Intelligent Systems Course Project

ISE100 INDEX FORECASTING

Supervisors:
Beatrice Lazzerini
Eleonora D'Andrea

Authors:
Alessio Apollonio
Daniele Battista

Computer Engineering
A.Y.2013/2014

Contents:

1. Introduction

The aim of the course project is to predict the value of ISE100 (Istanbul Stock Exchange), market index of Istanbul, based on previous values of this index or on previous values of other Market indexes.
All other indexes were chosen from a set of international indexes like: SP (Standars & Poor's), DAX (Stock market return index of Germany), FTSE (Stock market return index of UK), NIKKEI (Stock market return index of Japan), BOVESPA (Stock market return index of Brazil), EU (MSCI European index) and EM (MSCI emerging markets index).
All data collected are time series and they consist of 536 observations collected between January 5, 2009 to February 22, 2011.
The project needs the use of Matlab, in particular the use of the Neural Network Toolbox and the Genetic Algorithms.
We tried to solve all these problems in some steps, executing scripts created by Matlab and then modified to suit our purpose. Some values created by the functions we wrote and developed, were reported in order to evaluate some features of the system we were going to analyze and were used to develop all next steps.

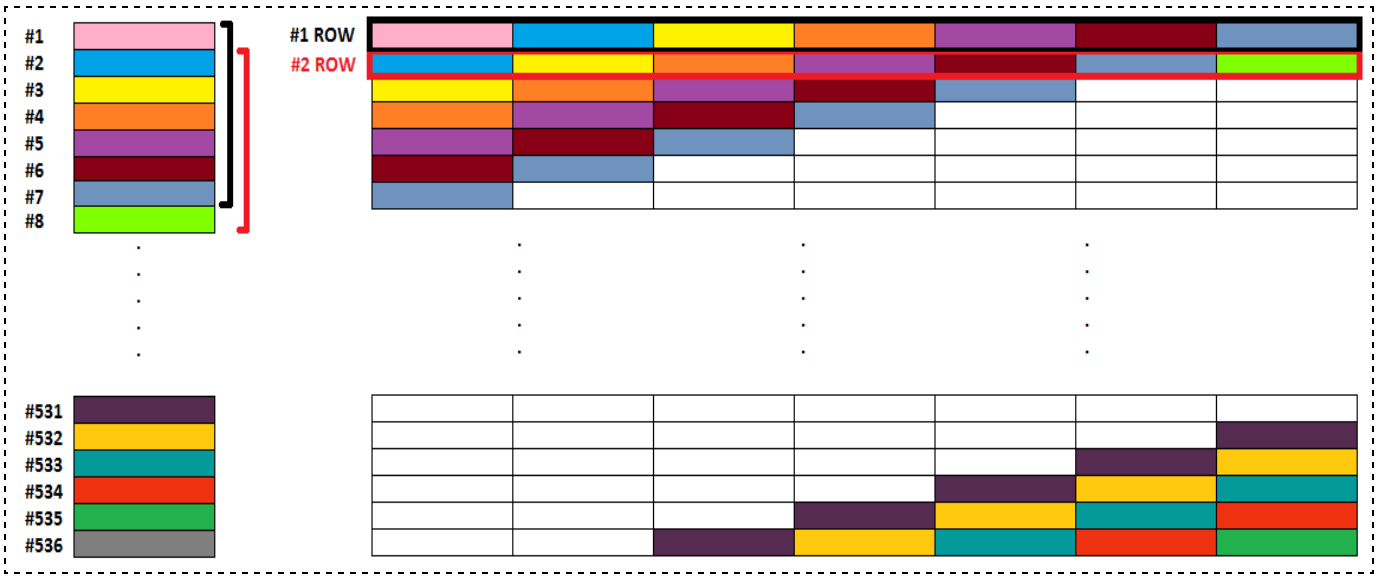
2. Feature Selection

In the very first phase of the project we were asked to use all indexes as predictors for ISE. We were asked to analyze all of them in order to discover which ones were the best to our goal. For each index we had to develop a neural network to predict the following observation of the ISE100, taking as inputs last seven observations of the index under analysis. Once done, we were asked to choose a number of best predictors useful for next step.

2.1. Constructing Data

In the file we download from the repository provided, we found a matrix of values organized this way: the matrix is composed of 10 columns, one for the date of the observation, 2 for the ISE100 values expressed in American Dollars and Turkish Lira, last seven for the values of all other indexes. On all rows there is the representation of all observations. We decided to take in consideration the ISE value expressed in American dollars. As far as our intent is to predict the following observation of ISE

considering last seven observations of one of the indexes listed above, we decided to build a matrix for each index organized this way: the matrix has 7 columns and 535 rows. 7 rows represent the 7 different days in which we take data, all rows represent sets of data useful to do the prediction. We excluded from our matrix the value for each index for the fact that we do not need it for our work.
All these considerations have been done for inputs. Passing to outputs, we created a vector done this way: it is a column vector that contains all data from the eighth observation (that is the first one to be predicted) to the last one, so that the number of rows coincides with the one of the matrix.



Of course, when we are going to use these data, we are not going to pass them to the network this way, but we have to transpose them in order to have 7 inputs and 1 outputs.
The code that executes this part is reported in the file *create_matrix.m* (where .m extension contain MATLAB code, either in the form of a script or a function)

```
function index_matrix = create_matrix( index )
        for i=1:529
        index_matrix(i,1) = index(i);
        index_matrix(i,2) = index(i+1);
        index_matrix(i,3) = index(i+2);
        index_matrix(i,4) = index(i+3);
```

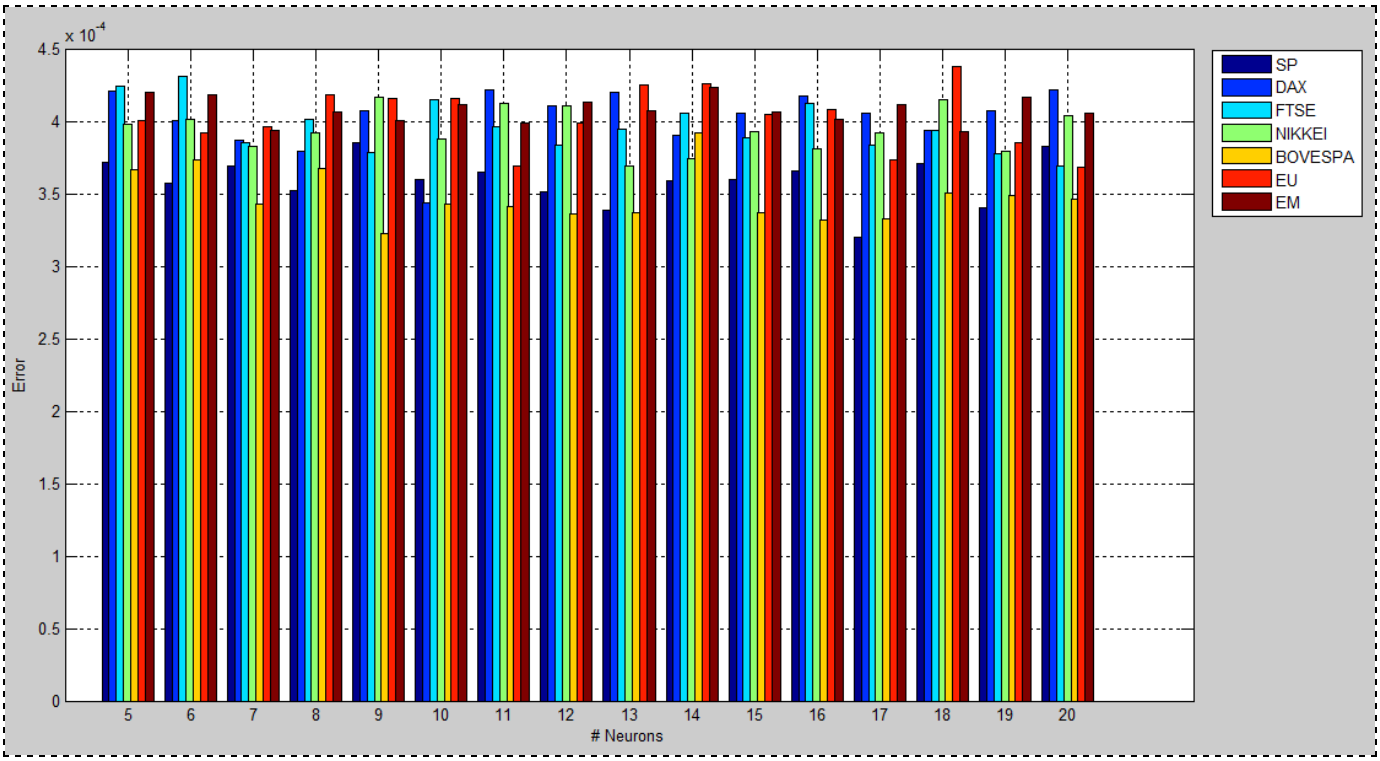
```
            index_matrix(i,5) = index(i+4);
            index_matrix(i,6) = index(i+5);
            index_matrix(i,7) = index(i+6);
            end
    end
```

2.2. Fitting

In this phase we started the interesting part of the project: we used the Neural Network Toolbox to see how good were our indexes at predicting our ISE100. We decided to develop a MLP neural network with our matrices as inputs and our ISE100 vector as output to predict.
Data had been saved into a file *dati_progetto.mat*, where the *.mat* extension specifies that our file contains MATLAB formatted data. This step is done in order to make possible to load the data by the command line:

```
load('dati_progetto.mat')
```

Thanks to *nnstart* tool we obtained the simple script that will be modified. Data were divided as Matlab does normally for each training in three parts: 70% for training, 15% for validating and 15% for testing.
As far as we were not aware of how many neurons to insert in out network, we implemented a solution in which there is a number of neurons going from 5 to 20. For each number of neurons we trained the network 10 times and to know if the chosen number was good or not, we summed all MSE values reported at the end of each training and, at the end of the 10 trainings, we computed the mean value, dividing this sum by a factor of 10. These operations had been done in order to achieve a statistical convergence.
Doing so, we saved all values of number of neurons and relative mean error in a matrix, from which we obtain a graph we show below.



Here we show the modified code from *train_function.m* that implements the above operations. The code refers to the nested loop cycles that implement the 10 trains referred to the network trained with different hidden layers number:

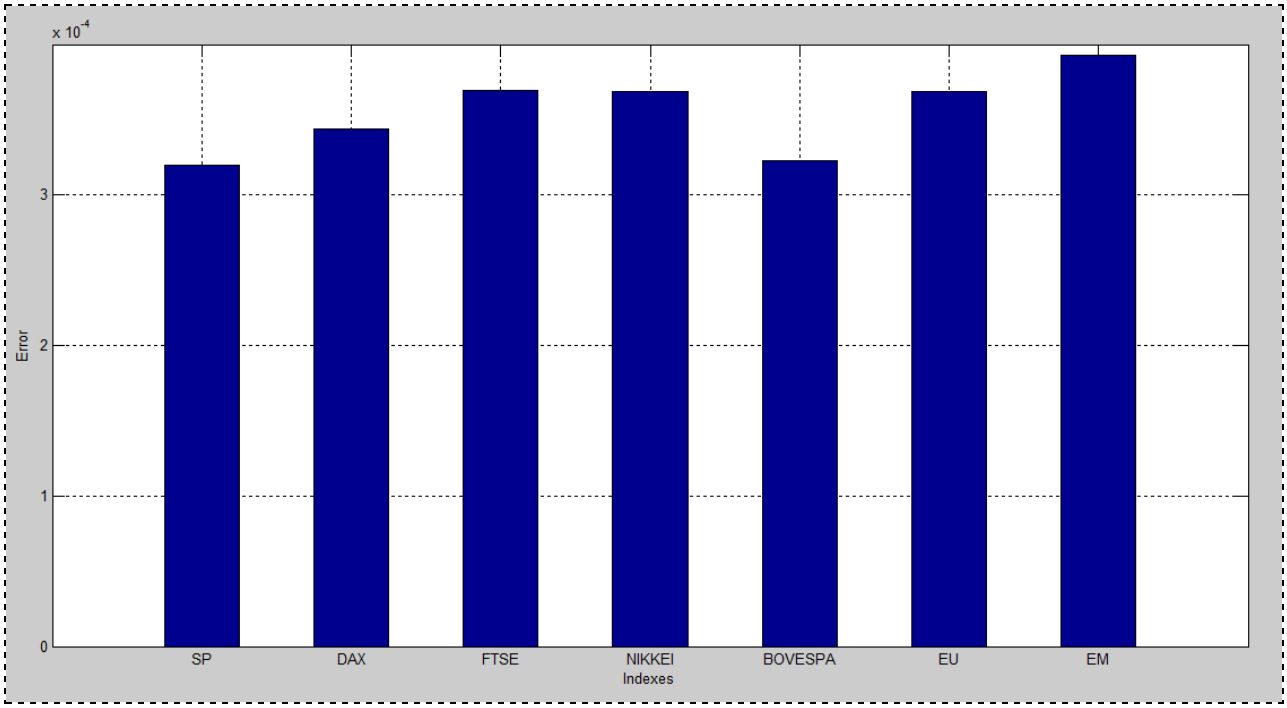
```
for i=5:20

    performance = 0;
    perf=0;

    hiddenLayerSize = i;
    net = fitnet(hiddenLayerSize);

    net.divideParam.trainRatio = 70/100;
    net.divideParam.valRatio = 15/100;
    net.divideParam.testRatio = 15/100;

    % Different train for statistical convergence
    for y=1:10
```

```
        [net,tr] = train(net,inputs,targets);

        outputs = net(inputs);
        % errors = gsubtract(targets,outputs);
        perf = perform(net,targets,outputs);

        performance = performance + perf;

        end

        mean = performance/10;
        results(i-4) = mean;

    end
```

Notice that *train_function.m* returns a row vector.
It will be *main_sheet.m* that will put together these different row vectors into a single matrix.

2.3. Classification

Once the matrix with number of neurons and relative mean error was created, we analyzed it to be aware of our best indexes and their related number of neurons.
Analyzing results coming from some simulations of this step, we saw that SP and BOVESPA were the two indexes that best predict the values of ISE. So, taking this into account, the minimum number of indexes to choose was for sure 2.
Then we discovered that sometimes DAX, FTSE, NIKKEI were better than the two indexes mentioned before in predicting ISE. We were not able to explain why, but our idea is that, once we do a new simulation, a new network is created and so we have new weights assigned to all branches. The position occupied by last three mentioned indexes in a possible indexes ranking was very rarely the first one and, a little bit less rarely, the second one. For sure one of them was able to occupy the third position. Only in one try we observed that the third position was occupied by EU.



For this reasons, we decided to take in consideration 3 indexes as best predictors.
The research of the three best indexes is done in *get_ranking.m* file. Here we do not show the whole code, but the part referred to one of the indexes and the part referred to the research of the best index:

```
min_SP = min(errors_matrix(1,:));
[rowSP,colSP] = find(errors_matrix(1,:) == min_SP);
ranking(1,1) = min_SP;
neurons_ranking(1,1) = colSP + 4;

...

ranking_2 = ranking;
best = find(ranking_2 == min(ranking_2));
ranking_2(1,best) = 1;
best_indeces(1) = best;
best_indeces(4) = neurons_ranking(1,best);
```

An important consideration is that, at this point, we have already stated that:

- three indexes will be used for next step;
- our network will be developed with a number of neurons equal to the maximum number between the three received from the first step.

The last assumption has to be explained this way: we have three indexes that, with a certain number of neurons, predict with the minimum error ISE100 index. But each of them, has a different number of neurons through which reaches the minimum error. As far as we have to decide a number good to continue our work, we choose the maximum for the fact that with more neurons there is more computation and probably a better prediction.

3. Forecasting Model Development

In this phase of the project we were asked to divide the available data in two periods, called estimation period and forecasting period, and to develop, through the *Time Series Tool*, a network that is able to predict the movement of ISE100 index. What we want to find is a couple of delays that is good for our job. To do this, we are advised to use a *Genetic Algorithm* that could give us a good couple of delays. We were also asked to evaluate the forecast accuracy through MSE, MAPE and the percentage of correctly forecasted days.

3.1. GA Development

We decided to proceed this way: to use a genetic algorithm for sure we need a fitness function that is able to tell us which is the best couple of delays. How to choose a suitable fitness function for our job?
Our intent is to see which is the couple of delays that minimize the error of the network we are going to generate. Since we fixed the number of neurons, we have to develop a network with the *Time Series Tool* that evaluates the error. So, our fitness function will be a *Time Series* that will report us the best couple to our purpose. The simulation requires a lower and an upper bound. While the upper bound values was trivial chosen as 1,

we decided as upper bound the value of 25. The upper bound could also be chosen different, but, after different simulations we notice that:

- when the upper bound was a small value (up to 10) the simulation was always returning the maximum available value;
- when the upper bound was greater than 30, the simulation returned a value between 20 and 25.

Considering this, the number of variables that has to be optimized is 2, as we declared in our script.
We decided that our GA will work on the following options:

- population of 30 individuals;
- number of generations equal to 70;
- crossover probability equal to 0.8;
- crossover function equal to crossover single point.

The last two options are taken following the default configuration of the *gaoptimset* function. The first two parameters, however, had been decided in order to obtain GA that is able to provide an acceptable result. Basically a *GA* simulation should require about a hundred of generations. We noticed that this number has lead to a very long simulation, so we decided to use a number of generations that could give us a good tradeoff between simulation time and goodness of result.
As far as the GA returns us a couple of delays, that generally is represented by two real number, we rounded them to nearest integer with the Matlab function *round()*.
Here we show the function code that is put into *genetic_algorithm.m* function:

```
function genetic_results = genetic_algorithm()

Fitfunction = @FitnessFunction;

numVar = 2;

LOWER_BOUND = [1 1];
UPPER_BOUND = [25 25];

options = gaoptimset('PopulationSize',30, 'CrossoverFraction', 0.8, 'Generations',
70, 'CrossoverFcn', @crossoversinglepoint);
```

```
[result_delays,fval,exitFlag,output]=ga(Fitfunction,numVar,[],[],[],[],LOWER_BOUND,
UPPER_BOUND,[],[],options);

genetic_results = [round(result_delays(1)) round(result_delays(2)) fval];

end
```

3.2. Fitness Function

Passing to the Fitness Function, we can say that it has been developed a *Open-loop* network through the *Time Series Tool*, that tries, with different delays that are provided by the *GA*, to see which is the best couple of them. Once a couple has been chosen, the created network is trained 10 times and the process to evaluate the goodness of the couple is done as said for the previous step: the sum of the error is divided by 10 in order to have the mean.
Data needed to this aim have been constructed this way: as far as we need data coming from the so-called estimation period, we built a matrix with 3 columns, one for each of the three best indexes and a number of rows equal to the number of data that make part of the estimation period. This part is implemented in the first part of our function.
In order to conclude we now show the code that is put into *FitnessFunction.m*:

```
input_data_indeces = [ matrix1( 1:268, best_indeces(1) ), matrix1( 1:268,
best_indeces(2) ), matrix1( 1:268, best_indeces(3) ) ];
ise_data_index = ISE1(1:268);

inputSeries = tonndata(input_data_indeces,false,false);
targetSeries = tonndata(ise_data_index,false,false);

neurons = [best_indeces(4) best_indeces(5) best_indeces(6)];

inputDelays = 1 : result_delays(1);
feedbackDelays = 1 : result_delays(2);
```

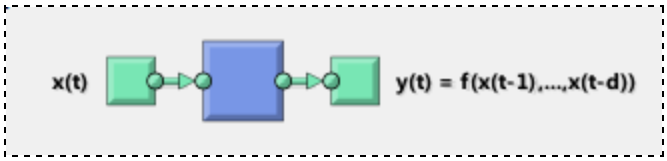
```
hiddenLayerSize = max(neurons);

net = narxnet(inputDelays,feedbackDelays,hiddenLayerSize);

[inputs,inputStates,layerStates,targets] = preparets(net,inputSeries,{},targetSeries);

net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

perf = 0;

% Loop used for statistical convergence
for i=1:10

    [net,tr] = train(net,inputs,targets,inputStates,layerStates);

    outputs = net(inputs,inputStates,layerStates);
    % errors = gsubtract(targets,outputs);
    perf = perf + perform(net,targets,outputs);

end

performance = perf/10;

end
```

At the end of these parts we have obtained:

- the best indexes that will be used;
- the chosen number of neurons;
- the delays values that will be next used (mainly done in this part).

4. One-step-ahead Forecasting

We now use a *Time Series* in order to create an *Open-loop network* that makes possible a *one-step-ahead* forecasting.
With *open-loop network* we are referring to a network where the output (our prediction) is not fed back to the input of the network itself. The concept is explained by the following picture:



This kind of network does not require the feedback delay, but only the input one and this is the main difference between the *open-loop network* and the *closed-loop* one (that will be described in a while).
The lack of the feedback delay, moreover, makes the *open-loop network* able to predict only the ISE100 value of the next day (we are looking one step in the future). This is the reason why we refer to this type of forecasting as *one-step-ahead*.
Similarly to the fitness function, we will give in input the first half part of the previously found best indexes values (the so called estimation period) in a network that works with the maximum number of neurons (between the ones just found).
Also in this case, for statistical convergence, for each iteration we train the network 10 times and take the mean of the computed error.
All the output are saved and computed in the way described in the chapter 4.1.
However, the most important result in this step is given by the saving of the trained network.
This is made possible by the following command:

```
save('network.mat','net')
```

As all the file with extension *.mat*, the network will be loaded and used at any time we need it (we will infact use it in the *n-step-ahead forecasting* part that is based on a *closed-loop network*).

We are now ready to show the code relative to this part, code that is saved into the *openLoopTraining.m* function:

```
function open_loop_performances = openLoopTraining(genetic_results)

load('usage.mat');
load('complete_data.mat');

input_data_indeces = [ matrix1( 1:268, best_indeces(1) ), matrix1( 1:268, best_indeces(2) ), matrix1( 1:268, best_indeces(3) ) ];
ise_data_index = ISE1(1:268);


inputSeries = tonndata(input_data_indeces,false,false);
targetSeries = tonndata(ise_data_index,false,false);
neurons = [ best_indeces(4) best_indeces(5) best_indeces(6) ];

inputDelays = 1:genetic_results(1);
feedbackDelays = 1:genetic_results(2);
hiddenLayerSize = max(neurons);
net = narxnet(inputDelays,feedbackDelays,hiddenLayerSize);

[inputs,inputStates,layerStates,targets] = preparets(net,inputSeries,{},targetSeries);

net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

performance = 0;

for t=1:10

        [net,tr] = train(net,inputs,targets,inputStates,layerStates);
```

```
        outputs = net(inputs,inputStates,layerStates);
        % errors = gsubtract(targets,outputs);
        performance = performance + perform(net,targets,outputs);

end

performance = performance/10;

perc_outputs = cell2mat(outputs);
perc_targets = cell2mat(targets);

open_loop_performances = calculate_performances(perc_outputs, perc_targets,
performance);

save('network.mat','net');

end
```

4.1. Performance Calculation

As requested in the project, to evaluate the goodness of our solution, we have to use three indicators like MSE (mean squared error), MAPE (mean absolute percentage error) and percentage of correctly forecasted days.
MSE has been calculated through the function *perform()* provided by Matlab on the number of evaluations or trainings done.
MAPE has been calculated through the related formula:

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{A_i - F_i}{A_i}\right| \%$$

where n is the number of samples predicted, A_i is the actual value and F_i is the forecasted value.

The percentage of correctly forecasted days could have been calculated in a lot of ways, but at the end we decided to implement a solution that considers the values correctly forecasted if an increment in the ISE100 target values correspond to an increment into the forecasted ones (in other words, when forecasted days follow the trend of the ISE values). This is done thanks to the following code put into *calculate_performances.m* function:

```
l = length(targets);

final_perc= 0;

        if (sign(targets(2) - targets(1)) == sign(outputs(2) - targets(1) ) )
                final_perc = final_perc + 1;
        end

for x=2:(l-1)
        if (sign(targets(x+1) - targets(x)) == sign(outputs(x+1) - outputs(x) ) )
                final_perc = final_perc + 1;
        end
end

final_perc = (final_perc / (l-1))*100;
```

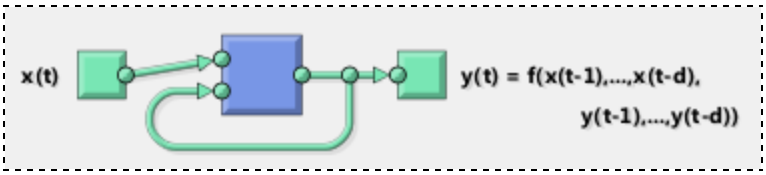
The estimation is made by considering the sign of the difference of two subsequent target values and the sign of the difference of the two forecasted values. If the value is increasing the sign function will return "1", whereas "-1". By comparing the sign we are able to understand if the two trend have the same direction. There are, however, two different situations to consider:

- ❏ the first forecasted value;
- ❏ from the second forecasted value on.

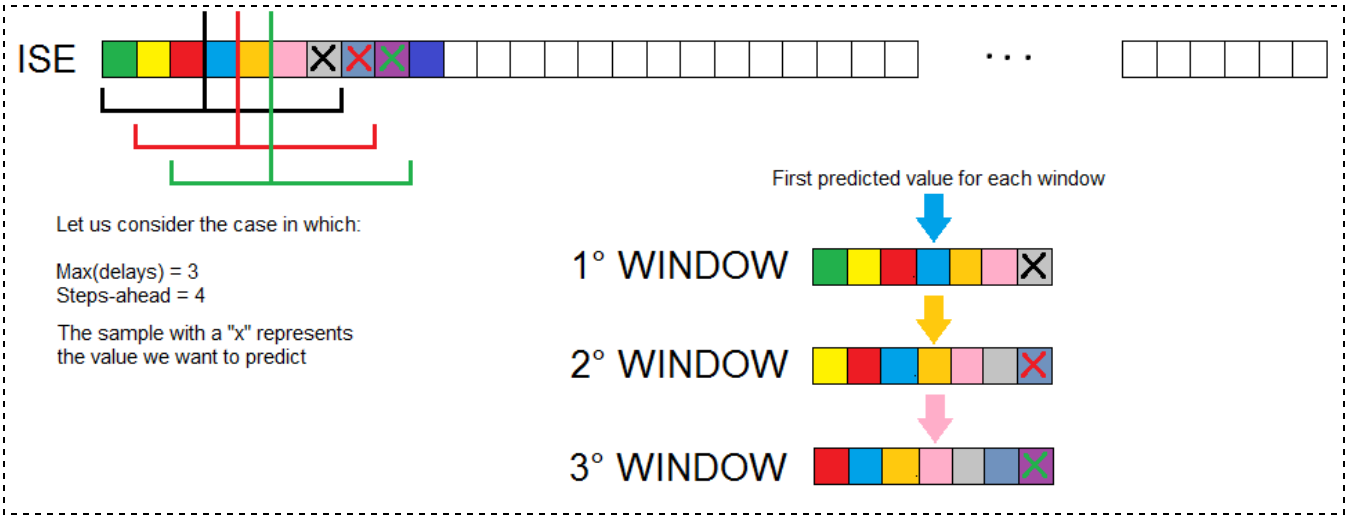
For the first forecasted value we do not have an output(1) value because we start forecasting from the second day on. We consider the difference between the first forecasted value and the first available target value. From the second forecasted value on we will consider the difference between subsequent forecasted values.

5. n-step-ahead Forecasting

In this part of the project we are arrived probably at the most interesting phase: the one in which we already have the network saved in our workspace and we want to know how good it can predict the value of ISE100 index some days ahead starting from “today”. The last sentence wants to say that we have a set data referred to certain days and we want to predict the value of ISE for a day that is after these certain days. Of course in this part we are going to work on the second part of the data that are constructed as said above. Moreover now, to do this part and to predict some steps ahead, we have to use a closed-loop network, able to use not only data coming from the input, but also the predictions coming from the output, computed in the previous steps. The figure below explains which kind of network we are going to use.



The day chosen for the forecasting might be just the following one, that represents the case shown above, or a day “in the future”. For this reason, we have to prepare data like windows of days that will be used to predict our values. This windows will be made of a number of days equal to number of steps we want to predict plus the maximum number of delays achieved by the previous part.
To be clearer, first of all we show an image that explain the windows creation:



This window is constructed this way because the maximum number of delays is already chosen by the Tool and so the first predicted day will be the first available after the ones used for closing the first loop. Instead we add a number equal to steps for the fact that our prediction aims to know the day indicated by the step. We create two windows, one for the inputs and one for the targets. The computations on these windows will return a vector containing the forecasted values.
Once all these data have been prepared, we pass to the part in which we take the network we already have, we close the loop and we do a number of trial equal to 268 minus the maximum of delays minus the number of steps plus one. This number has been obtained calculating the number of windows that can be built with these parameters.
In our implementation we decided to predict the value of ISE with a number of steps that goes from 1 to 9. We had chosen this maximum number of steps because, during a simulation up to 50 steps, we noticed that the MAPE was increasing more and more, by indicating a not satisfying result.
At the end of the day, for each of these trials, we observed the value of MSE calculated on the number of evaluations done, the value of MAPE and the percentage of correctly forecasted days.
This is the code that implement the *closed-loop network*.
It is put into the *closedLoopTraining.m* function:

```
delays = [ genetic_results(1) genetic_results(2) ];

input_data_indeces = [ matrix1( 269:536, best_indeces(1) ), matrix1( 269:536, best_indeces(2) ), matrix1( 269:536, best_indeces(3) ) ];
ise_data_index = ISE1(269:536);

for e=1:9

    evaluations = 268 - max(delays) - steps + 1;

    performance = 0;
    ending_outputs = zeros(1,steps);
    ending_targets = zeros(1,steps);

    for i=1:evaluations
```

```
window_input_data_indeces = input_data_indeces(i:i+steps+max(delays) - 1,:);
window_ise_data_index = ise_data_index(i:i+steps+max(delays) - 1);

close_input_Series = tonndata(window_input_data_indeces,false,false);
close_target_Series = tonndata(window_ise_data_index,false,false);

netc = closeloop(net);

[close_inputs,close_input_States,close_layer_States,close_targets]=preparets(n
etc,close_input_Series,{},close_target_Series);

close_outputs = netc(close_inputs,close_input_States,close_layer_States);
performance = performance + perform(netc,close_targets,close_outputs);

a = cell2mat(close_outputs);
b = cell2mat(close_targets);

ending_outputs(i) = a(steps);
ending_targets(i) = b(steps);

end
performance = performance/evaluations;

closed_loop_performances(e,:) = calculate_performances( ending_outputs,
ending_targets, performance)
 end
```

6. Prediction through Fuzzy Systems

In the last part of the project it is requested to develop a *fuzzy system* which forecasts the ISE100 index based on previous values of the indexes selected as best predictors for it, using the *Anfis* system or the one that uses *Mamdani fuzzy rules*.
Our intention was to proceed using first the *Anfis* system to have *Membership Functions* already generated and then export them in order to use the same *Membership Functions* with *Mamdani fuzzy rule*. As far as trying to produce Membership functions from scratch without knowing how data are formed, we exploit the *Anfis system* to have them.

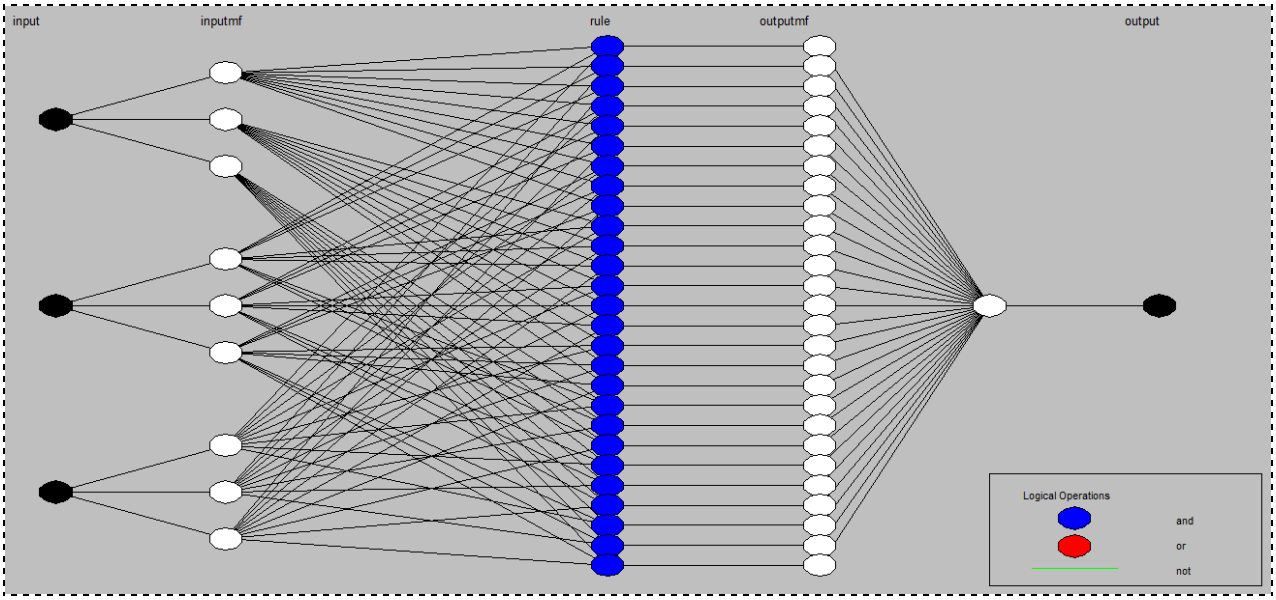
6.1. Anfis

Taking in consideration the Anfis System, at the beginning it is required to divide data in three parts: training, testing and checking. As far as we have 536 samples, we decided to discard last two samples and to take 534 samples dividing them in three parts. Each part contains 178 samples coming from the best 3 indexes chosen as predictors.
Passing to ISE100, we made the same partitioning. At this point we created 3 matrixes, each of which has four columns, one for each index (included ISE), and 178 rows. This operation is made thanks to the following code lines:

```
training = [matrix1(1:178,best_indeces(1)) matrix1(1:178,best_indeces(2))
matrix1(1:178,best_indeces(3)) ISE1(1:178) ];

testing = [matrix1(179:356,best_indeces(1)) matrix1(179:356,best_indeces(2))
matrix1(179:356,best_indeces(3)) ISE1(179:356) ];

checking = [matrix1(356:534,best_indeces(1))
matrix1(356:534,best_indeces(2)) matrix1(356:534,best_indeces(3))
ISE1(356:534) ];
```

These three matrixes are uploaded to the Anfis Editor to be used respectively in the training, testing and checking phase. The system

recognizes automatically that first three columns will be the input and the last one will be our output.
This model is shown in the following picture:

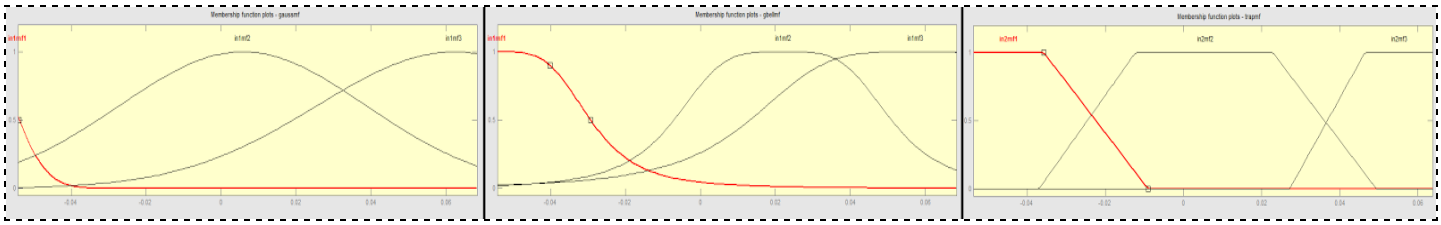


As we can see, the three inputs are combined together and, thanks to the created rules, we obtain the outputs.
We pass to anfis tool the created matrixes.
The second step is the one in which we have to generate the *FIS* (*Fuzzy Inference System*), we decided to set these options:

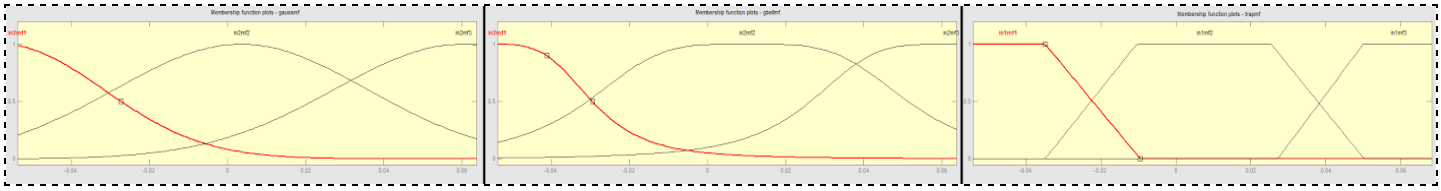
- Optimization method: hybrid;
- Error Tolerance: 0;
- Epochs: 50;
- Grid Partitioning;
- Membership Functions: gbell, gaussian and trap;
- MF is constant.

Once done, we tested the FIS on the three created partitions.
We show now the three different function obtained:

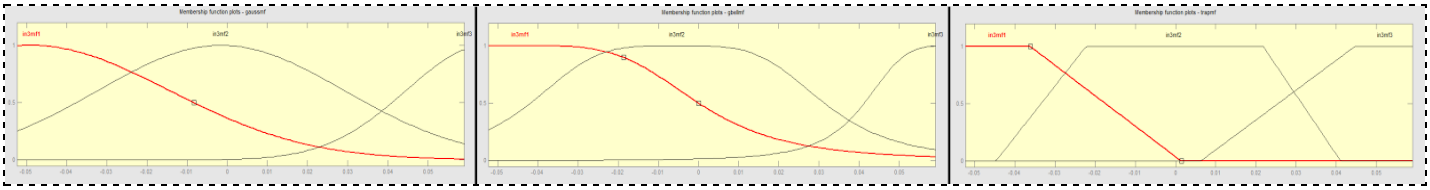
- training part:



- testing part

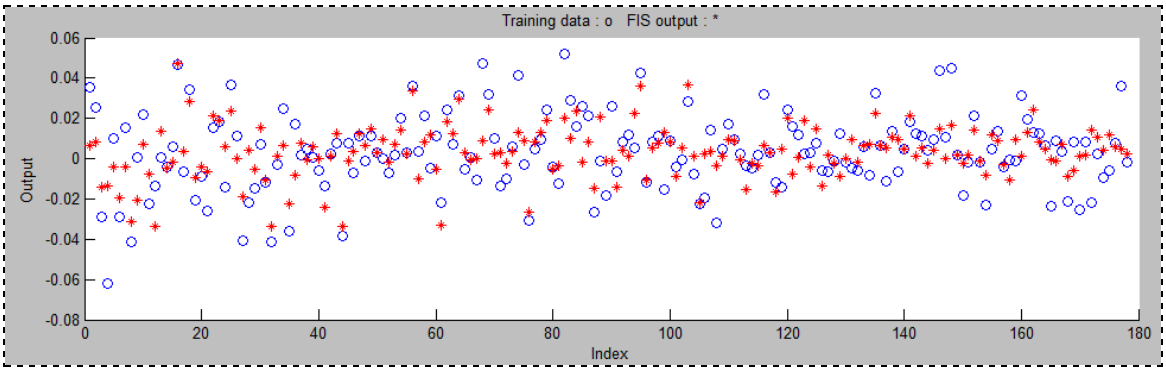


- checking part

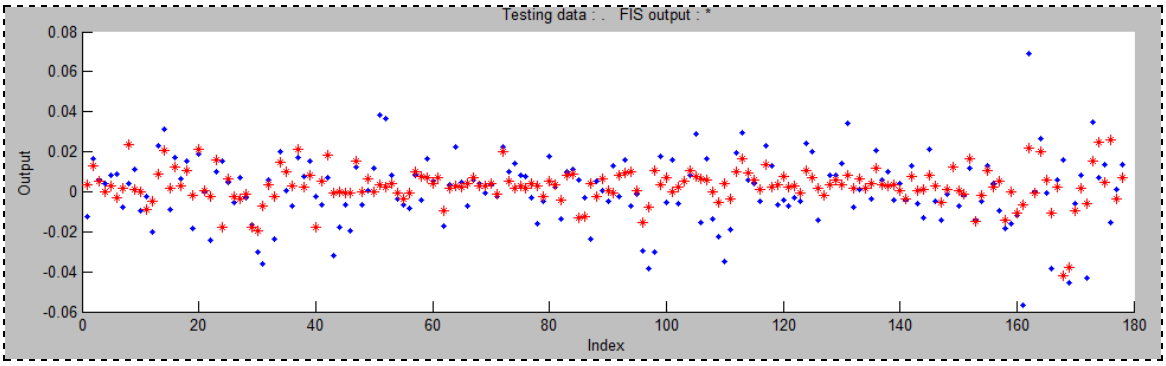


We show now the result obtained only for the Membership Functions *gbell*. In this way we will consider the obtained error by printing how the forecasted values differ from the real one for each data partition.

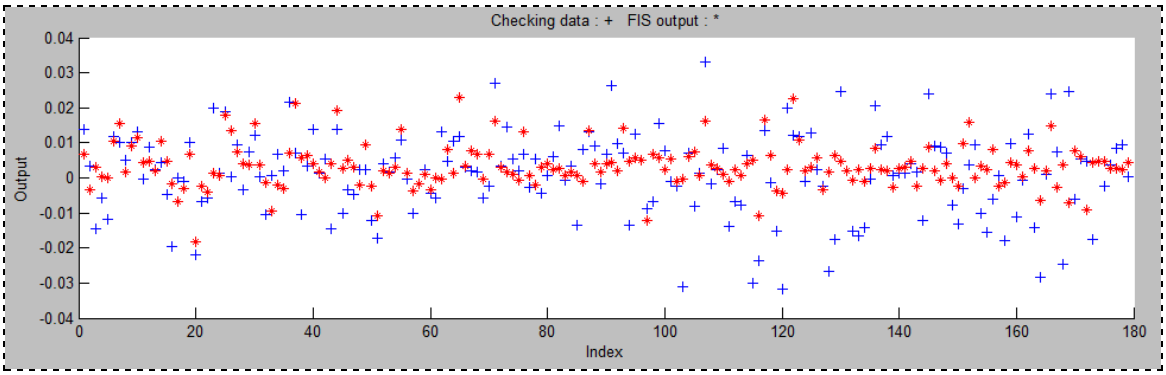
Average testing error for *training* = 0.013909



Average testing error for *testing* = 0.014585



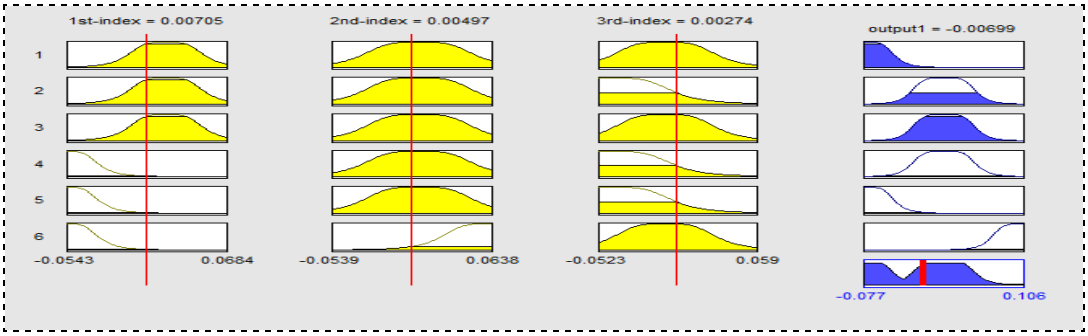
Average testing error for *checking* = 0.010736



As we can see, we obtain an average error that is higher than the one obtained by using the *Time Series*. Evaluations on the other Membership Functions have told us that the obtained average errors are always worse than the *Time Series* ones.

6.2. Mamdani

Reached the part in which we want to try and use *Mamdani fuzzy rules*, as it has been explained above, we reported the same Membership Functions in this system and we created some rules to see how the system works.
We take some samples from ISE100 and the three best indexes that has been found in the first part referred to the same days.
We have then reported from the previous step the *gbell* Membership Function for the three indexes and also fixed the ISE100 range between its minimum and maximum values.
By looking at the Memberships function we derived 6 rules that try to make an acceptable estimation.



We decided to implement and show only one Membership function type that is the *gbell*. Above we are going to show some images that explain the rule situation.

7. Results and Conclusions

We are now arrived at the end of this project. In order to complete the whole presentation of our work, we want to show you the code contained in the script that has to be launched to simulate the process, contained in the file *main_sheet.m*:

```
%First Part

SP_matrix = create_matrix( matrix(:,1) );
DAX_matrix = create_matrix( matrix(:,2) );
FTSE_matrix = create_matrix( matrix(:,3) );
NIKKEI_matrix = create_matrix( matrix(:,4) );
BOVESPA_matrix = create_matrix( matrix(:,5) );
EU_matrix = create_matrix( matrix(:,6) );
EM_matrix = create_matrix( matrix(:,7) );

errors_matrix(1,:) = train_function(SP_matrix,ISE);
errors_matrix(2,:) = train_function(DAX_matrix,ISE);
errors_matrix(3,:) = train_function(FTSE_matrix,ISE);
errors_matrix(4,:) = train_function(NIKKEI_matrix,ISE);
errors_matrix(5,:) = train_function(BOVESPA_matrix,ISE);
errors_matrix(6,:) = train_function(EU_matrix,ISE);
errors_matrix(7,:) = train_function(EM_matrix,ISE);

%Second Part

genetic_results = genetic_algorithm();

open_loop_performances = openLoopTraining(genetic_results);

closed_loop_performances = closedLoopTraining(genetic_results);
```

```
%Third Part

%Creating Data for Anfis System

training = [matrix1(1:178,best_indeces(1)) matrix1(1:178,best_indeces(2))
matrix1(1:178,best_indeces(3)) ISE1(1:178) ];
testing = [matrix1(179:356,best_indeces(1)) matrix1(179:356,best_indeces(2))
matrix1(179:356,best_indeces(3)) ISE1(179:356) ];
checking = [matrix1(356:534,best_indeces(1)) matrix1(356:534,best_indeces(2))
matrix1(356:534,best_indeces(3)) ISE1(356:534) ];
```

As it is possible to see, at the end we create also data needed for *Anfis System*. After this we would like to show also what is printed on the Matlab command line that shows some results obtained:

```
Creating Matrixes...
Matrixes Created.
Analyzing Indexes...
Indexes Analyzed and Classified.
Optimization terminated: average change in the fitness value less than options.TolFun.
The value returned by GA is: 1.279574520797430e-04
The value returned for input delay is: 25
The value returned for output delay is: 5
one-step-ahead Forecasting...
MSE     2.1649e-04
MAPE    147.5180
PERCENTAGE     75.6579

one-step-ahead Forecasting done.

n-step-ahead Forecasting...
We are going to predict ISE100 value with steps from 1 to 9.

Step:    1
MSE     2.9396e-04
MAPE    348.8131
PERCENTAGE     52.8926
```

Step: 2
MSE 2.9624e-04
MAPE 347.3205
PERCENTAGE 55.1867

Step: 3
MSE 2.9810e-04
MAPE 342.4968
PERCENTAGE 52.9167

Step: 4
MSE 2.9839e-04
MAPE 335.0707
PERCENTAGE 53.9749

Step: 5
MSE 2.9818e-04
MAPE 330.1023
PERCENTAGE 52.1008

Step: 6
MSE 2.9816e-04
MAPE 320.8223
PERCENTAGE 51.0549

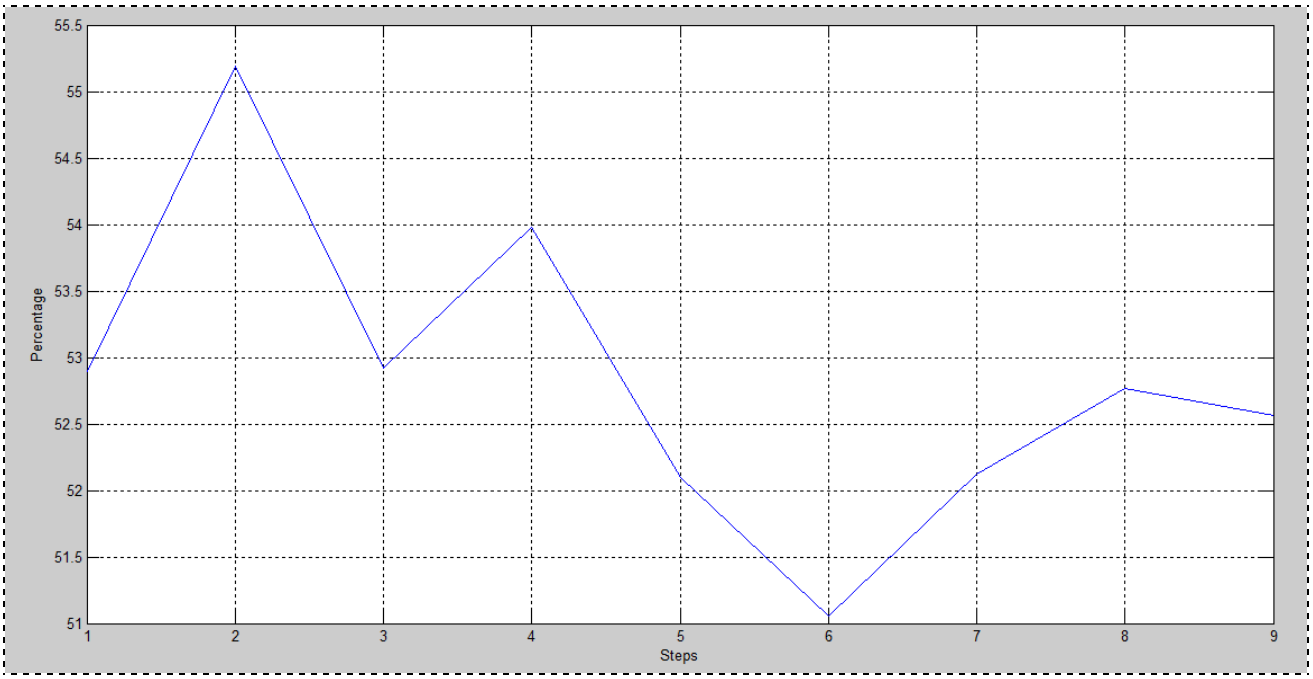
Step: 7
MSE 2.9846e-04
MAPE 319.9751
PERCENTAGE 52.1186

Step: 8
MSE 2.9863e-04
MAPE 325.3444
PERCENTAGE 52.7660

Step: 9
MSE 2.9908e-04
MAPE 324.8581
PERCENTAGE 52.5641

n-step-ahead Forecasting done.

As we can see from the report above, the percentage of correctly forecasted days starts from about 53% and at the ninth step reaches 52.5%. We can also appreciate this through the graph below:



The very last thing we wanted to fix is that, to be sure that our work has been done correctly, we tried and operate with the forecasting period data on an open-loop network and, with the same network, on a closed-loop network with one-step-ahead forecasting to see if really the percentage of correctly forecasted days was equal. After the trial, we observed that the results are equal, showing us the network was implemented in a correct way.
It was very interesting to work with neural networks and to develop a project regarding the index market forecast.