FEED-FORWARD NEURAL NETWORKS REGRESSION ANALYSIS WITH GENETIC ALGORITHMS: APPLICATIONS IN ECONOMICS AND FINANCE

Dr. Eleftherios Giovanis[4]

Abstract

In this paper feed-forward neural networks are examined using genetic algorithms in the training process instead of error backpropagation algorithm. Additionally, real encoding is preferred to binary encoding as it is more appropriate to find the optimum weights. Learning and momentum rates are used for the weight updating as in the case of the error backpropagation algorithm. Some empirical examples as well as the programming routines in MATLAB are provided in the paper.

Keywords: Feed-Forward Neural Networks, Genetic Algorithms, Time-Series, Stock Returns, Inflation Rate, Gross Domestic Product, Forecasting, MATLAB

JEL Codes: C15, C45, C63

4 Senior Lecturer, Manchester Metropolitan University, Business School, Department of Economics, Policy and International Business (EPIB), UK, giovanis95@gmail.com

GENETİK ALGORİTMALAR İÇEREN İLERİ BESLEMELİ SİNİR AĞLARI REGRESYON ANALİZLERİ: İKTİSAT VE FİNANS ALANINDA UYGULAMALAR

Eleftherios Giovanis

Özet

Bu çalışmada ileri beslemeli sinir ağları, hata geriye yayma algoritması yerine öğrenme sürecinde genetik algoritmalar kullanılarak incelenmiştir. İlave olarak, optimal ağırlıkları bulmada daha uygun olduğundan ikil kodlama yerine gerçek kodlamanın kullanımı tercih edilmiştir. Hata geriye yayma algoritmasında olduğu gibi ağırlıkların güncellenmesi için öğrenme ve momentum katsayıları kullanılmıştır. MATLAB'da yapılan ampirik örnekler ve program yordamları çalışmada sunulmuştur.

Anahtar Kelimeler: İleri Beslemeli Sinir Ağları, Genetik Algoritmalar, Zaman Serileri, Hisse Senedi Getirileri, Enflasyon Oranı, Yurtiçi Hasıla, Kestirim, MATLAB

Jel Kodları: C15, C45, C63

1. Introduction

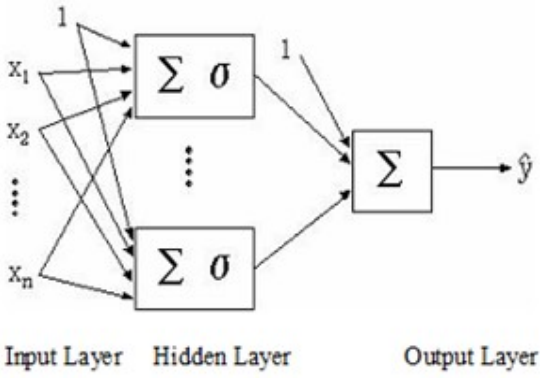
In the 1950s and the 1960s several computer scientists independently studied evolutionary systems with the idea that evolution could be used as an optimization tool for engineering problems. The idea in all these systems was to evolve a population of candidate solutions to a given problem, using operators inspired by natural genetic variation and natural selection. Genetic Algorithm (GA) is a method for moving from one population of "chromosomes" e.g., strings of ones and zeros, or "bits", to a new population by using a kind of "natural selection" together with the genetics−inspired operators of crossover, mutation, and inversion. Each chromosome consists of "genes", each gene being an instance of a particular "allele" (e.g., 0 or 1). The selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more offspring than the less fit ones. Crossover exchanges subparts of two chromosomes, roughly mimicking biological recombination between two single−chromosome organisms.

Mutation randomly changes the allele values of some locations in the chromosome. The purpose of our study is to present only the procedure and some empirical examples so we do not compare the results of other methods. Additionally, there are many models to compare with, as also even if we test for example ten different time-series in 50 countries it will not be enough. Furthermore, GAs are proposed for error minimizations, so GAs might be a good choice e.g. GARCH optimization among others.

2. Methodology

In its simplest "feed-forward" form, presented in figure 1, a neural network is a collection of connected activatable units -"neurons"- in which the connections are weighted, usually with real−valued weights. The activation coming into a unit from other units is multiplied by the weights on the links over which it spreads, and then is added together with other incoming activation. This process is meant to roughly mimic the way activation spreads through networks of neurons in the brain. In a feed-forward network, activation spreads only in a forward direction, from the input layer through the hidden layers to the output layer.

Figure 1. A Simple Feed-Forward Neural Network



We minimize function (1) in order to find the optimum parameters.

$$\frac{1}{2} mse\ (y - y^{t}) = \frac{1}{2} mse\ (e) \tag{1}$$

, where y^t is the target-actual, y is network's output variable and mse is the mean squared error The steps of genetic algorithms are (Bäck, 1996; Mitchell, 1996):

1. Start with a randomly generated population of n-bit chromosomes, which are the candidate solutions. In the case we do not use bit or binary encoding, but we use real number encoding based on the range of the input data. The chromosomes are equally with the number of weights for both input-to-hidden layer and hidden-to-output layer.

2. Calculate the fitness f(x) of each chromosome x in the population

3. Repeat the following steps until n offspring have been created:

a. select a pair of parents chromosomes of the current population and compute the probability of selection being an increasing function of fitness. In this case we take the roulette wheel selection algorithm. Also the selection process is one with replacement meaning that the same chromosome can be selected more than once to become a parent.

b. The next step is the crossover. We use one-point crossover process with probability p_c cross over the pair at a chosen point. If no crossover takes place we form two offspring that are exact copies of their respective parents.

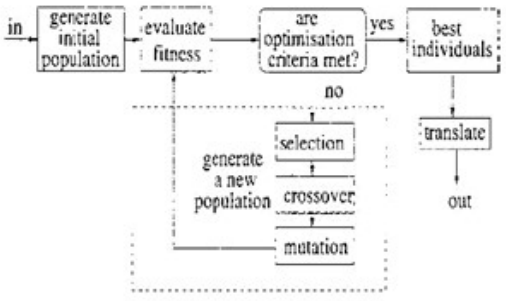
c. Mutate the two offspring with probability pm and place the resulting chromosomes in the new population.

4. Replace the current population with the new population.

5. Go to step 2.

In the initialisation, the first thing to do is to decide the coding structure. Coding for a solution is termed as chromosome in GA literature. In figure 2 a standard procedure for a genetic algorithm is presented.

Figure 2. Standard procedure of a canonical genetic algorithm



GA uses proportional selection, the population of the next generation is determined by n independent random experiments; the probability that individual x_i is selected from the tuple (x_1, x_2, . . . , x_m) to be a member of the next generation at each experiment is given by:

$$P\{x_i selected\} = \frac{f(x_i)}{\sum_{j=1}^{m} f(x_i)} > 0 \qquad (2)$$

This process (2) is the well-known roulette wheel selection Crossover is an important random operator in Canonical Genetic Algorithm (CGA) and the function of the crossover operator is to generate new or 'child' chromosomes from two 'parent' chromosomes by combining the information

extracted from the parents. The method of crossover used in CGA is the one-point crossover. Mutation operates independently on each individual by probabilistically perturbing each bit string. A usual way to mutate used in CGA is to generate a random number v between 1 and l and then make a random change in the v^{th} element of the string with probability ranging between 0 and 1. Usually probability pm takes values between 0.001 and 0.01. Holland's schema−counting argument seems to imply that GAs should exhibit worse performance on multiple−character encodings than on binary encodings. However, this has been questioned by some studies (Antonisse, 1989). Several empirical comparisons between binary encodings and multiple−character or real−valued encodings have shown better performance for the latter (Janikow and Michalewicz, 1991; Wright, 1991). But the performance depends very much on the problem and the details of the GA being used, and at present there are no rigorous guidelines for predicting which encoding will work best. The second approach is to introduce a weighted regression of the following form:

$$y = (\beta_0 + \beta_i x_i) \cdot w_{kj} \tag{3}$$

Where y and x_i are the dependent and independent variables respectively, β_0 is the constant or bias, β_i are the estimated coefficients for $i=1,2,...n$ inputs or independent variables and w_{kj} is the final weight matrix from the hidden to input layer after the training process with genetics. In this case the neural network procedure minimizes the squares error of the residuals of equation (3). Alternatively, (3) can be written as:

$$y = f((\beta_0 + \beta_i x_i) \cdot w_{kj}) \tag{4}$$

Where f denotes the transfer function. To be specific the dependent variable can be regressed on sigmoid, tangent hyperbolic, radial basis or any other function of the independent variables.

3. Empirical Applications

The first example is the inflation rate of USA in monthly data and we examine the period 1950-2009. The training period is 1960-2008 and the testing period is 2009. We use as input data the inflation rate with one lag and we do not take bias. The population size is set up at 30, with 20 iterations and chromosomes are equal with the number of weights, as mentioned previously, and differ in each case we examine. The crossover probability pc is set up at 0.2 and the mutation probability pm at 0.005. The transfer function from input to hidden is sigmoid and to the output is the linear. The out-of-sample forecasts are presented in Figure 3.

In the next example we examine again the inflation rate with AR(2) and with the second approach. Because the computation process takes a very long time we reduced the sample period and we obtain 1995-2007 as the training period and 2008 as testing periods. Additionally, we reduced the

number of iterations at 10. The population size is set up at 30. The crossover probability pc is set up at 0.2 and the mutation probability p_m at 0.005. The transfer functions from input to hidden and from hidden to output are linear. The in-sample and out-of-sample forecast are presented in Figures 4 and 5 respectively.

Figure 3. Out-of-sample forecasts for inflation rate

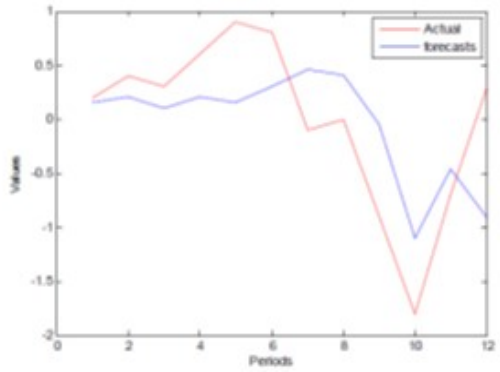


Figure 4. In-sample forecasts for inflation rate with the second approach

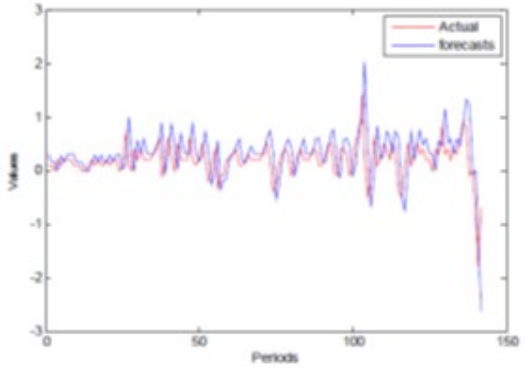
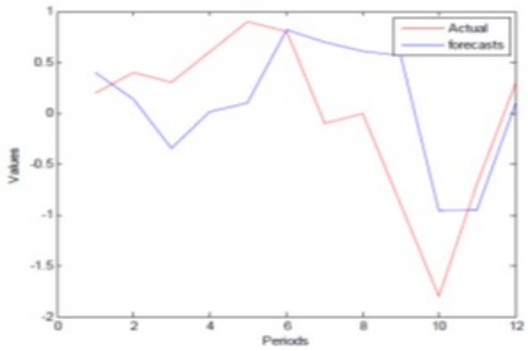


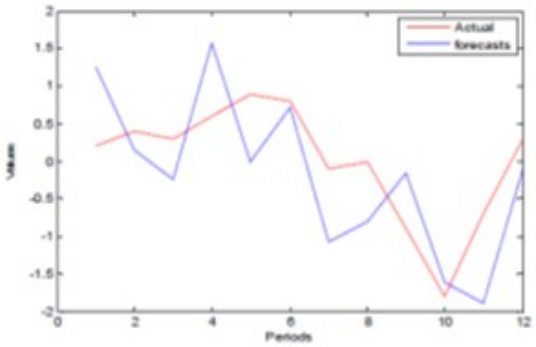
Figure 5. Out-of-sample forecasts for inflation rate with the second approach



In Figure 6 we present the out-of-sample forecasts with the second approach and AR(2) with

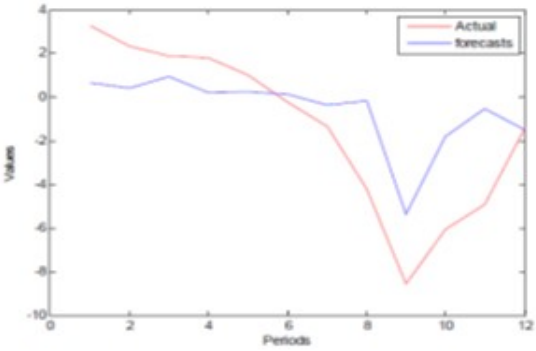
constant. The other settings remain the same. Also in table 1 the estimated coefficient and their respective t-statistics are reported.

Figure 6. Out-of-sample forecasts for inflation rate with the second approach and with constant



In the next numerical application example, we examine the growth rate of Japanese gross domestic product. The training period is 1995-2006 and the period 2007-2009 is taken as the test sample. The setting are the same with those me set up previously, while we estimate an AR(1) with no constant, with linear transfer functions. In figure 7 the out-of-sample forecasts are presented.

Figure 7. Out-of-sample forecasts for gross domestic product of Japan with the second approach and with no constant



In the last example we implement is the day of the week effect for FTSE 100 stock index returns. The regression we estimate is:

$$r_t = \beta_1 D_1 + \beta_2 D_2 + \beta_3 D_3 + \beta_4 D_4 + \beta_5 D_5 + \varepsilon_t \quad (5)$$

Where r_t denotes the stock index logarithmic returns, β_i denotes the estimated coefficients for $i=1,2..5$, D_1 is the dummy variable for Monday taking value 1 for returns on Monday and 0 otherwise and so on until dummy D_5 is the dummy variable for Friday. Our purpose is not to investigate if actually there is a day of the week effect but to show that this model can be useful for multi-period ahead forecasts. Because the independent variables are known, because we know the days, while in other models it is very difficult or even almost impossible to know with precision the actual values of independent variables in a long period ahead. Besides that, we can examine also calendar anomalies as this might be useful because the Boolean classification of one and zero is not anymore correct, because it is not enough to set up 1 for the returns on the specific day. More specifically, each day presents different returns on each week and the traditional classification is not clear. For this reason, different weights are assigned in each day and in each week. We take the trading days of 2008-2009. To be specific the last 50 trading days of 2009 are taken as the test sample while the remaining period is obtained for the in-sample or training period. The population size is set up at 30, the crossover probability pc is set up at 0.2 and the mutation probability pm at 0.003. The number of iterations is only 10, while we get satisfying results and this number can be increased. The transfer function from input to hidden layer is the sigmoid, while linear function is used from hidden to output layer. The process of taking the forecasts is very simple. Based on the estimated coefficient through the training period and knowing the values of the independent variables, which are the dummies, then we just have the following:

$$\hat{y} = \beta_i x_i \tag{6}$$

Where $\hat{y}$ are the predicted values, β_i are the estimated coefficients through the training period and xi is the test sample or a matrix containing the dummy variables. In Figure 8 we present the out-of-sample or test sample forecasts. We observe that there are deviations but we get the correct sign in most cases. Additionally, if we had estimated, with autoregressive, moving average, GARCH models or random walk, the forecasts would remind to us a dead line. We do not have to present the results in stock forecasting as are already very known and can be replicated. So the approach we propose is much more superior. Furthermore, we chose mutation probability pm equal with 0.003. Changing the probability, the forecasts can be changed significant as well. We change the mutation probability pm at 0.002 and we present the forecasts in Figure 9. It should be noticed that the sample period might be long, so a 10 period ahead might be more appropriate. Additionally, we take of four year period for FTSE 100 and we estimate EGARCH (1,1), proposed by Nelson (1991), and in figure 10 we present the out-of-sample forecasts. We observe how poor the predicted values are, so our argument is that if a model presents too poor forecasts, why should present also reliable estimations? The problem is not

only to solve for autocorrelation and ARCH effects, but is also the using of a model presenting good high forecasting performance. We estimated EGARCH process because simple Autoregressive (AR), Moving Average (MA), Autoregressive Moving Average (ARMA) models, or random walk and other GARCH models presented a lower forecasting power. Additionally, because these approaches give poor forecasts might not also be appropriate for the examination of the day of the week effect.

Figure 8. Out-of-sample forecasts for FTSE 100 with second approach and with mutation probability 0.003

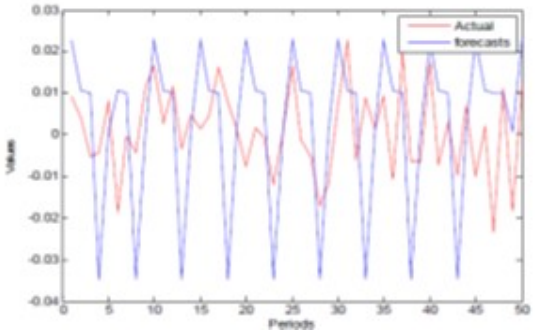


Figure 9. Out-of-sample forecasts for FTSE 100 with second approach and with mutation probability 0.002

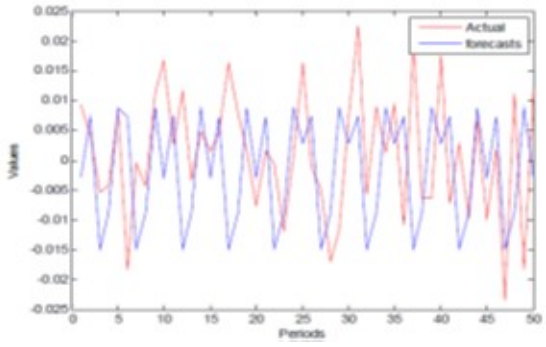
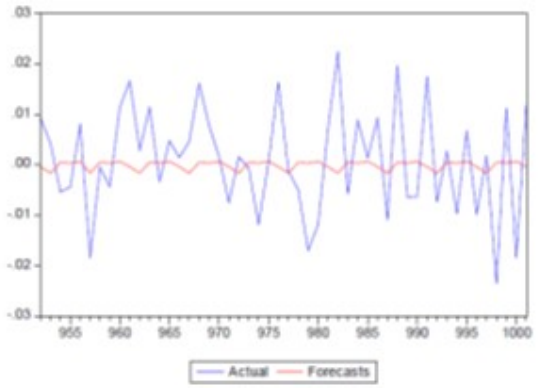


Figure 10. Out-of-sample forecasts for FTSE 100 with EGARCH (1,1)



It should be noticed that the length of sample plays a major and crucial role, because significant changes, in crossover and mutation probabilities, are needed in order to get satisfying forecasts.

4 Conclusion

We examined a simple approach of feed-forward neural networks combined with genetic algorithms. The encoding was based on real number, instead of binary encoding or bits, because this procedure is more appropriate to find the optimum weights. Additionally, we proposed a very simple regression based on the weighted independent variables derived by the training procedure.

References

Antonisse, J. (1989), "A new interpretation of schema notation that overturns the binary encoding constraint", In J. D. Schaffer, ed., Proceedings of the Third International Conference on Genetic Algorithms, pp. 86-91, George Mason University, USA

Bäck, T. (1996), Evolutionary Algorithms in Theory and Practice. Oxford University Press

Janikow, C. Z. and Michalewicz, Z. (1991), "An experimental comparison of binary and floating point representations in genetic algorithms", In R. K. Belew and L. B. Booker, eds., Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers

Mitchell, M. (1996), "An Introduction to Genetic Algorithms", MIT Press Cambridge, Massachusetts, London, England.

Nelson, D.B. (1991), "Conditional Heteroskedasticity In Asset Returns: A New Approach", *Econometrica*, 59,(2): 347-370.

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In G. Rawlins, ed., Foundations of Genetic Algorithms, Morgan Kaufmann Publishers

Appendix

MATLAB ROUTINE 1

For feed-forward neural networks and genetics

```
clear all; load file.mat
transfer_function=4       % 1 for log-sigmoid, 2 for tangent
hyperbolic, 3 for radbas and 4 for linear
if transfer_function==1 % 1 for log-sigmoid, 2 for tangent
hyperbolic, 3 for radbas and 4 for linear
transfer_delta_function=1
elseif transfer_function==2
transfer_delta_function=2
elseif transfer_function==3
transfer_delta_function=3
elseif transfer_function==4
```

```
transfer_delta_function=4

end

transfer_function_O=4  % 1 for log-sigmoid, 2 for tangent

hyperbolic, 3 for radbas and 4 for linear

constant=0

pc=0.2; % Crossover rate

pm=0.005;        % Mutation rate

nforecast=12

y=data(1:end-nforecast,1) x=data(1:end-nforecast,2)

if constant == 0 x=x

elseif constant==1 x=[ones(nk,1) x];

end

[nk ni]=size(x)

n_outputs = 1; n_inputs = ni;

num_hidden = ni;

chrom =num_hidden*n_inputs + num_hidden*n_outputs;

 chrom =num_hidden*(n_inputs+1) + (num_hidden + 1)*n_outputs;

 lb= min(min(x)); % Lower bound of the parameters to be optimized ub=max(max(x));; %Upper bound of the parameters to be optimized

popsize=30;

chromlength=chrom;

Range=repmat((ub-lb),[popsize chrom])

Lower = repmat(lb, [popsize chrom]); pop=rand(popsize, chromlength).*Range+ Lower

f=x

for iterations =1:20

% chromosoms' fitness evaluation [px,py] = size(pop);

Chromosome = pop; for i = 1:px

for j = 1:num_hidden*ni
```

```
w1(j) = Chromosome (i,j);
end
w1 = reshape(w1,num_hidden,ni);
k=1;
for j = n_inputs*num_hidden+1:(num_hidden*ni + n_outputs*num_hidden)
w2(k)= Chromosome (i,j); %w2=w2'
k=k+1;
end
[X Y]=meshgrid(w2) w2=Y(:,1)
k=1;
for j = (num_hidden*n_inputs + n_outputs*num_hidden + 1):(num_hidden*n_inputs + ...
n_outputs*num_hidden + num_hidden) input_hidden_Bias(k)= Chromosome (i,j);
k=k+1;
end
k=1;
for j = (num_hidden*n_inputs + n_outputs*num_hidden + num_hidden + 1):(num_hidden*n_inputs
+...
n_outputs*num_hidden + num_hidden +n_outputs) hidden_output_Bias(k)= Chromosome (i,j);
k=k+1;
end
 if transfer_function==1 h=logsig(f*w1');
elseif transfer_function==2 h=tansig(f*w1');
elseif transfer_function==3 h=radbas(f*w1');
elseif transfer_function==4 h=purelin(f*w1');
end
h1 = h
if transfer_function_O==1; y1=logsig(h1*w2);
elseif transfer_function_O==2; y1=tansig(h1*w2);
```

```
elseif transfer_function_O==3; y1=radbas(h1*w2);

elseif transfer_function_O==4; y1=purelin(h1*w2);

end

err = (y-y1);

err = reshape(err,nk*n_outputs,1); object_value(i)=1/2*mse(err)

end

fitvalue = object_value; totalfit=sum(fitvalue); fitvalue=fitvalue/totalfit; fitvalue=cumsum(fitvalue);
[px,py]=size(pop); ms=sort(rand(px,1)); fitin=1;

newin=1;

while newin<=px if(ms(newin))<fitvalue(fitin) newpop(newin,:)=pop(fitin,:); newin=newin+1;

else fitin=fitin+1; end

end

% crossover between chromosoms

 pop = newpop;

length_chrom = size(pop,2);

c_point = ceil(rand(size(pop,1)/2,1)*(length_chrom-1)); c_point = c_point.*(rand(size(c_point))<pc);

for i = 1:length(c_point);

newpop([2*i-1 2*i],:) = [pop([2*i-1 2*i],1:c_point(i)) pop([2*i 2*i-1],c_point(i)+1:length_chrom)];

end

% mutation of chromosoms pop = newpop;

mutated = find(rand(size(pop))<pm); newpop = pop;

newpop(mutated) = 1-pop(mutated);

% finding the best individual pop = newpop;

bestindividual=pop(1,:);

bestfit=fitvalue(1); for i=2:px

if fitvalue(i)<bestfit bestindividual=pop(i,:); bestfit=fitvalue(i);

end end

best_gen=bestindividual
```

```
for       j = 1:num_hidden*ni w1(j) = best_gen (1,j);

end

k=1;

for       j = ni*num_hidden+1:(num_hidden*ni + n_outputs*num_hidden) w2(k)= best_gen (1,j);

k=k+1;

end

iterations=iterations+1 iterations

arraygbest ( iterations )= bestfit; indexiter ( iterations ) = iterations; end

%end              test_sample=data(end-nforecast:end-1,2)              test_y=data(end-nforecast+1:end,1)
tt=length(test_sample)

if constant==0 test_sample=test_sample elseif constant==1

test_sample=[ones(size(test_y)) test_sample]

end

if transfer_function==1

 yfore=logsig(test_sample*w1) yfore=logsig(yfore*w2)

elseif transfer_function==2

yfore=tansig(test_sample*w1)

yfore=tansig(yfore*w2)

elseif transfer_function==3 yfore=radbas(test_sample*w1)

yfore=radbas(yfore*w2)

elseif transfer_function==4 yfore=test_sample*w1

yfore=yfore*w2 end


figure, plot(y,'-r'); hold on; plot(y1,'-b'); xlabel('Periods')

ylabel('Values') %title('In_sample forecasts')

h1 = legend('Actual','forecasts',1);

figure, plot(test_y,'-r'); hold on; plot(yfore,'-b'); xlabel('Periods')

ylabel('Values') %title('Out_of_sample forecasts')
```

```
h = legend('Actual','forecasts',1); figure, plot (indexiter , array_y); xlabel('epochs')
ylabel('Error') title('Number of Epochs')
```

MATLAB ROUTINE 2

For genetics with second approach and minimization of regression residuals

```
%function [bols] = ar_afnn_gen(y,lag,constant,criteria)
clear all; load file.mat
transfer_function=4 % 1 for log-sigmoid, 2 for tangent
hyperbolic, 3 for radbas and 4 for linear
if transfer_function==1 % 1 for log-sigmoid, 2 for tangent
hyperbolic, 3 for radbas and 4 for linear
transfer_delta_function=1
elseif transfer_function==2 transfer_delta_function=2
elseif transfer_function==3 transfer_delta_function=3
elseif transfer_function==4
 transfer_delta_function=4 end
transfer_function_O=4 % 1 for log-sigmoid, 2 for tangent hyperbolic, 3 for radbas and 4 for linear
nlag=2 % Lag order
constant=1
pc=0.2; % Crossover rate
pm=0.001; % Mutation rate
nforecast=12
for jj=nforecast:-1:1
y= data(1:end-jj,1);
clear x
for pp=1:nlag
x(:,pp)=lagmatrix(y,pp)
```

```
end
 i1        =        find(isnan(x));
i2         =        find(isnan(diff([x ; zeros(1,size(x,2))]) .* x));
if (length(i1) ~= length(i2)) ||     any(i1 - i2)
          error('Series cannot contain NaN).')
end
if any(sum(isnan(x)) == size(x,1))
          error('A realization of "x"          is completely missing
(all
NaN"s).')
end
first_Row  =  max(sum(isnan(x)))         + 1;
x                    =  x(first_Row:end       , :);
y=y(first_Row:end,:) [nk ni]=size(x)
if constant == 0 x=x
elseif constant==1 x=[ones(nk,1) x];
end
[nk ni]=size(x)
n_outputs = 1; n_inputs = ni;
num_hidden = ni;
chrom =num_hidden*n_inputs + num_hidden*n_outputs;
lb= min(min(x)); % Lower bound of the parameters to be optimized ub=max(max(x));; %Upper bound
of the parameters to be optimized
popsize=30;
chromlength=chrom;
Range = repmat((ub-lb),[popsize chrom]); Lower = repmat(lb, [popsize chrom]);
pop= rand(popsize,chrom) .* Range + Lower;
f=x
```

```
for iterations =1:10
% chromosoms' fitness evaluation [px,py] = size(pop);
Chromosome = pop; for i = 1:px
for j = 1:num_hidden*ni
w1(j) = Chromosome (i,j);
end
w1 = reshape(w1,num_hidden,ni);
k=1;
for j = n_inputs*num_hidden+1:(num_hidden*ni + n_outputs*num_hidden)
w2(k)= Chromosome (i,j);
k=k+1;
end
[X Y]=meshgrid(w2) w2=Y(:,1)
if transfer_function==1 h=logsig(f*w1);
elseif transfer_function==2 h=tansig(f*w1');
elseif transfer_function==3 h=radbas(f*w1');
elseif transfer_function==4
h=purelin(f*w1'); end
h1 = [h]
if transfer_function_O==1; y1=logsig(h1*w2);
elseif transfer_function_O==2; y1=tansig(h1*w2);
elseif transfer_function_O==3; y1=radbas(h1*w2);
elseif transfer_function_O==4; y1=purelin(h1*w2);
end
newx=x*w1
bols=inv(newx'*newx)*newx'*y err = y-newx*bols
err = reshape(err,nk*n_outputs,1); object_value(i)=1/2*mse(err)
end
```

```
fitvalue = object_value; totalfit=sum(fitvalue); fitvalue=fitvalue/totalfit; fitvalue=cumsum(fitvalue);
[px,py]=size(pop); ms=sort(rand(px,1)); fitin=1;

newin=1;

while newin<=px if(ms(newin))<fitvalue(fitin) newpop(newin,:)=pop(fitin,:); newin=newin+1;

else fitin=fitin+1; end

end

% crossover between chromosoms pop = newpop;

length_chrom = size(pop,2);

c_point = ceil(rand(size(pop,1)/2,1)*(length_chrom-1)); c_point = c_point.*(rand(size(c_point))<pc);

for i = 1:length(c_point);

newpop([2*i-1 2*i],:) = [pop([2*i-1 2*i],1:c_point(i)) pop([2*i 2*i-1],c_point(i)+1:length_chrom)];

end

 % mutation of chromosoms pop = newpop;

mutated = find(rand(size(pop))<pm); newpop = pop;

newpop(mutated) = 1-pop(mutated);

% finding the best individual pop = newpop;

bestindividual=pop(1,:);

bestfit=fitvalue(1); for i=2:px

if fitvalue(i)<bestfit bestindividual=pop(i,:); bestfit=fitvalue(i);

end end

best_gen=bestindividual

for       j = 1:num_hidden*ni w1(j) = best_gen (1,j);

end

k=1;

for       j = ni*num_hidden+1:(num_hidden*ni + n_outputs*num_hidden) w2(k)= best_gen (1,j);

k=k+1;

end

newx=x*w1
```

```
bols=inv(newx'*newx)*newx'*y
xp=y(end-nlag+1:end,:) %yhat_out3(ii,:)=xp'*bols if constant==0
yhat(jj,:)=[y(end,:) newx(end,1:end-1)]*bols elseif constant==1
yhat(jj,:)=[newx(end,1) y(end,:) newx(end,2:end-1)]*bols
end
iterations=iterations+1 iterations
arraygbest ( iterations )= bestfit; indexiter ( iterations ) = iterations; end
end
s2 = (y-newx*bols)'*(y-newx*bols)/(nk-ni);
Vb=s2*inv(newx'*newx);          %          Get the variance-covariance
matrix
se=sqrt(diag(Vb));          % Get   coefficient standard errors
 tstudent=bols./se;          % Get t-statistics
for iii=1:nforecast yfore(iii,:)=yhat(end-iii+1,:) iii=iii+1
end test_y=dat(end-nforecast+1:end,1)
figure, plot(y,'-r'); hold on; plot(y1,'-b'); xlabel('Periods')
ylabel('Values') %title('In_sample forecasts')
h1 = legend('Actual','forecasts',1);
figure, plot(test_y,'-r'); hold on; plot(yfore,'-b'); xlabel('Periods')
ylabel('Values') %title('Out_of_sample forecasts')
h = legend('Actual','forecasts',1); figure, plot (indexiter , arraygbest ); xlabel('epochs')
ylabel('Error') title('Number of Epochs')
```

MATLAB ROUTINE 3

An example for the day of the week effects in FTSE 100 and the second approach

```
clear all; load file.mat
```

% data is a matrix where the first column contains the stock returns and the remaining are the dummy variables.

```
transfer_function=1 % 1 for log-sigmoid, 2 for tangent
hyperbolic, 3 for radbas and 4 for linear
if transfer_function==1 % 1 for log-sigmoid, 2 for tangent
hyperbolic, 3 for radbas and 4 for linear transfer_delta_function=1
elseif transfer_function==2 transfer_delta_function=2
elseif transfer_function==3 transfer_delta_function=3
elseif transfer_function==4 transfer_delta_function=4 end
transfer_function_O=4 % 1 for log-sigmoid, 2 for tangent hyperbolic, 3 for radbas and 4 for linear
pc=0.2; % Crossover rate
pm=0.002; % Mutation rate
nforecast=50
y=data(1:end- nforecast,1) x=data(1:end-nforecast,2:end)
[nk ni]=size(x)
n_outputs = 1; n_inputs = ni;
num_hidden = ni;
chrom =num_hidden*n_inputs + num_hidden*n_outputs;
lb= min(min(x)); % Lower bound of the parameters to be optimized ub=max(max(x));; %Upper bound
of the parameters to be optimized
popsize=30;
chromlength=chrom;
Range = repmat((ub-lb),[popsize chrom]); Lower = repmat(lb, [popsize chrom]);
pop= rand(popsize,chrom) .* Range
f=x
for iterations =1:10
% chromosoms' fitness evaluation [px,py] = size(pop);
Chromosome = pop; for i = 1:px
for j = 1:num_hidden*ni
w1(j) = Chromosome (i,j);
```

```
end
w1 = reshape(w1,num_hidden,ni);
k=1;
for j = n_inputs*num_hidden+1:(num_hidden*ni + n_outputs*num_hidden)
w2(k)= Chromosome (i,j); k=k+1;
end
[X Y]=meshgrid(w2) w2=Y(:,1)
if transfer_function==1 h=logsig(f*w1);
 elseif transfer_function==2 h=tansig(f*w1');
elseif transfer_function==3 h=radbas(f*w1');
elseif transfer_function==4 h=purelin(f*w1');
end
h1 = [h]
if transfer_function_O==1; y1=logsig(h1*w2);
elseif transfer_function_O==2; y1=tansig(h1*w2);
elseif transfer_function_O==3; y1=radbas(h1*w2);
elseif transfer_function_O==4; y1=purelin(h1*w2);
end
newx=x*w1
bols=inv(newx'*newx)*newx'*y err = y-newx*bols object_value(i)=1/2*mse(err)
end
fitvalue = object_value; totalfit=sum(fitvalue); fitvalue=fitvalue/totalfit; fitvalue=cumsum(fitvalue);
[px,py]=size(pop); ms=sort(rand(px,1)); fitin=1;
newin=1;
while newin<=px if(ms(newin))<fitvalue(fitin) newpop(newin,:)=pop(fitin,:); newin=newin+1;
else fitin=fitin+1; end
end
% crossover between chromosoms pop = newpop;
```

```
length_chrom = size(pop,2);
c_point = ceil(rand(size(pop,1)/2,1)*(length_chrom-1)); c_point = c_point.*(rand(size(c_point))<pc);
for i = 1:length(c_point);
newpop([2*i-1 2*i],:) = [pop([2*i-1 2*i],1:c_point(i)) pop([2*i 2*i-1],c_point(i)+1:length_chrom)];
 end
% mutation of chromosoms pop = newpop;
mutated = find(rand(size(pop))<pm); newpop = pop;
newpop(mutated) = 1-pop(mutated);
% finding the best individual pop = newpop;
bestindividual=pop(1,:);
bestfit=fitvalue(1); for i=2:px
if fitvalue(i)<bestfit bestindividual=pop(i,:); bestfit=fitvalue(i);
end end
best_gen=bestindividual
for       j = 1:num_hidden*ni w1(j) = best_gen (1,j);
end
k=1;
for       j = ni*num_hidden+1:(num_hidden*ni + n_outputs*num_hidden) w2(k)= best_gen (1,j);
k=k+1;
end iterations=iterations+1 ( iterations )= bestfit;
indexiter ( iterations ) = iterations; end
test_y=data(end-nforecast+1:end,1) test_x=data(end-nforecast+1:end,2:end)
yfore=test_x*bols
s2 = (y-newx*bols)'*(y-newx*bols)/(nk-ni);
Vb=s2*inv(newx'*newx);          %          Get the variance-covariance
matrix
se=sqrt(diag(Vb));          % Get   coefficient standard errors
tstudent=bols./se;
```

```
figure, plot(test_y,'-r'); hold on; plot(yfore,'-b'); xlabel('Periods')
ylabel('Values') %title('Out_of_sample forecasts')
h = legend('Actual','forecasts',1); figure, plot (indexiter , array_y); xlabel('epochs')
ylabel('Error') title('Number of Epochs')
```