A Comparison of Standard Methods with a Neural Network Approach to Forecast Univariate Time Series

Karl KRYCHA

Forschungsbericht/
Research Memorandum No. 310
November 1992

Zusammenfassung

Die vorliegende Arbeit untersucht die Anwendung neuraler Netze zur Modellierung von Zeitreihen. Dazu wurden anhand einiger Datensets die Ergebnisse einer einfachen Netzwerkstopologie mit den Ergebnissen ökonometrischer Verfahren bzw. Verfahren des Operations Research verglichen. Die Schätzung der Netzwerksparameter erfolgte anhand verschiedener Trainingsmethoden (WIDROW/HOFF sowie Erweiterungen) und erbrachte teilweise verbesserte Ergebnisse. Alle Netzwerkberechnungen wurden mittels einer dokumentierten Programmroutine durchgeführt.

Abstract

The following paper tries to develop a simple neural network approach to analyse time series data. After an introductional part some standard approaches are presented. A neural network is developped and results on some different data sets are compared. It will be shown that even very simple and rather small network topologies may capture in a very efficient way time series characteristics.

CONTENTS

1. INTRODUCTION

Extrapolative forecasting procedures are methods for modeling predictions of future observations based only on current and past observations. This might appear to be a fairly simple task, yet the literature contains a vast number of techniques (HARVEY, 1984, p.245). These range from simple methods, such as the exponantially weigthed moving average, to the more complex procedures based on the ARIMA models of BOX and JENKINS (1976).

The subject of forecasting therefore has occupied an important role in the theory of stochastic processes and time series analysis. The earliest work (see MEHRA, 1979, p.75) on the subject is that of KOLMOGOROV (1941) for discrete-time stationary stochastic processes and that of WIENER (1949) for continuous-time processes.

Without going much into further details at this point it may be stated that investigating time series by means of statistical methods is some kind of an art. Furthermore every possible method has its own underlying assumptions which have to be accounted for very carefully.
On the other hand neural networks have a good reputation for pattern recognition and are able to learn and to adopt to changing circumstances. Neural networks may also incorporate several different inputs at the same time and therefore form a kind of "natural" multivariat tool (see below).

1.1. Neural Networks vs Statistics

A knowledge of statistics is excellent preparation for appreciating the power and flexibility of neural networks. In statistics, one must make many assumptions about the data, and must sometimes limit the analysis to a certain number of

possible interactions. By contrast from a practical point of view, neural networks are basically "non-parametric", although one can think of a neural network as being parametrized by its weigths. In addition, more terms can be examined for interaction by a neural network, since the network will place its emphasis on those inputs that help to predict the output. By allowing more data to be analyzed at the same time, more complex and subtle input interactions are possible. It should be stressed already at this point that statistics can be helpful in understanding the data, which can lead to developing a better neural network model (see NWorks, 1991, p.6).

1.2. Outline

This paper aims to compare some basic statistical methods in univariate time series analysis with a neural network approach. Six different time series will be investigated by

- two different exponential smoothing models
- ARIMA models
- and by a neural network approach

By means of an error statistic a valuation of all methods shall be effectuated; the results of this work will lead to an enhanced method for use in a material planning system. This *neural prognosis method* will account for information given from different points in a PPS system.
The former two approaches are effected by **RATS386 Version 3.11**. RATS is a computer package for econometrics, forecasting and statistical analysis. While its primary focus is upon time series data, it may also be used for cross sectional and panel data sets (DOAN, 1991).

For the neural network approach the author developped a program to ensure the necessary flexibility in testing different training approaches. The program is written in ANSI C (see KERNIGHAN/RICHIE, 1988).

As data input six empirical data series of a current research project were used. **APPENDIX I** gives the exact data; throughout this paper graphics only will show the first series. All other results will be compared by means of the Sum of Sqared Residuals ($SSR = e'e$).

2. Moving Averages and Exponential Smoothing

2.1. Moving Averages

Forecasts for observation z in period t are computed by the moving average M_t of a certain amount L of past demand values z_{t-1}, ... ,z_{t-L+1}, i.e. the estimation function is the arithmetic mean M_t of the last L periods. In this case we call the model moving average of order L. The estimation function

$$\hat{z}_{t+1} = M_t = \frac{z_t + z_{t-1} + \ \dots \ + z_{t-L+1}}{L}$$

minimizes the sum of sqared errors (see SCHNEEWEISZ, 1981, p. 90):

$$\sum_{j=0}^{L-1} (\hat{z} - z_{t-j})^2 .$$

This equation may be updated recursively and is found often in the following form:

$$z_t = z_{t-1} + \frac{1}{L}(z_t - z_{t-L}) .$$

2.2. Exponential Smoothing

As computing prognosis values is this simple, moving average methods are quite common in practice. A critical question is to determine the order L of the process as estimates are very sensible to it. An extreme example is is given in cases were L is set to $L = 1$ as in this form the estimate turns out to be $\hat{z}_t = z_{t-1}$ and therefore the last known value determines alone the prognosis value.
Higher order moving averages tend to "stabilize" the whole process and may lead to slow reactions to changes in the series pattern (op.cit., p.91).

The exponential smoothing technology chooses one from a small group of models (see Table 2-1) which focus upon the trend and

seasonal behaviour of the data. Because those two aspects overwhelmingly determine the variance of the series, properly chosen exponential smoothing models can do a good job relative to more complicated methods on a wide range of data series (DOAN, 1989, p.7-6).

Advantages of exponential smoothing techniques are their computational simplicity, so that they can be applied to a large number of series quickly. Plus, the small set of choices make specifications of the "best" model extremely simple, and make them ideal for very small data sets.

This leds to disadvantages of this class of models; it may be to narrow for some data series. For instance, a "no trend" model actually is a random walk type of model, so that it forecasts a constant level, which is very bad, especially at longer horizons, for series which return to a mean level (op.cit., p.7-6).

Table 2-1 lists the error-correction forms of the models used for the different combinations of seasonal (top) and trend (left).

Table 2-1: Exponential Smoothing Models; op.cit., p.14-69

S_t	...	smoothed level of the series
T_t	...	trend rate
I_t	...	seasonal index (factor)
e_t	...	period t forecast error

	NONE	LINEAR	EXPON
NONE	$S_t = S_{t-1} + \alpha e_t$ $I_t = I_{t-p} + \delta(1-\alpha)e_t$	$S_t = S_{t-1} + \alpha e_t$ $I_t = I_{t-p} + \delta(1-\alpha)e_t + S_t$	$S_t = S_{t-1} + \alpha e_t + I_{t-p}$
LINEAR	$S_t = S_{t-1} + T_{t-1} + \alpha e_t$ $T_t = T_{t-1} + \alpha\gamma e_t$ $I_t = I_{t-p} + \delta(1-\alpha)e_t$	$S_t = S_{t-1} + \alpha e_t$ $T_t = T_{t-1} + \alpha\gamma e_t$ $I_t = I_{t-p} + \delta(1-\alpha)e_t + S_t$	$S_t = S_{t-1} + T_{t-1} + \alpha e_t + I_{t-p}$ $T_t = T_{t-1} + \alpha\gamma e_t + I_{t-p}$
EXPON	$S_t = S_{t-1}T_{t-1} + \alpha e_t$ $T_t = T_{t-1} + \alpha\gamma e_t + S_{t-1}$ $I_t = I_{t-p} + \delta(1-\alpha)e_t$	$S_t = S_{t-1}T_{t-1} + \alpha e_t$ $T_t = T_{t-1} + \alpha\gamma e_t + S_{t-1}$ $I_t = I_{t-p} + \delta(1-\alpha)e_t + S_t$	$S_t = S_{t-1}T_{t-1} + \alpha e_t + I_{t-p}$ $T_t = T_{t-1} + \alpha\gamma e_t + (I_{t-p}S_{t-1})$

2.3. Computational results

The given time series were analysed by two different exponential smoothing models, namely by one using a linear trend factor and the second one using a linear trend again as well as a multiplicative seasonal factor. Computing an exponential smoothing model by RATS needs making a decision between setting the parameters to some values or estimating them. In the case of estimating factors, RATS chooses them by non-linear least squares, minimizing the in-sample squared one-step forecast errors (DOAN, 1989, p.7-6).

Subsequently the output file with results for series kk1a is printed. Table 2-2 gives an overview of all estimated parameters for all considered series as indicated in **APPENDIX I**. The copy file of the procedure is not printed.

```
env columns=110
open data a:\kk1a
open output a:\esmokk1a.out
open copy a:\esmokk1a.dat
all 0 66
data(org=obs) / ts1
graph(header='Observations') 1
# ts1
 esmooth(estimate,trend=linear,smooth=ts2) ts1

 CONVERGENCE REACHED ON ITERATION     19
 Estimated Coefficients    0.4875477  -0.0232977
 Sum of Squared Errors    180.53146
 esmooth(estimate,trend=linear,seas=mult,smooth=ts3) ts1

 CONVERGENCE REACHED ON ITERATION     12
 Estimated Coefficients  -0.0159287   1.7733172   0.6874097
 Sum of Squared Errors    182.37491
 graph(header='EXSMOOTH kk1a',number=1,subhead='Trend only') 2
# ts1
# ts2
graph(header='EXSMOOTH kk1a',number=1,subhead='Trend and Saison') 2
# ts1
# ts3
copy(org=obs,format='(F6.3,F6.3,F6.3)') / ts1 ts2 ts3
end
```

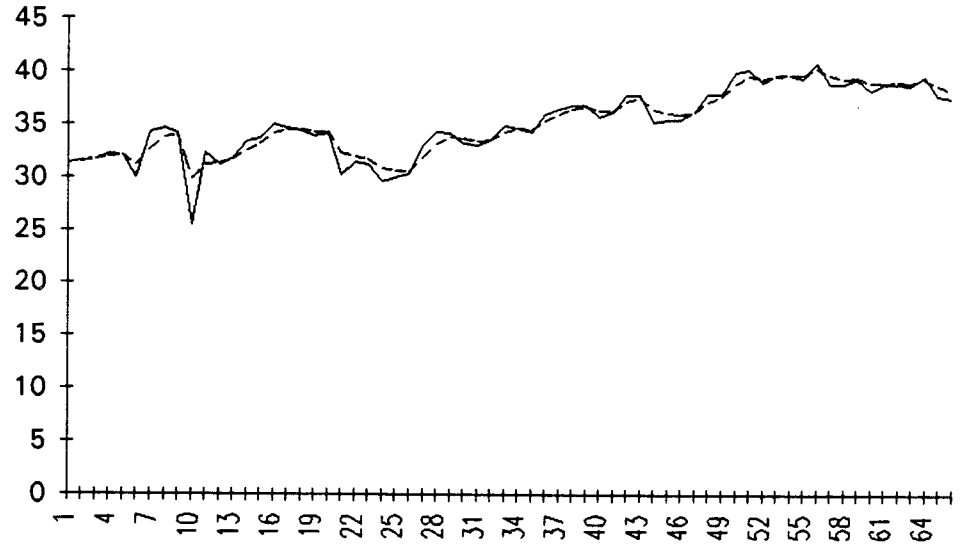
Table 2-2: Estimation and evaluation results
Exponential Smoothing models

series	SSR	α	γ	SSR	α	γ	δ
kk1a	181	0.49	-0.02	182	-0.02	1.77	0.69
kk1b	454	0.30	-0.01	483	0.34	0.01	0.29
kk1c	680	0.32	-0.01	738	-0.02	0.04	0.62
kk3a	142	0.66	-0.05	137	0.22	-0.28	0.63
kk3b	472	0.1	-0.06	604	-0.02	0.5	0.62
kk3c	443	0.78	-0.04	678	0.3*	0.3*	0.3*

* standard starting values as numerical difficulties with this data set appeared

See for detailed results concerning series kk1a figure 2-1. The original series will be printed throughout the paper in solid lines whereas estimates are indicated in dotted lines.

Figure 2-1: Exponential Smoothed series kk1a
Model with linear trend only;



3. ARIMA Modeling Techniques

Generally speaking this method offers a broad collection of models for fitting serial correlation pattern. Because of the number of possible parameters, it can handle many type of time series, but choosing an appropriate model is something of an art (see DOAN, 1989, p.7-3).

3.1. The BOX-JENKINS modeling procedure

The objective of "modeling" a time series in this case is to find the mechanism that converts original observations to a series of serially uncorrelated values called white noise. This mechanism, or filter, does not change over time, and thus it can be used to predict future values of the time series strictly from its own past. The filter should be chosen so that all deterministic and systematic parts of the time series are captured, leaving only the white noise component unaccounted for (see HANSSEN et al., 1990, p.115 f).

YULE proposed that time series observations are generated by an underlying stochastic process:

$$p(z_1, z_2, \ldots, z_{t-1}, z_t, \ldots z_T)$$

in such a way that each observation z_t is a *linear combination* of all previous shocks:

$$z_t = \alpha_t + \psi_1 \alpha_{t-1} + \psi_2 \alpha_{t-2} + \ldots + \psi_t \alpha_0 \, .$$

This underlying stochastic process is assumed to be stationary, i.e., it remains the same over time. In practice, we check the stationarity assumption by verifying that the first two moments of the distribution, i.e., the mean and the variance, are constant (see HANSSEN et.al., 1990, p.117):

$$E(Z_t) = \mu$$

and

$$E(Z_t - \mu)^2 = \sigma^2 \, .$$

The analysis of stationary data tries to transform it to white noise by means of two distincte linear filters, the *autoregressive* (AR) *filter* and the *moving-average* (MA) *filter*. The AR filter is generally denoted by $\Phi(L)$ and operates on the left-hand side of the time series equation (z_t); the MA filter $\Theta(L)$ operates on the right-hand side (α_t). Thus, the basic univariate time series model is (op.cit., p.119)

$$\Phi(L)z_t = \Theta(L)\alpha_t \ .$$

Rearranging terms shows that the linear filter $\Psi(L)$ has been decomposed in an AR and MA component:

$$z_t = \frac{\Theta(L)}{\Phi(L)}\alpha_t = \Psi(L)\alpha_t \ .$$

Thus, the observed stationary series $z_t(t = 1,2, \ldots ,T)$ is generated by an unobservable white noise series $(\alpha_0,\alpha_1, \ldots ,\alpha_t)$, filtered by the linear filter $\Psi(L)$:

$$\begin{aligned} z_t &= \alpha_t + \psi_1\alpha_{t-1} + \psi_2\alpha_{t-2} + \ \ldots \\ &= \alpha_t + \psi_1 L\alpha_t + \psi_2 L^2\alpha_t + \ \ldots \\ &= (1 + \psi_1 L + \psi_2 L^2 + \ \ldots \ )\alpha_t \\ &= \Psi(L)\alpha_t \ . \end{aligned}$$

where L is the lag operator, e.g., $L^k z_t = z_{t-k}$.

The AR filter is infinite, so its weigths must converge in order to preserve stationarity. This implies that the parameter ϕ must be less than 1 in absolute value:

$$(1 - \phi L)^{-1} = 1 + \phi L + \phi^2 L^2 + \phi^3 L^3 + \ \ldots$$

as long as $-1 < \phi < 1$. The MA filter is finite, so stationarity is always implied (see op.cit., pp.120f). This model may now be generalized to an autoregressive process of order p ($AR(p)$) and a moving-average process of order q ($MA(q)$). The combination produces an $ARMA(p,q)$ model, which can be written as

$$(1-\phi_1 L-\phi_2 L^2-\ \ldots\ -\phi_p L^p)z_t=(1-\theta_1 L-\theta_1 L^2-\ \ldots\ -\theta_q L^q)\alpha_t\ .$$

BOX and JENKINS (see op.cit., pp131ss.) proposed a three-step modeling procedure, consisting of identification, estimation, and diagnostic checking; they also proposed two useful tools for the identification process. The *autocorrelation function* (ACF) at lag *k* is simply the correlation of two data points that are *k* periods apart. The *partial autocorrelation function* (PACF) at lag *k* is a similar correlation, but it holds constant all (*k-1*) observations between the two data points.
In the above mentioned process the modeler first selects a candidate ARMA process by inspecting the ACF-PACF of the original series. Next, the model's parameters are estimated and the residuals are stored. If the candidate model is adequate, i.e., if the proposed filter does whiten the data, then the ACF-PACF of the residuals should be flat. If not, then some spikes will remain and their pattern will suggest ways of improving the original model.

Seasonal time series are not necessarily non-stationary and can exhibit seasonal patterns even after differencing (*d* indicates the number of regular differencings). The BOX-JENKINS model takes such patterns into account by adding seasonal autoregressive and moving-average parameters to the base equation (see op.cit., p.136):

$$\Phi(L)\Phi_s(L)(1-L)^d(1-L^s)^{d_s}z_t=c+\Theta(L)\Theta_s(L)\alpha_t\ ,$$

where $\Phi_s(L)$ denotes the seasonal autoregressive process and $\Theta_s(L)$ the seasonal moving-average parameters. The model is multiplicative and is generally represented as ARIMA(p,d,q) $(p_s,d_s,q_s)_s$, where s is the length of the season.

3.2. Computational results

As in chapter 2 all series have been analysed according to the scheme proposed by BOX and JENKINS. After identification for every series the best fitting model has been estimated. All computations have been done by RATS; in this software the procedure BOXJENK uses the Gauss-Newtown algorithm (see DOAN, 1991, p.14-17) with numerical derivatives. See table 3-1 for results.

The output file for series kk1a is partly printed below. See figure 3-1 for original (solid) and fitted (dotted) series kk1a.

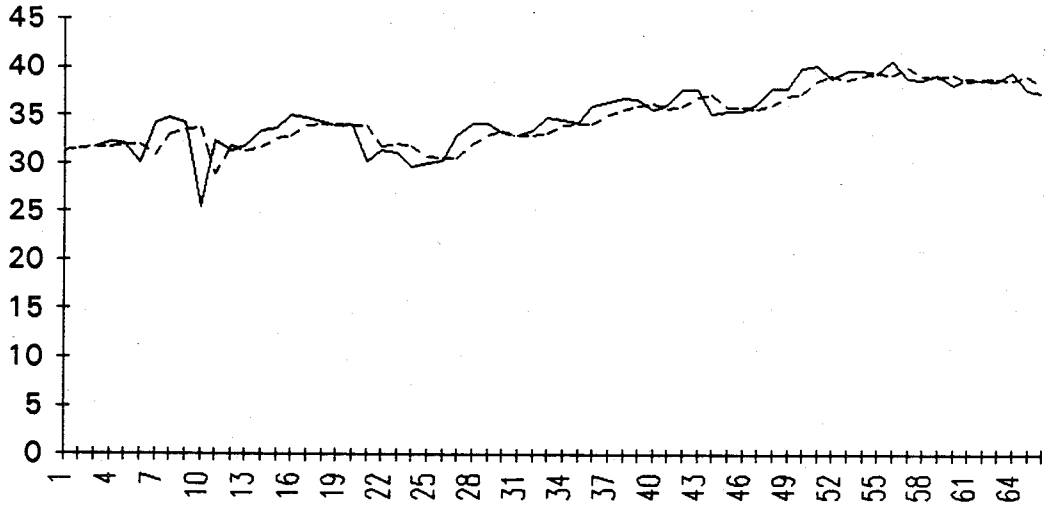
```
all 0 66
data(org=obs) / oo
*
*         oo ... original observations
*
     :              :
     :              :
*
correlate oo 1 66 16 ACF
AUTOCORRELATIONS OF SERIES    1     OO            VARIANCE  10.678
NUMBER OF OBSERVATIONS   66
   1 :  0.829051  0.766415  0.704650  0.676067  0.618568  0.549697
   7 :  0.529855  0.529542  0.525534  0.444368  0.447933  0.379818
  13 :  0.342837  0.314243  0.265086  0.217987

partial oo 1 66 16 PACF
PARTIAL AUTOCORRELATIONS OF SERIES    1     OO
NUMBER OF OBSERVATIONS   66
   1 :  0.829051  0.252945  0.069256  0.115310 -0.032516 -0.090428
   7 :  0.098065  0.129633  0.072615 -0.206256  0.116881 -0.180136
  13 : -0.053818  0.103402 -0.077591 -0.116020
*
     :              :
     :              :
*
BOXJENK(diffs=1,ar=1,ma=1) oo / RESIDS

CONVERGENCE REACHED ON ITERATION    10


DEPENDENT VARIABLE     1     OO
TOTAL OBSERVATIONS      64     SKIPPED/MISSING          0
USABLE OBSERVATIONS     64     DEGREES OF FREEDOM      62
R**2            0.72983611     RBAR**2         0.72547863
SSR           182.80447        SEE           1.7171078
DURBIN-WATSON  2.02233476
Q(  24)=   14.3194        SIGNIFICANCE LEVEL 0.93910459
NO.      LABEL     VAR  LAG     COEFFICIENT    STAND. ERROR    T-STATISTIC
***     *******    ***  ***    ************    ************   ************
  1      AR          1   1       0.2678333       0.2268348      1.180742
  2      MA          2   1      -0.7070789       0.1673503     -4.225144


graph(header='residuals ARMA(1,1,1)') 1
# RESIDS
*
     :              :
     :              :
*
end
```

Table 3-1: Estimation and evaluation results
ARIMA Modeling

series	fitted model	SSR	Parameters	t^{Stat}
kk1a	ARIMA(1,1,1)	183	AR 0.268	1.18
			MA -0.707	-4.23
kk1b	ARIMA(0,1,1)	466	MA -0.656	-6.92
kk1c	ARIMA(0,1,1)	695	MA -0.659	-7.01
kk3a	ARIMA(1,1,0)	150	AR -0.323	-2.71
kk3b	ARIMA(0,1,1)	527	MA -1.718	-8.24
kk3c	ARIMA(1,2,1)	989	AR -0.06	-0.468
			MA -1.086	-578

However, the SSR statistics has to be interpretet carefully. It allows for comparisions between different models for one series only. Intermodel differences may be due to scale differences or number of observations and so on. In our case the SSR shall serve as goodness of fit indicator for different models applied to the same series and has therefore been selected.

Figure 3-1: Filtered time series kk1a
ARIMA modelling



4. Neural Prognosis Techniques

4.1. Neural Network Computations

This chapter aims to solve the modeling problem by an application of a connectionist network. A neural network consists of a set of computational units (cells) and a set of one-way data connections joining these units (see Gallant, 1992, p.1). Each of these directed arcs has a numerical weigth, w_{ij}, that specifies the influence of cell u_j on cell u_i. A subset of cells, $\{u_1, \ldots, u_p\}$, are considered as network inputs that are set externally and that do not recompute their outputs (op.cit., p.9).

In this application we use also a finite set of training examples $\{E^k\}$. E^k is a p-vector of values that gives settings for the corresponding input cells. We will apply supervised learning by adopting time series adequately; at the same time correct responses $\{C^k\}$ will be constructed: Each example E^k is therefore associated with a correct response C^k (see op.cit., p.11).
By convention there is a cell u_0 whose output is always $+1$ that is connected to every other cell u_i (except for input cells). The corresponding weigths ($w_{i,0}$) are called biases. Weigths $w_{i,*}$ are refered to as weigths of cell u_i and will be stated as W (or W^k, see chapter 4.3).

As time series data contains usualy of real valued data, a Backpropagation Network (BPN) topology will be taken into consideration. In this case inputs and activations are continuous, assuming values on [0,1] or [-1,+1]. Activations may be computed as follows (op.cit., p.17):

$$S_i = \sum_{j=0}^{i-1} w_{i,j} u_j$$

$$u_i = \begin{cases} \dfrac{1}{1+e^{-S_i}} & \text{if activations in } [0,1] \text{ are used} \\ -1 + \dfrac{2}{1+e^{-S_i}} & \text{if activations in } [-1,+1] \text{ are used} \end{cases}$$

4.2. Gradient Descent Learning

For a linear network a suitable set of weigths may be explicitly computed using the *pseude-inverse* method (HERTZ et.al., 1991, p.102). However, this method requires the input patterns to be linearly independent. If on the contrary there exists a linear relationship (as assumed in the linear filter theory, see chapter 3.1.) between them, then outputs cannot be independly chosen and the problem is normaly insoluble.

A major one-layer learning system of the 1950s and early 1960s, namely the *least-mean-sqare* (LMS) learning procedure of WIDROW and HOFF (1960) shall be considered.
The LMS procedure makes use of the delta rule for adjusting connection strengths; the perceptron converge procedure is very similar, differing only in that linear threshold units are used instead of units with continuous-valued outputs (McCLELLAND/RUMELHART, 1988, p.126 ss.).

As some preliminary tests with use of an activation function as indicated above led to no computational improvements (in a strictly layered network any activation function undertakes only a linear transformation of the inputs and therefore does not affect gradient descent), linear units have been used. In the case of purely linear output units the activation of an output unit, u_j, is simply given by

$$S_i = \sum_j w_{i,j} u_j \quad .$$

The error function, as indicated by the name least-mean-sqare, is the summed sqared error. That is, the total error, ε, is defined to be (op.cit., p.127)

$$\varepsilon = \sum_k \varepsilon_k = \sum_k \sum_i (S_{k,i} - C_{k,i})^2$$

where the index k ranges over the set of training examples, i ranges over the set of output units, and ε_k represents the error on pattern k. The variable $C_{k,i}$ is the desired output, or target, for the i^{th} output unit when the k^{th} pattern has been presented, and $S_{k,i}$ is the actual output of the i^{th} output unit when pattern k has been presented. The object is to find a set of weigths W (or W^*) that minimizes this function.

If there are N training examples $\{E^k\}$ with corresponding outputs $\{C^k\}$ then the mean sqared error (MSE) is given by (see GALLANT, 1992, p.128)

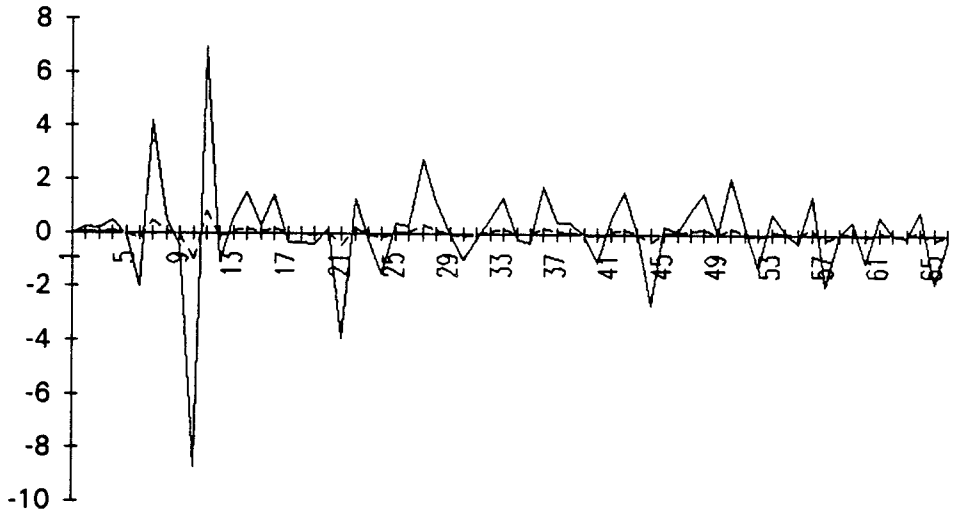
$$MSE = \frac{1}{N}\sum_{k=1}^{N}(W \cdot E^k - C^k)^2 \ .$$

There is a fast, convenient gradient descent algorithm for minimizing MSE, called either the WIDROW-HOFF Rule or LMS algorithm. In its final form it dictates to take a step in the direction $-\nabla MSE$, yielding an updating procedure of (op.cit., p.129):

$$W^* = W + \rho(C - S)E \ .$$

This equation tells us to update by taking the signed difference between the correct output and the weigthed sum, multiplying by the small step size ρ , and then taking a step in the direction of the cell's inputs (or the opposite direction if $(C\text{-}S)<0$).

Figure 4-1: Transformed and scaled transformed series kk1a



In order to apply this technique to time series data a preliminary transformation of the data has been undertaken. Instead of considering original values with sometimes big fluctuations the net is trained by scaled first differences

$d_t=sf(Z_{t+1}-Z_t)$. The scaling factor *sf* transformes this detrended data to an intervall of [-1,+1]. This allows the net to train on similar values and therefore getting better results. Figure 4-1 shows both differences (solid line) and scaled differences (dotted line) for input series kk1a.

Figure 4-2: A single layer feed forward approach
linear units

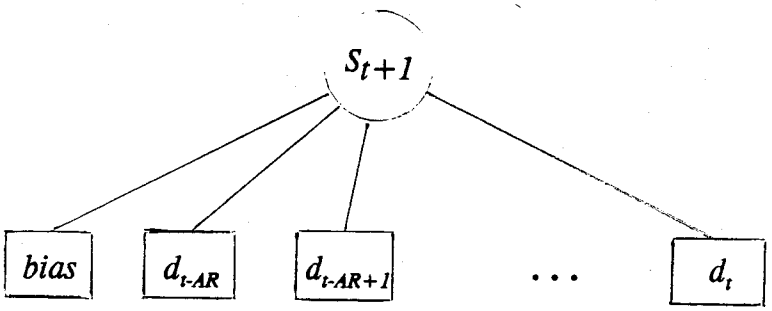
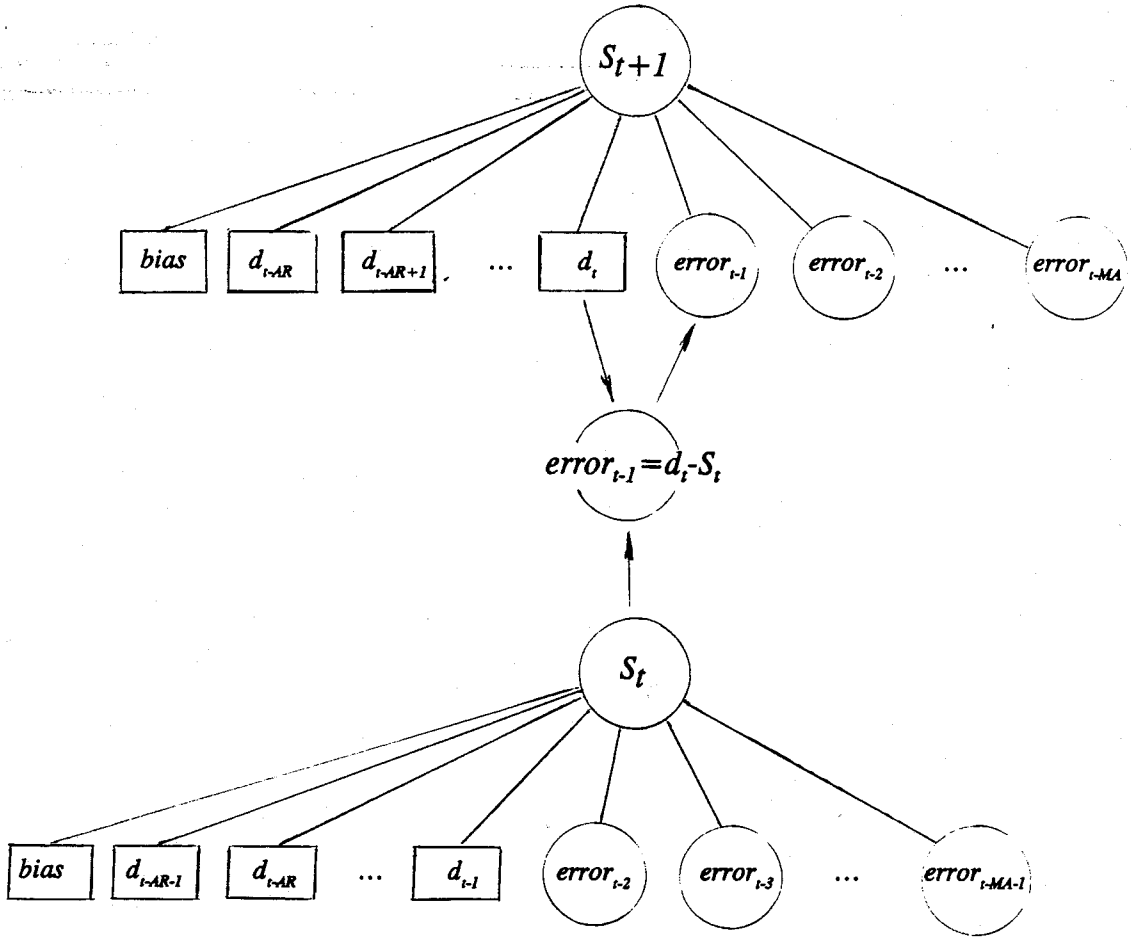


Figure 4-3: A two layer recurrent approach
linear units



The submitted program allows for both a strictly layered feed forward approach as indicated in figure 4-2 as well as for a modified form of a JORDAN-Network (JORDAN, 1986). In this case the MSE algorithm modifies to an application of BPTT learning (*backpropagation through time*; see figure 4-3).

To run a data set as feed forward net simply specify only autoregressive parameters AR. The program therefore will modify input data to generate a training set as pictured in figure 4-2; in the case of working with error terms the net topology enhances to a recurrent form as indicated in figure 4-3.

4.3. NTSP - Neural Time Series Prognosis

4.3.1. Program Description

NTSP allows to analyze in the compiled version time series with rather smaller data sets up to 70 observations. It is oriented versus econometric methods, i.e., BOX-JENKINS models. The number of parameters in the compiled form must not exceed *5*; $AR + MA \leq 5$.

NTSP may use standard WIDROW-HOFF or a modified training method, called *Pattern Sum Errors* (PSE) algorithm. The PSE-algorithm was developped to overcome problems due to rather small sample sizes. It minimizes in every executional run (this means the number of times the training set is shown to the net) the sum of errors of every pattern ε_k. In the next run this weights are updated and so on. Naturally this leds to k different sets of weights. An econometric counterpart to this method is called KALMAN filtering (SCHNEEWEISZ, 1977, p.17). Every new observation leds to a justification of the prediction equation and allows to fit data through time. A disantvantage of this method is its lack of generality, therefore a MSE algorithm minimizing ε will be used to compare results.

Before starting NTSP a setup routine (**[d]:\setup**) has to be runned which takes care of data handling. NTSP will read/write data only from/to the rendered floppy disc. All analyzed series are saved under **[d]:\series\kk1a**, **[d]:\series\kk1b**, ... , **[d]:\series\kk3c**. Simply copy a series **to [d]:\obs.ntp**, run NTPS and save the results to any file.

NTSP is controlled by a file called **[d]:\ini.ntp**. The following specifications have to be made:

```
TIME HORIZON (TT)
NUMBER OF AUTOREGRESSIVE PARAMETERS (AR)
NUMBER OF MOVING AVERAGE PARAMETERS (MA)
ITERATIONS (II)
NUMBER OF EXECUTIONAL RUNS (ER)
PRINTSTEP (PRINTSTEP)
STARTING WEIGHTS (ww)
LEARNING RATE RHO (rho)
ERROR CRITERIA (ecrit)
TRAINING METHOD (etarg)
DISPLAY MODE (dm)
```

All characters in brackets give the name of the parameter as used in the program (see source code in **Appendix II** and on the disc in **[d]:\series\ntsp**). The following options are possible:

TIME HORIZON: Any integer value from 0 up to 70 is possible

NUMBER OF AUTOREGRESSIVE/MOVING AVERAGE PARAMETERS:
Any integer value from 0 up to 5 possible but remember $AR + MA \leq 5$.

ITERATIONS: Any integer value; this option becomes relvant only in the case of using the PSE routine (see below). Any training example trys to adjust its weights according to the remaining error of the previous executional run.
After reaching a certain error criteria $(ecrit \geq \varepsilon_k^2 > ecrit \cdot 0.1)$ the learning rate is adopted to $\rho' = \rho \cdot 0.1$. The same adjustment $(\rho'' = \rho' \cdot 0.1)$ is made if ε_k is going to be in the interval of $\varepsilon_k^2 \leq ecrit \cdot 0.1$; this aims to prevent the algorithm of overadjusting once a minimum has already nearly been encountered. If the display mode has been set to print on the screen every time this adjustment is undertaken "|" is displayed.

In the case of reaching the indicated error criteria $\varepsilon_k^2 \leq ecrit \cdot 0.05$ the MSE algorithm breaks. In this case or if the number of iterations elapsed the remaining error is stored and shown in the next executional run to the net (recurrent if *MA>0*) or simply taken to further adjust individual weights.

In the case of running the MSE routine ITERATIONS is set automatically to be *1*!

NUMBER OF EXECUTIONAL RUNS: see above; any integer value.

PRINTSTEP: Any integer value smaller as your indicated number of iterations. Relevant only in combination DISPLAY MODE beeing *2*.

PRINTSTEP forces the program to print only every indicated time its gradient descent computation information to **[d]:\cmp.ntp** as well as to the screen.
Be careful to set this parameter reasonable high to prevent the file from beeing not manageable.

STARTING WEIGHTS: Select a floating-point number to initialise computations; the PSE-algorithm will assign in its first example the adjusted weights of its predecessors. In all subsequent executions every example adjusts its own weights without influencing other example weights.

LEARNING RATE: Select a floating-point number to initialise the learning rate ρ. In the PSE routine ρ will be updated as indicated above whereas in the MSE routine no adoption will arise at all.

ERROR CRITERIA: The error criteria may speed up the PSE routine as it serves as breaking and adoption value as stayed above. In the MSE routine it will not be used; select a floating-point number.

TRAINING METHOD: Options:
[1] ... MSE training algorithm
[2] ... PSE training algorithm

DISPLAY MODE: Options:
[0] ... No screen printout; file **[d]:\trd.ntp** for information on trained series and on training examples; file **[d]:\wts.ntp** for final weights of the network.
[1] ... Screen printout is turned on; see figures 4-5 and figure 4-6; files as under [0].
[2] ... Full printout, see [1]; additional a file containing information on gradient descent computations (**[d]:\cmp.ntp**) will be written. Be carefull about the PRINTSTEP parameter.

Figure 4-4: An example input file [d]:\ini.ntp

```
66              66 observations will be read from [d]:\obs.ntp
3               three autoregressive input cells
2               two moving average (recurrent) input cells
100             100 iterations on each example at most in the PSE routine
15              15 executional runs of the trainings set
5               every fifth computation is printed to [d]\cmp.ntp
0.11            starting weights
0.77            learning rate
0.01            error criteria
2               execute PSE algorithm
0               do not print any output on to the screen
```

All output files are printed and commented in **APPENDIX III.** For this reason a smaller example has been worked through by the NTPS PSE-routine; subsequently figures 4-5 and figure 4-6 give comments on the screen output both for i) the PSE- and ii) the MSE-routine.

Figure 4-5: Screen output for PSE algorithm

```
            :           :
            :           :
EXECUTION 1

...-0.07
-0.03                                   <in the first run only
||.......-0.07                         remaining errors of each
|.................-0.11                input pattern are
.............|.....0.24                 printed>
|............|.....-0.07
            :           :
            :           :
0.04
0.06
0.01
0.07                                    <rho has been adopted
....|.........|......-0.08              twice>
-0.05

EXECUTION 2

-0.07:-18.810305
-0.03:-18.810305
-0.07:-18.810305
|..|.......-0.07:-18.810305
|................|..0.09:-18.810305
            :           :
            :           :
0.04:-18.810305
0.06:-18.810305         <starting with the second replication
0.01:-18.810305         both remaining errors of each training
0.07:-18.810305         example as well as the total sum of
.-0.07:-18.810305       errors of the previous replication is
-0.04:-18.810305        printed>

final SSE = 20.587
                        <remaining Sum of Sqared Errors>
```

Figure 4-6: Screen output for MSE algorithm

```
     :          :
     :          :
  6 PSE: -4.228113
  7 PSE: -3.790818
  8 PSE: -3.400717
  9 PSE: -3.052326
 10 PSE: -2.740848
 11 PSE: -2.462111
 12 PSE: -2.212473
 13 PSE: -1.988736
     :          :
     :          :
111 PSE: -0.000071
112 PSE: -0.000064
113 PSE: -0.000058
114 PSE: -0.000052
115 PSE: -0.000047
116 PSE: -0.000042
117 PSE: -0.000038      <only the total sum of errors
118 PSE: -0.000034      on all input patterns of the
119 PSE: -0.000031      previous replications is
120 PSE: -0.000028      printed>

final SSE = 278.606
                        <remaining Sum of Sqared Errors>
```

As several test runs proofed the PSE algorithm converges with proper parameters to a perfect representation of any series with remaining SSE zero or nearly zero (try, e.g., ini.ntp=(66,5,0,200,15,1,0.11,0.2,0.001,2,1)!
(REMARK: If the gradient descent algorithm in the PSE routine does not converge it is generally advisable to try first a smaller learning rate , e.g., 0.1.)

4.3.2. Computational results

To compare the network approach results with results of the other presented methods the MSE routine with two equal parameter settings (ini.ntp=(66,0,5,1,120,1,0.11,0.2,0.01,1,1), i.e. a fully recurrent MA=5 topology and ini.ntp=(66,5,0,1,120,1,0.11,0.2,0.01,1,1) i.e. a strictly feed forward AR=5 topology) for each series will be investigated.

Additionally NTSP will be used to compute a mixed model (ini.ntp=(66,3,2,1,120,1,0.11,0.2,0.01,1,1)) for series kk1 to demonstrate its effects of shaping the smoothed approximation (see figures 4-7, 4-8 and 4-9). A survey on computational results is given in table 4-1; it should be easy possible to further improve these results by changements in the parameter settings.

Table 4-1: Estimation and evaluation results
NTSP Modeling

series	recurrent SSE	forward SSE
kk1a	MA(0,1,5) 240.1	AR(5,1,0) 242.4
kk1b	MA(0,1,5) 506.96	AR(5,1,0) 516.5
kk1c	MA(0,1,5) 911.4	AR(5,1,0) 925.3
kk3a	MA(0,1,5) 174.91	AR(5,1,0) 175.2
kk3b	MA(0,1,5) 235.4	AR(5,1,0) 244.4
kk3c	MA(0,1,5) 232.4	AR(5,1,0) 246.4

Figure 4-7: Recurrent approximation of kk1a; SSE = 240
ini.ntp=(66,0,5,1,120,1,0.11,0.2,0.01,1,1)

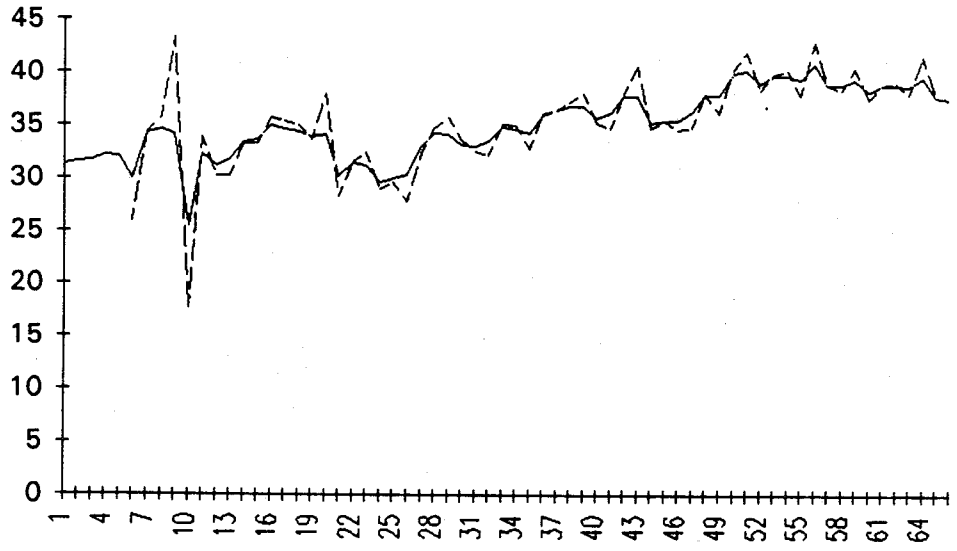


Figure 4-8: Feed forward approximation of kk1a; SSE = 242
ini.ntp=(66,5,0,1,120,1,0.11,0.2,0.01,1,1)

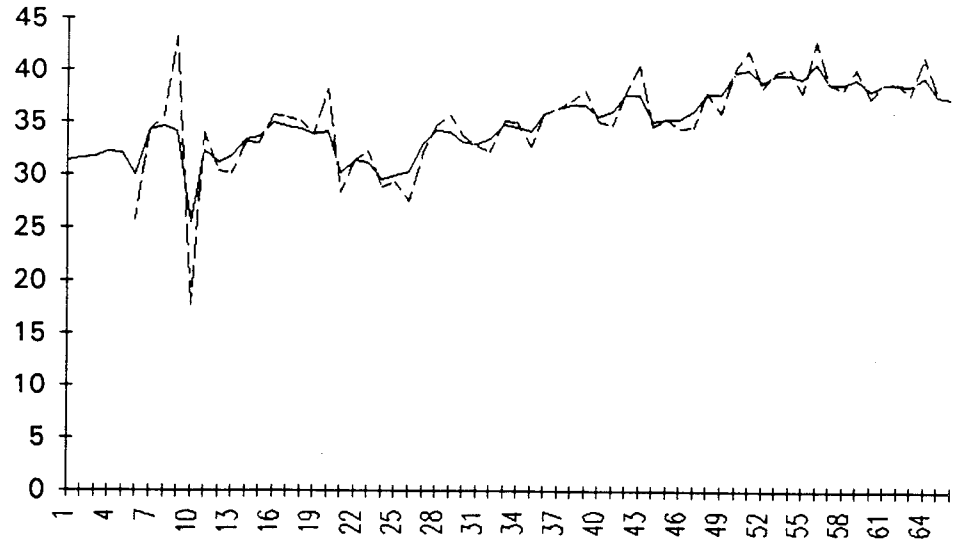
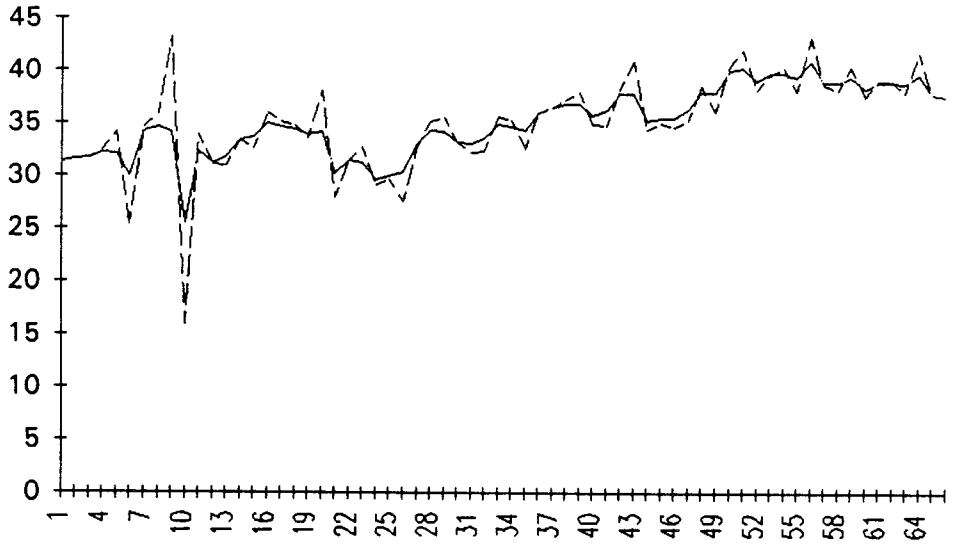


Figure 4-8: Generalized approximation of kk1a; SSE = 282
ini.ntp=(66,3,2,1,120,1,0.11,0.2,0.01,1,1)



5. Conclusion

This paper tried to establish a simple network for time series aqnalysis tasks. Results in chapter 4 show that in most cases MSE approximations are slightly worse than ARIMA results except in two cases where the neural net captures the series dramatically better than other approaches. The proposed new approach (PSE algorithm) represents with proper parameters every series perfectly.

As figure 4-6 -figure 4-8 indicate the shape of the smoothed series fluctuates more than Exponential-Smoothing- or ARIMA-Models.

Improvements to this pitfalls may be effectuated by means of an augmentation of the number of input cells more than to any changement of the proportion of AR/MA terms. Another way of improving this approach would be to apply adopted learning algorithms and/or adding hidden units (see PLUTOWSKI et. al., 1992).

REFERENCES

BOX, G.E.P. and JENKINS, G.M.: "Time Series Analysis: Forecasting and Control", 1976

DOAN, T.A.: "User's Manual to RATS, Version 3.02", 1989 and update RATS386 Version 3.11, 1991

HANSSEN, D.M., PARSON, L.J. and SCHULTZ, R.L.: "Market Response Models. Econometric and Time Series Analysis", 1990

HARVEY, A.C.: "A Unified View of Statistical Forecasting Procedures", Journal of Forecasting, 1984, Vol.3, pp. 245-275

HERTZ, J., KROGH, A. and PALMER, R.G.: "Introduction to the Theory of Neural Computation", 1991

JORDAN, Michael I.: "Serial Order: A Parallel Distributed Processing Approach", 1986

KERNIGHAN, B.W. and RICHIE, D.M.: "The C Programming Language", 1988

KOLMOGOROV, A.N.: "Interpolation and extrapolation of stationary random sequences", 1941

McCLELLAND, J.L. and RUMELHART, D.E.: "Explorations in parallel distributed processing", 1988

MEHRA, R.K.: "Kalman Filters and their Applications to Forecasting", TIMS Studies in the Management Sciences 12, 1979, pp. 75-94

NWorks, "Using NWorks; An extended Tutorial for NeuralWorks Professional II/PLUS and NeuralWorks Explorer", 1991

PLUTOWSKI, M., COTRELL, G. and WHITE, H.: "Learning Mackey-Glass from 25 examples, plus or minus 2.", 1992

SCHNEEWEISZ, CH.: "Inventory-Production Theory", 1977

SCHNEEWEISZ, Ch.: "Modellierung industrieller Lagerhaltungssysteme", 1981

WIDROW, G. and HOFF, M.E.: "Adaptive switching circuits", Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4, pp.96-104, 1960

WIENER, N.: "The Extrapolation, Interpolation and Smoothing of Stationary Time Series with Engineering Applications", 1949

APPENDIX I: Time series data

kk1a	kk1b	kk1c	kk3a	kk3b	kk3c
31.35	25.34	25.15	23,93	24,22	24,22
31.59	25.46	25.56	23,27	24,22	24,22
31.75	34.13	32.95	21,33	24,22	24,22
32.21	31.18	32.46	21,35	24,22	0
32.08	30.73	32.74	21,43	7,35	6,9
30.07	32.25	33.94	21,51	22,47	22,47
34.26	33.67	33.05	21,83	27,11	27,25
34.71	24.28	23.93	23,39	27,25	27,25
34.2	31.3	33.11	23,29	27,25	27,25
25.47	33.23	32.57	23,61	27,25	27,25
32.41	35.6	35.97	23,88	27,25	27,25
31.31	32.4	33.6	29,68	27,25	27,25
31.89	32.6	35.02	29,99	27,25	27,3
33.41	28.19	28.44	29,74	27,25	27,25
33.66	28.06	27.5	29,9	27,25	27,25
35.08	32.16	31.42	25,28	27,25	27,25
34.73	34.35	34.88	29,87	27,25	27,25
34.39	32.31	32.5	29,81	27,25	27,25
33.96	27.88	27.86	29,72	27,25	27,25
34.13	31.55	33.01	29,55	27,25	27,25
30.22	32.07	32.55	29,71	27,25	27,25
31.49	32.25	31.69	29,71	27,25	27,25
31.22	31.2	30.89	29,6	27,25	27,25
29.72	32.02	31.62	27,6	27,36	27,42
30.07	33.39	34.58	26,65	32,3	32,47
30.34	32.94	34.79	26,68	32,47	32,47
33.1	33.01	32.52	26,65	32,47	32,47
34.29	31.63	32.47	30,03	27,47	27,47
34.22	30.24	30.64	25,12	32,47	32,47
33.25	32.71	33.5	30,34	32,47	32,47
33.03	33.3	33.53	30,24	32,47	32,47
33.59	31.54	33.23	30,09	32,47	32,47
34.89	33.19	34.27	30,29	29,26	32,47
34.68	31.77	33.91	30,13	32,27	32,47
34.34	32.68	34.68	30,34	27,47	27,47
36.07	31.3	30.73	33,89	32,47	32,47
36.5	31.35	31.82	34,74	32,47	32,47
36.89	33.17	33.89	34,77	32,47	32,47
36.82	33.89	34.09	34,71	33,25	33,99
35.72	32.42	24.86	34,66	33,02	35,48
36.29	33.93	32.7	34,7	32,76	33,4
37.83	36.68	32.58	34,71	34,42	33,39
37.9	34.82	34.53	35,03	32,94	33,61
35.26	30.43	30.91	34,7	33,23	34,26
35.51	31.61	32.78	34,81	33,33	33,39
35.59	35.47	37.49	34,33	32,47	34,44
36.43	36.56	37.96	32,92	33,76	33,3
37.95	37.27	38.84	32,88	33,4	34,44
37.94	38.12	38.22	32,92	33,05	33,61
40.05	37.73	37.83	33,11	33,02	33,51
40.3	37.26	37.63	32,89	33,54	34,03
39.04	37.54	38.94	32,66	34,1	33,76
39.79	37.85	38.82	32,64	29,34	29,59
39.85	37.24	38.61	32,73	33,76	33,54
39.49	38.4	41.33	32,75	32,83	33,76
40.92	32.57	32.4	33,51	32,88	34,07
38.96	36.59	38.9	32,8	33,56	32,94
38.93	36.77	36.7	32,68	32,99	33,04
39.39	36.16	38.85	32,75	33,54	34,97
38.33	38.14	38.25	32,54	33,21	33,43
38.97	37.08	39.55	32,63	32,74	32,85
38.94	30.15	29.97	32,74	33,68	32,78
38.81	31.09	30.14	32,63	34,19	34,67
39.67	34.79	37.74	32,78	32,89	33,3
37.82	38.97	38.28	32,8	33,36	33,3
37.55	35.79	36.25	32,79	32,66	34,16

APPENDIX II: Source Code NTSP

Neural Time Series Prognosis

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*     NEURAL TIME SERIES PROGNOSIS - NTSP                                 */
/*                                                                         */
/*                                                                         */
/*     N     ...   number of training examples                             */
/*     Ek    ...   set of training examples                                */
/*     PSE   ...   Pattern Sum of Squared Errors                           */
/*     SSE   ...   Sum of Squared Errors                                   */
/*     MSE   ...   Mean Squared Error                                      */
/*     errork      residual error on example k                             */
/*     S     ...   weighted sum for output cell up+1                       */
/*     I     ...   number of iterations per training example               */
/*     rho   ...   iteration stepsize                                      */
/*     ER    ...   number of training (executional) runs                   */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main()
{
int dd;
int dm;
int er;                                  /*-----------------------------*/
int etarg;                               /* etarg==1 ... MSE descent ---*/
int hh;                                  /* etarg!=1 ... PSE descent ---*/
int ii;                                  /*-----------------------------*/
int kk;
int ll;
int mm;
int oo;
int pp;
int qq;
int AR;
int ER;
int II;
int MA;
int NN;
int PRINTSTEP;
int TT;
int HighestI[70];
int LowestI[70];

float ecrit;
float hf;
float MSE;
float PSE;
float helpPSE;
float SSE;
float sse;
float r;
float rho;
float scale;
float ww;
float error[70];
float E[70][8];
float S[70];
float w[8];
float W[70][8];
float z[70];
float Z[70];
```

```
float TRANSF();
int MAXIMUM();
int VEKTORLARGEST();

FILE *ini;
FILE *ser;
FILE *cmp;
FILE *wts;
FILE *trd;

system("cls");
printf("\n  N T S P  Neural Time Series Prognosis");
printf("\n Institute for Advanced Studies - Vienna");
printf("\n            SemPap/BAOR KRYCHA Karl, 1992\n");
printf("           get file [a:\\ini.ntp]\n");
ini=fopen("a:\\ini.ntp","r");
 fscanf(ini,"%d",&TT);
 fscanf(ini,"%d",&AR);
 fscanf(ini,"%d",&MA);
 fscanf(ini,"%d",&II);
 fscanf(ini,"%d",&ER);
 fscanf(ini,"%d",&PRINTSTEP);
 fscanf(ini,"%f",&ww);
 fscanf(ini,"%f",&rho);
 fscanf(ini,"%f",&ecrit);
 fscanf(ini,"%d",&etarg);
 fscanf(ini,"%d",&dm);
fclose(ini);

if (etarg==1) II=1;

printf("           get file [a:\\obs.ntp]");
ser=fopen("a:\\obs.ntp","r");
 for (dd=1;dd<=TT;dd++)
    {
    fscanf(ser,"%f",&hf);
    Z[dd]=hf;
    }
fclose(ser);

for (dd=0;dd<=70;dd++) { z[dd]=0; }
for (dd=1;dd<TT;dd++)
    {
    z[dd]=Z[dd+1]-Z[dd];
    z[dd]=z[dd]*z[dd];
    }
kk=VECTORLARGEST(z,(TT-1));
scale=sqrt(z[kk]);
for (dd=1;dd<TT;dd++)
    {
    z[dd]=((Z[dd+1]-Z[dd])/scale);
    }
pp=AR+MA;
for (ii=0;ii<=pp;ii++)
    {
    w[ii]=ww;                                  /*-----------------------------*/
    }                                          /* Gradient descent ---------- */
                                               /* Least Mean Square --------- */
r=rho;                                         /* algorithm ----------------- */
if (dm>=1)                                     /*-----------------------------*/
{
printf("\n\n*** gradient descent *********");
printf("\n*** WIDROW HOFF  LMS-algorithm\n");
printf("\ninitial stepsize is %.3f",rho);
printf("\nerror criteria is %.3f",ecrit);
printf("\nnumber of original observations is %d",TT);
printf("\nnumber of autoregressive parameters is %d",AR);
printf("\nnumber of moving average parameters is %d",MA);
            {
if (etarg==1) printf("\ntraining procedure uses MSE algorithm;\n    rho
will not be adopted.");
```

```
else          printf("\ntraining procedure uses PSE algorithm");
              }
printf("\nnumber of executional runs is %d",ER);
printf("\nnumber of iterations is %d",II);
printf("\nPRINTSTEP of gradient descent is %d",PRINTSTEP);
printf("\n\ntype any key to continue ... "); getch(); system("cls");
}
if(dm>=2)
  {
cmp=fopen("a:\\cmp.ntp","w");
fprintf(cmp,"\n*** gradient descent *********");
fprintf(cmp,"\n*** WIDROW HOFF  LMS-algorithm\n");
  }                                          /*------------------------------*/
SSE=0;                                       /* initialisation ------------ */
MSE=0;                                       /*------------------------------*/
PSE=0;
for (kk=1;kk<=TT;kk++) { LowestI[kk]=2000; HighestI[kk]=0; }
for (ll=0;ll<=TT;ll++) { S[ll]=0; error[ll]=0; }
oo=0;
qq=MAXIMUM(AR,MA);                           /*------------------------------*/
for (er=1;er<=ER;er++)                       /* executional runs ---------- */
{                                            /*------------------------------*/
 if (dm>=2) fprintf(cmp,"\nTHIS IS THE %d. EXECUTION OF THE TRAININGS
EXAMPLE SET",er);
 if (dm>=1 && etarg!=1) printf("\nEXECUTION %d\n\n",er);
 helpPSE=(PSE/(TT-2-qq));
 PSE=0;
                                             /*------------------------------*/
 for (kk=qq;kk<TT-1;kk++)                    /* training examples --------- */
 {                                           /*------------------------------*/
 if (etarg!=1)
    {
    if (er>1) { for (ll=0;ll<=pp;ll++) w[ll]=W[kk][ll]; }
    }
 if (etarg==1)
    {
    if (er>1) { for (ll=0;ll<=pp;ll++) w[ll]=W[TT-2][ll]; }
    }
                                             /*------------------------------*/
E[kk][0]=1;                                  /* The following section ----- */
for (ll=1;ll<=AR;ll++)                       /* computes learning examples  */
   {                                         /*------------------------------*/
   E[kk][ll]=z[kk-AR+ll];
   }
for (ll=1;ll<=MA;ll++)
   {
   E[kk][ll+AR]=error[kk-MA+ll-1];
   }
E[kk][AR+MA+1]=z[kk+1];

mm=0;
if(dm>=2)
{
fprintf(cmp,"\n\n  Actual starting values for weigths are\n");
for (ii=0;ii<=pp;ii++) fprintf(cmp,"  w%d   ",ii);
fprintf(cmp,"\n");
for (ii=0;ii<=pp;ii++)
              {
              if (w[ii]<0)
                 {
                 if (w[ii]<=-10) fprintf(cmp,"%.2f ",w[ii]);
                 else            fprintf(cmp," %.2f ",w[ii]);
                 }
              else
                 {
                 if (w[ii]>=10)  fprintf(cmp," %.2f ",w[ii]);
                 else            fprintf(cmp,"  %.2f ",w[ii]);
                 }
               }
fprintf(cmp,"\n  Example %d is going to be executed
                 the %d. time",(kk-qq+1),er);
```

```
    {
    if (E[kk][pp+1]>=0)
                {
                if (kk>=0 && kk<10)
                fprintf(cmp,"\n  Target output value
                        C%d = %.3f ********************",kk,E[kk][pp+1]);
                else
                   {
                if (kk>=10 && kk<100)
                fprintf(cmp,"\n  Target output value
                        C%d = %.3f ********************",kk,E[kk][pp+1]);
                else
                fprintf(cmp,"\n  Target output value
                        C%d = %.3f *****************",kk,E[kk][pp+1]);
                   }
                }
    else
                {
                if (kk>=0 && kk<10)
                fprintf(cmp,"\n  Target output value C%d = %.3f
                             ******************",kk,E[kk][pp+1]);
                else
                   {
                   if (kk>=10 && kk<100)
                   fprintf(cmp,"\n  Target output value C%d = %.3f
                                ******************",kk,E[kk][pp+1]);
                   else
                   fprintf(cmp,"\n  Target output value C%d = %.3f
                                ****************",kk,E[kk][pp+1]);
                   }
                }
    }
fprintf(cmp,"\nitn    ");
for (ii=0;ii<=pp;ii++) fprintf(cmp,"  w%d    ",ii);
fprintf(cmp,"\n");
}                                       /*-----------------------------*/
for (hh=1;hh<=II;hh++)                  /* iterations ---------------- */
  {                                     /*-----------------------------*/
  rho=r;
  oo=oo+1;
      if (dm>=2 && oo==PRINTSTEP)
      {
      if (hh>=0 && hh<10)   fprintf(cmp,"  %d ",hh);
      if (hh>=10 && hh<100) fprintf(cmp," %d ",hh);
      if (hh>=100)          fprintf(cmp,"%d ",hh);
      }
  if (etarg!= 1 && dm>=1 && oo==PRINTSTEP) { printf("."); }

  S[kk]=0;
  for (ii=0;ii<=pp;ii++)
      {
      S[kk]=S[kk]+(w[ii]*E[kk][ii]);
      }
  error[kk]=(E[kk][pp+1]-S[kk]);
  PSE=PSE+error[kk];
                                        /*-----------------------------*/
  if (etarg!=1)                         /* weigths ------------------- */
     {                                  /* gradient descent ---------- */
     for (ii=0;ii<=pp;ii++)             /* for PSE algorithm --------- */
         {                              /*-----------------------------*/
         w[ii]=w[ii]+(rho*error[kk]*E[kk][ii]);
         }
     }

  if (etarg==1)                         /*-----------------------------*/
     {                                  /* weigths ------------------- */
     for (ii=0;ii<=pp;ii++)             /* gradient descent ---------- */
         {                              /* for MSE algorithm --------- */
     w[ii]=w[ii]+(rho*helpPSE*E[kk][ii]);/*-----------------------------*/
         }
     }
```

```
   if (dm>=2 && oo==PRINTSTEP)
       {
       for (ii=0;ii<=pp;ii++)
               {
               if (w[ii]<0)
                  {
                  if (w[ii]<=-10) fprintf(cmp,"%.2f ",w[ii]);
                  else            fprintf(cmp," %.2f ",w[ii]);
                  }
               else
                  {
                  if (w[ii]>=10)  fprintf(cmp," %.2f ",w[ii]);
                  else            fprintf(cmp,"  %.2f ",w[ii]);
                  }
               }
          {
          if (S[kk]<0)  fprintf(cmp,"  S%d=%.2f;\n",kk,S[kk]);
          else          fprintf(cmp,"   S%d=%.2f;\n",kk,S[kk]);
          }                                  /*------------------------------*/
       }                                     /* convergence error --------- */
                                             /* criterias (ecrit) --------- */
       {                                     /*------------------------------*/
       if ((error[kk]*error[kk])<=ecrit &&
(error[kk]*error[kk])>(0.1*ecrit) && mm==0)
          {
          rho=rho*0.1;
            {
  if (dm>=1 && etarg!=1) { printf("|"); }
  if (dm>=2)
  {
  if (kk>=0 && kk<10)
   fprintf(cmp," * rho adopted after %d iterations;
           rho=%.3f *******\n",hh,rho);
   else
     {
     if (kk>=10 && kk<100)
     fprintf(cmp," * rho adopted after %d iterations;
             rho=%.3f ******\n",hh,rho);
     else
     fprintf(cmp," * rho adopted after %d iterations;
             rho=%.3f *****\n",hh,rho);
     }
  }
            }
          mm=1;
          }
        if ((error[kk]*error[kk])<=(ecrit*0.1) && mm==1)
          {
          rho=rho*0.1;
            {
  if (dm>=1 && etarg!=1) { printf("|"); }
  if (dm>=2)
  {
  if (kk>=0 && kk<10)
   fprintf(cmp," * rho readopted after %d iterations;
           rho=%.3f ****\n",hh,rho);
   else
     {
     if (kk>=10 && kk<100)
     fprintf(cmp," * rho readopted after %d iterations;
             rho=%.3f ***\n",hh,rho);
     else
     fprintf(cmp," * rho readopted after %d iterations;
             rho=%.3f **\n",hh,rho);
     }
  }
            }
          mm=2;
          }
        if ((error[kk]*error[kk])<=(ecrit*0.05))
          {
```

```
            mm=3;
             {
if(dm>=2)
 {
 if (kk>=0 && kk<10)
 fprintf(cmp," * converged after %d iterations
        ********************\n\n",hh);
 else
   {
   if (kk>=10 && kk<100)
   fprintf(cmp," * converged after %d iterations
          ******************\n\n",hh);
   else
   fprintf(cmp," * converged after %d iterations
          ******************\n\n",hh);
   }
 }
            }
           oo=0;
           goto L1;                          /*-----------------------------*/
           }                                 /* end of convergence criteria */
       }                                     /*-----------------------------*/
    if (oo==PRINTSTEP) oo=0;
    }                                        /*-----------------------------*/
    if(dm>=2) fprintf(cmp,"\n");             /* end of iterations loop ---- */
                                             /*-----------------------------*/
    L1:
    if (mm!=3)
    {
    if(dm>=2)
        {
        if (kk>=0 && kk<10)
         {
         fprintf(cmp," WARNING: Convergence not reached
                ****************\n",II);
         fprintf(cmp,"          after %d iterations
                ********************\n",II);
         }
        else
         {
         if (kk>=10 && kk<100)
             {
             fprintf(cmp," WARNING: Convergence not reached
                    ****************\n",II);
             fprintf(cmp,"          after %d iterations
                    ********************\n",II);
             }
             else
              {
              fprintf(cmp," WARNING: Convergence not reached
                     ***************\n",II);
              fprintf(cmp,"          after %d iterations
                     ********************\n",II);
              }
       }
      }
    }
    if (dm>=1 && etarg!=1)
       {
       if (er==1) printf("%.2f\n",error[kk]);
       if (er>1)  printf("%.2f:%f\n",error[kk],(helpPSE*(TT-2-qq)));
       }
    if (dm>=1 && etarg==1 && kk==(TT-2))
       {
       if (er>=0 && er<10)
          printf("  %d PSE: %f\n",er,(helpPSE*(TT-2-qq)));
       if (er>=10 && er<100)
          printf(" %d PSE: %f\n",er,(helpPSE*(TT-2-qq)));
       if (er>=100 && er<1000)
           printf("%d PSE: %f\n",er,(helpPSE*(TT-2-qq)));
       }
```

```
      oo=0;
      if (hh<LowestI[kk])  LowestI[kk]=hh;
      if (hh>HighestI[kk]) HighestI[kk]=hh;
      for (ll=0;ll<=pp;ll++) { W[kk][ll]=w[ll]; }
      if(dm>=2)
         {
         fprintf(cmp,"final output value S%d,%d = %.3f\n",kk,hh,S[kk]);
         fprintf(cmp,"target output value C%d = %.3f\n",kk,E[kk][pp+1]);
         fprintf(cmp,"final error = %.3f\n",error[kk]);
         fprintf(cmp," ----------------------------------------------\n\n");
         }
 }
 hf=PSE;
                                                /*-----------------------------*/
}                                               /* end of training examples -- */
                                                /* end of executional runs --- */
                                                /*-----------------------------*/
PSE=hf;
if (dm>=1)
{
sse=0;
for (kk=qq;kk<TT-1;kk++)
     { sse=sse+((error[kk]*scale)*(error[kk]*scale)); }
printf("\nfinal SSE = %.3f\n",sse);
}
if (dm>=2)
   {
   fprintf(cmp,"final SSE = %.4f\n",sse);
   fclose(cmp);
   }

printf("\nwriting to [a:\\wts.ntp]");
if (etarg!=1)
{
wts=fopen("a:\\wts.ntp","w");
for (kk=qq;kk<TT-1;kk++)
          {
    for (ii=0;ii<=pp;ii++)
             {
             if (W[kk][ii]<0)
                {
                if (W[kk][ii]<=-10) fprintf(wts,"%.2f ",W[kk][ii]);
                else                fprintf(wts," %.2f ",W[kk][ii]);
                }
             else
                {
                if (W[kk][ii]>=10)  fprintf(wts," %.2f ",W[kk][ii]);
                else                fprintf(wts,"  %.2f ",W[kk][ii]);
                }
              }
           fprintf(wts," %.d|%.d\n",LowestI[kk],HighestI[kk]);
           }
fprintf(wts,"  ----------------------------------------\n\n");
fprintf(wts,"  For each training example best last weights\n");
fprintf(wts,"  and informations on gradient descent\n");
fprintf(wts,"  (lowest|highest number of iterations) are\n");
fprintf(wts,"  listed above.\n\n");
fclose(wts);
}
if (etarg==1)
{
wts=fopen("a:\\wts.ntp","w");
    for (ii=0;ii<=pp;ii++)
             {
             if (W[TT-2][ii]<0)
                {
                if (W[TT-2][ii]<=-10) fprintf(wts,"%.2f ",W[TT-2][ii]);
                else                  fprintf(wts," %.2f ",W[TT-2][ii]);
                }
             else
                {
```

```
                if (W[TT-2][ii]>=10)  fprintf(wts," %.2f ",W[TT-2][ii]);
                else                  fprintf(wts,"  %.2f ",W[TT-2][ii]);
                }
              }
fprintf(wts,"\n  ---------------------------------------\n");
fprintf(wts,"  MSE - trained weights are listed above.\n");
fclose(wts);
}

sse=0;
for (kk=qq;kk<TT-1;kk++) { sse=sse+((error[kk]*scale)*(error[kk]*scale)); }
printf("\nwriting to [a:\\trd.ntp]\n");
trd=fopen("a:\\trd.ntp","w");
fprintf(trd,"final SSE = %.3f  \n\n",sse);
 {
 if (Z[1]<0)                fprintf(trd,"%.2f  \n",Z[1]);
 else                       fprintf(trd," %.2f  \n",Z[1]);
 }

 if (qq>2)
 for (dd=2;dd<=qq;dd++)
 {
    {
    if (Z[dd]<0)            fprintf(trd,"%.2f  ",Z[dd]);
    else                    fprintf(trd," %.2f  ",Z[dd]);
    }
    {
    if ((z[dd-1]*scale)<0) fprintf(trd,"%.2f  ",(z[dd-1]*scale));
    else                    fprintf(trd," %.2f  ",(z[dd-1]*scale));
    }
    {
    if (z[dd-1]<0)          fprintf(trd,"%.2f  \n",z[dd-1]);
    else                    fprintf(trd," %.2f  \n",z[dd-1]);
    }
 }

qq=MAXIMUM(2,qq);
 for (dd=qq+1;dd<=TT-1;dd++)
 {
    {
    if (Z[dd]<0)            fprintf(trd,"%.2f  ",Z[dd]);
    else                    fprintf(trd," %.2f  ",Z[dd]);
    }
    {
    if ((z[dd-1]*scale)<0) fprintf(trd,"%.2f  ",(z[dd-1]*scale));
    else                    fprintf(trd," %.2f  ",(z[dd-1]*scale));
    }
    {
    if (z[dd-1]<0)          fprintf(trd,"%.2f  ",z[dd-1]);
    else                    fprintf(trd," %.2f  ",z[dd-1]);
    }
    {
    if ((Z[dd]-(scale*error[dd-1]))>100)
                            fprintf(trd,"%.3f  ",(Z[dd]-(scale*error[dd-
1])));
    else                    fprintf(trd," %.3f  ",(Z[dd]-(scale*error[dd-
1])));
    }
    fprintf(trd," 1 ");
    for (ll=1;ll<=(AR+MA+1);ll++)
       {
       if (E[dd-1][ll]<0)  fprintf(trd,"%.2f ",E[dd-1][ll]);
       else                fprintf(trd," %.2f ",E[dd-1][ll]);
       }
    fprintf(trd,"\n");
 }
    {
    if (Z[TT]<0)            fprintf(trd,"%.2f  ",Z[TT]);
    else                    fprintf(trd," %.2f  ",Z[TT]);
    }
```

```
    {
    if ((z[TT-1]*scale)<0) fprintf(trd,"%.2f  ",(z[TT-1]*scale));
    else                   fprintf(trd," %.2f  ",(z[TT-1]*scale));
    }
    {
    if (z[TT-1]<0)         fprintf(trd,"%.2f  \n",z[TT-1]);
    else                   fprintf(trd," %.2f  \n",z[TT-1]);
    }
fclose(trd);

printf("run.execution terminated;\n");
printf("           read files\n");
printf("                [a:\\trd.ntp]\n");
printf("                [a:\\wts.ntp]\n");
if(dm>=2) printf("                    [a:\\cmp.ntp]\n");
printf("           for results.\a\n");
}


int VECTORLARGEST(candidates,dimension)
int dimension;
float candidates[70];
{
int ii;
int jj;
int kk;
int Help;
int vector[70];

for(ii=0;ii<dimension;ii++)
   vector[ii]=ii;
   ii=1;
   while(ii<dimension)
      {
      jj=ii;
      while(jj >= 1)
         {
         if (candidates[vector[jj-1]]>=candidates[vector[jj]])
{
         jj=jj-1;
         continue;
}
              else
              {
              Help=vector[jj-1];
              vector[jj-1]=vector[jj];
              vector[jj]=Help;
              jj=jj-1;
              }
         }
         ii++;
      }
      return (vector[0]);
}

int MAXIMUM(aa,bb)
int aa;
int bb;
{
int maxhelp;
if (aa>=bb) maxhelp=aa;
else        maxhelp=bb;
return maxhelp;
}
```

APPENDIX III: Example Output Files series kk1a

files ini.ntp, cmp.ntp, trd.ntp, wts.ntp

File [d]:\ini.ntp; see chapter 4 for comments:

```
10
3
2
120
2
1
0.11
0.2
0.001
2
2
```

File [d]:\cmp.ntp; output of gradient descent computations:

```
*** gradient descent *********
*** WIDROW HOFF  LMS-algorithm

THIS IS THE 1. EXECUTION OF THE TRAININGS EXAMPLE SET

  Actual starting values for weigths are
  w0     w1     w2     w3     w4     w5
  0.11   0.11   0.11   0.11   0.11   0.11
  Example 1 is going to be executed the 1. time
  Target output value C3 = -0.031 ******************
itn     w0     w1     w2     w3     w4     w5
  1   0.08   0.11   0.11   0.11   0.11   0.11    S3=0.13;
  2   0.05   0.11   0.11   0.10   0.11   0.11    S3=0.10;
  3   0.03   0.11   0.11   0.10   0.11   0.11    S3=0.07;
  4   0.01   0.10   0.11   0.10   0.11   0.11    S3=0.05;
  5   0.00   0.10   0.11   0.10   0.11   0.11    S3=0.03;
  6  -0.01   0.10   0.11   0.10   0.11   0.11    S3=0.02;
  7  -0.02   0.10   0.11   0.10   0.11   0.11    S3=0.01;
  8  -0.02   0.10   0.10   0.10   0.11   0.11    S3=0.00;
  9  -0.03   0.10   0.10   0.09   0.11   0.11   S3=-0.00;
 * rho adopted after 9 iterations; rho=0.020 *******
 10  -0.03   0.10   0.10   0.09   0.11   0.11   S3=-0.01;
 11  -0.04   0.10   0.10   0.09   0.11   0.11   S3=-0.01;
 12  -0.04   0.10   0.10   0.09   0.11   0.11   S3=-0.02;
 13  -0.04   0.10   0.10   0.09   0.11   0.11   S3=-0.02;
 14  -0.04   0.10   0.10   0.09   0.11   0.11   S3=-0.02;
 * rho readopted after 14 iterations; rho=0.020 ****
 15  -0.05   0.10   0.10   0.09   0.11   0.11   S3=-0.02;
 * converged after 15 iterations ******************

final output value S3,15 = -0.024
target output value C3 = -0.031
final error = -0.007
 ------------------------------------------------


  Actual starting values for weigths are
  w0     w1     w2     w3     w4     w5
 -0.05   0.10   0.10   0.09   0.11   0.11
  Example 2 is going to be executed the 1. time
  Target output value C4 = -0.480 ******************
itn     w0     w1     w2     w3     w4     w5
  1  -0.13   0.10   0.09   0.10   0.11   0.11   S4=-0.03;
  2  -0.21   0.09   0.09   0.10   0.11   0.11   S4=-0.12;
  3  -0.26   0.09   0.08   0.10   0.11   0.11   S4=-0.20;
```

```
   4  -0.31    0.09    0.08    0.10    0.11    0.11    S4=-0.25;
   5  -0.34    0.09    0.07    0.10    0.11    0.11    S4=-0.30;
   6  -0.37    0.09    0.07    0.10    0.11    0.11    S4=-0.34;
   7  -0.40    0.09    0.07    0.10    0.11    0.11    S4=-0.37;
   8  -0.41    0.09    0.06    0.10    0.11    0.11    S4=-0.39;
   9  -0.43    0.09    0.06    0.10    0.11    0.11    S4=-0.41;
  10  -0.44    0.09    0.06    0.11    0.11    0.11    S4=-0.42;
  11  -0.45    0.09    0.06    0.11    0.11    0.11    S4=-0.43;
  12  -0.46    0.09    0.06    0.11    0.11    0.11    S4=-0.44;
  13  -0.46    0.09    0.06    0.11    0.11    0.11    S4=-0.45;
  * rho adopted after 13 iterations; rho=0.020 ******
  14  -0.47    0.09    0.06    0.11    0.11    0.11    S4=-0.46;
  15  -0.47    0.08    0.06    0.11    0.11    0.11    S4=-0.46;
  16  -0.47    0.08    0.06    0.11    0.11    0.11    S4=-0.46;
  17  -0.48    0.08    0.06    0.11    0.11    0.11    S4=-0.47;
  18  -0.48    0.08    0.06    0.11    0.11    0.11    S4=-0.47;
  * rho readopted after 18 iterations; rho=0.020 ****
  19  -0.48    0.08    0.06    0.11    0.11    0.11    S4=-0.47;
  20  -0.48    0.08    0.06    0.11    0.11    0.11    S4=-0.47;
  * converged after 20 iterations ******************

final output value S4,20 = -0.474
target output value C4 = -0.480
final error = -0.006
 ------------------------------------------------


   Actual starting values for weigths are
   w0      w1      w2      w3      w4      w5
 -0.48    0.08    0.06    0.11    0.11    0.11
   Example 3 is going to be executed the 1. time
   Target output value C5 = 1.000 ******************
itn      w0      w1      w2      w3      w4      w5
   1  -0.18    0.12    0.05   -0.04    0.11    0.11    S5=-0.53;
   2   0.05    0.14    0.04   -0.15    0.11    0.11    S5=-0.15;
   3   0.23    0.16    0.03   -0.23    0.11    0.11     S5=0.14;
   4   0.36    0.18    0.03   -0.29    0.10    0.11     S5=0.35;
   5   0.45    0.19    0.03   -0.34    0.10    0.11     S5=0.51;
   6   0.53    0.19    0.03   -0.38    0.10    0.11     S5=0.63;
   7   0.58    0.20    0.02   -0.40    0.10    0.11     S5=0.73;
   8   0.62    0.21    0.02   -0.42    0.10    0.11     S5=0.79;
   9   0.65    0.21    0.02   -0.44    0.10    0.11     S5=0.85;
  10   0.68    0.21    0.02   -0.45    0.10    0.11     S5=0.88;
  11   0.69    0.21    0.02   -0.46    0.10    0.11     S5=0.91;
  12   0.71    0.21    0.02   -0.46    0.10    0.11     S5=0.93;
  13   0.72    0.22    0.02   -0.47    0.10    0.11     S5=0.95;
  14   0.72    0.22    0.02   -0.47    0.10    0.11     S5=0.96;
  15   0.73    0.22    0.02   -0.47    0.10    0.11     S5=0.97;
  * rho adopted after 15 iterations; rho=0.020 ******
  16   0.73    0.22    0.02   -0.48    0.10    0.11     S5=0.98;
  17   0.74    0.22    0.02   -0.48    0.10    0.11     S5=0.98;
  18   0.74    0.22    0.02   -0.48    0.10    0.11     S5=0.99;
  19   0.74    0.22    0.02   -0.48    0.10    0.11     S5=0.99;
  * rho readopted after 19 iterations; rho=0.020 ****
  20   0.74    0.22    0.02   -0.48    0.10    0.11     S5=0.99;
  * converged after 20 iterations ******************

final output value S5,20 = 0.993
target output value C5 = 1.000
final error = 0.007
 ------------------------------------------------


   Actual starting values for weigths are
   w0      w1      w2      w3      w4      w5
   0.74    0.22    0.02   -0.48    0.10    0.11
   Example 4 is going to be executed the 1. time
   Target output value C6 = 0.107 ******************
itn      w0      w1      w2      w3      w4      w5
```

```
  1   0.71   0.22   0.03  -0.51   0.10   0.11    S6=0.25;
  2   0.70   0.22   0.04  -0.52   0.10   0.11    S6=0.18;
  3   0.69   0.22   0.04  -0.53   0.10   0.11    S6=0.15;
  4   0.69   0.22   0.05  -0.54   0.10   0.11    S6=0.13;
 * rho adopted after 4 iterations; rho=0.020 *******
  5   0.68   0.22   0.05  -0.54   0.10   0.11    S6=0.12;
  6   0.68   0.22   0.05  -0.54   0.10   0.11    S6=0.11;
 * rho readopted after 6 iterations; rho=0.020 *****
  7   0.68   0.22   0.05  -0.54   0.10   0.11    S6=0.11;
 * converged after 7 iterations ********************

final output value S6,7 = 0.111
target output value C6 = 0.107
final error = -0.004
 ---------------------------------------------------


  Actual starting values for weigths are
  w0     w1     w2     w3     w4     w5
  0.68   0.22   0.05  -0.54   0.10   0.11
  Example 5 is going to be executed the 1. time
  Target output value C7 = -0.122 ******************
itn     w0     w1     w2     w3     w4     w5
  1   0.54   0.29  -0.09  -0.56   0.10   0.11    S7=0.57;
  2   0.47   0.32  -0.17  -0.56   0.10   0.11    S7=0.26;
  3   0.43   0.34  -0.21  -0.57   0.10   0.11    S7=0.09;
  4   0.40   0.35  -0.23  -0.57   0.10   0.11   S7=-0.01;
  5   0.39   0.36  -0.24  -0.57   0.10   0.11   S7=-0.06;
  6   0.38   0.36  -0.25  -0.57   0.10   0.11   S7=-0.09;
  7   0.38   0.37  -0.25  -0.57   0.10   0.11   S7=-0.10;
 * rho adopted after 7 iterations; rho=0.020 *******
  8   0.38   0.37  -0.26  -0.57   0.10   0.11   S7=-0.11;
  9   0.38   0.37  -0.26  -0.57   0.10   0.11   S7=-0.12;
 * rho readopted after 9 iterations; rho=0.020 *****
 * converged after 9 iterations ********************

final output value S7,9 = -0.116
target output value C7 = -0.122
final error = -0.006
 ---------------------------------------------------


  Actual starting values for weigths are
  w0     w1     w2     w3     w4     w5
  0.38   0.37  -0.26  -0.57   0.10   0.11
  Example 6 is going to be executed the 1. time
  Target output value C8 = -2.084 ******************
itn     w0     w1     w2     w3     w4     w5
  1  -0.20  -0.21  -0.32  -0.50   0.10   0.11    S8=0.78;
  2  -0.54  -0.55  -0.36  -0.46   0.10   0.11   S8=-0.38;
  3  -0.74  -0.75  -0.38  -0.44   0.10   0.11   S8=-1.07;
  4  -0.86  -0.87  -0.39  -0.42   0.11   0.11   S8=-1.48;
  5  -0.93  -0.94  -0.40  -0.41   0.11   0.11   S8=-1.72;
  6  -0.98  -0.99  -0.40  -0.41   0.11   0.11   S8=-1.87;
  7  -1.00  -1.01  -0.41  -0.41   0.11   0.11   S8=-1.96;
  8  -1.02  -1.03  -0.41  -0.40   0.11   0.11   S8=-2.01;
  9  -1.03  -1.04  -0.41  -0.40   0.11   0.11   S8=-2.04;
 10  -1.03  -1.04  -0.41  -0.40   0.11   0.11   S8=-2.06;
 * rho adopted after 10 iterations; rho=0.020 ******
 11  -1.03  -1.04  -0.41  -0.40   0.11   0.11   S8=-2.07;
 12  -1.04  -1.05  -0.41  -0.40   0.11   0.11   S8=-2.07;
 * rho readopted after 12 iterations; rho=0.020 ****
 13  -1.04  -1.05  -0.41  -0.40   0.11   0.11   S8=-2.08;
 * converged after 13 iterations ******************

final output value S8,13 = -2.078
target output value C8 = -2.084
final error = -0.006
 ---------------------------------------------------
```

```
THIS IS THE 2. EXECUTION OF THE TRAININGS EXAMPLE SET

  Actual starting values for weigths are
  w0      w1      w2      w3      w4      w5
 -0.05    0.10    0.10    0.09    0.11    0.11
  Example 1 is going to be executed the 2. time
  Target output value C3 = -0.031 ******************
itn      w0      w1      w2      w3      w4      w5
  1  -0.05    0.10    0.10    0.09    0.11    0.11    S3=-0.03;
 * converged after 1 iterations ********************

final output value S3,1 = -0.026
target output value C3 = -0.031
final error = -0.005
 ------------------------------------------------


  Actual starting values for weigths are
  w0      w1      w2      w3      w4      w5
 -0.48    0.08    0.06    0.11    0.11    0.11
  Example 2 is going to be executed the 2. time
  Target output value C4 = -0.480 ******************
itn      w0      w1      w2      w3      w4      w5
  1  -0.48    0.08    0.06    0.11    0.11    0.11    S4=-0.47;
 * converged after 1 iterations ********************

final output value S4,1 = -0.475
target output value C4 = -0.480
final error = -0.005
 ------------------------------------------------


  Actual starting values for weigths are
  w0      w1      w2      w3      w4      w5
  0.74    0.22    0.02   -0.48    0.10    0.11
  Example 3 is going to be executed the 2. time
  Target output value C5 = 1.000 *******************
itn      w0      w1      w2      w3      w4      w5
  1   0.74    0.22    0.02   -0.48    0.10    0.11     S5=1.00;
 * converged after 1 iterations ********************

final output value S5,1 = 0.995
target output value C5 = 1.000
final error = 0.005
 ------------------------------------------------


  Actual starting values for weigths are
  w0      w1      w2      w3      w4      w5
  0.68    0.22    0.05   -0.54    0.10    0.11
  Example 4 is going to be executed the 2. time
  Target output value C6 = 0.107 *******************
itn      w0      w1      w2      w3      w4      w5
  1   0.68    0.22    0.05   -0.54    0.10    0.11     S6=0.11;
 * converged after 1 iterations ********************

final output value S6,1 = 0.110
target output value C6 = 0.107
final error = -0.002
 ------------------------------------------------


  Actual starting values for weigths are
  w0      w1      w2      w3      w4      w5
  0.38    0.37   -0.26   -0.57    0.10    0.11
  Example 5 is going to be executed the 2. time
  Target output value C7 = -0.122 ******************
```

```
itn      w0      w1      w2      w3      w4      w5
  1    0.38    0.37   -0.26   -0.57    0.10    0.11    S7=-0.12;
 * converged after 1 iterations ********************

final output value S7,1 = -0.118
target output value C7 = -0.122
final error = -0.003
 ------------------------------------------------


  Actual starting values for weigths are
  w0      w1      w2      w3      w4      w5
-1.04   -1.05   -0.41   -0.40    0.11    0.11
  Example 6 is going to be executed the 2. time
  Target output value C8 = -2.084 ******************
itn      w0      w1      w2      w3      w4      w5
  1   -1.04   -1.05   -0.41   -0.40    0.11    0.11    S8=-2.08;
 * converged after 1 iterations ********************

final output value S8,1 = -2.080
target output value C8 = -2.084
final error = -0.004
 ------------------------------------------------

final SSE = 0.0019
```

File [d]:\wts.ntp; final weights for each training example
in the PSE-routine;
in the MSE-routine only **one** set of weights

<bias>	$<d_{t-2}>$	$<d_{t-1}>$	$<d_t>$	$<error_{t-2}>$	$<error_{t-1}>$	1	
-0.05	0.10	0.10	0.09	0.11	0.11	1	15
-0.48	0.08	0.06	0.11	0.11	0.11	1	20
0.74	0.22	0.02	-0.48	0.10	0.11	1	20
0.68	0.22	0.05	-0.54	0.10	0.11	1	7
0.38	0.37	-0.26	-0.57	0.10	0.11	1	9
-1.04	-1.05	-0.41	-0.40	0.11	0.11	1	13

For each training example best last weights and informations on gradient descent (lowest|highest number of iterations) are listed above.

File [d]:\trd.ntp; trained series and information on training examples set

```
final SSE = 0.002
```

<original series>	<first differences and first scaled differences>		<trained series>	<training examples set; s.a.>					<dt+1>
31.35									
31.59	0.24	0.06							
31.75	0.16	0.04							
32.21	0.46	0.11	32.233	1	0.06	0.04	0.11	0.00	0.00 -0.03
32.08	-0.13	-0.03	32.101	1	0.04	0.11	-0.03	0.00	-0.01 -0.48
30.07	-2.01	-0.48	30.050	1	0.11	-0.03	-0.48	-0.01	-0.00 1.00
34.26	4.19	1.00	34.269	1	-0.03	-0.48	1.00	-0.00	0.00 0.11
34.71	0.45	0.11	34.724	1	-0.48	1.00	0.11	0.00	-0.00 -0.12
34.20	-0.51	-0.12	34.216	1	1.00	0.11	-0.12	-0.00	-0.00 -2.08
25.47	-8.73	-2.08							