x_j} + p_{i0})}{\sum \mu^i} \tag{2.3}$$

2.1 From fuzzy to neuro-fuzzy

In a conventional fuzzy approach the membership functions and the consequent models are fixed by the model designer according to a priori knowledge. If this knowledge is not available but a set of input-output data is observed from the process f, the components of the fuzzy system (membership and consequent models) can be represented in a parametric form and the parameters tuned by a learning procedure. In this case the fuzzy system turns into a *neuro fuzzy* approximator [3]. Neuro-fuzzy systems are a powerful trade off in terms of readability and efficiency between a human-like representation of the model and a fast learning method. However, what mainly distinguishes neuro-fuzzy estimators from other kinds of non linear approximators is their potentiality for combining available a priori first principle models with data driven modeling techniques [4]. In fact, while learning methods provide the adaptation of the inference system to the observed data, the fuzzy architecture allows an easy integration into the system of available knowledge about the process to be modeled.

Let us see now in detail our neuro-fuzzy learning procedure.

2.2 Structural and parametric learning in neuro-fuzzy inference systems

In a neuro-fuzzy system, two types of tuning are required, namely *structural* and *parametric tuning.*

Structural tuning aims to find a suitable number of rules and a proper partition of the input space. Once available a satisfactory structure, the parametric tuning searches for the optimal membership functions together with the optimal parameters of the consequent models. There may be a lot of structure/parameter combinations which make the fuzzy model behave in a satisfactory way. The problem can be formulated as that of finding the structure complexity which will give the best performance in generalization [10]. In our approach we choose the number of rules as the measure of complexity to be properly tuned on the basis of available data. We adopt an incremental approach where different architectures having different complexity (i.e. number of rules) are first assessed in cross-validation and then compared in order to select the best one. The whole learning procedure is represented in the flow chart in Fig. 2.1.

The initialization of the architecture is provided by a hyper-ellipsoidal fuzzy clustering procedure inspired by Babuska & Verbruggen [2]. This procedure clusters the data in the input-output domain obtaining a set of hyper-ellipsoids which are a preliminary rough representation of the input/output mapping. Methods for initializing the parameters of a fuzzy inference system from the outcome of the fuzzy clustering procedure are described in [1]. Here, we use the axes of the ellipsoids (eigenvectors of the scatter matrix) to initialize the parameters of

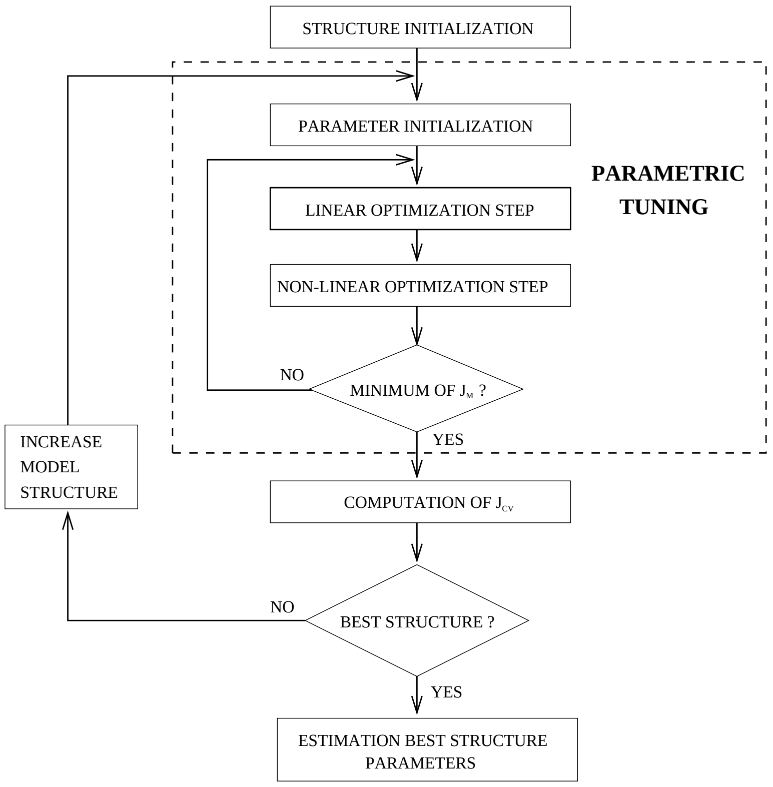


Figure 2.1: Flow-chart of the neuro-fuzzy learning procedure.

the consequent functions f^i, we project the cluster centers on the input domain to initialize the centers of the antecedents and we adopt the scatter matrix to compute the width of the membership functions. An example of fuzzy clustering in the case of a single-input-single-output function modeled by a fuzzy inference system with Gaussian antecedents is represented in Fig. 2.2.

Once the initialization is done, the learning procedure begins. Two optimization loops are nested: the parametric and the structural one. The parametric loop (the inner one) searches for the best set of parameters by minimizing a sum-of-squares cost function J_M which depends exclusively on the training set. In the case of linear TS models this minimization procedure can be decomposed in a least-squares problem to estimate the linear parameters of the consequent models f^i [6] and a nonlinear minimization (Levemberg-Marquardt) to find the parameters of the membership functions A^i_j [3].

The structural identification loop (the outer one) searches for the best structure, in terms of optimal number of rules, by increasing gradually the number of local models. The different structures are assessed and compared according to their performance J_{CV} in *K-fold* cross-validation [7]. This procedure uses a high proportion of the available data to train the current model structure and

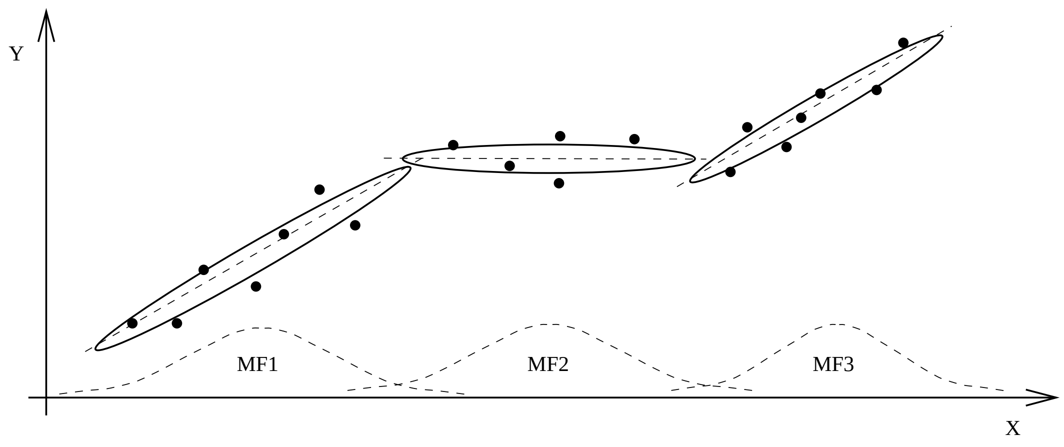


Figure 2.2: The hyper-ellipsoidal clustering initialization procedure.

gives a reliable estimate of the performance in generalization. Unfortunately, the training process has to be repeated as many times as the number K of partitions of the training set, making the whole learning process computationally expensive.

The model with the best cross-validation performance is then selected to represent the input-output mapping and consequently trained on the whole dataset.

Chapter 3

The toolbox

The toolbox has two versions

Command line It is called by

>> *fuzzy_b;*

The different options are set by uncommenting the respective lines in the m-file *fuzzy_b*

Graphical It is called by

>> *fuzzy_g;*

The options are set through a graphical interface called by the command *fuzzy_b* (see Fig. 3.1)

3.1 The fuzzy inference system in Matlab

The fuzzy inference system (2.1) is represented in the toolbox by 3 matrices:

1. The matrix $[r, n]$ *centers* which contains the location of the centers of the membership functions of the antecedents.

2. The matrix $[r, n]$ *bases* which contains the bases (standard deviation) of the triangular(Gaussian) membership functions of the antecedents.

3. The matrix *par*, which contain the parameters of the consequent model of the rules. It is a matrix $[r, 1]$ in the case of constant models and $[r, n + 1]$ in the case of linear models.

4. The matrix $[1, 1]$ bias which contains the bias term. It is the bias term used the linear step of the nonlinear parametric optimization.

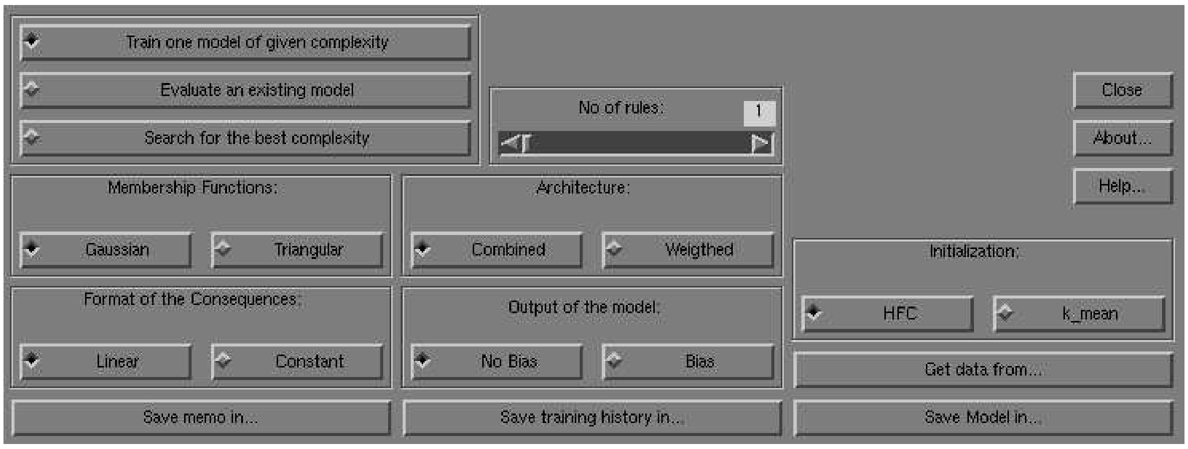


Figure 3.1: The graphical interface

3.2 Structural parameters

The program provides a set of structural alternatives in the definition of the fuzzy model. In particular the user may choose

Shape of the membership functions of the antecedents It can be Gaussian

$$\mu^i(x) = \prod_{j=1}^{n} e^{-\frac{\left(x_j - c_j^i\right)^2}{(b_j^i)^2}} \tag{3.1}$$

or triangular

$$\mu^i(x) = \prod_{j=1}^{n} \max\left(0, 1 - \frac{|x_j - c_j^i|}{b_j^i}\right). \tag{3.2}$$

where the Matlab matrix *centers(i,j)* represents the value of c_j^i and the Matlab matrix *bases(i,j)* represents the value of b_j^i.

Parametric form of the consequent model It can be constant

$$y^i = p_i \tag{3.3}$$

where the Matlab vector *par(i)* represents the value of p_i or linear (Eq. 2.3)

$$y_i = \sum_{j=1}^{n} p_{ij} x_j + p_{i0} \tag{3.4}$$

where the Matlab matrix *par* is so that *par(i,j)* contains p_{ij} and *par(i,n+1)* contains p_{i0}'

Combination method of the rules It can be a weighted combination (Eq. (2.2)) or a non weighted combination.

Bias term It can be used or not in the linear step of the nonlinear parametric optimization.

Clustering initialization policy It can be *k-mean* or *Hyperplane Fuzzy Clustering* and serve to provide the identification algorithm with good initial values of the centers and bases of the membership functions.

3.3 Program features

The toolbox runs in three features:

Training of a model with fixed complexity The user chooses the architecture of the fuzzy inference system, fixes the number of rules, chooses the initialization method and provides the program with a dataset of input-output samples. The program returns the trained model in the form of membership function parameters and consequent parameters. See Section 3.4.1 for the format of the file.

Model selection The user chooses the architecture of the fuzzy inference system, the initialization method and a range of number of rules, which will be explored by the model selection feature. The program searches also for the 'right' complexity (number of rules) of the architecture by adopting a procedure of cross-validation on the available data set. It starts with a minimal number of rules and at each step increase the number of rules by restarting the global procedure, until a maximum number of rules is reached (the user is free to set properly what is the desired range of complexity to range over). The program will plot the cross-validation error against the number of rules in order to help the designer with the choice of the best structure.

Prediction The user provides the program with a model, either trained by the program or fixed by the designer, and set of input samples. The program will return the set of predictions.

3.3.1 The command line environment

The desired feature must be selected before running the program 'uncommenting' the corresponding line in the module fuzzy_b.m. Three options (mutually exclusive) are available and correspond to the above cited features:

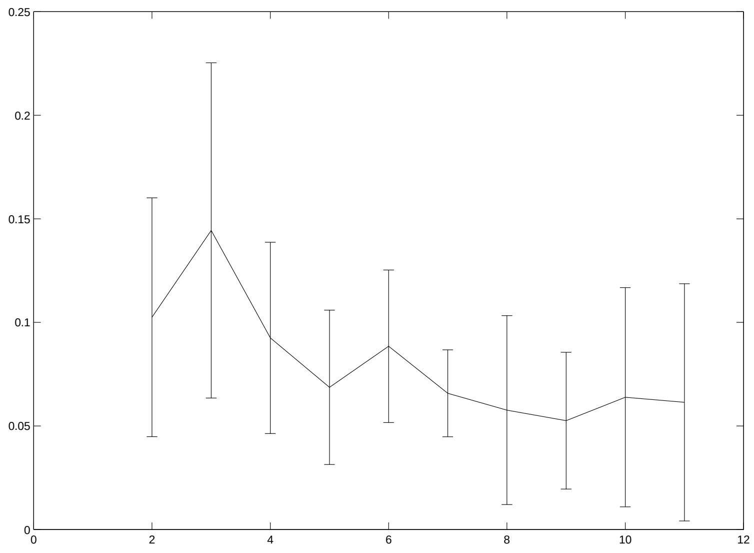


Figure 3.2: Cross-validation error vs. complexity diagram.

1. *pr='one_mod'*. By setting this option the program performs the training with fixed complexity (number of rules set in the variable *comp*). The user has to provide a training set with the format specified in Section 3.4.1 and set the path in the proper line in the module *fuzzy_b.m*.

2. *pr='comp_err'*. By setting this option the program performs the model selection feature. It searches for the best structure (in terms of number of rules) by ranging the models over the set *[comp_min comp_max]*. The numbers *comp_min* and *comp_max* represent the bounds of the complexity range (minimum and maximum number of rules, respectively).

3. *pr='evaluate'*. By setting this option the program performs the prediction feature. The user has to provide an input test set with the format specified in Section 3.4.2.

For the first two modes, the user can choice the desired model configuration in terms of membership shape (Gaussian or triangular), local model structure (constant or linear), rule of model composition (weighted or not), initialization procedure (k-means or fuzzy clustering).

3.4 Data files

3.4.1 Training and model selection

The training set must be a *.mat* file containing a rectangular matrix $[N, n+1]$ named *data*. The number N represents the number of training samples, n is the number of regressors (or inputs) and the last column of the matrix *data* contains the training output. Note that the number of outputs is always restricted to one. All the inputs and the output should be scaled between -1 and 1. Automatic scaling will be a feature of the next version.

3.4.2 Prediction

The input samples for which a prediction is required must be formatted in a matrix $[N_{ts}, n]$ named *in*.

The number N_{ts} is the number of samples for which a prediction is required while n is the number of inputs which has to be the same as the number used in the training set. This matrix must be save in a file *.mat*.

Chapter 4

Example

4.1 The problem

Consider the problem of modeling the input-output relation

$$y = 4\sin(\pi x_1) + 2\cos(\pi x_2) + N(0, 0.05) \tag{4.1}$$

where the variables x_1 and x_2 range over the domain $[-1, 1]$ and $N(0, 0.05)$ represents a Gaussian random noise signal with zero mean and standard deviation equal to 0.05.

4.2 Data generation

Assume that only a training set of $N = 100$ points is available. First we generate the training set:

```
>> N=100;
>> inputs=[2*rand(N,1)-1 2*rand(N,1)-1];
>> output=4*sin(pi*inputs(:,1))+2*cos(pi*inputs(:,2))+0.05*randn(N,1); >> data=[inputs
output];
```

then we save it:

```
>> save data_example data;
```

4.3 Model selection

Now, we search for the optimal structure configuration by running the toolbox in the *model selection* mode. We call the graphical interface

```
>> fuzzy_g
```

We select the following options:

Functioning mode : *Search for the best complexity*

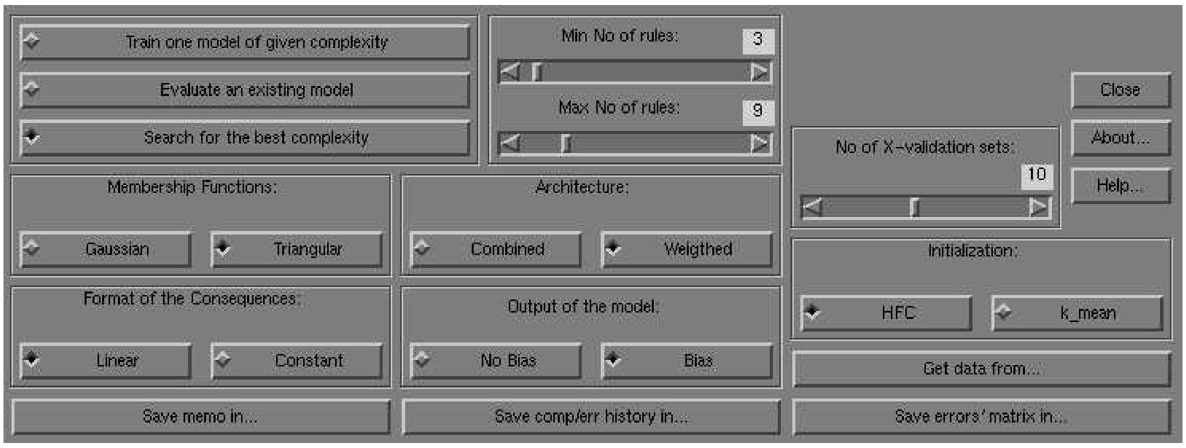


Figure 4.1: The graphical interface in the model selection mode

Membership function : *Triangular*

Format of the consequences : *Linear*

Architecture : *Weighted*

Output of the model : *Bias*

Min. no. rules : 3

Max. no. rules : 9

No. cross validation sets : 10

Initialization : *HFC*

The window will appear as in Fig. 4.1 We click on the button *Get data from..* and we select the file *data_example.mat.* This is the only mandatory operation. The *memo*, *history* and *errors* informations will be saved in default files.

Now click on *OK* and wait.
>> *FIS is working*
Note that this a quite time consuming operation but it makes easier the task of model selection which must be performed by a model designer each time he deals with nonlinear approximator. To make shorter the waiting time, you could choose a smaller number of cross-validation folds.

After 12 minutes of computation in my machine the following message will appear in the Matlab shell
...Done.
FUZZY INFERENCE SYSTEM memo:

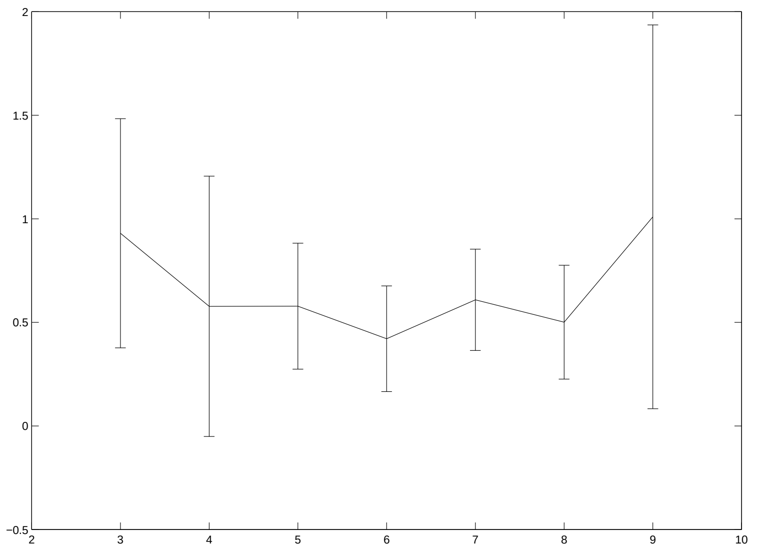


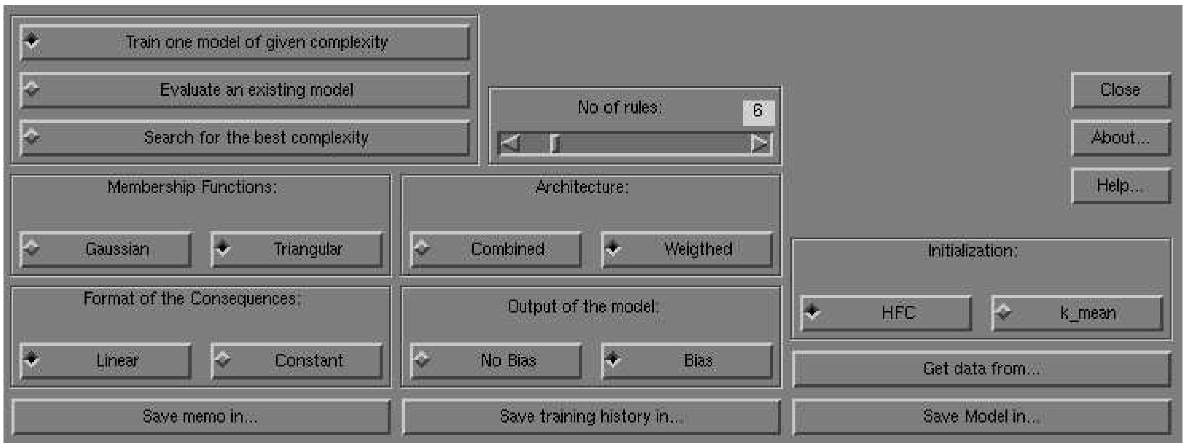
Figure 4.2: Cross-validation error vs. complexity diagram.

Number of rules vs Mean Square Error
Number of cross validation sets: 10
Shape of the membership functions: triangular
Output of each local model: linear
Architecture of the fuzzy model: weighted
Initialization: HFC
Bias vs No-Bias: bias
Minimum Number of Rules: 3
Maximum Number of Rules: 9
The matrix errors vs complexity has been saved in the file:
.../errors.mat
The diary of the computation has been saved in the file:
.../ehist.txt
and the diagram in Fig. 4.2 will be plotted.

We choose 6 as the optimal complexity of the fuzzy inference architecture.

4.4 Model training

Now we can train a fuzzy inference model with 6 rules and having the same options as specified before. We type again
>> *fuzzy_g*
we select the option *Train one model...* and we retype the options (Fig. 4.3). We reload the training set and we push *OK*.
>> *FIS is working...*



[✓ Train one model of given complexity] Evaluate an existing model Search for the best complexity

No of rules: 6

Membership Functions: Gaussian [✓ Triangular]

Architecture: Combined [✓ Weigthed]

Initialization: [✓ HFC] k_mean

Format of the Consequences: [✓ Linear] Constant

Output of the model: No Bias [✓ Bias]

Figure 4.3: The graphical interface

This time the waiting time will be shorter. After 1 minute in my machine the following message appears.
Done.
FUZZY INFERENCE SYSTEM memo:
Shape of the membership functions: triangular
Output of each local model: linear
Architecture of the fuzzy model: weighted
Initialization: HFC
Bias vs No-Bias: bias
Number of Rules: 6 The trained model has been saved in the file: .../model.mat
The diary of the computation has been saved in the file: .../mhist.txt

The model for prediction has been saved in the default file *model.mat.*

Let us see the content of the file. It contains the matrices described in Section 3.1
centers =
-0.5496 0.1093
0.5547 0.2757
-0.5687 -0.3290
-0.0458 -0.1863
0.3531 -1.1947
0.3764 0.7919
and
bases =
5.3562 4.8935
0.8293 3.1366

```
5.3806 4.8379
2.5144 3.7915
5.5308 4.4605
2.5245 3.0970
```
which describe the membership functions of the antecedents (Eq. 3.2) and the matrix

```
par =
10.0753 -67.0159 -0.1751
-29.1802 5.0977 19.2551
-30.9553 71.9611 2.7797
71.6720 12.4489 11.8340
0.4000 -17.0518 -17.2704
9.4213 -2.0301 -14.1028
```

which return the parameters of the consequent linear models (Eq. 3.4).

4.5 Model prediction

We generate now a test set of $N_{ts} = 50$ points.

```
>> N_ts=50;
>> inputs_ts=[2*rand(N_ts,1)-1 2*rand(N_ts,1)-1];
>> output_ts=4*sin(pi*inputs_ts(:,1))+2*cos(pi*inputs_ts(:,2));
>> in=inputs_ts;
>> save test_example in
```

Type for the last time

```
>> fuzzy_g
```

and select the *Evaluate an existing model* mode. Load the input test set *test_example.mat*, load the model *model.mat* and push *OK*.

```
>> FIS is working...Done.
FUZZY INFERENCE SYSTEM memo:
Shape of the membership functions:triangular
Output of each local model:linear
Architecture of the fuzzy model:weigthed
Bias vs No-Bias:bias
Number of Rules:6
The predicted output has been saved in the file:
.../out_hat.mat
```

Finally, the last step is to compare the fuzzy prediction saved in *out_hat.mat* and the target output (see Fig. 4.4).

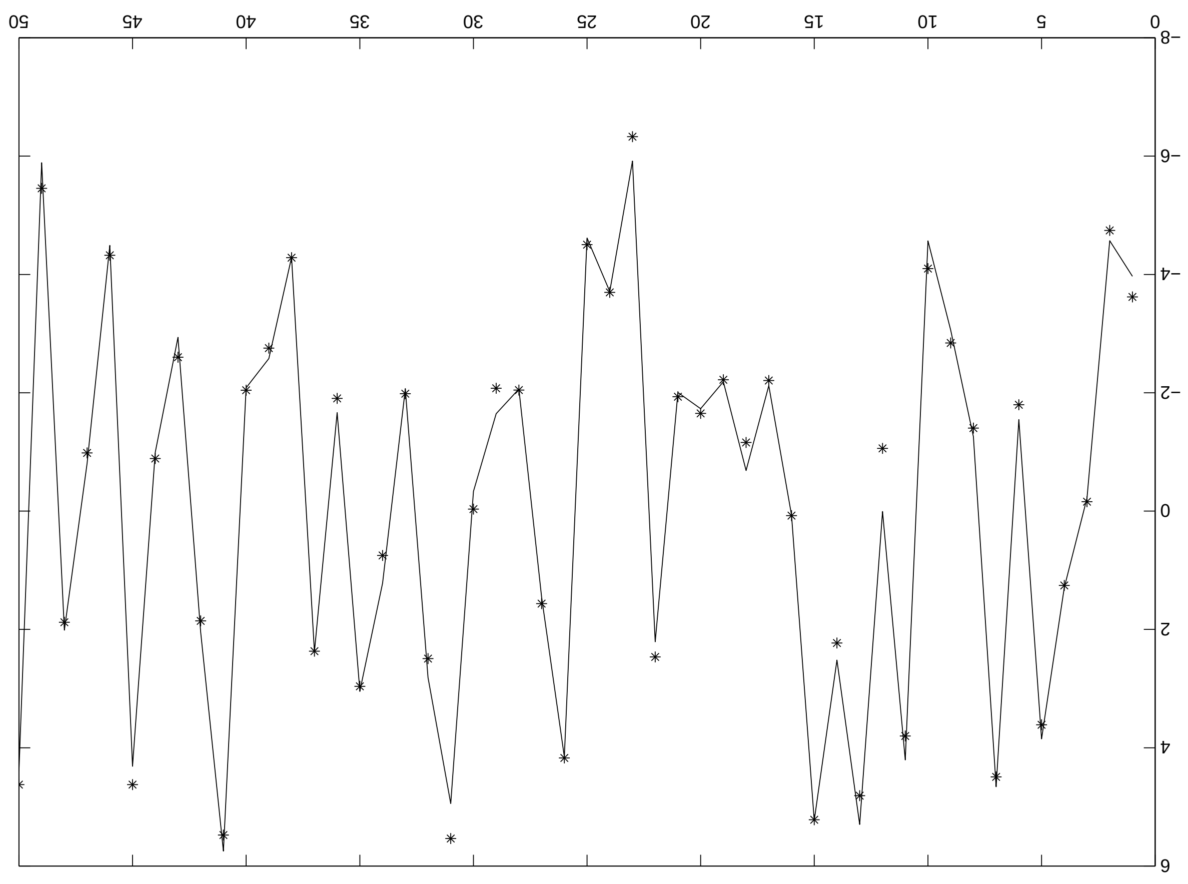


Figure 4.4: Fuzzy prediction (dotted) vs. target values (solid).

Bibliography

[1] R. Babuska. *Fuzzy Modeling and Identification.* PhD thesis, Technische Universiteit Delft, 1996.

[2] R. Babuska and H. B. Verbruggen. Fuzzy set methods for local modelling and identification. In R. Murray-Smith and T. A. Johansen, editors, *Multiple Model Approaches to Modeling and Control*, pages 75–100. Taylor and Francis, 1997.

[3] H. Bersini and G. Bontempi. Now comes the time to defuzzify the neuro-fuzzy models. *Fuzzy Sets and Sytems*, 90(2):161–170, 1997.

[4] G. Bontempi and H. Bersini. Identification of a sensor model with hybrid neuro-fuzzy methods. In A. B. Bulsari and S. Kallio, editors, *Neural Networks in Engineering systems (Proceedings of the 1997 International Conference on Engineering Applications of Neural Networks (EANN '97), Stockolm, Sweden)*, pages 325–328, 1997.

[5] G. Bontempi, H. Bersini, and M. Birattari. The local paradigm for modeling and control: From neuro-fuzzy to lazy learning. *Fuzzy Sets and Systems*, 1999. in press.

[6] J. S. R. Jang, C. T. Sun, and E. Mizutani. *Neuro-Fuzzy and Soft Computing.* Matlab Curriculum Series. Prentice Hall, 1997.

[7] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B*, 36(1):111–147, 1974.

[8] M. Sugeno and G. T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28:15–33, 1988.

[9] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(1):116–132, 1985.

[10] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, NY, 1995.