Predicting a chaotic time series using TSK with backpropagation as learning method

Christopher Bekos (mpekchri@auth.gr)

October 1, 2017

Abstract

Fuzzy systems consist of multiple rules, of which everyone is in the form :

$$R_i \ : \ IF \ x_1 \ is \ A_1^i \ AND \ ... \ x_m \ is \ A_m^i \ THEN \ y \ is \ B^i$$

where A_j are the fuzzy sets in the hypothesis section ,$x_1, ..., x_m$ are the inputs and y is the output of the system . Each input/output is a fuzzy set , in case of singleton inputs we may fuzzify them before we use them in our model , and then , in most cases , we need to deffuzify the output y .

These are the classic fuzzy models and are referenced in the literature as **Mamdani models**. Mamdani models can qualitatively describe the behavior of a system ,which allows user to understand it ,however , their accuracy of approach is not satisfactory , unless we add a huge amount of rules . Since adding new rules increases the complexity of the model , **new fuzzy TSK (Takagi-Sugeno-Kang) type models where developed** .

TSK models include unclear rules whose conception section is described generally by algebraic functions of the inputs and they can achieve better better accuracy using less rules .

In this document we will discuss the usage of such a TSK model in order to predict a chaotic time series (desired values will be created by the solution of the Mackey-Glass equation) .

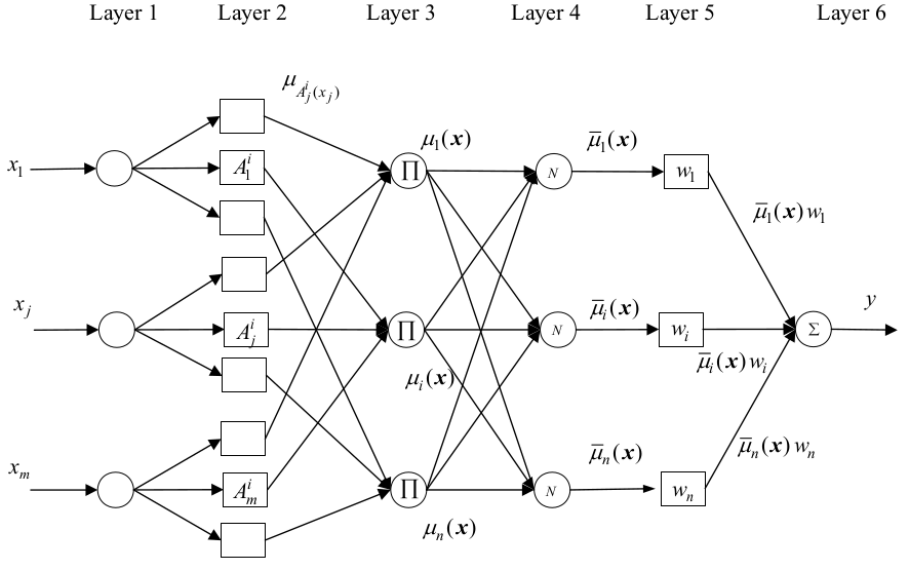
1 Zero order - TSK models

TSK rules may have different forms , regarding zero order TSK , we use singleton rules, which form is described as :

$$R_i \ : \ IF \ x_1 \ is \ A_1^i \ AND \ ... \ x_m \ is \ A_m^i \ THEN \ y \ is \ w^i$$

, where w_i is a real number .

Each rule R_i forms a fuzzy region which is described by the multidimensional fuzzy set $A_i = (A_1^i, A_2^i, ..., A_m^i)$. Singleton rule R_i depicts the fuzzy set A_i to a constant w_i in the conclusion section .The form zero order TSK is shown in the figure below :



1.1 Layer 1

Nodes in this layer are called input nodes ,and they just pass their input value to the next layer . Layer's 1 size is equal to the number of inputs.

1.2 Layer 2

Nodes in this layer are called linguistic term nodes , and they act as membership functions $\mu_{A_j^i}(x_j)$. Their output is represents degree of participation of input x_j input, to A^i fuzzy set .
Layer's 2 size is equal to $\sum_{j=1}^{m} k_j$, where k_j is the number of fuzzy sets in which input x_j is being divided.

1.3 Layer 3

Nodes in this layer are called rule nodes , and each node combines the inputs $\mu_{A_j^i}(x_j)$ and computes the degree of fulfillment $\mu_i(\mathbf{x})$, for each rule R_i .
Layer's size is equal to the number of rules .

1.4 Layer 4

Nodes in this layer compute the normalized degree of fulfillment for each rule (deffuzifier CAD may be used) .

1.5 Layer 5

Nodes in this layer calculate the weighted contribution of the conclusions section of each rule $\mu_i(\mathrm{x})w_i$, where μ_i is the output of layer 4 and w_i is the value of the conclusion of rule R_i .

1.6 Layer 6

This layer consists of only one layer, which summarizes all outputs of layer 5 and outputs the conclusion .

2 Initialization of the rule base

In general two main approaches may be used : **Grid partitioning** , where the total entry space is completely divided and the number of rules is determined in accordance with the number of fuzzy sets that are defined for each input x_j . Usually we define the fuzzy sets A_l for each input in such a way that there is enough overlap between them . The second approach is **Subtractive Clustering** which will be discussed in another project .

3 Learning method

Learning method aims at defining the proper w_i's (layer's 5 parameters) and the fuzzy sets for each input (layer's 2 parameters), in a way that our system is having the desirable behavior .
Here we will use **backpropagation** method in order to train our TSK. If we define a vector θ ,as the vector of all parameters for which we seek a better value, then we may say that each learning method aims to minimize some error $E(\theta)$, with respect to θ .

3.1 Error Criterion - MSE

For each input set x_i (be aware that x_i is a vector here) , we define error as :

$$E(\theta) = \tilde{y} \text{ - } y_d$$

where $\tilde{y}$ is the output of the TSK model to the x_i input , and y_d is the output of the real system

for the same input .
More often we use the **Mean Squared Error (MSE)** which is defined as :

$$E(\theta) = \frac{1}{M}\sum_{k=1}^{M}(\tilde{y} - y_d)^2$$

where M equals to number the of different inputs we use to train our model (training sets) .

3.2 Learning with backpropagation

Since we have define our Error function E(θ) we start a loop for t times where, for each set of inputs ,x_i , we compute the $E_i(\theta)$, and we update values of θ according to the equation :

$$\theta_i(t+1) = \theta_i(t) - n\frac{\partial E(\theta_i(t))}{\partial \theta_i(t)}$$

notice that t refers to the iteration of the loop ,while i refers to an input vector x_i, y_d^i, also n is called step size and is a vector (different step size for each θ's element) .
In order to compute the term $\frac{\partial E(\theta_i(t)(}{\partial \theta_i(t)}$ for each layer we need backpropagation . The idea is that we compute the system output for an input x_i (feedforward) and then we can easy compute $E(\theta_i)$ for the last layer . Now we backpropagate the error of layer 6 to the find the errors of the rest layers (namely this way we compute all values of vector $\frac{\partial E(\theta)}{\partial \theta}$).

3.3 Hybrid learning

Hybrid learning combines **BP** (backpropagation) method and **LSE** (Least Squares Estimate) method, in order to train our model . The Hybrid algorithm takes advantage of the fact that while the output of the TSK model is non-linear with respect to the parameters of the conclusion section (inputs to our model) , on the contrary, the output is a linear function with the parameters of the conclusion section (weights) .
According to the above , Hybrid algorithm uses LSE method for readjusting TSK's weights , and BP method in order to readjust the form of fuzzy sets for each input .
According to the literature,Hybrid learning achieves much better accuracy than BP when it is used to train all network parameters.

4 Strategies for parameter adjustment using datasets

In general, there are two strategies for weighting a network based on a dataset :
Pattern learning : where each sample is examined separately based on a series, and gives a specific correction .
Batch learning : where all data are processed simultaneously and give a total weight correction .

5 Train,check and validation sets

Most ML models are described by two sets of parameters. The 1st set consists in "regular" parameters that are "learned" through training. The other parameters, called hyperparameters or meta-parameters are parameters which values are set before the learning starts .
Obviously, different values for those parameters may lead to different (sometimes by a lot) generalization performance for our Machine Learning model therefore we need to identify a set of optimal values for them and this is done by training multiple models with different values for the hyperparameters (how to chose those values falls under the name of hyper parameter optimization https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning))
Now, imagine you have you data and you need to run a supervised ML algorithm on it. You split the data into :

- **training** - this is the data for which your algorithm knows the "labels" and which you will feed it to the training process to build your model.
- **test** - this is a portion of the data that you keep hidden from your algorithm and only use it after the training takes places to compute some metrics that can give you a hint on how your algorithm behaves. For each item in you test dataset you predict it's "value" using the built model and compare against the real "value" .

Now, back to the context of hyper parameter optimization. If you run the same algorithm (train on training, evaluate on test) for multiple sets of hyper parameter and chose the model with the best "performance" on the test set you risk over fitting this test set. To avoid this problem of over fitting the test data, the training set is split once more into :

- **actual training** - a subset of the training set that is used to optimize the model
- **validation** - another subset of the training set that is used to evaluate the model performance for each run / set of hyper parameter values.

Multiple training sessions are run on the actual training set, for various hyper parameter values and the models are evaluated again the validation dataset. The model with the best performance is then chosen - remember that so far the algorithm has not yet seen the test data therefore there is no suspicion of over fitting it. After choosing the best model (and implicitly the values for the hyperparameters) this model is evaluated against the test dataset and the performance is reported .

6 The Mackey-Glass equation

——- TO BE FILLED ——-

7 Datasets creation in matlab

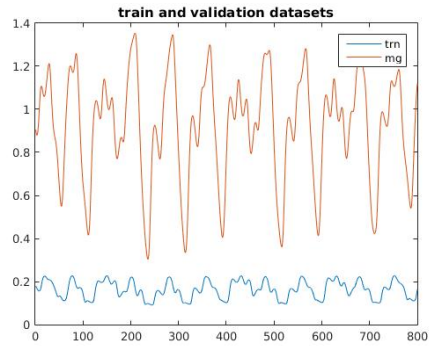
Script named 'create-trainning-sets.m' is used for this purpose . At first we load MG equation's solution for N = 3000 steps . We now define our train dataset (Dtrn) , check dataset (Dchk) and validation dataset (Dval) , avoiding the first 500 solutions in order to eliminate the effect of the initial value on the time series variation .
A dataset set D_s is created as : $D_s = [x(t-(q-1*p)\Delta), x(t-(q-2*p)\Delta), x(t), x(t+p)$ *(desired-output)* , index] .
Index is used in order to proper plot the results after we shuffle the datasets .
Now , in order to eliminate the autocorrelation's effect between close solutions of MG equations , we shuffle our data . Datasets are ready to be used .

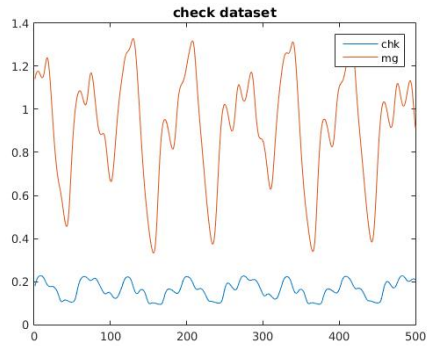
8 Task 1

We use a zero-order-TSK , with bell shaped membership functions (which are overlapping to about 0.5) . We train our model using BP and then using Hybrid learning method. Here are the results :

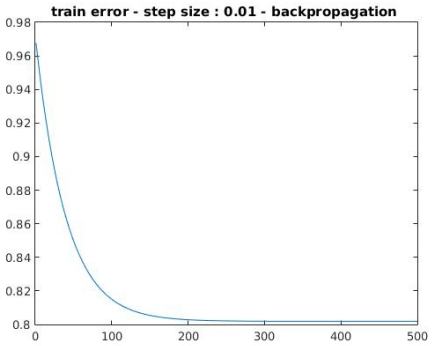
8.1 Learning using BP method

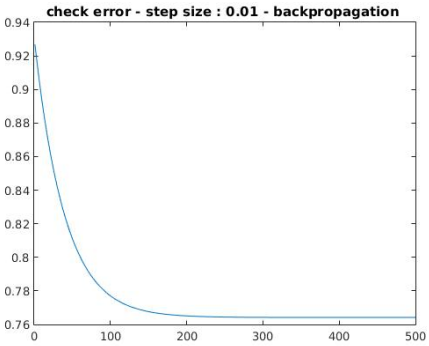
Estimations of our TSK model along with the desired values



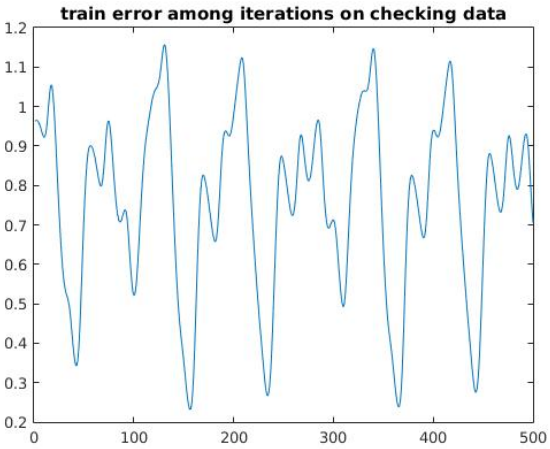


Train and check error among iterations (**Learning curves**)



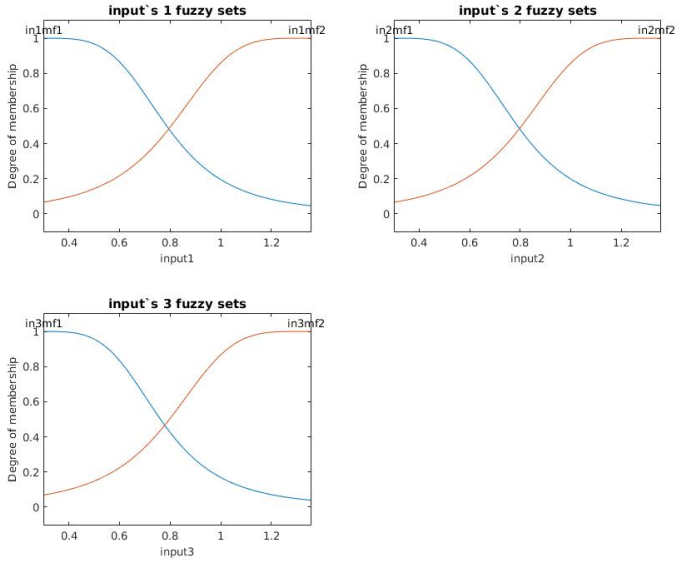


Estimation's error in check dataset



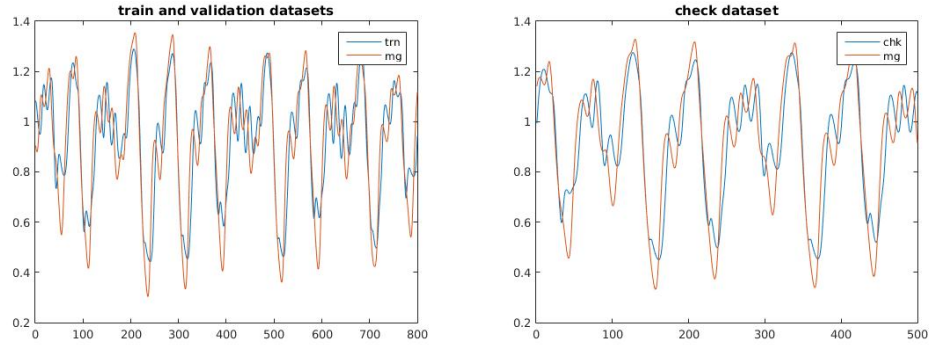
Results for error indicators :
MSE = 0.6254 , RMSE = 0.7908 , NMSE = 0.0017 , NDEI = 0.0406

Final membership function of the trained TSK model

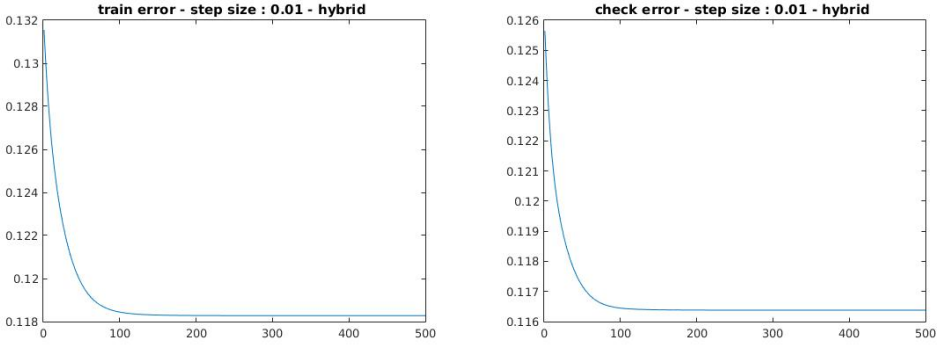


8.2 Learning using Hybrid method

Estimations of our TSK model along with the desired values

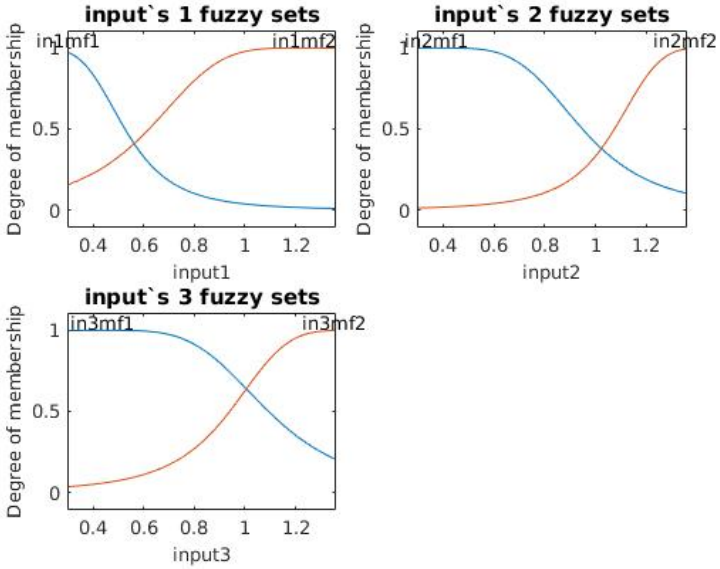


Train and check error among iterations (**Learning curves**)



Estimation's error in check dataset

Results for error indicators :
MSE = 0.0140 , RMSE = 0.1185 , NMSE = 0.0178 , NDEI = 0.1334
Final membership function of the trained TSK model

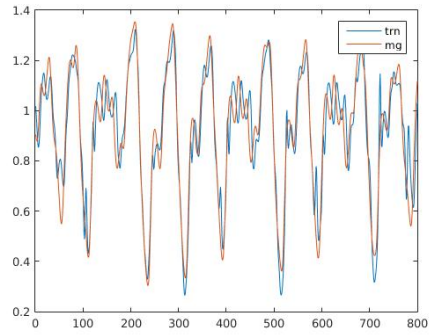


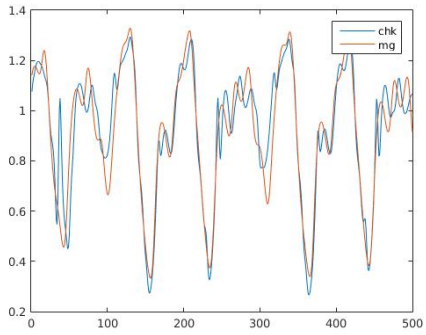
8.3 Conclusions

As expected ,since the Hybrid algorithm takes advantage of the fact TSK's output is a linear function with the parameters of the conclusion section , while BP completely ignores this fact , Hybrid method leads to better estimations .
Matlab's script 'ergasia4-question1-ver2.m' was used to create the above figures .

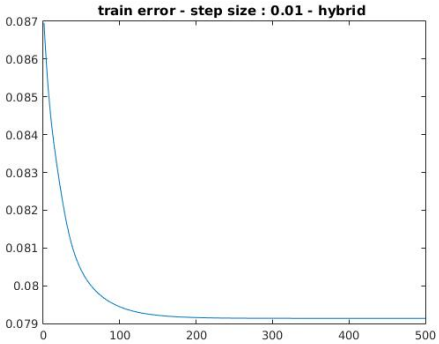
9 Task 2 - TSK with polynomial functions in the conclusion section

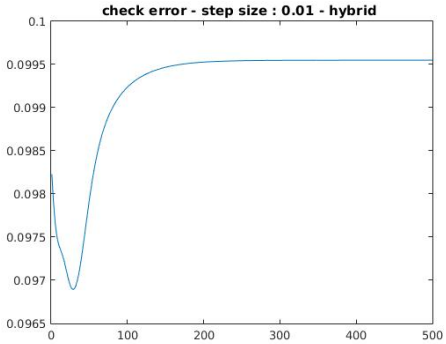
Estimations of our TSK model along with the desired values





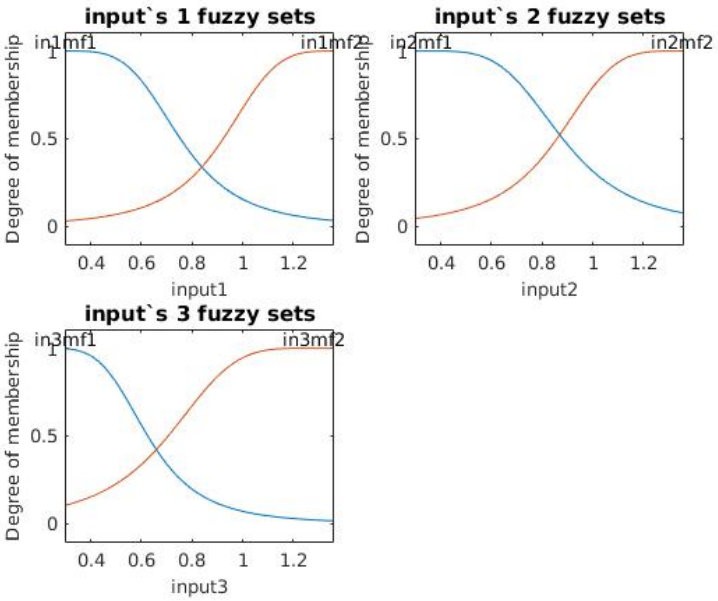
Train and check error among iterations (**Learning curves**)





Estimation's error in check dataset

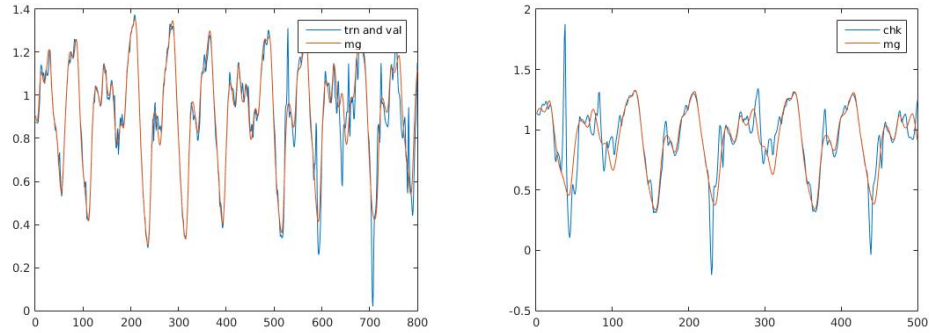
Results for error indicators :
MSE = 0.0091 , RMSE = 0.0954 , NMSE = -0.0123 , NDEI = 0.1107
Final membership function of the trained TSK model



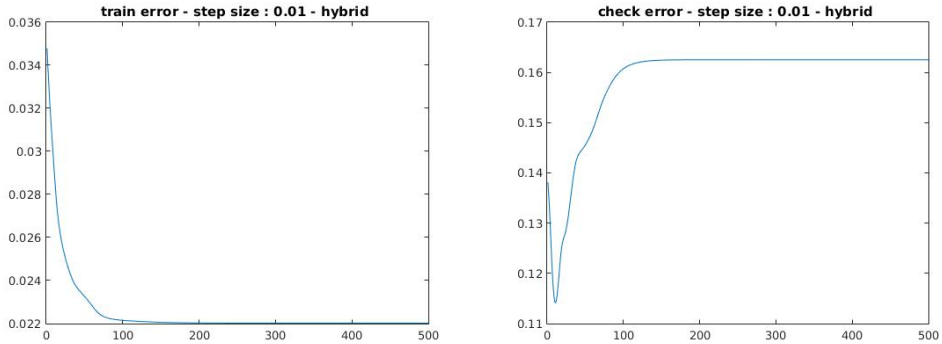
Matlab's script 'ergasia4-question1-ver2.m' was used to create the above figures .

10 Task 3

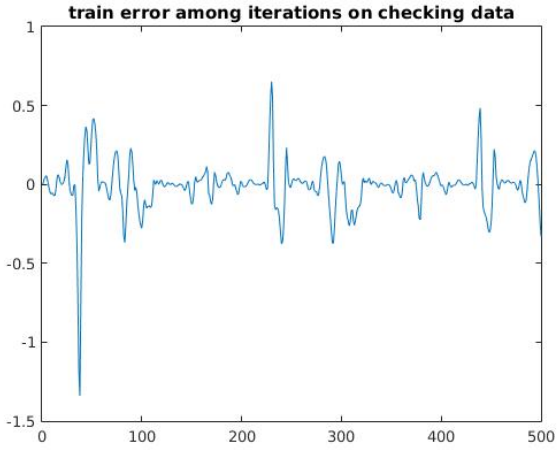
We may say that our previous TSK model results poor estimations , so a simple thought for increasing it's accuracy could be increasing the number of membership functions used by our model . Well the answer is not that simple ,because increasing the number of membership functions may cause our system to **Over-learning** as shown below :
Estimations of our TSK model along with the desired values



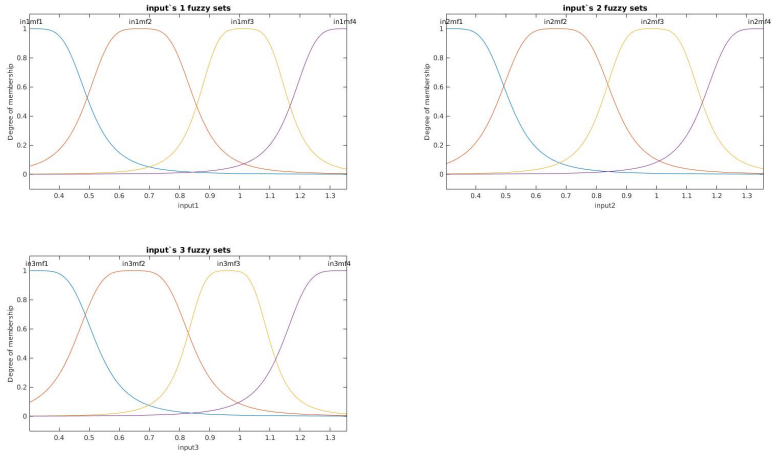
Train and check error among iterations (**Learning curves**)



Estimation's error in check dataset



Results for error indicators :
MSE = 0.0249 , RMSE = 0.1577 , NMSE = -0.0044 , NDEI = 0.0666
Final membership function of the trained TSK model



As you can clearly see , because we use more membership functions , our model's estimations are fitted in training set way too good , but they can not fit in check or even validate datasets . This is called Over-learning . **It has to be mentioned that in order to emphasize the possibility of occurrence of the phenomenon of Over-learning when using more MFs , the above results where calculated without datasets shuffling . However , if we combine the usage of validation dataset and dataset shuffling , we get better results .**
Matlab's script 'ergasia4-question3.m' was used to create the above figures .

11 Task 4

Until now, all of our models were created assuming that the total entry space is completely divided and the number of rules is determined in accordance with the number of fuzzy sets that are defined for each input (grid partitioning) . As mentioned above , there is a second approach , named **Subtractive Clustering** .

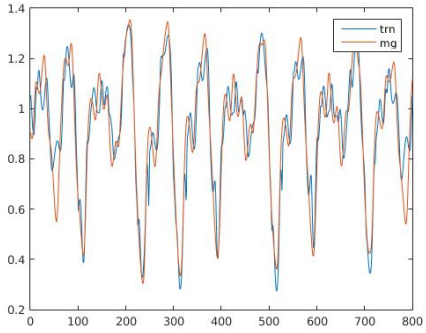
11.1 Subtractive Clustering

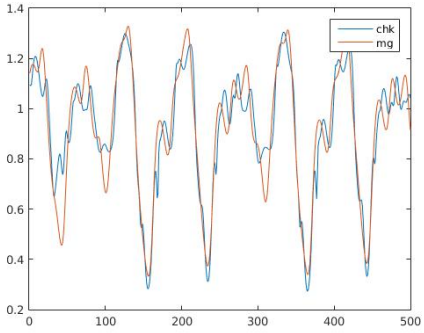
Suppose we don't have a clear idea how many clusters there should be for a given set of data. Subtractive clustering, , is a fast, one-pass algorithm for estimating the number of clusters and the cluster centers in a set of data. The cluster estimates obtained from the subclust function can be

used to initialize iterative optimization-based clustering methods (fcm) and model identification methods (like anfis). The subclust function finds the clusters by using the subtractive clustering method.
The genfis2 function builds upon the subclust function to provide a fast, one-pass method to take input-output training data and generate a Sugeno-type fuzzy inference system that models the data behavior.

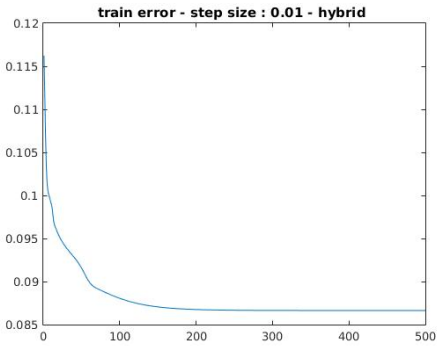
11.2 Results of TSK model with 3 clusters

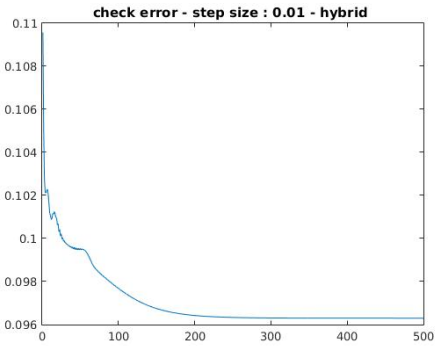
Estimations of our TSK model along with the desired values





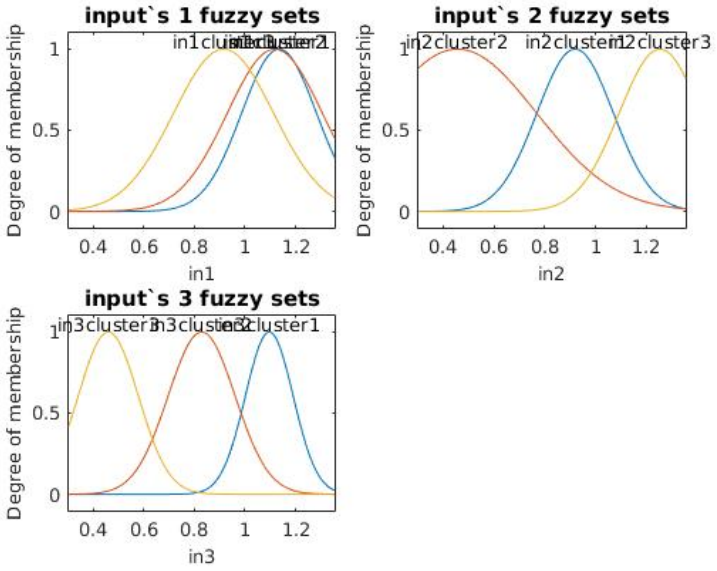
Train and check error among iterations (**Learning curves**)





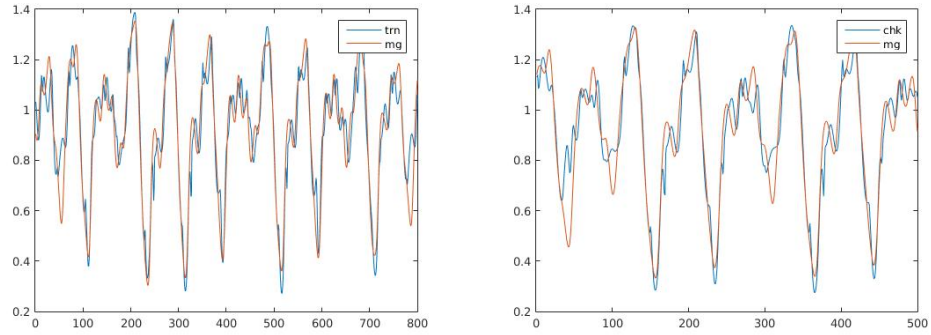
Estimation's error in check dataset

Results for error indicators :
MSE = 0.0086 , RMSE = 0.0926 , NMSE = 0.0063 , NDEI = 0.0795
Final membership function of the trained TSK model

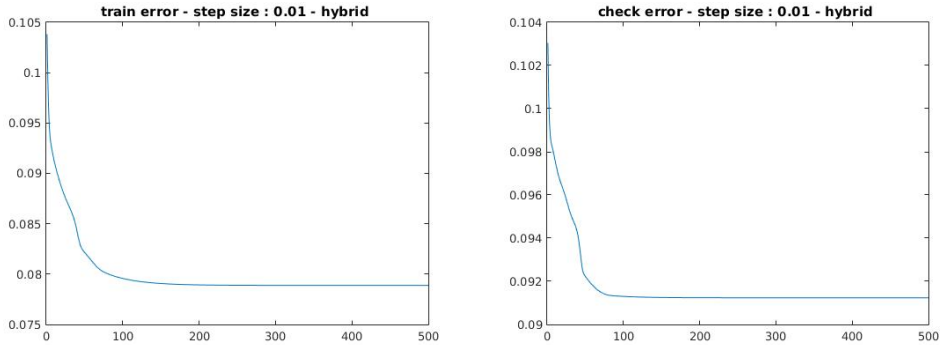


11.3 Results of TSK model with 4 clusters

Modifying the subclast options we got a TSK with 4 clusters.
Estimations of our TSK model along with the desired values

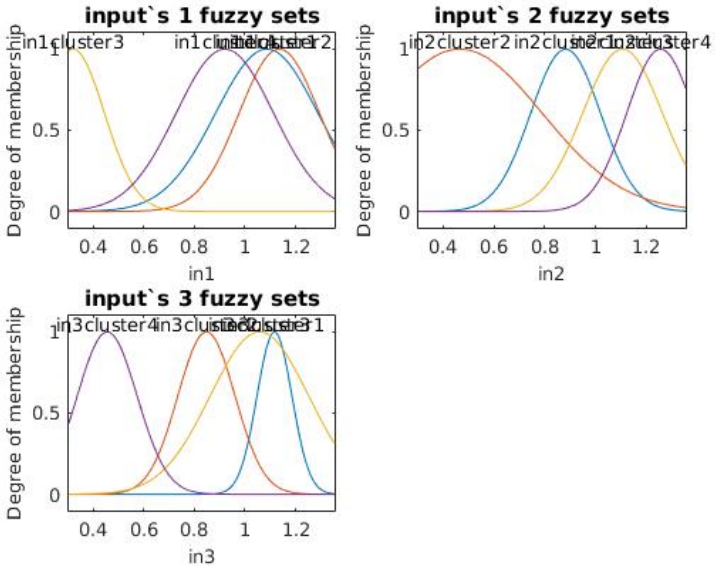


Train and check error among iterations (**Learning curves**)



Estimation's error in check dataset

Results for error indicators :
MSE = 0.0076 , RMSE = 0.0869 , NMSE = 0.0052 , NDEI = 0.0723
Final membership function of the trained TSK model

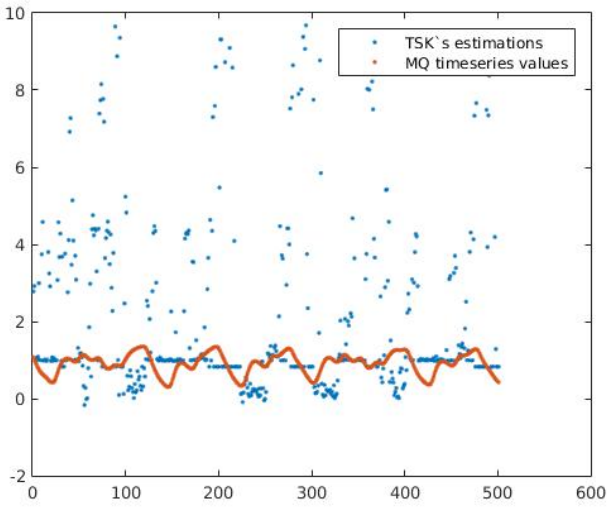


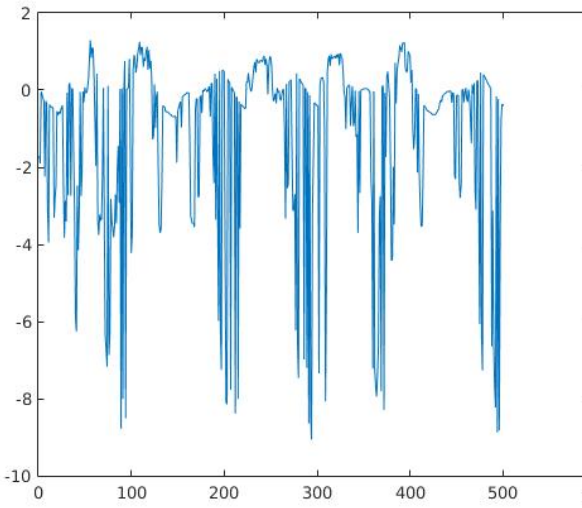
11.4 Conclusions

From all the TSK models tested , the one using subtractive clustering for defining input's fuzzy sets , hybrid algorithm for learning and polynomial functions in the conclusion section had the best performance .
Matlab's script 'ergasia4-q4.m' was used to create the above figures .

12 Multiple step ahead prediction

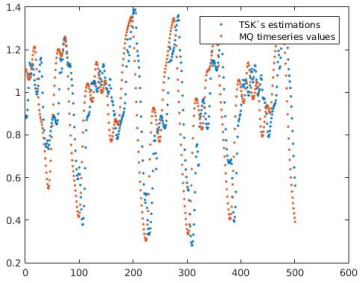
Using the TSK model that we created in task4 (the one with 4 clusters) we will make predictions of MG equation for P = 96 steps ahead . Each prediction is created as :
$x(t-p), x(t), \tilde{x}(t+p)$ -> $\tilde{x}(t+2*p)$
$x(t), \tilde{x}(t+p), \tilde{x}(t+2*p)$ -> $\tilde{x}(t+3*p)$
$\tilde{x}(t+p), \tilde{x}(t+2*p), \tilde{x}(t+3*p)$ -> $\tilde{x}(t+4*p)$ and so on ...
Script 'ergasia4-q5.m' simulates this processes :
For P = 96 and N = 500 iterations, we have :



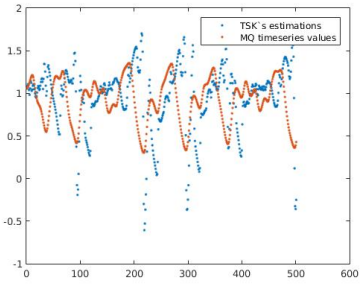


Results for error indicators :
MSE = 7.0077 , RMSE = 2.6472 , NMSE = 4.8470 , NDEI = 2.2016

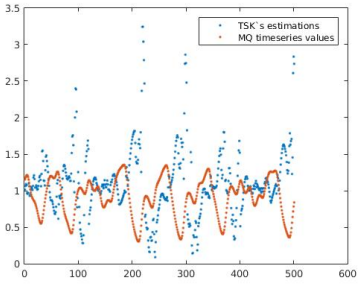
For some other choices over P , we got :



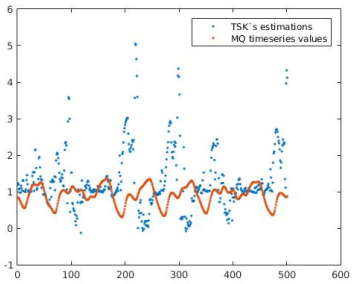
P = 18



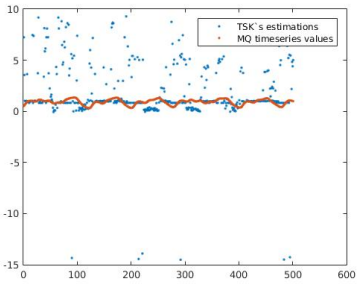
P = 24



P = 30



P = 48



P = 120

It's clear that Multiple step ahead prediction method "adds" the error of each prediction , resulting a bad prediction for P greater than some value (even a small one).
For applying Multiple step ahead prediction you may use script 'ergasia4-q5.m'