Applications of Neural Network Radial Basis Function in Economics and Financial Time Series

Eleftherios Giovanis

Abstract

In this paper we present the Radial Basis Neural Network Function. We examine some simple numerical examples of time-series in economics and finance. The forecasting performance is significant superior, especially in financial time-series, to traditional econometric modeling indicating that artificial intelligence procedure are more appropriate. Some MATLAB routines are presented for further application research.

Keywords: Radial Basis Function, Neural Networks, Time-Series, Forecasting, stock returns, MATLAB, error backpropagation algorithm

1. Introduction

Radial Basis Functions emerged as a variant of artificial neural network in late 80's. However, their roots are entrenched in much older pattern recognition techniques as for example potential functions, clustering, functional approximation, spline interpolation and mixture models. RBF networks have been successfully applied to a large diversity of applications including interpolation, chaotic time-series modelling, system identification, control engineering , electronic device parameter modeling, speech recognition, image restoration (Broomhead and Lowe, 1988; Casdagli, 1989; Niranjan and Fallside, 1990; Chen *et al*., 1991; Bors and Gabbouj, 1994; Sanner and Slotine, 1994) among many others.

2. Methodology

Radial Basis Functions (RBF's) are embedded in a two layer neural network, where each hidden unit implements a radial activated function. The output units implement a weighted sum of hidden unit outputs. The input into an RBF network is nonlinear while the output is linear. In order to use a Radial Basis Function Network we need to specify the hidden unit activation function, the number of processing units,

a criterion for modeling a given task and a training algorithm for finding the parameters of the network. Finding the RBF weights is called network training.

The radial basis function (Powell, 1987), introduces a set of N basis function one for each point, which take the following form:

$$\phi \| x - x^n \| \tag{1}$$

, where φ(.) is a non-linear function. The n^{th} such function depends on the Euclidean Distance (or even we can take Manhattan distance) ||x — xn|| between x and xn. The output is taken to be linear combination of the basis functions such as:

$$h(x) = \sum_n w_n \phi(\| x - x^n \|) \tag{2}$$

The most common basis function is the Gaussian:

$$\phi(x) = \exp\left(-\frac{x^2}{2\sigma^2} \right) \tag{3}$$

Additional function can be incorporated as Multiquadrics, Reciprocal Multiquadrics, Logistic, Thin-Plate Spline and others, as we present in MATLAB routine in appendix. The radial basis function is

$$y_k(x) = \sum_{j=1}^{M} w_{kj} \phi_j(x) + w_{ko} \tag{4}$$

, where w_{kj} are the weights and w_{ko} are biases, that can be absorbed into the summation by including an extra function φ_0, whose activation is 1 (Bishop , 1995). Our purpose is to find a function h(x) that

$$h(x^n) = t^n \tag{5}$$

, which can be written in matrix form

$$\Phi w = t \tag{6}$$

, where $t= t^n$, $w= w_n$ and the square matrix Φ has elements $\Phi_{nn'} = \varphi(\|x^n — x^{n'}\|)$. If there is the inversed matrix of Φ then we solve for *w*

$$w = \Phi^{-1} t \tag{7}$$

So w_{kj} from equation (4) can be written as

$$w_{kn} = \sum_{n'} (\Phi^{-1})_{nn'} t_k^{n'} \tag{8}$$

, and for the case of the Gaussian function is

$$\phi_j(x) = \exp(-\frac{\| x - \mu_j \|}{2\sigma^2{}_j}) \tag{9}$$

, where x is the d-dimensional input vector with elements x_{ii} and μ_j is the vector, which determines the centre of the of the basis function φ_j and has elements μ_{ji} (Bishop , 1995).

There are various optimization solutions for the radial basis function ,as to set the function centres μ_j equal to a random subset of the input sectors from the training set but this method is not the proposed. The optimization method we apply in the specific paper is the K-means clustering algorithm. If there are N data points x^n total , we try to find the K vectors μ_j , where j = 1,2,….K and this algorithm categorize the data points x^n into K disjoint subsets S_j which contain N_j data points in such a way to minimize the sum of squares of the clustering function, which is

$$J = [\sum_{j-1}^{K} \sum_{n \in S_j} \| x^n - \mu_j \|]^2 \tag{10}$$

, where μ_j is the mean of the data points in set S_j. The K-means algorithm begins with the random setting of K sets and then computes the mean vector of data points in each set. Then each point is re-assigned in a new set to the nearest mean vector. The mean of the sets are recomputed. This iterative procedure is repeated until further change in the group of the data points is not needed (Bishop, 1995).

3. Numerical Applications

In the first numerical application we implement is the day of the week effect for FTSE 100 and DAX stock index returns. More specifically the inputs are the dummies representing the returns corresponding in the specific trading weekdays and the output is the index stock returns. 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 procedure 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. We use error backpropagation algorithm, with learning rate 0.1, momentum rate 0.1, lambda value at 0.1, Euclidean distance method, with error_performance, weights and sigma=1 in MATLAB routine and Gaussian function. The other functions give almost the same results. We set up the number of maximum epochs at 40 and the goal error at 0.001. Because we know the values of the dependent variables as we know already what is the next day. We examine the year of 2009, where the last 50 trading days are left as the out-of-sample or the testing period. In figures 1 and 2 we present respectively the in-sample and out-of-sample forecasts of FTSE-100. Similarly in figures 3 and 4 we present the respective forecasts for DAX index returns. It should be noticed that the in-sample forecasts can be perfect. This can be done by increasing lambda value but with this way the out-of-sample forecasts might be deteriorated. So for this reason we keep low the lambda value.

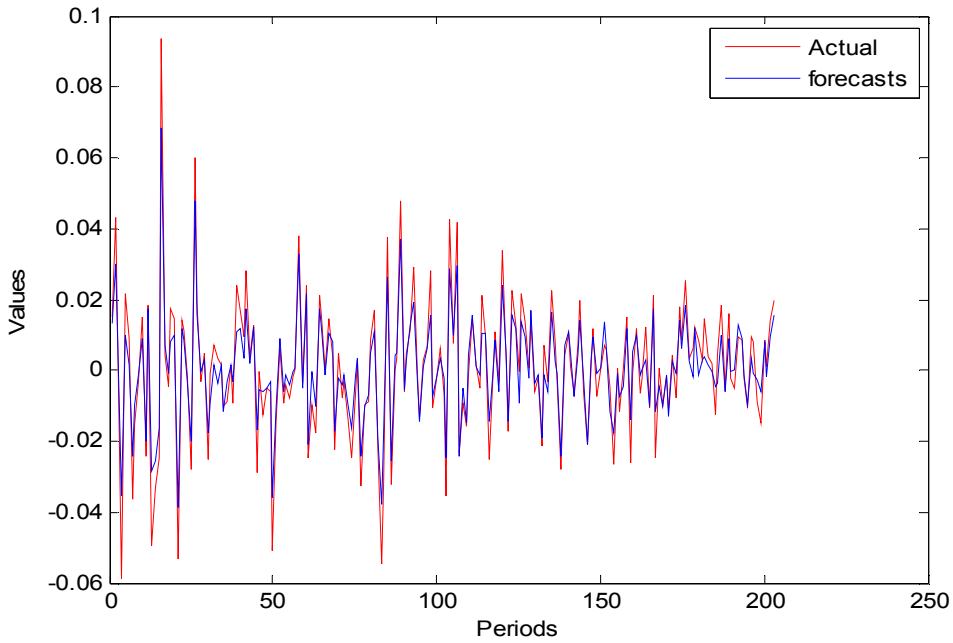

Fig. 1 In-sample forecasts for FTSE-100 index returns

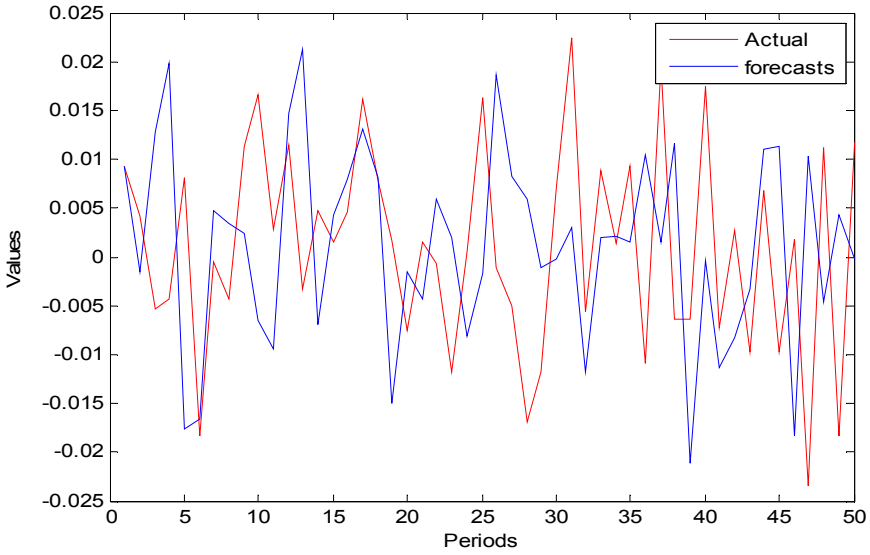

Fig. 2 Out-of-sample forecasts for FTSE-100 index returns

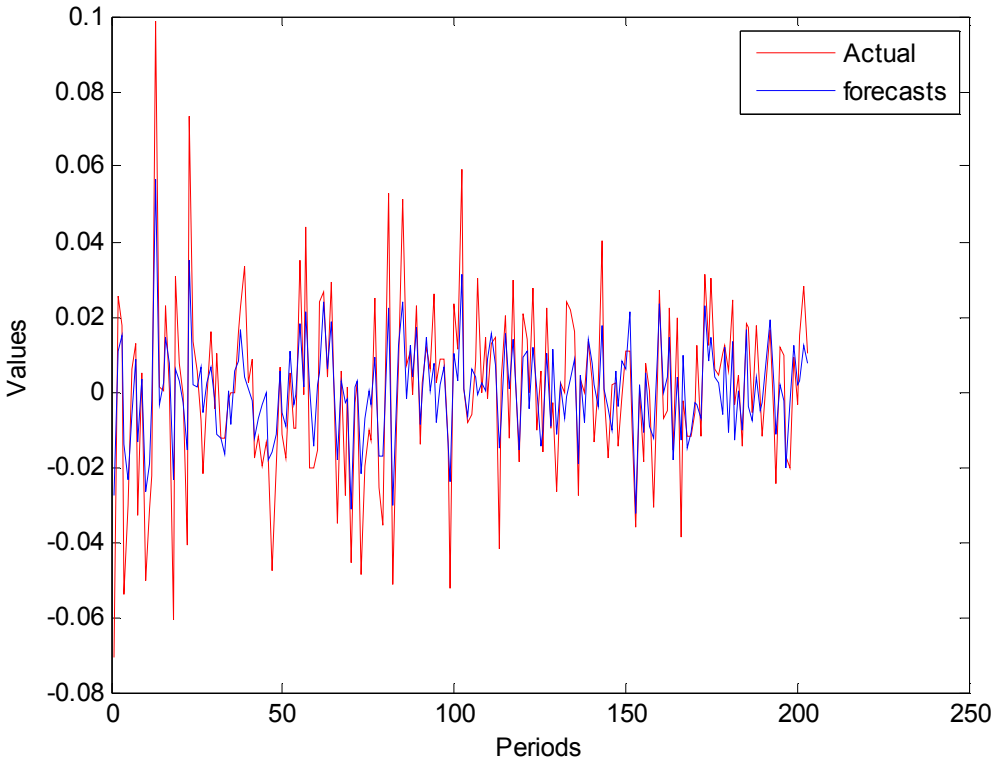

Fig. 3 In-sample forecasts for DAX index returns

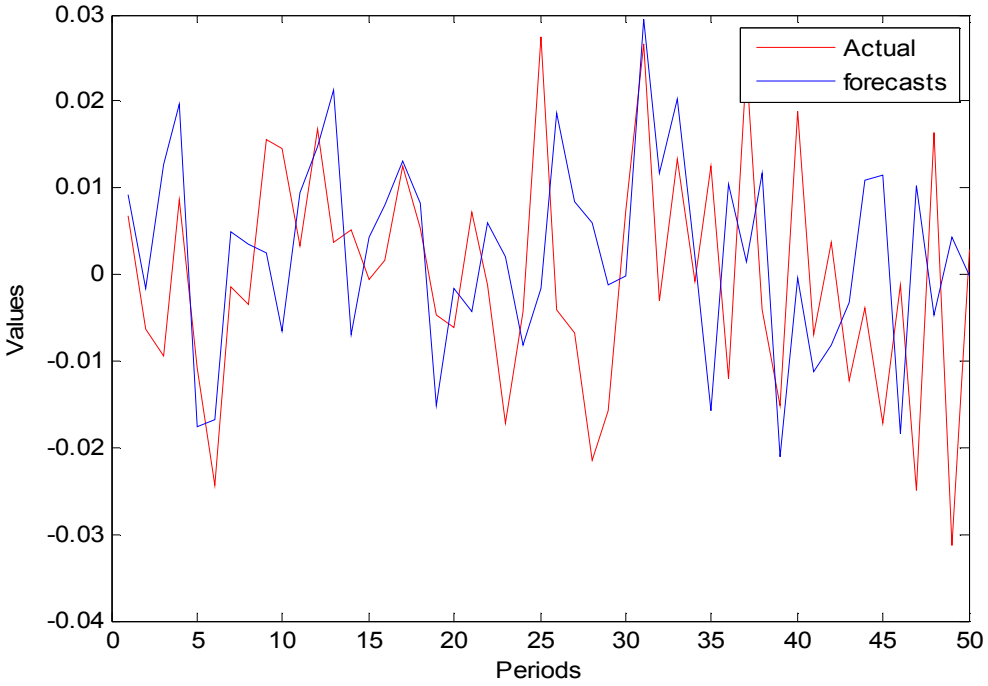

Fig. 4 Out-of-sample forecasts for DAX index returns

In the second application example we examine the inflation rate of USA. We estimate an AR(2) with no constant, maximum epochs at 40, the lambda value is set up at 0.3, error_performance=2 and we apply Cauchy function. The error reached 0.000926 after 44 epochs. The training period is 1968-2008 and 2009 is left for testing. In figures 5 and 6 the in-sample and out-of-sample forecasts respectively are presented, while in figure 7 we present the error after through the training process. Additionally in table 1 we present the estimated coefficients of AR(2) for the inflation ate of USA.

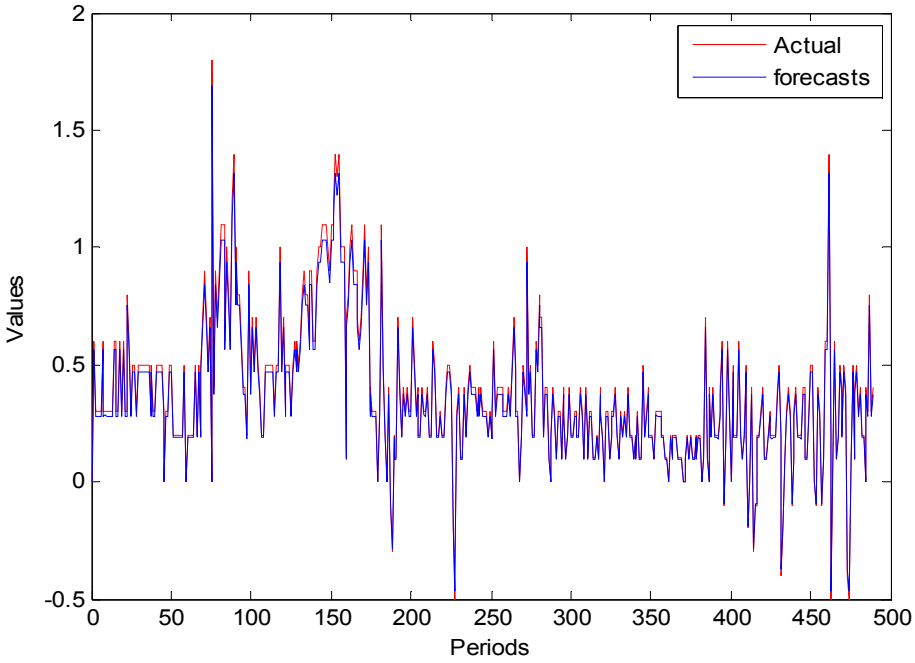

Fig. 5 In-sample forecasts for inflation rate

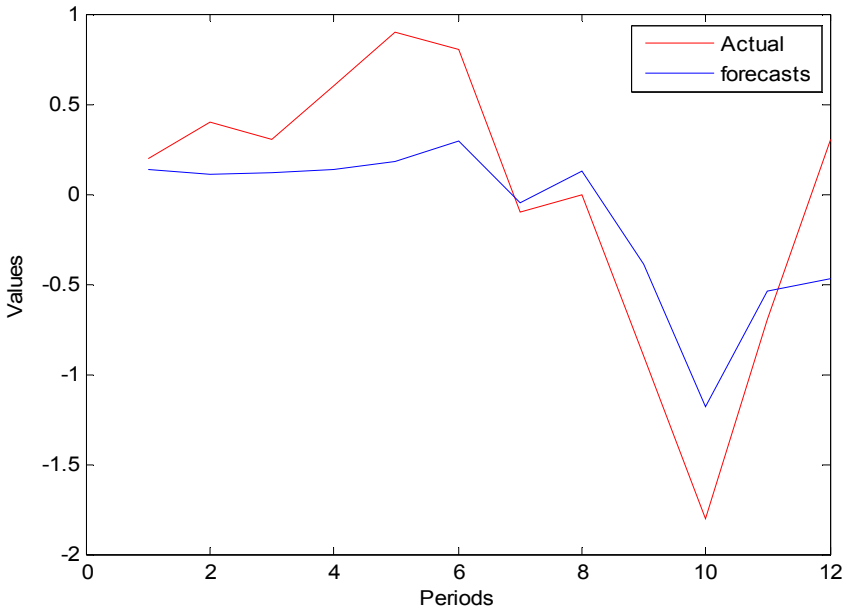

Fig. 6 Out-of-sample forecasts for inflation rate

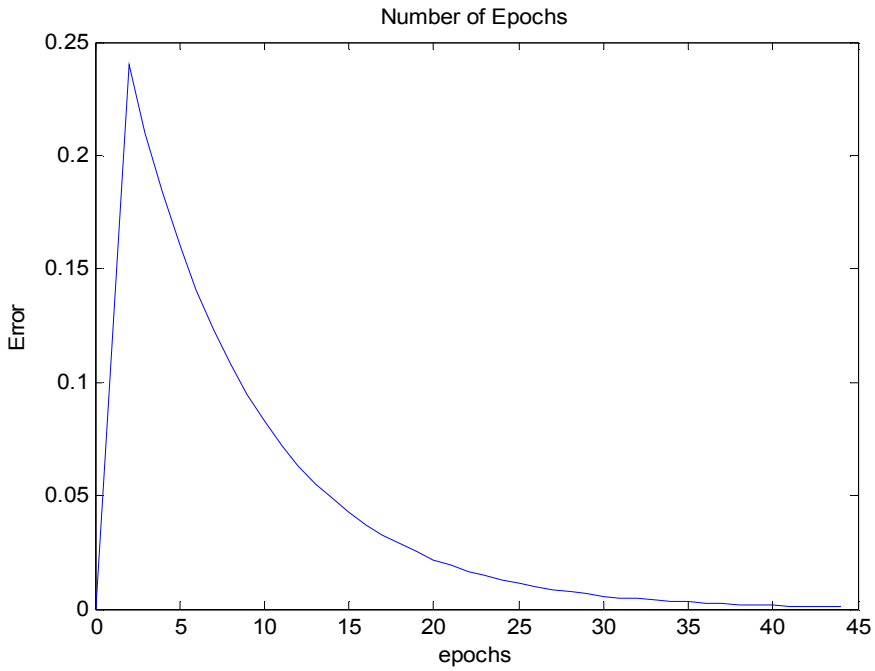

Fig. 7 Error minimization through the training process

Table 1. Estimation AR(2) for US inflation rate

Estimated coefficients		
β_0	β_1	β_2
1.007	0.2072	0.1809
(8.907)*	(8.907)*	(7.783)*

t-statistics in parentheses, * denotes significance in α=0.01

It should be noticed that in routine we take the last value of *y* and *x* as we already know them in order to forecast the new values.

In the last example we examine FTSE-100 stock index returns and we estimate an AR(2) process with no constant, Gaussian function, number of maximum epochs set up at 100 and the goal error at 0.0001. Learning, momentum rates and lambda are set up at 0.1, 0.1 and 0.3 respectively. The error reached its goal after 81 epochs and was equal with 0.0000931. We took the trading days of 2009, where the last 50 of them are obtained as the testing period. The in-sample (train period) and the out-of-sample (test period) are presented in figures 8 and 9 respectively. For comparison we present in figure 10 an Autoregressive AR(2) process, as also we have tested ARMA, MA and GARCH processes with different lag orders to get the best forecasts, but in all cases

the situation is similar with that f figure 10. So we conclude that the traditional financial time-series analysis and forecasting in finance give forecasts reminding "dead lines".

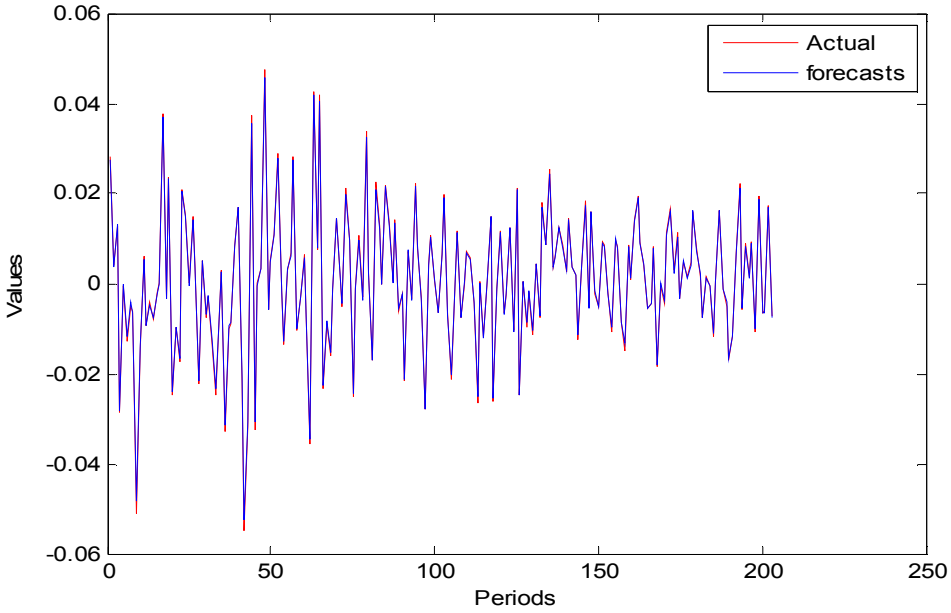

Fig. 8 In-sample forecasts for FTSE-100

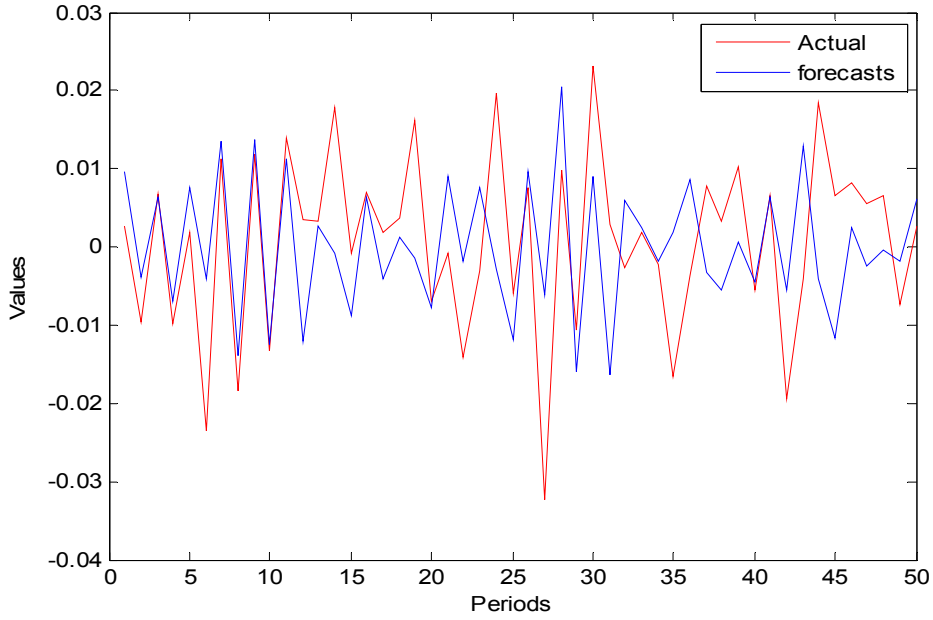

Fig. 9 Out-of-sample forecasts for FTSE-100

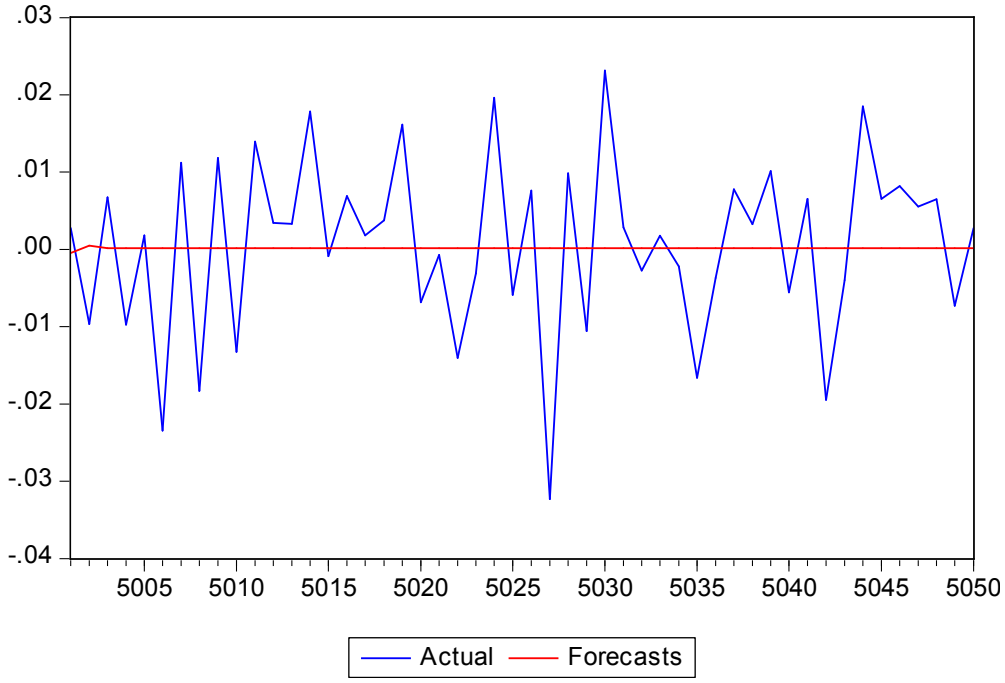

Fig. 10 Out-of-sample forecasts for FTSE-100 with AR(2)

Conclusions

We presented a simple approach for radial basis neural network function. We examined some time-series as the inflation rate and the day the week effect and we have shown how strong the forecast can be using the RBF.

References

Bishop C.M., (1995). Neural Networks for Pattern Recognition. Clarendon Press, Oxford

Bors, A. G. and Gabbouj, G. (1994). Minimal topology for a radial basis function neural network for pattern classification. *Digital Signal Processing: a review journal*, Vol. 4, No. 3, pp. 173-188

Broomhead, D.S. and Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems,* Vol. 2, pp. 321-355.

Casdagli, M., (1989). Nonlinear prediction of chaotic time series. *Physica*, Vol. 35, pp. 335-356.

Chen, S., Cowan, C. F. N. and Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Trans. On Neural Networks*, Vol. 2, No. 2, pp. 302-309.

Niranjan, M. and Fallside, F. (1990). Neural networks and radial basis functions in classifying static speech patterns *Computer Speech and Language*, Vol. 4, pp. 275-289.

Powell, M. J. D. (1987). Radial basis functions for multivariable interpolation.: A review. In J. C. Mason and M. G. Cox (Eds.), Algorithms for Approximation, pp. 143-167. Oxford: Clarendon Press

Sanner, R. M. and Slotine, J.-J. E. (1994). Gaussian networks for direct adaptive control. *IEEE Trans. On Neural Networks*, vol. 3, no. 6, pp. 837-863.

.

Appendix

MATLAB routine 1
Radial basis neural network function with Gaussian and other functions.

This routine is for inflation rate example while similarly can be formulated for other applications

```
clear all;
load inflation1.mat
nforecast=12
y=data(1:end-nforecast,1)
x=data(1:end-nforecast,2:3)

constant=0


[nk1,ni]=size(x)

eta=0.5             % learning rate
spread=1            % 1  for estimating the sigma, 3 for input sigma
                    % of your choise
                    % 3 is the default sigma
default_sigma=1; % The default sigma for option spread=3.

descent=2;          % Set the descent method. 1 for learning rate, 2
                    % for both learning
                    % and momentume rates and 3 with lamda approach
mom=0.4             % Set the momentum rate for the descent method=2.
distance_method=1; % Set the distance method between row i of input
                    % matrix (x)
                    % and  row j of center matrix (center_X). 1 for
                    % Euclidean distance method and 2 for Manhattan
                    % distance method
weights=1;          % Set the weight initialization. 1 for random and
                    % 2 for random and
                    % Nguyen - Widrow initialization
                    % Set the AR process for autoregresive models
lamda=0.3           % Set the smoothing parameter. Very important.
                    % Possible values ranging 0.1-0.5 but not
                    % restricted in those.

if constant == 0
        x=x
elseif constant==1
 x=[ones(nk1,1) x];
end

options='c'         % Set the radial masis function

 cont=2; % Set the continuity value. Takes only values of 0,2,4 and 6

 error_performance=2;       % This option is important for
                            % over-fitting estimations.

[nk1,ni]=size(x)
[nk2,nj]=size(y)
```

```
num_hidden=ni;              % Set the number of hidden neurons



N=ni   % Number of clusters
x=x'
centers = x(:,1:N);
clusters = rem([0:ni-1]',N) + 1;
x=x'


   for i = 1:N
      dist2 = zeros(N,1);
      for j = 1:N
         dista(j) = (x(i,:)'-centers(:,j))'*(x(i,:)'-centers(:,j));
      end
      [mind,mini] = min(dista);
      clusters(i,:) = mini;
   end

   for i = 1:N
      centers = mean(x(find(clusters==i),:));
   end

center_matrix=y - ones(nk1,1)*centers;


 % choose function
for option = options

        if option == 'g'      % Gaussian
            model = 0;
        elseif option == 'c'  % Cauchy
            model = 1;
        elseif option == 'm'  % Multiquadrics
            model = 2;
        elseif option == 'r'  % Reciprocal Multiquadrics
            model = 3;
        elseif option == 'l'  % logistic.
            model = 4;
        elseif option == 's'  % Thin-Plate Spline
            model = 5;
        elseif option == 'w'  % Wendland
            model = 6;
        else
            error('Illegal option of RBF function')
        end
    end


 [nk3,nh]=size(center_matrix);

if nk1~=nk3;
  error('x and center_X mismatched');
end;


% Set up the weight matrix
```

```
if weights==1;
a=-0.05
b=0.05
%rand('state',sum(100*clock))         % Resets it to a different
state each time.
rand('state',0)                       % Resets the generator to its
initial state.
w=a + (b-a) *rand(num_hidden,nk1);

elseif weights==2;
gamma = 0.7*ng^(1/u);
a=-0.05
b=0.05
%rand('state',sum(100*clock))           % Resets it to a different
state each time.
rand('state',0)                       % Resets the generator to its
initial state.
w=a + (b-a) *rand(ni,nk1);
w = gamma*w/sqrt(sum(sum(w.^2)));

end


% Compute sigma
if spread==1;

sigma = 1/(2*size(center_matrix, 1)^(1/nk1)-
2)/sqrt(2*log(2))*ones(num_hidden, 1);
sigma=sigma(1,1)


elseif spread==2;
    sigma=input('Input spread value')

elseif spread==3;
   sigma= default_sigma
end

goal_error=0.001;
maxepochs=50;
epochs=1;
error = 10;

 % Start the training process
  while (epochs < maxepochs & error > goal_error)


%if nj == 1,
    %distance = abs(x'*ones(nk1,nj)-ones(nk2,ni)'*center_X);
    %elseif ni >= nj,
    %for i = 1:nj,
    %   distance(i,:) = sqrt(sum(((ones(nk1,nj)*x(i,:)'-
center_X(:,1))').^2));
    %end
    %else
    %for i = 1:ni,
        %distance(:,i) = sqrt(sum(((x(:,i)-
ones(nk1,nj)*center_X(i,:)')').^2))';
        %end
```

```
        %end

if distance_method==1;
distance=dist(x',center_matrix)
elseif distance_method==2;
distance=mandist(x',center_matrix)
end


for j = 1:nj
    if model == 0                              % Gaussian

if numel(sigma) ==1
    s =  exp(distance/sigma);
    s = lamda*s;
elseif size(sigma,1)> 1           % when each center has its own
width
   s= zeros(size(distance));
   s = lamda*s;
    for i = 1:size(distance,2)
        s(:,i) = exp(distance(:,i)/sigma(i));
        s = lamda*s;
    end
else                              % when each dimension has its own
width
    s =  exp(-distance);          % distance has already been weighted
end

    elseif model == 1                          % Cauchy
         s =1./((exp((-1/(2*sigma^2)*distance.^2))+1));
         s = lamda*s;

    elseif model == 2                          % Multiquadrics
        if numel(sigma) ==1
    s = sqrt(distance + sigma^2);
    s = lamda*s;
elseif size(sigma,1)> 1 % when each center has its own width
   s= zeros(size(distance));
   s = lamda*s;
   for i = 1:size(distance,2)
      s(:,i) = sqrt(distance(:,i) + sigma(i)^2);
      s = lamda*s;
   end
else                     % when each dimension has its own width
    s =  sqrt(distance + 1); % dis has already been weighted
    s = lamda*s;
end

    elseif model == 3                          % Reciprocal
Multiquadrics
       if numel(sigma) ==1
    s = 1./sqrt(distance + sigma^2);
    s = lamda*s;
elseif  size(width,1)> 1 % when each center has its own width
   s= zeros(size(distance));
   s = lamda*s;
   for i = 1:size(distance,2)
      s(:,i) = 1./sqrt(distance(:,i) + sigma(i)^2);
      s = lamda*s;
   end
```

```
else
     %when each dimension has its own width
    s =  1./sqrt(distance + 1); % dis has already been weighted
    s = lamda*s;
end


    elseif model == 4                          % Logistic
        distance= sqrt(distance);
if numel(sigma) ==1
    s = 1./(1 + exp(distance/sigma));
    s = lamda*s;
elseif size(sigma,1)> 1% when each center has its own width
   s= zeros(size(distance));
   for i = 1:size(distance,2)
      s(:,i) = 1./(1 + exp(distance(:,i)/sigma(i)));
      s = lamda*s;
   end
else
   s = 1./(1 + exp(distance));
   s = lamda*s;
end

       elseif model == 5                           % Thin-Plate Spline
        if numel(sigma) ==1
    r= distance/sigma^2;
    s =  (r).*log(r)/2;
    s = lamda*s;
elseif size(sigma,1)> 1 % when each center has its own width
   s= zeros(size(distance));
   s = lamda*s;
   for i = 1:size(distance,2)
      ri = distance(:,i)/sigma(i)^2;
      s(:,i) =(ri).*log(ri)/2 ;
      s = lamda*s;
   end
else % when each dimension has its own width
    s =  (distance).*log(distance)/2;
    s = lamda*s;
end

  elseif model == 6                          % Wendland
   distance= sqrt(distance);
if numel(sigma) ==1
    r = distance/sigma;
elseif size(width,1)> 1 % when each center has its own width
   r= zeros(size(distance));
   for i = 1:size(distance,2)
      r(:,i) = distance(:,i)/sigma(i);
   end
else %when each dimension has its own width
    r = distance;% dis has already been weighted
end


ell = floor(ni/2) + cont/2 + 1;
switch cont
   case 0
         exponent = ell;
         polycoeff = 1;
```

```
   case 2
         exponent = ell +1;
         polycoeff = [ell+1, 1];
   case 4
         exponent = ell +2;
         polycoeff = [(ell^2 + 4*ell + 3), (3*ell+6) , 3];
   case 6
         exponent = ell +3;
         polycoeff = [(ell^3 + 9*ell^2 + 23*ell + 15), (6*ell^2 +
36*ell + 45), (15*ell+45) , 15];
end

s = (1-r).^(exponent);
s = (r<1).*s;
s = s.*reshape(polyval_mex(polycoeff,r), size(r));
s = lamda*s;
end
end


yhat=w'*s       % in-sample forecasts

  target_y=y
  err1 =target_y -yhat ;
  if error_performance==1;
     error=err1'*err1
  elseif error_performance==2;
error = sum(sum(err1.^2))/prod(size(err1));;
elseif error_performance==3;
gamma=0.2;                    % is the performance ratio
perfe = sum(sum(err1.^2))/prod(size(err1));
perfw = sum(sum(w.^2))/length(w);
error = gamma*perfe + (1-gamma)*perfw
end



  % Plot the trained data versus the actual to examine if there is a
  % satisfying approach. Try for values of lamda to see which is the
best
  % fitted and choose this value.


  % Back Propagation for output layer
    dyhat = -2*(y - yhat);  % dE/dyhat
    ddw = s*dyhat';

    % Back Propagation for hidden layer
    dds = dyhat.'*w';           % dE/dyhat
    dsSigma = s.*(distance.^2*diag(sigma.^(-3)));

    ddSigma = sum(dds.*dsSigma')';

    dCenter = diag(sigma.^(-2))*((dds*s)*y'-
diag(sum(dds*s))*center_matrix');


    % Simple steepest descent
    dw = -eta*ddw;
```

```
    if descent==1;
        w = w + dw;
    elseif descent==2;
        w1=w
        w = w + dw;
        w = w + mom*(w-w1);

    end

    dSigma = -eta*ddSigma;
    XdCenter = -eta*dCenter;
    sigma = sigma + dSigma;
    center_matrix=center_matrix+XdCenter';

 epochs=epochs+1
array_y ( epochs )= error;
indexiter ( epochs ) = epochs;

end


% CGV criterion. Try for different values of lamda and learning rate
and
% the lowest values for CGV criterion are preferred.
I=eye(size(x*x'))
CGV=(yhat'*(I-x*x').^2*yhat)/((trace(I-x*x'))^2)
% Error criterion. Try for different values of lamda and learning
rate and
% the lowest values for Error criterion are preferred.
ss = s*s';
syhat = s*yhat';
[p, m] = size(s);
L = diag(lamda * ones(p,1));
A = inv(ss + L);
W = A * syhat;
P = eye(m) - s'*A*s;
Pyhat = P * yhat;
yhatP = sum(diag(Pyhat, Pyhat'));
g = p - trace(P);
psi = p^2 / (p - g)^2;
e = [error  psi * yhatP / p];
e=e*e'


y_test=data(end-nforecast+1:end,1)
x_test=data(end-nforecast+1:end,2:3)
if constant == 0
        x_test=x_test
elseif constant==1
 x_test=[ones(size(y_test)) x_test];
end

w=w'
for i=1:nk2
    for ii =1:ni
   newx(i,ii)=w(i,ii).*x(i,ii);
    end
```

```
end

bols=inv(newx'*newx)*newx'*y
%bols=pinv(newx)*y
res=y-newx*bols
s2 = (y-newx*bols)'*(y-newx*bols)/(nk1-ni);
Vb=s2*inv(newx'*newx);              % Get the variance-covariance
se=sqrt(diag(Vb));                  % Get coefficient standard errors
tstudent=bols./se;                  % Get t-statistics
yf1=x_test*bols                     % Get out-of-sample forecasts

figure, plot(y,'-r'); hold on; plot(yhat,'-b');
xlabel('Periods')
ylabel('Values')
%title('In_sample forecasts')
h1 = legend('Actual','forecasts',1);
figure, plot(y_test,'-r'); hold on; plot(yf1,'-b');
xlabel('Periods')
ylabel('Values')
%title('Out_of_sample forecasts')
h1 = legend('Actual','forecasts',1);
figure, plot (indexiter , array_y );
xlabel('epochs')
ylabel('Error')
title('Number of Epochs')
```

MATLAB routine 2
Test sample for the day of the week effect, while the process is different.

```
if weights==1;
a=-0.05
b=0.05
%rand('state',sum(100*clock))          % Resets it to a different
state each time.
rand('state',0)                        % Resets the generator to its
initial state.
w_test=a + (b-a) *rand(num_hidden,nk1);

elseif weights==2;
gamma = 0.7*ng^(1/u);
a=-0.05
b=0.05
%rand('state',sum(100*clock))            % Resets it to a different
state each time.
rand('state',0)                        % Resets the generator to its
initial state.
w_test=a + (b-a) *rand(ni,nk1);
w_test= gamma*w_test/sqrt(sum(sum(w_test.^2)));

end


yf1=w_test'*s

figure, plot(y,'-r'); hold on; plot(yhat,'-b');
xlabel('Periods')
ylabel('Values')
```

```
%title('In_sample forecasts')
h1 = legend('Actual','forecasts',1);
figure, plot(y_test,'-r'); hold on; plot(yf1,'-b');
xlabel('Periods')
ylabel('Values')
%title('Out_of_sample forecasts')
h1 = legend('Actual','forecasts',1);
figure, plot (indexiter , array_y );
xlabel('epochs')
ylabel('Error')
title('Number of Epochs')
```