Internal Report 09-04 May 2009

Universiteit Leiden

Opleiding Informatica

Investigating an Evolutionary Strategy to Forecast Time Series

Kerstin Wurdinger

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden

Abstract

Recently many investigations have been published about finding good solutions to forecast time series. Different linear and non-linear approaches and hybrid models have been used successfully. Approaches of using Genetic Algorithms and Evolutionary Programming could demonstrate suitability, too. However, a third current mainstream of Evolutionary Algorithms – Evolutionary Strategies – is hardly investigated for this issue. This study is aimed to contribute research to the question whether and how suitable an Evolutionary Strategy can be for forecasting time series. Test sequences of different origins were subject to research. We used a combined model of Autoregressive Moving Average Model and a neural network for our experiments; the overall architecture and the parameters of both components have been optimized by a Covariance Matrix Adaptation Evolutionary Strategy. Our experiments revealed that the Evolutionary Strategy is very well suitable to forecast time series of natural data and computer generated sequences, but less good in forecasting financial time series.

Acknowledgements

I thank Prof. Dr. Thomas Bäck for his valuable feedback during the period of research and writing, and Meneer Jansen for his unconditional support.

Contents

Introduction

Recently a whole string of investigations has been published about the challenge to find quantitative and qualitative good solutions to forecast time series. Several authors used different neural networks [2][3][5][7][8][27][31][40][41][47] or fuzzy systems [13][19][21][29][39][47] with non-linear characteristics as approximation model for forecasts. Also the application of classical linear models as pure Autoregressive Model (AR), pure Moving Average Model (MA), Autoregressive Moving Average Model (ARMA), Autoregressive Integrated Moving Average Model (ARIMA) or Vector multivariate Autoregressive model (VAR) [5][7][8][12][40][41][43] has been researched. Furthermore Genetic and Evolutionary Programming (GP and EP) were used to demonstrate the generation of suitable models themselves [15][16][23][24][25][45]. A few authors combined a selection of these models experimenting with hybrid models [1][5][6][28][57].

Evolutionary Algorithms (EA) are mostly used to solve parameter optimization problems [53]. Since they are global random search algorithms they appear to be of use to forecast time series as well. EAs can either optimize the architecture of a certain model or estimate its parameters or they can do both. While for Genetic Algorithms (GA) and extensions of it reasonable research results are available in literature [1][9][10][11][13][14][17][19][21][22][28][35][39][42][44][46], Evolutionary Strategies (ES) are hardly investigated for this issue. At this moment only a few publications use ES for optimization tasks connected to time series forecasting. These authors investigated ESs for the parameter estimation or learning algorithm selection together with fuzzy systems [19], neural networks [20], ARMA based models [26] or Gene Regulatory Networks [35] as basic models.

This study is aimed to contribute further research to the question whether and how suitable ESs can be for forecasting time series. Therefore, we suggest an ensemble of linear (ARMA) and non-linear models (neural network) to approximate the time series. A hybrid model should bring together advantages of both models resulting from their different characteristics. The present study investigates an ES algorithm on this suggested approximation model and tests its implementation with several sets of time series data. Among ESs, the ES with Covariance Matrix Adaptation (CMA-ES) [33][34][36][38] is the state of the art algorithm and therefore chosen for these experiments. To our knowledge, CMA-ES has been one time only applied to forecast a time series: as successful parameter estimation for a Gene Regulatory Network [35]. So, when our study started we faced the challenge to look into an algorithm without having much previous research of the subject to our reference.

1 Forecasting a time series

1.1 Definitions

A time series is a sequence of chronologically ordered data where each single information is measured at a specific time [10][51]. Such a collection of data points can either consist of real experimental data from different subject areas as physics or astronomy, biogeochemistry or finance or it consists of computer generated data. The experimental data are derived from observations of natural phenomena (e.g. solar cycle, climate change), economic life (e.g. sales figures), or financial markets (e.g. stock price indices). Computer generated time series can be calculated by solving equations of some determined nonlinear dynamic systems at a specific time. With a certain set of parameters, the solution of these equations or systems of equations show deterministic chaotic behaviour. Lorenz attractor, Henon map and Mackey-Glass time series are well-known examples for computer generated time series [51].

When studying the literature about modelling time series one frequently meets different terminology for estimating data points in the future: *forecasting* and *prediction*. *Prediction* is "generally interpreted as a process of estimating a future event based on subjective considerations" [49]. This study will use the term *forecast* as suggested in [49] to describe the process of estimating a future event by scientifically analyzing past data, discovering underlying patterns and using the information in a predetermined way.

Recently, the application of Evolutionary Algorithms (EAs) for parameter optimization and model generation increased in importance. Evolutionary Algorithms are biologically-inspired search and optimization algorithms taking organic evolution as the model [52]. They use processes of mutation, selection and recombination for the search process and environmental information as feedback. Beside Genetic Algorithms (GAs) and Evolutionary Programming (EP), Evolutionary Strategies (ESs) are one of the currently three mainstreams of Evolutionary Algorithms. Originally developed by Bienert, Rechenberg and Schwefel almost five decades ago [52], the Covariance Matrix Adaptation (CMA-ES) proposed by Hansen and Ostermeier [36][34] is currently state-of-the-art among Evolutionary Strategies. This strategy does not only contain mutation, selection and recombination processes for optimizing object variables, but also the capability of optimizing the inherent parameters of the strategy in dependence of the actual object variables. In addition, the important concept of *adaptation by accumulation* was added, that is, a wise use of past information during the evolution [53]. Especially this high performant capability of self-adaptation makes the CMA-ES a candidate for dynamical optimization problems in general and for time series forecast in particular.

1.2 Difficulties of classical methods in forecasting a time series and the aim of this study

For the last four decades, linear and nonlinear approaches have been developed for modelling a time series forecast [5]. At first researchers either have used several autoregressive models (e.g. Autoregressive Integrated Moving Average Model or Vector multivariate Autoregressive model) or they have used different neural networks (e.g. Radial Basis Function Neural Networks or Multilayer

Perceptrons). The parameters of one or the other model have been estimated by classical methods. Parameters of ARIMA models have been designed e.g. by Holt-Winters or Box-Jenkins methodology [10]. Learning methods like Backpropagation [4][18][32] or Levenberg-Marquardt [1][5][11] have been used is the case of neural networks.

What difficulties do classical methods face in forecasting a time series? For neural networks the key problems turn out to be a) the method to determine the appropriate architecture of the neural network and b) the decision what learning algorithm should be used for the successful training [30]. A third and possibly the most important issue is the appropriate selection of the number of past observations. There is no theory available for guidance for this selection [6][46]. In the absence of theories appropriate search algorithms for optimizing a model appeared to be a solution. Evolutionary Algorithms have entered as Genetic Algorithms [10][14][17] or as Evolutionary Strategies [1][20][35] the area of parameter estimation since they are powerful tools in optimization tasks for one ore more overall criteria [52]. In the case of linear approaches the main problem to get a successful model lies in the capability to identify and possibly transform the data of the time series such that it fits a model with certain components [6][10]. Autocorrelation and cross-correlation functions as measure of linear dependence support the process to determinate whether it is necessary to select e.g. for an ARIMA model an autoregressive part and/or an moving average part and/or perhaps to use an integration part [50].

However, even after one has determined the architecture and estimated the parameters of the model adequately, there are still two further issues that can interfere in the desire to achieve a successful forecast. The model is not necessarily optimized. The different patterns of a time series themselves are the reason for that. First, linear and nonlinear approaches do have the inherent strength to capture linear and nonlinear patterns respectively within a time series. A time series with significant nonlinear data would request a model beyond the limits of a linear approach as ARIMA [5]. Second, there is always the risk that a model overfits the original data. Overfitting leads to a very exact model for past data (in-sample) but an unsuitable model for the forecast accuracy (out-sample), the model can correctly process training data but it fails in generalization. In the literature we can find different approaches to overcome both latter issues of models with a previously determined architecture. Several authors tried to adapt the approach for time series to specific pattern (to name a few of them: the bilinear model, the threshold autoregressive model, the smoothing transition autoregressive model [5]). That way, the models lost their flexibility to be applicable for forecasting time series data universally. A second suggestion was the development of hybrid models, models that are capable to capture linear contributions to a time series as much as nonlinear ones. Combinations of a neural network, ARIMA or a fuzzy system could outperform single models [1][5][6][28]. A whole bunch of research also was made to support the determination of an appropriate model architecture. Lacking theoretical methods to design an initial neural network have been successfully replaced by Evolutionary Algorithms. Evolutionary Programming (GP and EP) dominate the research up to now [15][16][23][24][25][45]. For the design of the architecture by Evolutionary Strategies (ES) is, as far as we know, no research available.

With the present study we want to partly reduce this lack of studies. Evolutionary Strategies are search and optimization algorithms. They can directly use the concept of Pareto Dominance [54]. Based on this concept an Evolutionary Strategy shall be useful at different levels of optimization: to partly evolve structural parameters of known forecasting methods and to tune parameters. In

particular, in this study shall be investigated whether an Evolutionary Strategy (i.e. the Covariance Matrix Adaptation (CMA-ES)) can mean an advantage when a) being used as training algorithm, i.e. for parameter estimation, b) selecting the number of past observations, i.e. for optimizing the architecture design on a low level, and c) determining linear and nonlinear contributions to a combined model, i.e. for optimizing the architecture design on a high level.

2 Classical forecasting methods for system modelling

One characteristic of a method to model a system is its fundamental mathematical structure. For this property, classical methods can be divided into linear and non-linear ones. In the current chapter we want to introduce shortly two respective representatives and selected those methods we use in our experiments later on, too: autoregressive moving average model (ARMA) and a neural network. The strength of neural networks is their ability to discover patterns, to learn to deal with new situations during training and finally to generalize to them [55] while ARMA models can fit on time series very well if no discontinuities occur [51].

2.1 Autoregressive Moving Average Model (ARMA)

The autoregressive moving average model is a combination of two models: an autoregressive model (AR) and an moving average model (MA). It is based on the assumption that a present value y_t of a time series can be calculated by a linear function of past values and white noise. The autoregressive part of the ARMA-model considers the dependencies of the present value y_t on past values:

$$y_t = y_{t-1} \cdot \phi_1 + y_{t-2} \cdot \phi_2 + \ldots + y_{t-p} \cdot \phi_p + \varepsilon_t \qquad (1)$$

In equation *(1)*, p determines the number of lags that necessarily have to be used to reproduce the present value. ϕ_m are the parameters and ε_t is the actual value of the white noise series with mean=0 and a determined variance at time *t*.

The moving average part of the ARMA-model contributes the dependencies of the present value y_t from past values of a white noise series:

$$y_t = \varepsilon_{t-1} \cdot \vartheta_1 + \varepsilon_{t-2} \cdot \vartheta_2 + \ldots + \varepsilon_{t-q} \cdot \vartheta_q \qquad (2)$$

Here, q is the order of the process for the moving average model and has the same function as p for the autoregressive model does. ϑ_n are the parameters.

The combination of *(1)* and *(2)* leads to the representation of the present value by the ARMA-model:

$$y_t = y_{t-1} \cdot \phi_1 + y_{t-2} \cdot \phi_2 + \ldots + y_{t-p} \cdot \phi_p + \varepsilon_t + \varepsilon_{t-1} \cdot \vartheta_1 + \varepsilon_{t-2} \cdot \vartheta_2 + \ldots + \varepsilon_{t-q} \cdot \vartheta_q \qquad (3)$$

To get the optimal value of the orders p and q and to find the optimal parameters ϕ_m and ϑ_n for an ARMA-model it is necessary to appropriately analyse the data and possibly to transform them. After that the dependence orders of the model have to be identified, the model components to be estimated and as a last step the adequacy of the model to be examined [50].

2.2 Neural Networks (NN)

To forecast a time series a neural network must be capable to generate a particular output sequence in response to a determined input sequence. This task is called *Temporal Association* [55]. It includes the capability to a) recognize a sequence and to b) reproduce a sequence. Since simple neural networks meet the requirement of sequence recognition only we need a network with recurrency for time series forecasting. Recurrency refers to the quality of a network to have "connections allowed both ways between pairs of units, and even from a unit to itself" [55]. Networks with recurrency are the state of the art for the tasks of temporal pattern classification, system identification and nonlinear time series forecast [63].

One of several possible architectures of neural networks is the layered feed-forward one. It has been studied around three decades ago [55]. With additional direct or indirect loops of connection this network fulfils the requirement of recurrency and is therefore suitable for time series forecasts [31]. Since a small amount of connections only are feedback connections such a network is called partially recurrent. It uses context units. Context units store the output of the hidden layer of one or more previous time steps and serve as additional input units. That way they provide information from the past to the current state of the network. Neural networks with context units have been suggested by Elman, Kohonen, Jordan and Mozer [55].

We have chosen the Elman network for parts of our experiments because it is recurrent and because of the simplicity to use a function of the MATLAB Toolbox. The Elman network is well suited to generate and reproduce time sequences [55]. Its capability to store information in internal states for later use made it an efficient tool for identification once the network is trained [60][62]. In the literature we find evidence that it has been successfully applied to solve forecasting tasks [59][61][62][63]. However, if trained with classical backpropagation algorithm the network can suffer from finding a suboptimal solution. Recently, several authors suggested alternative learning methods to overcome this weakness to a certain extent [60][62][64].

Figure 1(a) shows its general architecture ([55], p. 180) and *Figure 1(b)* an example with 3 hidden units and a context layer with 1 lag.

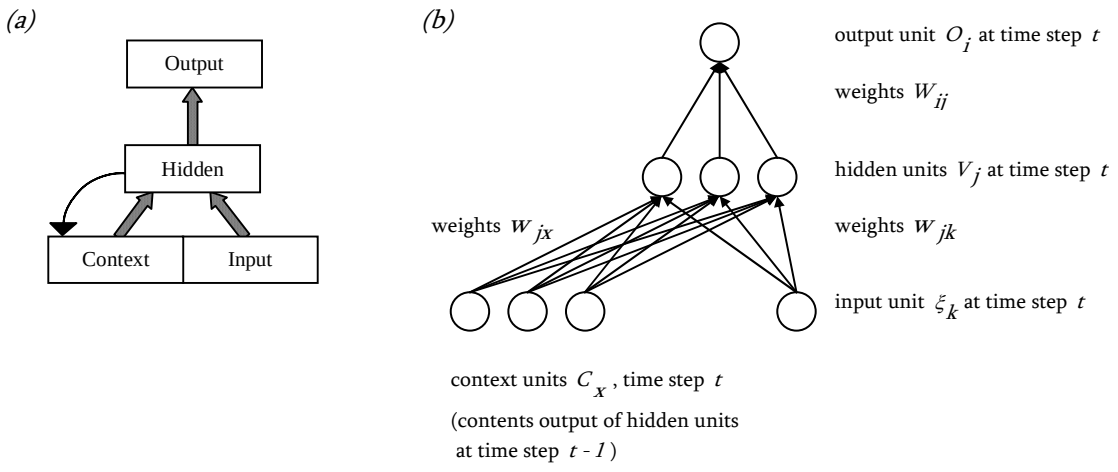


Figure 1. Elman network

The network consists of one input layer with units ξ_k, hidden layer units V_j, context units C_x and output layer units O_i. The connections between the units are the weights w_{jk} and W_{ij}. If the task is a time series forecast the model contains one input unit and one output unit only. The context units contain past information from one or more than one time steps, i.e. additional lags m are possible as well. The number of past observations is dependent on the test cases and defined – as well as the numbers of the different units - during the design of the network architecture. In addition, bias coefficients[1)] can be defined for the hidden layer units and the output units.

A time series is learned by training the network with a subset of the whole time series. A classic algorithm to do this task is called backpropagation. In the process, the network learns a training set of input-output pairs $\left[\xi_{k_t}, \xi_{k_{t+1}}\right]$. Backpropagation is applied as follows. The context units and the weight matrices are initialized with random values. The first item of the training set of data is presented to the input unit. As a result of that, the input/context to the hidden units *(4a)* and the output from the hidden units *(4b)* are achieved:

$$h_j = \sum_k w_{jk} \cdot \xi_k + \sum_x w_{jx} \cdot C_x \qquad (4a)$$

$$V_j = g(h_j) = g\left(\sum_k w_{jk} \cdot \xi_k + \sum_x w_{jx} \cdot C_x \right) \qquad (4b)$$

The activation function g for h_j and h_i is usually a sigmoid function [55] and determines how input values cause a change in the output of the node *(5)*. β is a parameter $\frac{1}{k_B T}$.[2)]

$$g(h) = \frac{1}{1 + exp(-2\beta h)} \qquad (5)$$

After that, the input to and the output from the output unit, i.e. the output of the whole network, is gradually calculated by *(6a)* and *(6b)*.

$$h_i = \sum_j W_{ij} \cdot V_j \qquad (6a)$$

$$O_i = g(h_i) = g\left(\sum_j W_{ij} \cdot V_j \right) \qquad (6b)$$

---

1) the bias coefficients are not drawn in *Figure 1(b)* for the reason of clarity

2) k_B = Boltzmann's constant, T = temparature in K

The output unit is representing the output of the whole network. Its calculation can be summarized to *(7)*:

$$O_i = g\left(\sum_j W_{ij} \cdot g\left(\sum_k w_{jk} \cdot \xi_k + \sum_x w_{jx} \cdot C_x\right)\right) \tag{7}$$

Now, the output of the network at time step t O_{i_t} is compared with the input at time step $t+1$ $\xi_{k_{t+1}}$. The comparison leads us – using the definition that $\xi_{k_{t+1}} = \zeta_i$ – to equation *(8)*, the definition of the error E. The index μ represents the set of patterns that is presented to the input nodes during training.

$$E = \frac{1}{2}\sum_{\mu i}\left(\zeta_i - O_i\right)^2 \tag{8}$$

The error is processed to provide deltas δ that are necessary to update the weight matrices for the hidden-to-output connections *(9a)*, the input-to-hidden connections *(9b)* and the context-to-hidden connections *(9c)*. This update process starts at the output unit.

$$\begin{aligned}\Delta W_{ij} &= -\eta \cdot \frac{\partial E}{\partial W_{ij}} = \eta \cdot \sum_{\mu}\left(\zeta_i{}^{\mu} - O_i{}^{\mu}\right)\cdot g'(h_i{}^{\mu})V_j{}^{\mu} \\ &= \eta \cdot \sum_{\mu} \delta_i{}^{\mu} \cdot V_j{}^{\mu} \qquad \text{with } \delta_i{}^{\mu} = \left(\zeta_i{}^{\mu} - O_i{}^{\mu}\right)\cdot g'(h_i{}^{\mu})\end{aligned} \tag{9a}$$

$$\begin{aligned}\Delta w_{jk} &= -\eta \cdot \frac{\partial E}{\partial w_{jk}} = -\eta \cdot \frac{\partial E}{\partial V_j{}^{\mu}}\frac{\partial V_j{}^{\mu}}{\partial w_{jk}} \\ &= \eta \cdot \sum_{\mu i}\left(\zeta_i{}^{\mu} - O_i{}^{\mu}\right)\cdot g'(h_i{}^{\mu})W_{ij} \cdot g'(h_j{}^{\mu})\xi_k{}^{\mu} \\ &= \eta \cdot \sum_{\mu} \delta_j{}^{\mu}\xi_k{}^{\mu} \qquad \text{with } \delta_j{}^{\mu} = g'(h_j{}^{\mu}) \cdot \sum_i \delta_i{}^{\mu} W_{ij}\end{aligned} \tag{9b}$$

$$\begin{aligned}\Delta w_{jx} &= -\eta \cdot \frac{\partial E}{\partial w_{jx}} = -\eta \cdot \frac{\partial E}{\partial V_j{}^{\mu}}\frac{\partial V_j{}^{\mu}}{\partial w_{jx}} \\ &= \eta \cdot \sum_{\mu i}\left(\zeta_i{}^{\mu} - O_i{}^{\mu}\right)\cdot g'(h_i{}^{\mu})W_{ij} \cdot g'(h_j{}^{\mu})C_x{}^{\mu} \\ &= \eta \cdot \sum_{\mu} \delta_j{}^{\mu} C_k{}^{\mu} \qquad \text{with } \delta_j{}^{\mu} = g'(h_j{}^{\mu}) \cdot \sum_i \delta_i{}^{\mu} W_{ij}\end{aligned} \tag{9c}$$

Finally the whole procedure is repeated until the forecasting error is sufficiently small.

3 The approach of a combined model using the Evolutionary Strategy CMA-ES

3.1 Architecture design and parameter estimation of the combined model

Authors in [5] describe that a broad kind of studies confirm the idea that a combination of methods for forecasting performs better than an isolated one. The basic idea behind a combination of models is to use the specific strength of a model for modelling a specific time series to optimize the performance of a forecast. In fact, there is much evidence that it is dependent on the test cases whether e.g. a linear or a nonlinear model is more effective to forecast a time series [5]. Linear and nonlinear approaches have the inherent strength to capture linear and nonlinear patterns respectively within a time series [6]. While nonlinear models theoretically should be able to capture linear patterns, on one hand a linear approach would be the straight solution to form a model it is highly specialized for. On the other hand, nonlinear models can also be specialized by design for a certain nonlinear pattern with the consequence that their capability to capture a different (non-)linear pattern is decreased. That way even a linear model could outperform them in modelling a nonlinear test case [5].

We assume that the advantage of using a combined model also holds in the case of combining an ARMA model and a neural network. Evidence for this assumption can be found for the forecast of different time series in [1][5] and [6]. Further, a hybrid model of ARIMA and a neural network has been implemented successfully before in short-term forecasting speed time series for traffic regulation [57]. Because of that we decided to use an combined model, too. In this way we expect to cover linear and non-linear characteristics as well if the necessity of one or both of them should occur. For the general model *(10)* we use the common definition that the time series value of the overall model $y_{t_{sum}}$ consists of the time series values y_t of every part that is contributing to. There is also a coefficient c for every y_{ti} respectively. Our own model will be a linear combination of two models, thus n = 2.

$$y_{t_{sum}} = \sum_{i=1}^{n} c_i \cdot y_{t\,i} \qquad (10)$$

However, there are different approaches how the combination shall be established, how the weights shall be determined. If one considers the task as a multiobjective optimization, usually the general sum of the weights is equal to 1 [58]. Authors in [5] adopt this assumption and use a method with equal weights. The authors appear to justify this choice with previous research. The method with equal weights is simple and robust, in addition it is concluded that it is not outperformed by more sophisticated approaches. A different approach is tried in [1]. The authors consider the task still as a multiobjective optimization and proposes the estimation of coefficients of a linear combination. However, in contrast to [58] and [5], the coefficients are real numbers between –1 and 1 and the results show that the sum of the coefficients does not necessarily have to be 1 for a competitive performance. In our study we want to take advantage of a strength of CMA-ES: the capability to

handle multiobjective optimization tasks. Therefore we chose the combined model with coefficents for the included models instead of weigthing them equally *(11)*. We also decide to use a model where the general sum of weights has not to be 1, because we want to set as little constraints as possible if the algorithm should meet an instable ARMA model during the search.

$$y_{t_{sum}} = c_{ARMA} \cdot y_{t_{ARMA}} + c_{NN} \cdot y_{t\,NN} \tag{11}$$

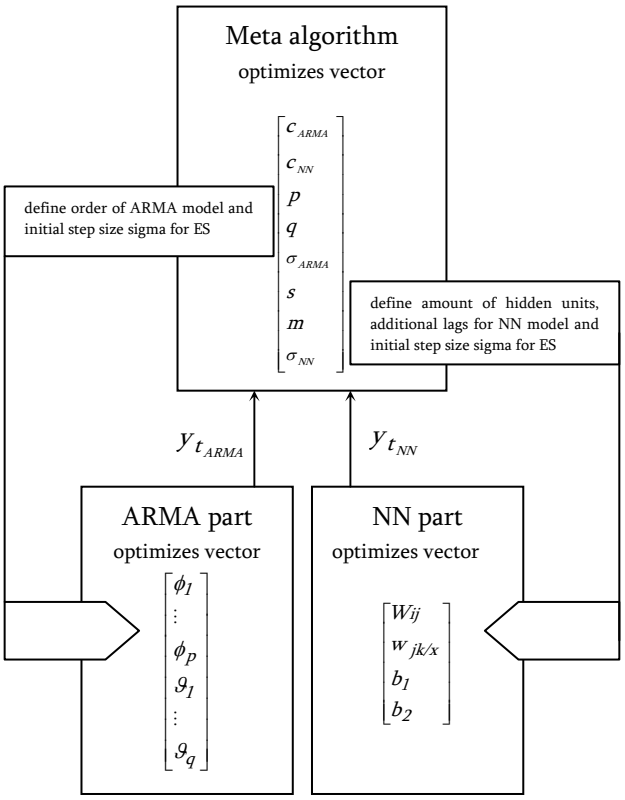


Figure 2. Block diagram of combined model

Figure 2 shows a block diagram of our model. We will refer to this part of our model, that calculates $y_{t_{sum}}$, as meta algorithm (see *Figure 3*). To calculate the contributing time series values y_{t_i} we suggest two separate two low level algorithms that are further explained in paragraph 3.2 and paragraph 3.3. The meta algorithm uses the CMA-ES algorithm [37]. It optimizes a vector of 8 objective variables and fulfils three main tasks. First, the coefficients c_{ARMA} and c_{NN} are directly subject to the optimization. Second, the meta algorithm will provide the parameters for the architecture for the two low level algorithms that calculate $y_{t_{ARMA}}$ and $y_{t\,NN}$. In detail this means that it optimizes a) the orders p and q for the ARMA part and b) the number of hidden nodes s and the number of lags m for the context layer for the neural network part. Third, both low level algorithms receive their initial step size sigma, i.e. the value to create the vector of standard deviation for the ES, from the meta algorithm. The Sum of Squared Errors (SSE) serves to measure the quality of the forecast by using the error $\hat{y}_t - y_t$ between the actual time series value $\hat{y}_t$ and the modelled time series value y_t. The number of forecasts is denoted by N_f:

$$SSE = \sum_{t=1}^{N_f} (\hat{y}_t - y_t)^2 \qquad (12)$$

Since choosing the best model is also an issue to avoid overfitting, the Bayesian Information Criterion (BIC) [10] is used as fitness function on the meta level and on the low levels. It takes additionally into consideration that the model uses an appropriate number of parameters to keep the ability to generalize new data. In equation *(13)*

$$BIC = N_{ex} \cdot ln\left(\frac{SSE}{N_{ex}}\right) + N_{par} \cdot ln(N_{ex}) \qquad (13)$$

N_{ex} denotes the number of training examples and N_{par} the number of model parameters. The number of parameters N_{par} is raised by the two coefficient parameters c_i when the BIC is calculated for the meta algorithm.

```
generate population
initialize strategy parameters
while stopping_condition ≠ true do
    for k=1 to lambda
              mutate population
              calculate y_ARMA using low level algorithm ARMA
              calculate y_NN using low level algorithm NN
              calculate y_sum = c_ARMA .y_ARMA + c_NN . y_NN
              if constraints_met = false
                        apply penalties
              end if
              evaluate fitness of y_sum (using training data)
    end for
    select best offspring mju
    recombine best offspring
    update strategy parameters
end do
apply fittest model to test data
```

Figure 3. Meta algorithm

The meta algorithm penalizes if necessary values of single parameters that determine the architecture of the ARMA and the neural network part respectively. The penalties are introduced to limit the dynamics of the search process, considering that we calculate 1000 generations only. Especially for the neural network part the number of parameters can otherwise increase very much. However, our goal is to find optimal architectures for the low level algorithms. This goal is usually indicated by a smaller BIC and an excessively large number of parameters would promote a grow of the BIC.

3.2 Architecture design and parameter estimation of ARMA part

The calculation of the coefficients of the ARMA model is the task of the first of the two low level optimizations. The model receives the order p for the autoregressive model and the order q for the moving average model from the meta algorithm. Further it gets the initial step size sigma for running the fixed amount of generations. The number of generations depends on the orders of the model and is – in contrast to the original model by N. Hansen [37] – considerably reduced to 25 times the total orders of the model. Tests with the original number of generations in the ARMA part of the algorithm showed that except of a larger effort in computation time the results of the optimization as a whole did not differ in such a way that additional computational effort is justified.

The coefficients for the ARMA model are optimized by the ES. The standard function *predict* from the MATLAB System Identification Toolbox is used to calculate $y_{t_{ARMA}}$ for a one-step-ahead prediction. As for the meta algorithm the BIC *(13)* is again representing the fitness. The ARMA(p,q) process can be completely specified with the number of parameters in equation *(14)* [10]. We use $N_{par_{ARMA}}$ for the calculation of the BIC in the ARMA part.

$$N_{par_{ARMA}} = p + q + 1 \qquad (14)$$

3.3 Architecture design and parameter estimation of neural network part

The two weight matrices and the two bias vectors of an Elman network (see paragraph 2.2) are subject to optimization in the neural network part. This low level algorithm receives the number of hidden nodes s and the number of lagged values in the input vector m for the architecture of the model. The initial step size sigma is handed over to the neural network part. The ES carries out the training. The number of cycles to train the network is limited to 500 cycles. Again, the computational effort is the main reason for a limitation. Our tests showed that the dynamic process of searching the optimum for the neural network was sufficiently finished after this number of steps. We concluded this by examining the course of the fitness in diagrammatic form.

The standard function *nnt2elm* from the MATLAB Neural Network Toolbox is used to create the neural network. The matrices for weights and biases for the neural network model are optimized by the ES. The author in [4] states the suitability to use an ES for this task, that "ESs are, in principle, capable of successful weight tuning in neural networks" [4]. The function *sim* applies the model to the set of training data and calculates $y_{t_{NN}}$. The BIC *(13)* is again the fitness function. The number of parameters for the neural network part is

$$N_{par_{NN}} = s \cdot (m + 2) + 1 \qquad (15)$$

assuming, that the number of lags m is equal to 1 when the model does not contain context units C_x, i.e. it consists exclusively of input units ξ_k [56].

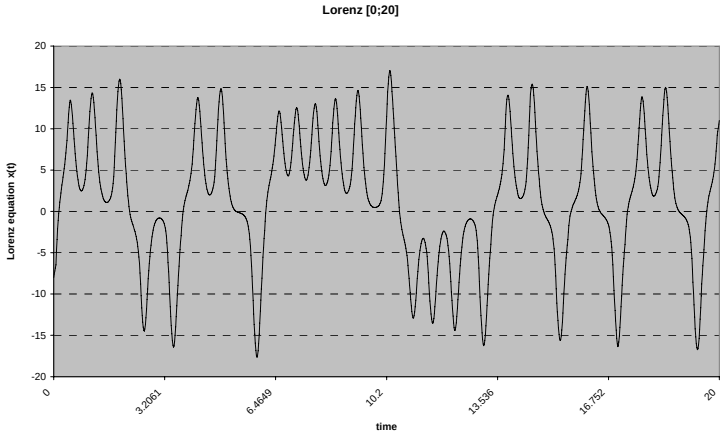
4 Introduction of the test data

The current chapter is organized as follows. Every time series investigated is plotted as a whole to get the general picture. The source of a time series is explicitely provided, also the manner of splitting the data points into a set of training data and test data respectively. If in previous research information about the size of a time series and the splitting ratio has been documented, we used the data in the same way in our experiments. This way we are able to compare results with our new model also with results of previous research and not with our own reference values of classical methods only.

4.1 Computer generated time series

4.1.1 Lorenz

The collection consists of 1213 datapoints.



The time series is generated with MATLAB code at *http://www.math.tamu.edu/~kfu/matode.pdf*. To our knowledge, a completely documented experiment (i.e. including information about the initial values of all parameters and about the coordinates of the starting point) has not been published where we could compare the Root Mean Squared Error[1)] (RMSE) or the Mean Squared Error (MSE) with the results of our model. That is why we decided to use the values as shown in *Figure 4*. The data has been split in ratio 1:1: the training data consists of 607 data points and the test data consists of 606 data points. The x-value of the Lorenz equation is used for our experiments.

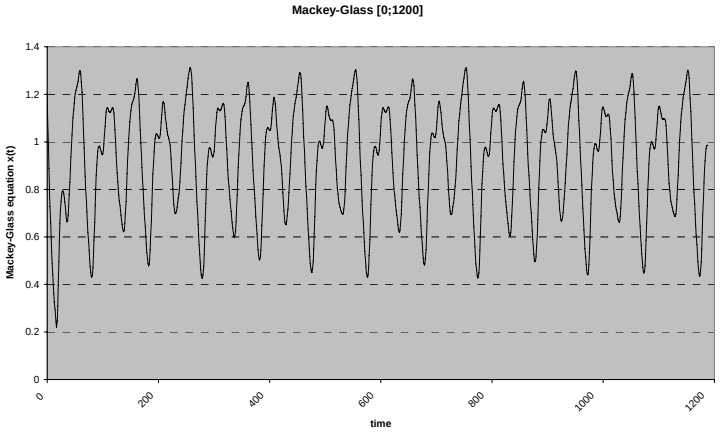
```
sigma=10;
beta=8/3;
rho=28;
x(0)=[-8 8 27];
lorenz(t,x)=[-sigma*x(1) + sigma*x(2); rho*x(1) - x(2) - x(1)*x(3); -beta*x(3) + x(1)*x(2)];
```

Figure 4. Generation of coordinates for Lorenz equation

[1)] Root Mean Squared Error $RMSE = \sqrt{\sum_{t=1}^{N} [\hat{y}_t - y_t]^2}$

4.1.2 Mackey-Glass

The collection consists of 1201 datapoints.



We use for this time series the data from the MATLAB Fuzzy Logic Toolbox, file mgdata.dat. Also for this time series we are not aware of any publication that is sufficient for a comparison as we want to do. The initial coordinates used in mgdata are x(0)=1.2 and x(t)=0 for t<0 and the parameter τ=17 for the Mackey-Glass time delay differential equation:

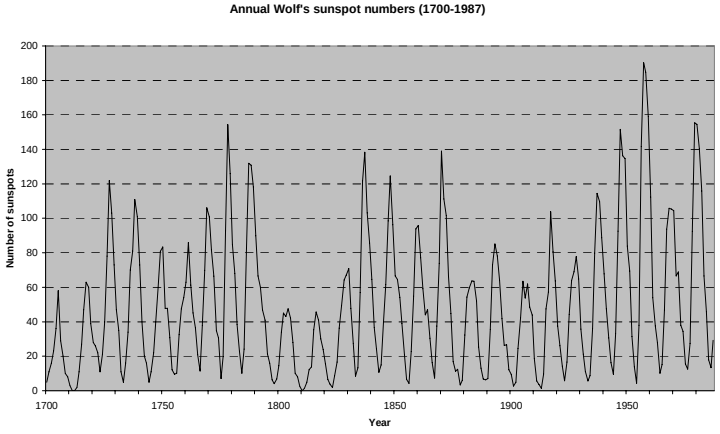
$$\dot{x}(t) = \frac{0.2\,x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1\,x(t) \qquad (10)$$

The data has been split in ratio 1:1: the training data consists of 601 data points and the test data consists of 600 data points.

4.2 Natural phenomena

4.2.1 Annual Wolf's sunspot numbers

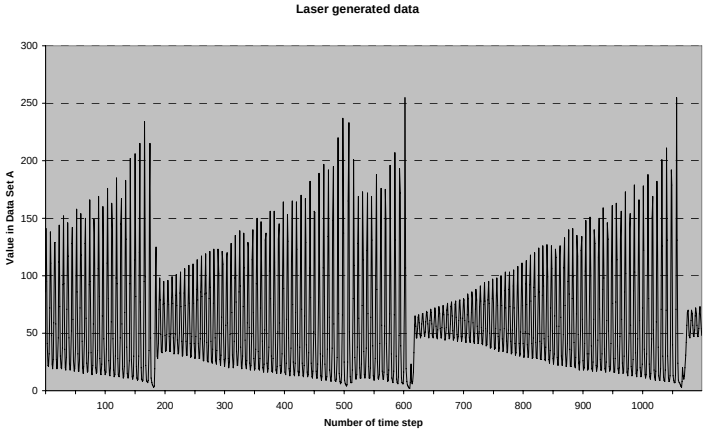
The collection consists of 288 datapoints.



The origin of the data is to find at *http://sidc.oma.be/DATA/yearssn.dat*. Authors of [6] investigated this time series already and provided information about the MSE and the exact data sets. To be able to use the results in [6] for comparison we are using the same data and the same split of data sets: the training data consists of 221 data points (1700-1920) and the test data consists of 67 data points (1921-1987).

4.2.2 Laser generated data

The collection consists of 1100 datapoints.

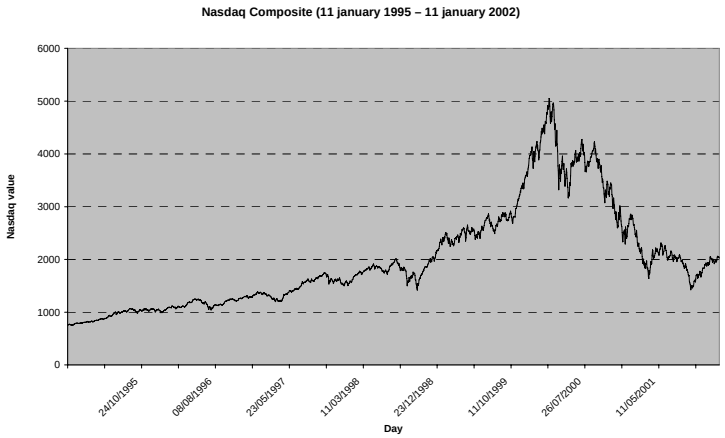


For this experiment we used data *from http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe/A.dat* and from *http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe/A.cont*. [48] provides an RMSE of Support Vector Regression and Multi-scale Support Vector Regression. For that reason we take over the exact data points and the split. The training set consists of 1000 records and the test set consists of 100 data points.

4.3 Financial problems, stock price indices

4.3.1 Daily stock closing price of Nasdaq

The collection consists of 1765 datapoints.

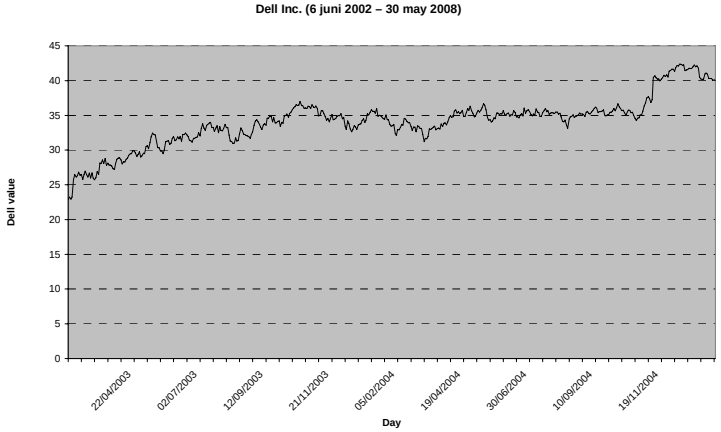


Data stem from *http://ichart.finance.yahoo.com/table.csv?s=%5EIXIC&a=00&b=11&c=1995&d=00&e=11&f=2002 &g=d&ignore=.csv*. The time series of Nasdaq stock index has been investigated before in [1] and [9]. Authors in [9] used an amount of 8428 data points though, they used in addition to training and test data also data for validation, and the time series was normalized. The authors in [1] used 1765 records (11 january 1995 – 11 january 2002), the ratio of training set and test set was 1:1 (883 records for training and 882 records for testing). This publication provides results for the RMSE of several algorithms. However, the RMSE value appears to be calculated differently to most other

publications. Additionally, the forecast models are based on opening, closing and maximum values of the index while we want to use the daily closing price only. We use the sample composition from [1] also when there is not really the opportunity for a comparision of the results.

4.3.2 Daily stock closing price of Dell

The collection consists of 492 datapoints.



For this experiment we used data from *http://ichart.finance.yahoo.com/table.csv?s=DELL&a=01&b=10&c=2003&d=00&e=21&f=2005&g=d&ignore=.csv*. In [44] is an Mean Absolute Percentage Error (MAPE) documented. We use the same datasets: 400 data points for the training set (10 february 2003 – 10 september 2004) and 92 ones for the test set (13 september 2004 – 21 january 2005).

5 Application of the new model to the test data

This chapter contains the results that we obtained by the application of our new, ES based model to the test data. To be able to evaluate the quality of these results we document on top of that results in forecasting the same time series with two classical methods alone: ARMA and a neural network. Information about results of previous research is integrated if available. The overall comparison is presented in tables and shows how the new model performs in the forecasts. Plots of training set and test set can be found in the appendix Diagrams.

5.1 Classical forecasting methods for comparison

To achieve data for a benchmark we performed forecasts with all time series using ARMA models and Elman networks. For both methods we did a sequential search for the best model. In case of the ARMA model we set the boundaries for the autoregressive part to [2..10] and for the moving average part to [0..10]. The coefficients for the ARMA model are calculated by the MATLAB function *ARMAX*. For searching the best ARMA model we also used the optional possibility *FixedParameter* of this function to exclude determined lags. In case of the Elman network we sequentially searched with learning rates between 10^{-5} and 1.0. The boundaries for the number of units in the hidden layer we defined in the range [1..10] and the number of additional lags of the context units was limited to 0 and 1. The network was trained by backpropagation.

5.2 Computer generated time series

5.2.1 Lorenz

The ES performed a good forecast of the Lorenz time series. While the best ARMA model was superior for modelling the the training data, the CMA-ES performed slightly, but consistently best for the test data sets (see *Table 1*). During our experiments it occured solely in case of the Lorenz time series that different models turned out to be the respective best models for a certain part of our examination. For every other time series we researched and documented the results for in this chapter we are able to determine a certain model performing best over all areas of the examination.

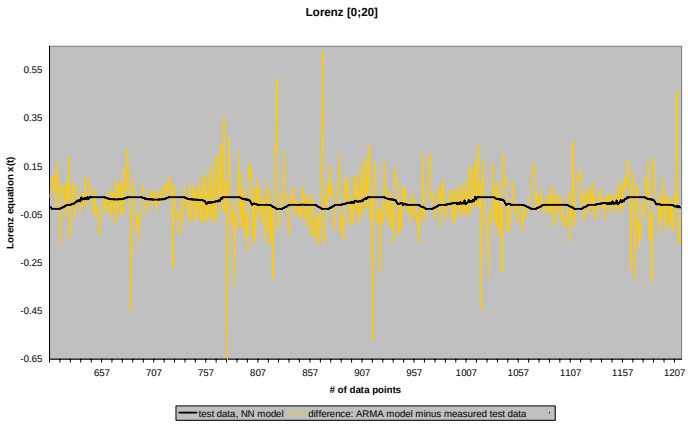


Figure 5. Neural network part compensates deficiencies of the ARMA part

The meta algorithm had chosen an ARMA(6,8)-model and a neural network with 8 hidden nodes and 2 additional lags of the context units in the last generation. The coefficient c_{ARMA} was calculated with 1.0017, c_{NN} = -0.0021 (see equation *(5)*). On its own the ARMA(6,8)-model performs with a BIC = -2887 and an RMSE = 0.1080 for 606 points ahead. The BIC of -2887 is better than our reference of the best ARMA model (BIC = -2879). A large amount of coefficients (9 out of 14) of the *ARMAX* model had the same mathematical sign as the ES calculated ones. The BIC of the chosen neural network alone was 1123, the RMSE = 2.4040. This result is considerably worse than the reference model for a neural network. However, the neural network contributes a partial time series with a slight phase difference and a negative coefficient c_{NN} . The value of c_{NN} is very small. Apparantly the neural network is able to fine tune and improve areas where the ARMA part contributes to the ES model with a less correct time series (see *Figure 5*) The RMSE is for out-of-sample forecasts improved. We will come back to this characteristic result in paragraph 5.3.1.

The chosen ARMA and neural network models performed as part of the combined ES model better than if modelled by classical methods (see *Table 1*). This result indicates that both low level algorithms are capable to optimize the parameters of the component models very well.

Model	training data		test data	
	BIC	RMSE	RMSE, forecast 303 points ahead	RMSE, forecast 606 points ahead
ARMA model: 6 10 {4 11} [1)]	**-2879**	**0.0865**	0.1125	0.1111
NN model: 4 0 {0.001} [2)]	-651	0.5464	0.6341	0.6083
CMA-ES model: equation *(5),* sigma 0.4 consisting of:	-2595	0.0868	**0.1124**	**0.1076**
1.0017 *ARMA model 6 8	-2887	0.0867	0.1129	0.1080
-0.0021 * NN model 8 2	1123	2.0350	2.4791	2.4040
using *ARMAX* and *nnt2elm*: [3)]				
ARMA model: 6 8	-2811	0.0923	0.1174	0.1162
NN model: 8 2	1169	2.1154	2.5763	2.4264

Table 1. Forecast accuracy comparison Lorenz

5.2.2 Mackey-Glass

The results for training set and test set of the Mackey Glass time series are shown in *Table 2*. Our algorithm based on ES was able to find models to reproduce the training data and to perform the forecast. It produced in the last generation of the meta algorithm an ARMA(4,10)-model and an NN model with 9 hidden units and 3 additional lags of the context units. Further we got coefficients

---

[1)] notation of best ARMA model: " *p q {FixedParameter}*"

[2)] notation of best neural network: "number of *s m {learning rate}*"

[3)] the component models evolved by CMA-ES are calculated for comparison reasons by classical methods, too (using the MATLAB functions *ARMAX* and *nnt2elm)*

c_{ARMA} = 0.3092 and c_{NN} = 0.6773. The accuracy of the forecast was considerably worse than with the best ARMA model and still worse than with the best neural network. A few reflections can lead to possible reasons why the ES performed less successful than the other models.

The ARMA(4,10)-model performs with a BIC = -6801 and an RMSE = 0.0007 for 600 points ahead when it works on its own and using the MATLAB function *ARMAX*. Within the ES algorithm we get a BIC = 7365 and an RMSE = 0.0018. These results indicate that the choice of the ARMA model by the meta algorithm is acceptable. Although we find big differences in the achieved coefficients – the majority of coefficients calculated by the ES does have opposite signs to the ones calculated by *ARMAX* – might the ES optimize all ARMA coefficients in our experiments suffiently.

The neural network model chosen by the ES calculates a BIC = - 3176 and an RMSE = 0.0677 for 600 points ahead. This result is away from the best NN model and only acceptable in the frame of pareto optimal choice if the whole ES model would perform better than it did. It appears to be obvious that there are shortcomings in the choice of the neural network model. The question is where they can have their origins. Already with using classical backpropagation the training process was to recognize as very dependent on initial values of the parameters, especially on the initial learning rate. Therefore we kept the initial values the same for every experiment with the classical method. The low level algorithm for the neural network though creates for every generation of the meta algorithm a new set of parameters that are subject to optimization. Additionally, it receives the initial step size from the meta algorithm. The complexity of optimization increases especially in cases where the number of additional lags in the context units grows. In [4] the author points out that exactly in those situations the ES turns out to be a less reliable method for training than classical ones and that "... the 'correct' parametrization gets crucial".

Model	training data		test data	
	BIC	RMSE	RMSE, forecast 300 points ahead	RMSE, forecast 600 points ahead
ARMA model: 10 10 {10 13 14 16 17}	**-7556**	**0.0017**	**0.0013**	**0.0013**
NN model: 9 0 {0.6}	-4568	0.0193	0.0638	0.0462
CMA-ES model: equation *(5),* sigma 0.4 consisting of:	-6100	0.0539	0.0656	0.0505
0.3092 * ARMA model 4 10	-7365	0.0020	0.0018	0.0018
0.6773 * NN model 9 3	-3176	0.0532	0.0916	0.0677
using *ARMAX* and *nnt2elm*:				
ARMA model: 4 10	-6801	0.0033	0.0007	0.0007
NN model: 9 3	-2040	0.1371	0.1497	0.1403

Table 2. Forecast accuracy comparison Mackey-Glass

5.3 Natural phenomena

5.3.1 Annual Wolf's sunspot numbers

For the sunspot time series the ES performed excellent. It achieved the best result (*Table 3*). The meta algorithm had chosen an ARMA(2,9)-model and a neural network with 2 hidden nodes and 3 additional lags of the context units. c_{ARMA} was calculated with 1.0751, c_{NN} = -0.0443.

On its own the ARMA(2,9)-model performs with a BIC of 896 and an RMSE = 9.0784 for 67 points ahead. The BIC of 896 is better than our reference of the best ARMA model (BIC = 1215). The coefficients agreed in the majority of the mathematical signs for *ARMAX* and those obtained by the ES. The BIC of the chosen neural network alone was 1520, the RMSE = 45.6115. This model already deviates much from the best competitors. Nevertheless it contributes to the whole ES model in such a way that the performance of the latter improves for the test data set. In this case we would dare to speak of a pareto optimal choice. It is remarkable that the calculated values $y_{t\,NN}$ are opposite to the values of the actual time series (*Figure 6*). Thus, the neural network appears to compensate deficiencies of the ARMA part as a "leading" method. The negative sign of the coefficient c_{NN} and the very small value of c_{NN} suggest this idea as well.

Model	training data		test data	
	BIC	RMSE	RMSE, forecast 35 points ahead	RMSE, forecast 67 points ahead
Heuristic EA [10]	1377	n. d.[1)]	n. d.	17.9 ± 0.0
Meta-EA [10]	1362	n. d.	n. d.	17.6 ± 0.2
Holt-Winters [10]	n. d.	n. d.	n. d.	28.3
Box-Jenkins [10]	n. d.	n. d.	n. d.	21.4
ARIMA [6]	n. d.	n. d.	14.730[2)]	17.495[2)]
Artificial NN [6]	n. d.	n. d.	14.328[2)]	18.740[2)]
Hybrid [6]	n. d.	n. d.	13.668[2)]	14.428[2)]
ARMA model: 8 5 {3 10}	1215	13.5598	14.1640	16.1655
NN model: 2 0 {0.2}	1360	20.0028	21.2189	28.1384
CMA-ES model: equation *(5),* sigma 0.8 consisting of:	**993**	**6.8059**	**9.4494**	**9.0064**
1.0751 * ARMA model: 2 9	896	6.7341	9.9866	9.0784
-0.0443 * NN model: 2 3	1520	26.7948	34.7796	45.6115
using *ARMAX* and *nnt2elm*:				
ARMA model: 2 9	1249	14.9578	15.7133	19.9696
NN model: 2 3	1815	52.3600	59.0872	69.8733

Table 3. Forecast accuracy comparison sunspot numbers

[1)] not documented

[2)] original documented as MSE

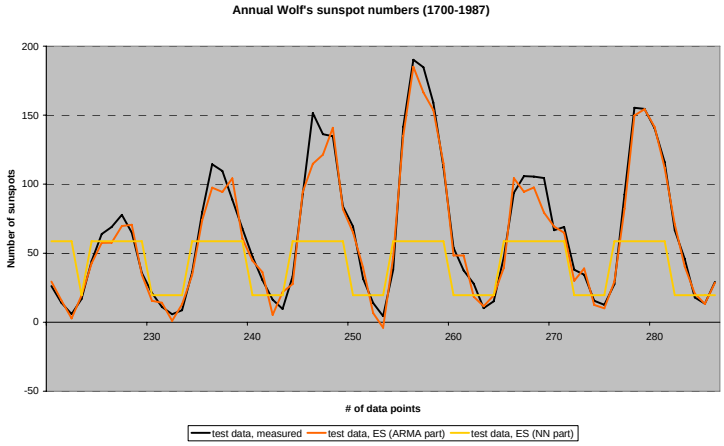


Figure 6. Separate forecasts of the ARMA and NN part, last generation of meta algorithm

In contrast to the Mackey-Glass time series the BICs of the best ARMA and the best neural network model respectively can be found in the same range. This fact indicates that both methods can model the sunspot series in a similar quality. The ES seems to be less vulnerable with regard to the problems we discussed in paragraph 5.2.1. Althought we recognize that both ES components optimize the parameters of the same model already better than if modelled by classical methods, we consider it to be possible that a further revision of the two low level algorithms with regard to fitness function and initial states of the parameters can mean another improvement to the forecast of the series. The experiment with the sunspot time series demonstrates that the ES is capable to use the respective advantages of either of the two methods to perform very well. However, the ES carefully calculates the influence of the neural network part as we have noticed with the proportionally very small coefficient for this part.

5.3.2 Laser generated data

The most critical region for the forecast of this time series turned out to be the region betweeen data point 1060 and 1080 (see *Figure 7*). The capability to model this region reflects the accuracy to forecast the whole series of the test data. Our ES model performed slightly better than the ARMA model, however, the neural network and especially the support vector regressions were superior to these both (*Table 4*).

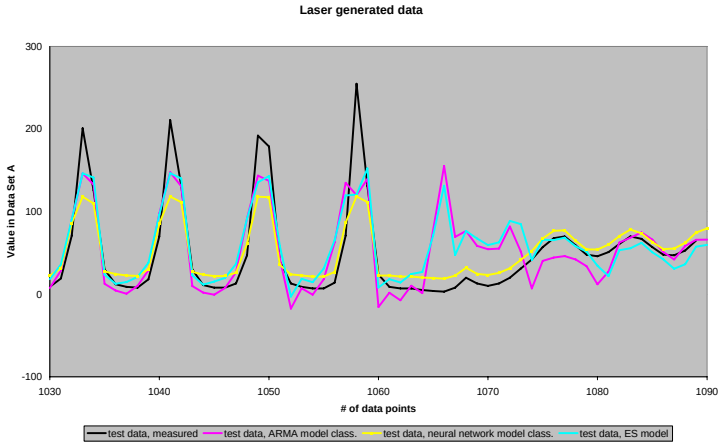


Figure 7. Comparison forecasting results of ARMA, NN, ES model

The ES calculated for the ARMA part a coefficient c_{ARMA} = 0.8634 and for the neural network part c_{NN} = 0.2027. For forecasting the test data we got an ARMA(7,9)-model (BIC = 5980, RMSE = 32.0834 for 100 points ahead) and a neural network with 8 hidden nodes and 2 additional lags. The coefficients of the ARMA model do almost completely have the same mathematical signs as for the coefficients calculated by *ARMAX*. Noticeable for the result of the neural network is that the model chosen by ES contributes a sort of base load and the graph is slightly out of phase in comparison to the original time series. These characteristics are similar to what we found for the sunspot series and the Lorenz series as well and modestly for Mackey-Glass, too. The combined ES model performs with regard to the RMSE for out-of-sample forecasts better than its best component (*Figure 8*).

As for the sunspot series the BICs of the best ARMA and neural network model can be found in the same range. Differently to the experiments with the sunspot series the ES does not improve the results of either of them. A common BIC can be therefore excluded to be the reason for the excellent performance of the ES that we had seen in paragraph 5.3.1. What could be the a reason for this result? Do ARMA model and neural network contribute differently to the ES model in the case of the laser generated series? Indeed. For the laser generated series the neural network performs better than the ARMA method in terms of the RMSE. Possible weaknesses of the neural network part in the ES should have a larger negative influence on the total result since the neural network actually should contribute a larger part. Also the results of [48] suggest that the neural network appears to be the more suitable model for this time series. Support vector regression is said to be even better in performance for the laser generated data than the usually used radial basis function neural networks [48]. An additional reason is the fact that the best ES component does not perform that extremely better than the best reference model as we have seen in case of the sunspot series.

Model	training data		test data	
	BIC	RMSE	RMSE, forecast 50 points ahead	RMSE, forecast 100 points ahead
SVR [48]	n. d.	n. d.	n. d.	5.80[1)]
MS-SVR [48]	n. d.	n. d.	n. d.	**3.92**[1)]
ARMA model: 8 10 {15 18}	5980	18.7882	21.2427	31.8554
NN model: 1 0 {0.7}	5932	19.1614	30.9614	26.8748
CMA-ES model: equation *(5),* sigma 0.4 consisting of:	6186	17.9206	21.5228	30.5317
0.8634 * ARMA model: 7 9	5980	19.0358	21.6969	32.0834
0.2027 * NN model: 8 2	6886	27.1842	40.6424	35.9125
using *ARMAX* and *nnt2elm*:				
ARMA model: 7 9	6213	21.2442	24.1671	37.6452
NN model: 8 2	8912	74.8662	90.4219	76.7956

Table 4. Forecast accuracy comparison laser generated data

[1)] original documented as RMSE using preprocessed data points $\hat{y}_t = \frac{\hat{y}_t}{100}$

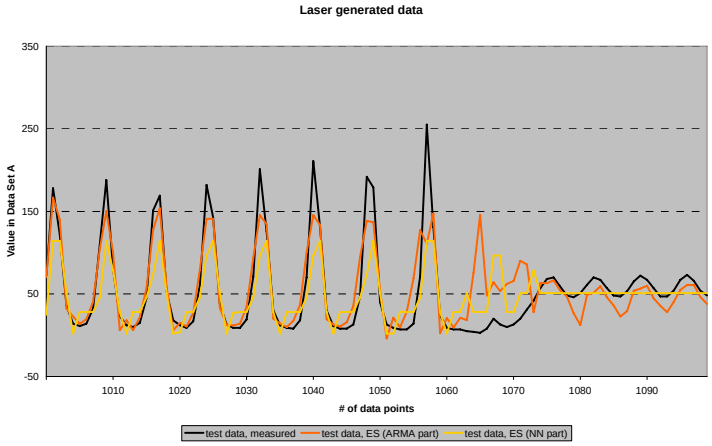


Figure 8. Separate forecasts of the ARMA and NN part, last generation of meta algorithm

5.4 Financial problems, stock price indices

5.4.1 Daily stock closing price of Nasdaq

For the forecast of the Nasdaq time series the RMSE of the best ARMA model was slightly better than the one of our ES model. The neural network proved to be completely unsuitable (*Table 5*). The results of our experiments with the ES confirmed this as well. The coefficient for the neural network part was very small (c_{NN} = 0.0105) and the ARMA part did the main contribution to the y_t values of the whole model (c_{ARMA} = 0.9940).

Model	training data		test data	
	BIC	RMSE	RMSE, forecast 442 points ahead	RMSE, forecast 883 points ahead
ARMA model: 9 10 {2 3 6 9 10 11 12 13 14 18}	**4682**	**13.3041**	**58.5468**	**69.5685**
NN model: 6 1 {0.6}	10285	309.8101	1705.1664	1668.0966
CMA-ES model: equation *(5),* sigma 0.4 consisting of:	4970	14.2376	60.4490	72.8601
0.9940 * ARMA model: 2 8	4698	13.8303	60.1338	72.9281
0.0105 * NN model: 7 1	12600	1132.9690	2695.0953	2660.2634
using *ARMAX* and *nnt2elm*:				
ARMA model: 2 8	4700	13.8428	59.9220	73.3556
NN model: 7 1	10313	309.8783	1697.5595	1661.5403

Table 5. Forecast accuracy comparison Nasdaq

The last generation of the meta algorithm had chosen an ARMA(2,8)-model and a neural network with 7 hidden units and 1 additional lag of the context units. The ARMA(2,8)-model performs on its own with BIC = 4698 and an RMSE = 72.9281 for 883 points ahead. The neural network has a BIC = 12600 and an RMSE = 2660.2634. It provides simply an almost constant value of the time series to the whole model. However, the ES was still capable to achieve a forecast that was better in RMSE than the ARMA part that is contributing to. This might be an indication that the neural network is not

completely useless for our ES model – even when we had in this case expected that the algorithm mainly models an ARMA behaviour only. Theoretically, the ES should find our best reference model, i.e. the ARMA(9,10). Actually, the ES already found an ARMA model within the first mutated population that was better than the reference model (ARMA(5,9) with a BIC = 4567). We assume that we face here a negative influence on the dynamic behaviour of the algorithm because the optimum is situated very close to a constraint of the ES. A second possible option is that the inital values of the coefficients c_{ARMA} and c_{NN} are too large; we had chosen for coefficients with normally distribution and variance = 2. However, we could not find evidence that the additional constraint $c_{ARMA} + c_{NN} = 1$ would improve the combined ES model; the algorithm reached sooner convergence than without constraint at a BIC between 5150 and 5200.

Taking a look at the quality of parameter estimation of the components of the combined ES it becomes apparent that both of them perform worse than the classical methods do for the same model. Since the coefficients are optimized in the low level algorithm we are tempted to question the BIC as fitness function. It appears to be necessary to use the RMSE or another representation for the modelling error for the optimization of the coefficients. This can be supported by the fact that the combination of different fitness functions (BIC in meta algorithm and RMSE in a low level optimization) has been successfully used before in [10].

5.4.2 Daily stock closing price of Dell

Our experiments with the time series of Dell's stockprices showed that the ES created a suitable optimization. However, in comparison to other models – also to models published by other authors – it could not reach the best models with regard to the forecast accuracy (*Table 6*). As already for the time series in the previous paragraph the neural network was less suitable in general. Again, this issue is also visible for the ES model. We got in the last generation of the meta algorithm an ARMA(6,8)-model and a neural network with 7 hidden nodes and 1 additional time lag. The coefficient for the ARMA part was 0.9344, for the neural network one 0.1084. When we compare the ES optimized coefficients of the ARMA model with the MATLAB function ARMAX we find 10 out of 14 coefficients having an opposite sign.

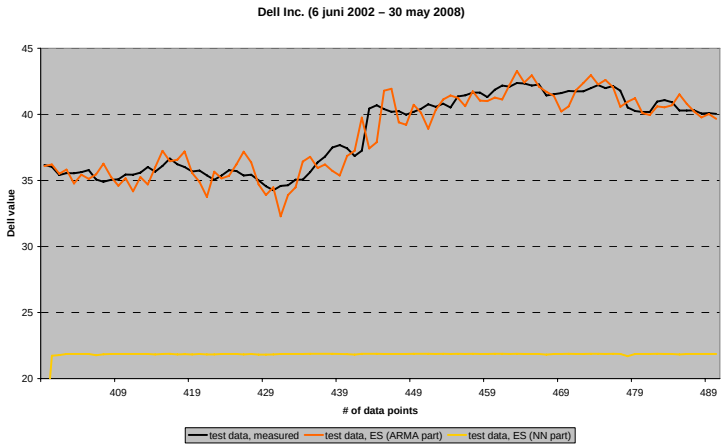


Figure 9. Separate forecasts of the ARMA and NN part, last generation of meta algorithm

Model	training data		test data	
	BIC	RMSE	RMSE, forecast 50 points ahead	RMSE/MAPE[1], forecast 92 [2] points ahead
Hidden Markov Model followed by interpolation [44]	n. d.	n. d.	n. d.	n. d./1.0117
Fusion model (Artificial NN, GA, Hidden Markov Model with weighted average) [44]	**n. d.**	**n. d.**	**n. d.**	**n. d./0.6992**
ARMA model: 7 8 {9 12}	-566	0.4454	0.5672	0.4964/0.8307
NN model: 9 0 {0.0001}	28	0.8416	2.1403	4.4140/8.9288
CMA-ES model: equation *(5),* sigma 1.0 consisting of:	146	0.8505	1.1270	0.9862/1.9830
0.9344 * ARMA model: 6 8	-45	0.8820	1.1624	0.9707/1.8965
0.1084 * NN model: 7 1	2.133	11.6093	14.6830	16.9834/43.1755
using *ARMAX* and *nnt2elm*:				
ARMA model: 6 8	-535	0.4660	0.5858	0.5110/0.8248
NN model: 7 1	588	1.6831	3.2651	5.2328/11.4939

Table 6. Forecast accuracy comparison Dell

We assume that the ES attempted a compensation with the relatively large contribution of the neural network part (10%). The neural network seems to provide a sort of a gliding, almost constant value for $y_{t\,NN}$ (*Figure 9*). As in the previous paragraph we also can find the characteristics that ARMA part and neural network part of the combined ES perform an insufficient parameter estimation.

---

[1] Mean Absolute Percentage Error $MAPE = \frac{1}{N_f} \cdot \sum_{t=1}^{N_f} \frac{|\hat{y}_t - y_t|}{\hat{y}_t} \cdot 100$

[2] documented MAPE in [44] contents in fact 91 points only

6 Conclusions

The combined ES model could perform with regard to the RMSE for out-of-sample forecasts better than the classical reference models in 3 out of 6 test cases. One time series – the sunspot time series - could additionally perform with a better accuracy than the reference methods with regard to the training data. Main reasons for the good performance were found in the capability to estimate the parameters of the components very well. The algorithm also consequently combines both components in such a manner that the forecast usually reaches the RMSE of the best component but often even improves it with regard to data in the future.

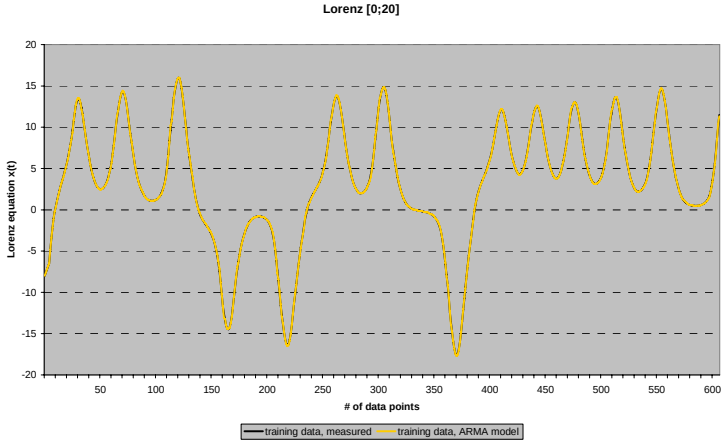
Many questions remain after analyzing the results that we achieved in forecasting several time series with an ES. Has the sunspot time series the smallest number of data points and we are not aware that it avoids overfitting this way? How large is the influence of the initial parameters and starting points of search? What would happen if we prepare a template of a randomized matrix one time in advance and we would use this template for generating every initial population of the low level algorithms instead of creating a randomized matrix for every initial population anew? Differs the issue of initial parameters and starting points in dependence on the time series that is used? Should this matter change if the ratio changes between training data and test data, when the time series is split differently? Can results be changed if there are changes in the fitness function of the low level algorithms and if yes: how would they change? How would results change, if ES is replaced by backpropagation for the training of the neural network and possibly the ES is replaced by ARMAX approximation for the ARMA part? Finally, does the natural origin of the data have an influence on the performance of the ES?

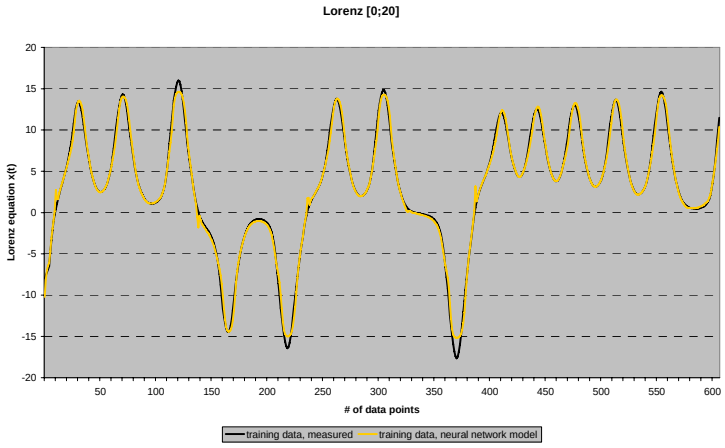
The documented experiments provide evidence that the combined ES model is very well suitable for time series forecasting of natural data and computer generated sequences. For the forecast of financial time series it appears to be less suitable. It remains to explicitely determine and to test the circumstances that the ES can perform on a high level.

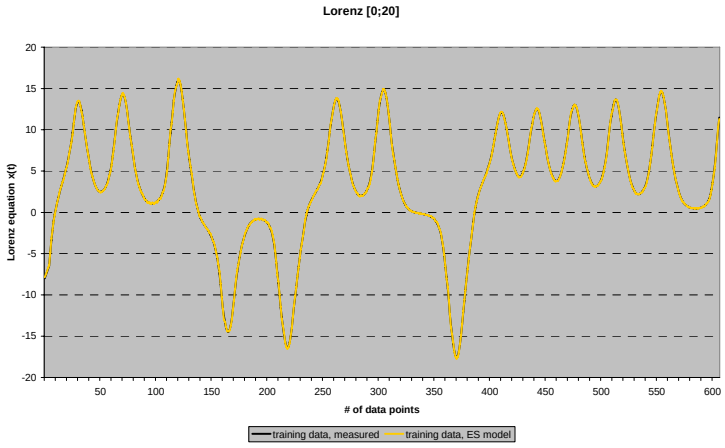
Appendices

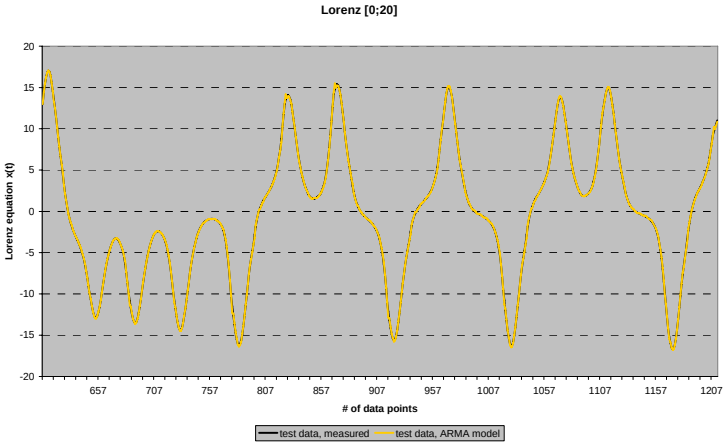
Diagrams

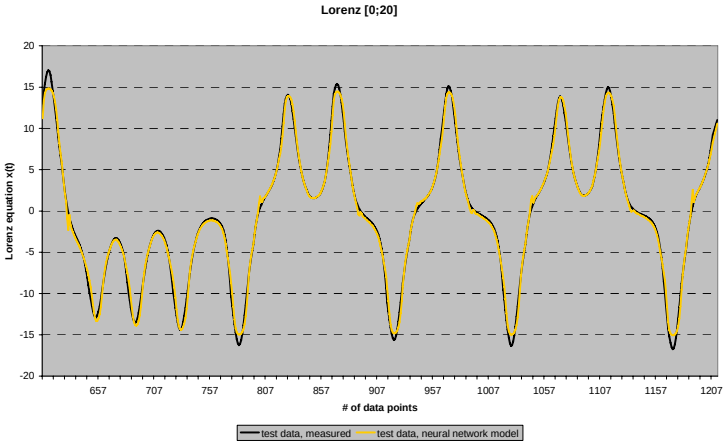
Lorenz

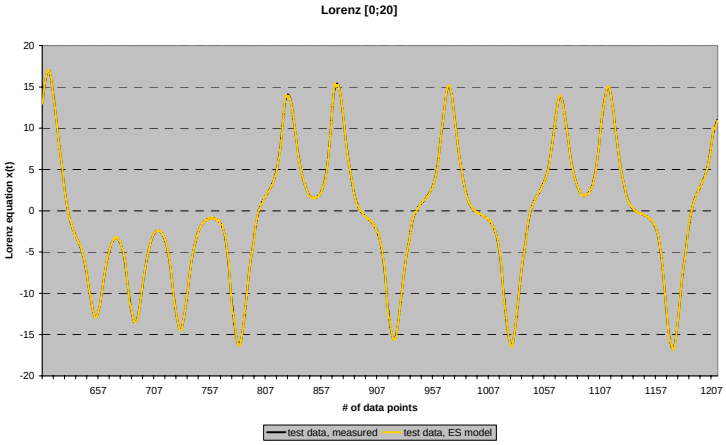




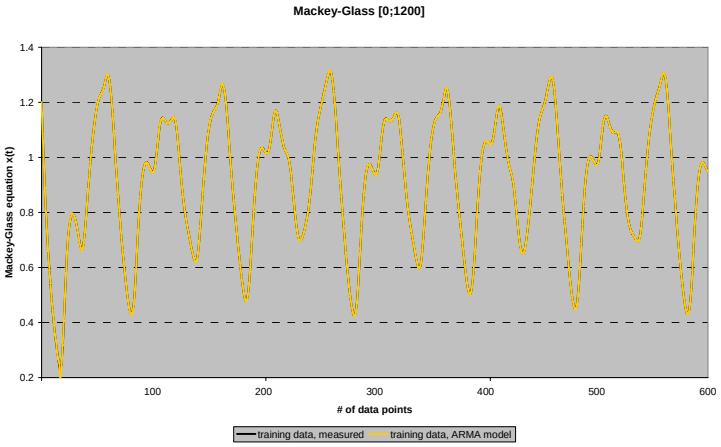


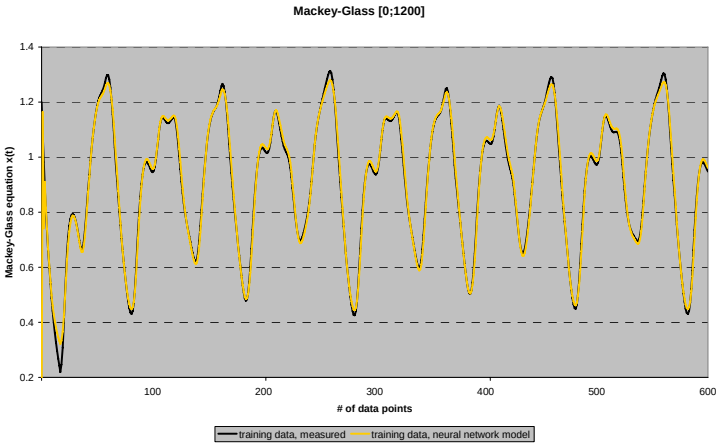

Lorenz [0;20]
Lorenz equation x(t)
20
15
10
5
0
-5
-10
-15
-20
657
707
757
807
857
907
957
1007
1057
1107
1157
1207
# of data points
test data, measured
test data, ARMA model

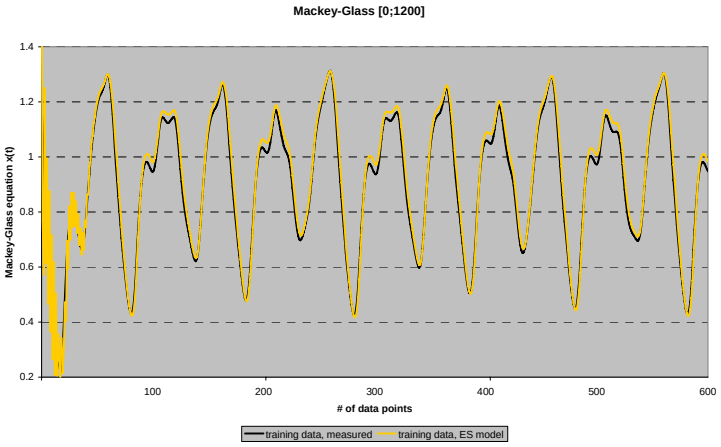

Lorenz [0;20]
Lorenz equation x(t)
20
15
10
5
0
-5
-10
-15
-20
657
707
757
807
857
907
957
1007
1057
1107
1157
1207
# of data points
test data, measured
test data, neural network model

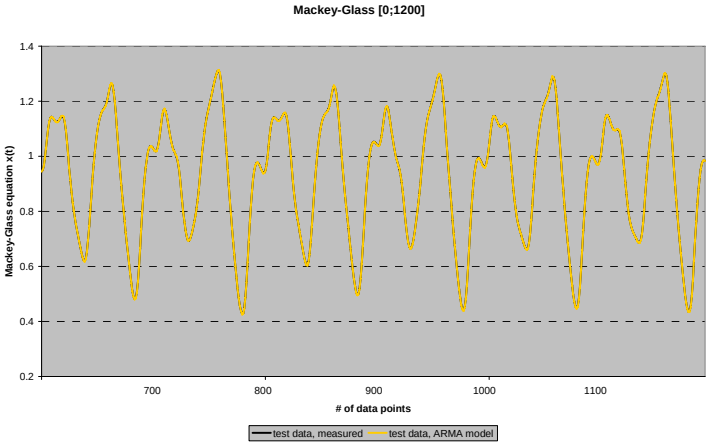

Lorenz [0;20]
Lorenz equation x(t)
20
15
10
5
0
-5
-10
-15
-20
657
707
757
807
857
907
957
1007
1057
1107
1157
1207
# of data points
test data, measured
test data, ES model

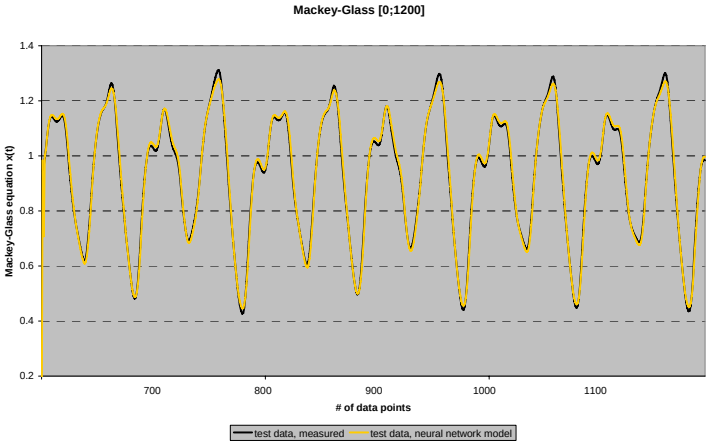
Mackey-Glass

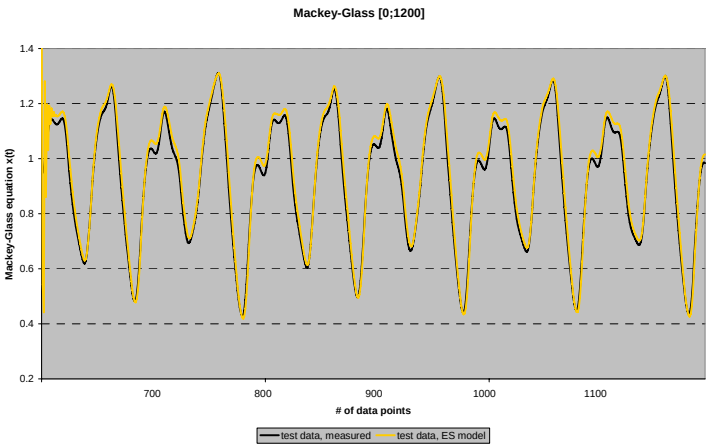




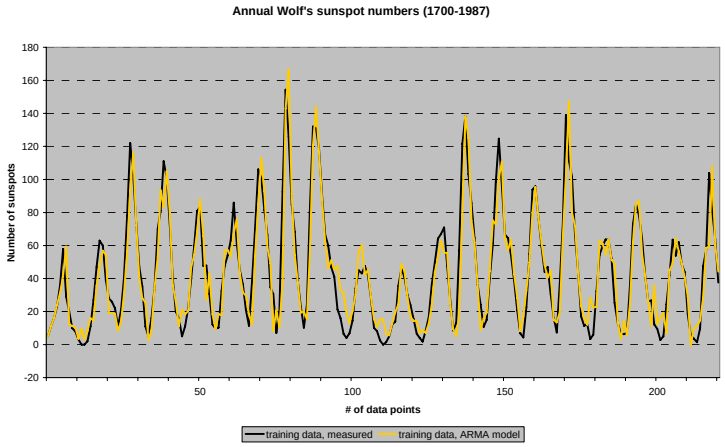


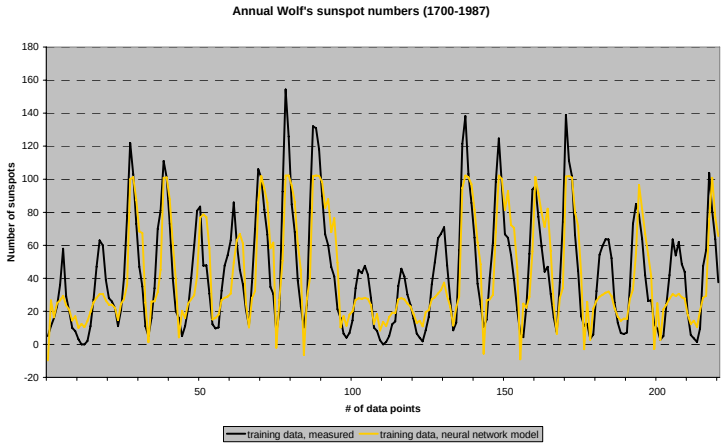

Mackey-Glass [0;1200]
Mackey-Glass equation x(t)
1.4
1.2
1
0.8
0.6
0.4
0.2
700
800
900
1000
1100
# of data points
test data, measured
test data, ARMA model

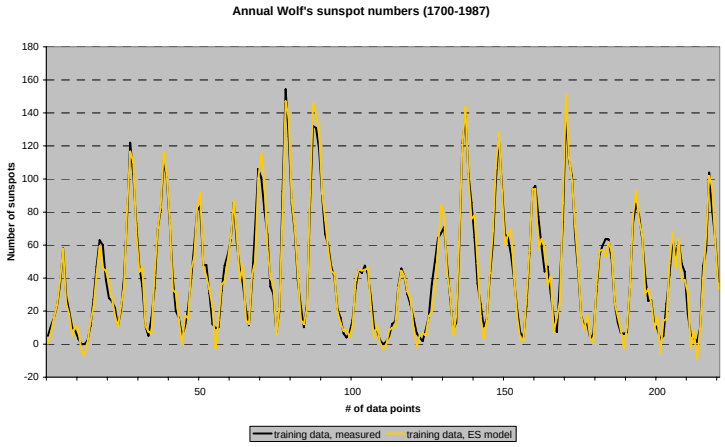

Mackey-Glass [0;1200]
Mackey-Glass equation x(t)
1.4
1.2
1
0.8
0.6
0.4
0.2
700
800
900
1000
1100
# of data points
test data, measured
test data, neural network model

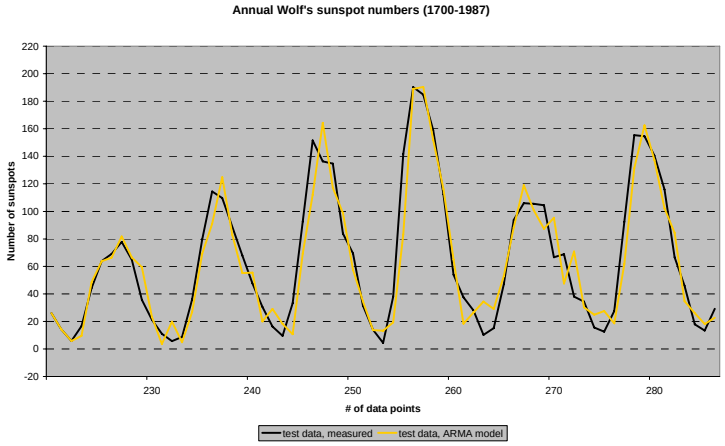

Mackey-Glass [0;1200]
Mackey-Glass equation x(t)
1.4
1.2
1
0.8
0.6
0.4
0.2
700
800
900
1000
1100
# of data points
test data, measured
test data, ES model

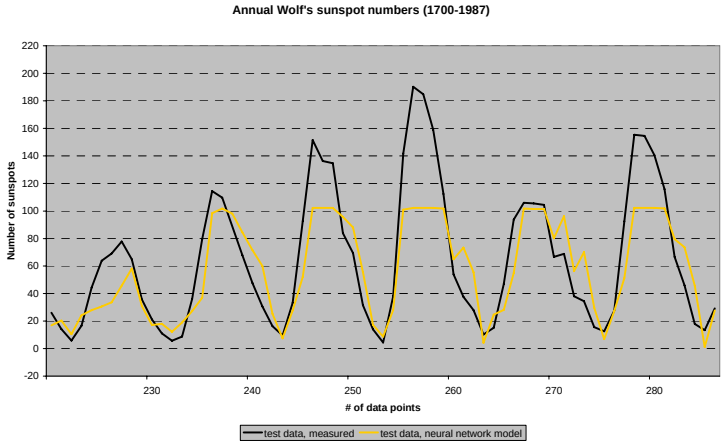
Annual Wolf's sunspot numbers

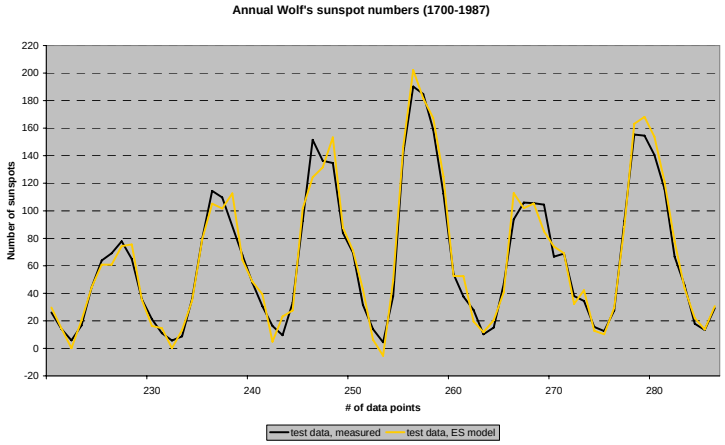




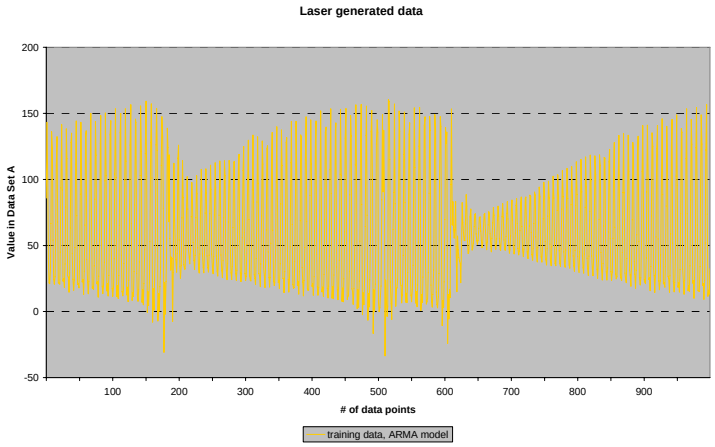


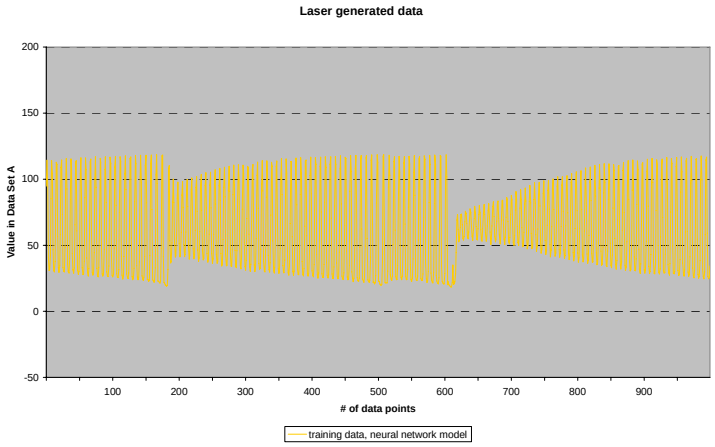

Annual Wolf's sunspot numbers (1700-1987)
Number of sunspots
# of data points
test data, measured
test data, ARMA model

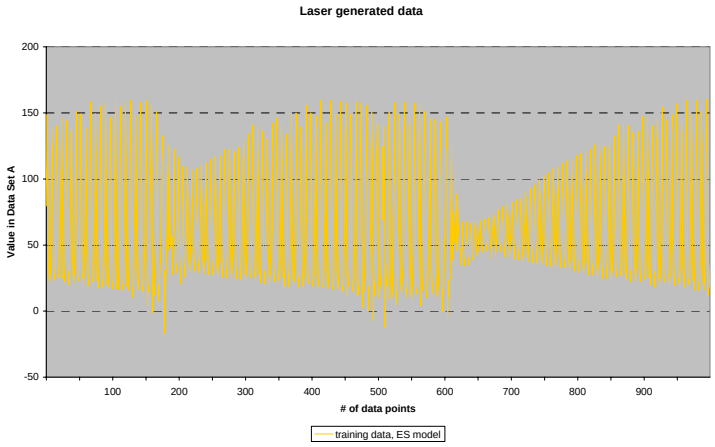

Annual Wolf's sunspot numbers (1700-1987)
Number of sunspots
# of data points
test data, measured
test data, neural network model


Annual Wolf's sunspot numbers (1700-1987)
Number of sunspots
# of data points
test data, measured
test data, ES model

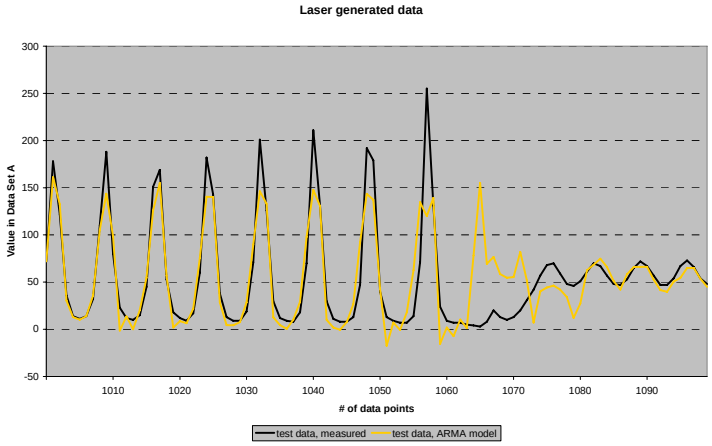
Laser generated data [1)]

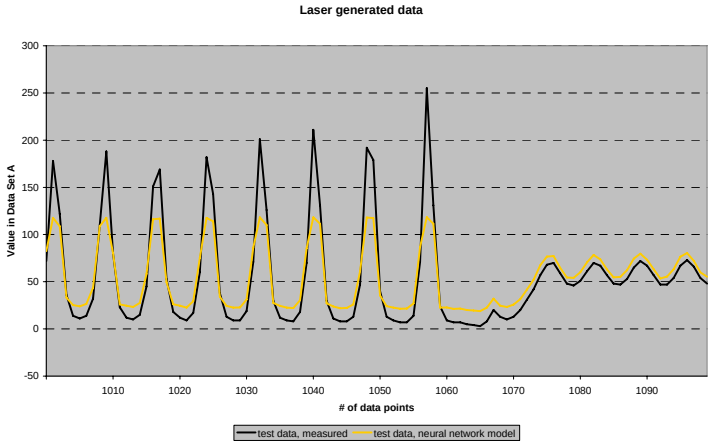


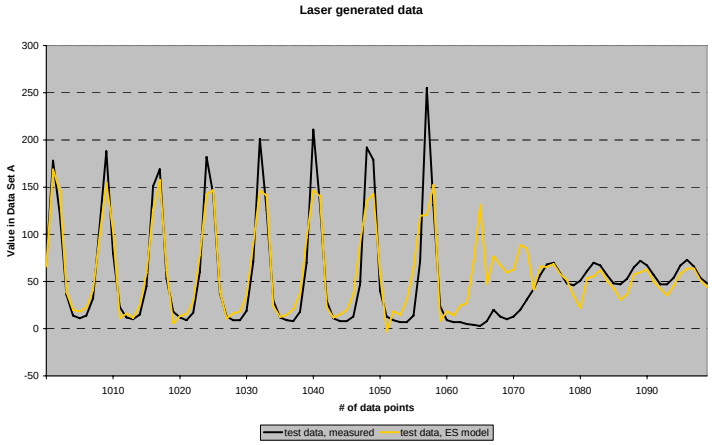




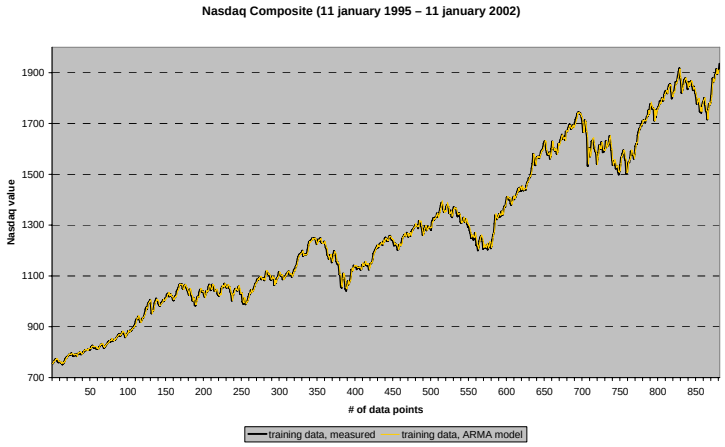
[1)] The measured training data are not included in the first 3 diagrams for the reason of clarity. We refer to paragraph 4.2.2 for a plot of the whole data set.

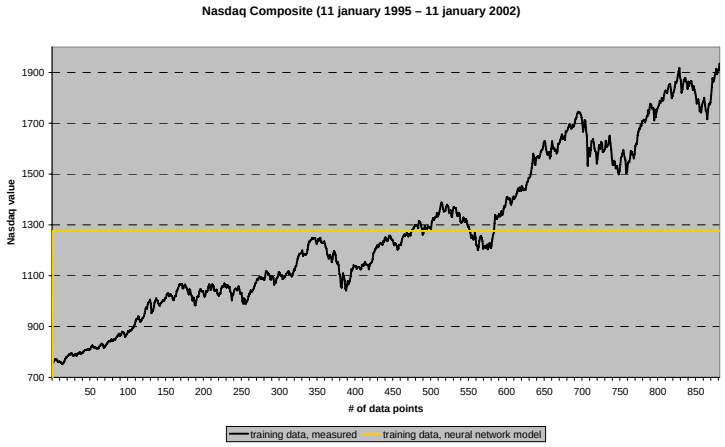

Laser generated data
Value in Data Set A
300
250
200
150
100
50
0
-50
1010
1020
1030
1040
1050
1060
1070
1080
1090
# of data points
test data, measured
test data, ARMA model

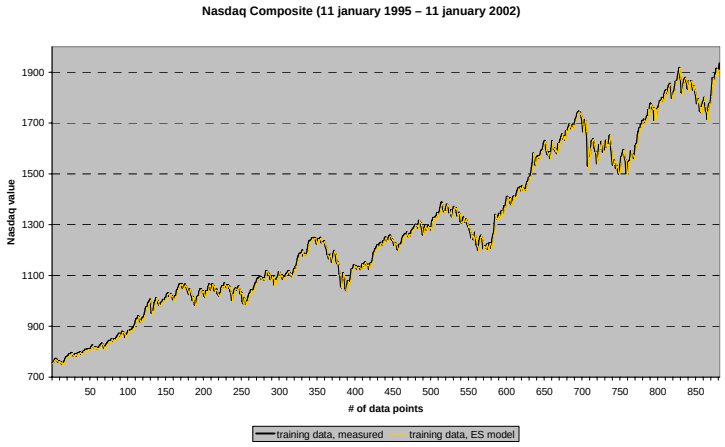

Laser generated data
Value in Data Set A
300
250
200
150
100
50
0
-50
1010
1020
1030
1040
1050
1060
1070
1080
1090
# of data points
test data, measured
test data, neural network model

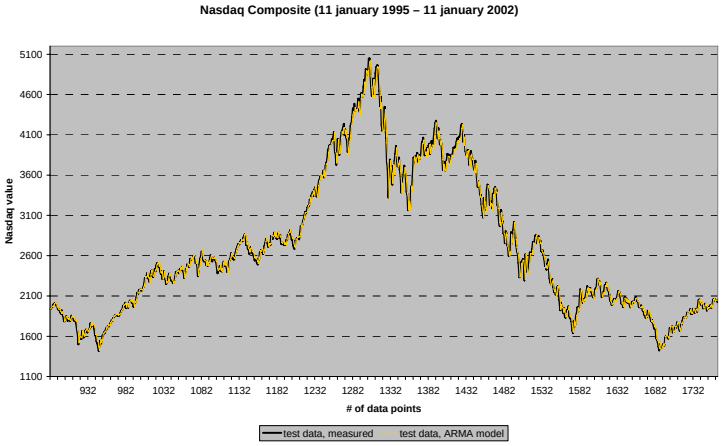

Laser generated data
Value in Data Set A
300
250
200
150
100
50
0
-50
1010
1020
1030
1040
1050
1060
1070
1080
1090
# of data points
test data, measured
test data, ES model

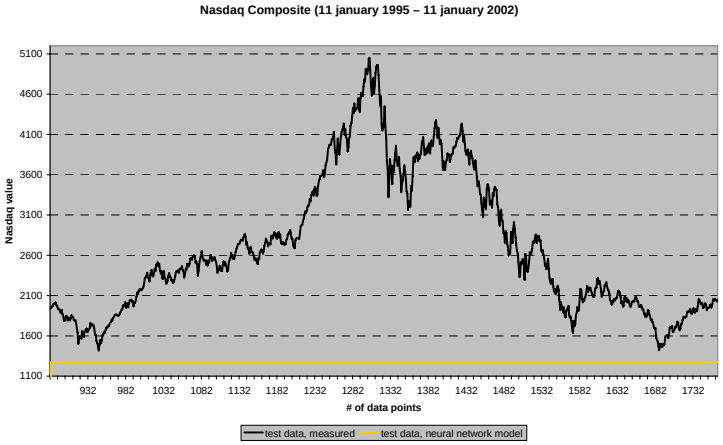
Daily stockclosing price of Nasdaq

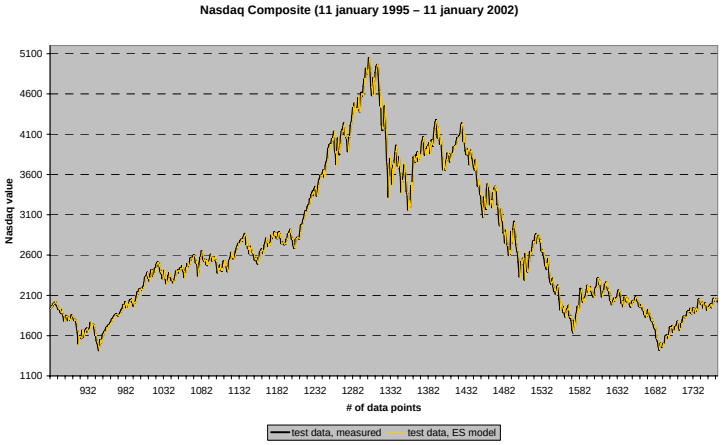




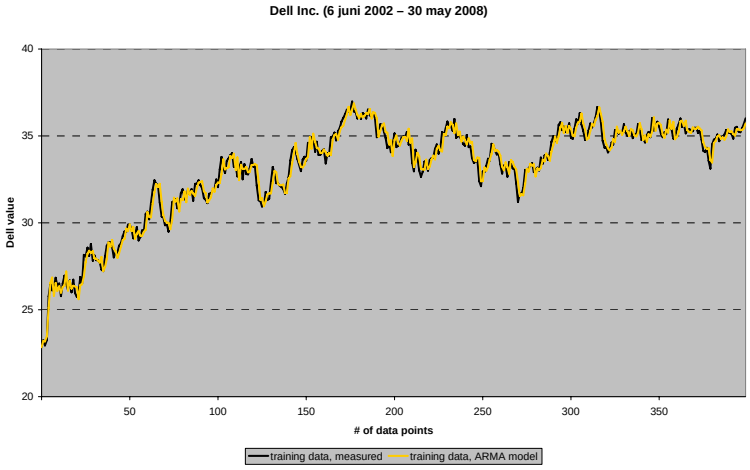


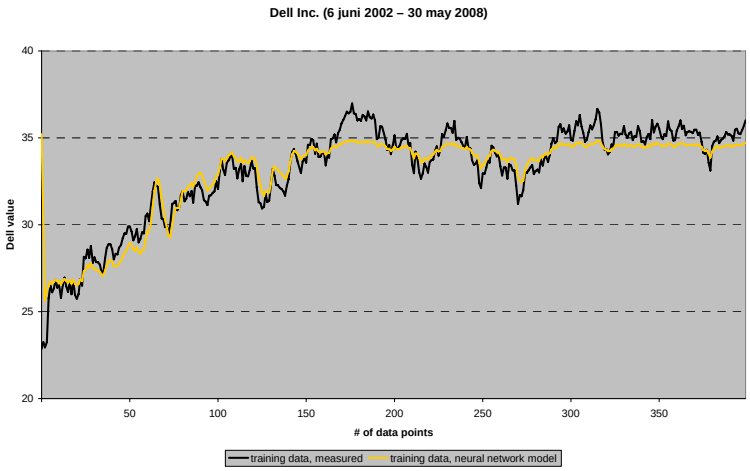

Nasdaq Composite (11 january 1995 – 11 january 2002)
Nasdaq value
5100
4600
4100
3600
3100
2600
2100
1600
1100
932 982 1032 1082 1132 1182 1232 1282 1332 1382 1432 1482 1532 1582 1632 1682 1732
# of data points
test data, measured
test data, ARMA model

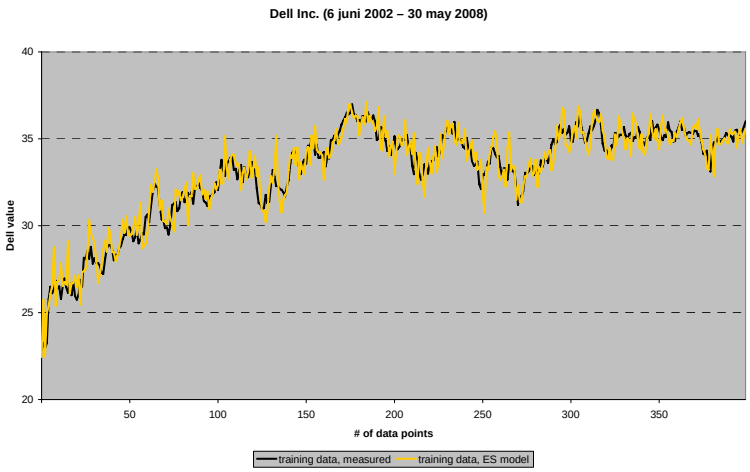

Nasdaq Composite (11 january 1995 – 11 january 2002)
Nasdaq value
5100
4600
4100
3600
3100
2600
2100
1600
1100
932 982 1032 1082 1132 1182 1232 1282 1332 1382 1432 1482 1532 1582 1632 1682 1732
# of data points
test data, measured
test data, neural network model

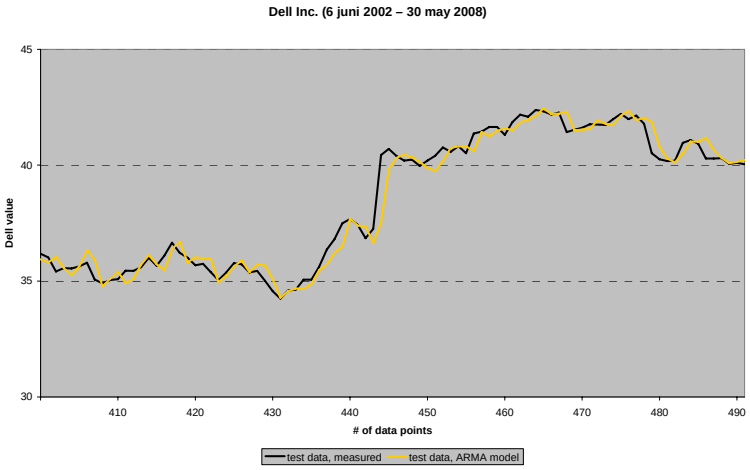

Nasdaq Composite (11 january 1995 – 11 january 2002)
Nasdaq value
5100
4600
4100
3600
3100
2600
2100
1600
1100
932 982 1032 1082 1132 1182 1232 1282 1332 1382 1432 1482 1532 1582 1632 1682 1732
# of data points
test data, measured
test data, ES model

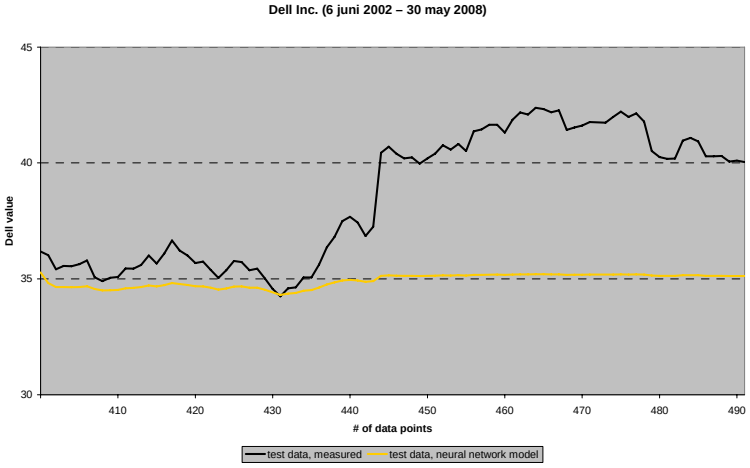
Daily stock closing price of Dell

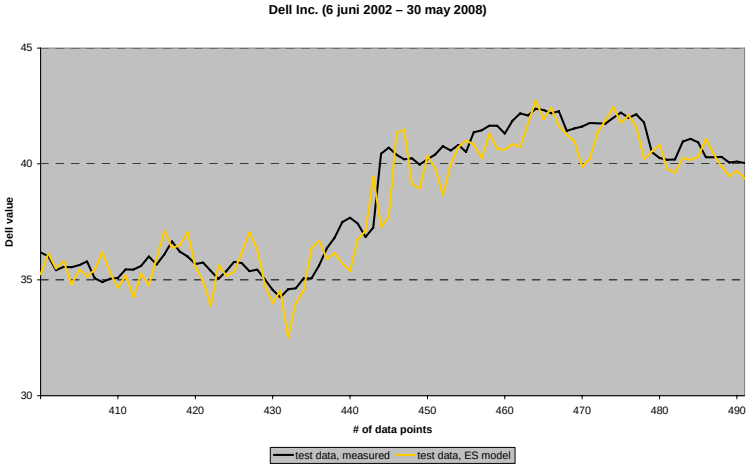







Dell Inc. (6 juni 2002 – 30 may 2008)
Dell value
45
40
35
30
410
420
430
440
450
460
470
480
490
# of data points
test data, measured
test data, ARMA model


Dell Inc. (6 juni 2002 – 30 may 2008)
Dell value
45
40
35
30
410
420
430
440
450
460
470
480
490
# of data points
test data, measured
test data, neural network model


Dell Inc. (6 juni 2002 – 30 may 2008)
Dell value
45
40
35
30
410
420
430
440
450
460
470
480
490
# of data points
test data, measured
test data, ES model

Software

MATLAB code, meta algorithm

```
function [y_cmaes_test]= cmaes_meta(y_measured_training, y_measured_test, nr_records_part, nr_records_total)

% algorithm with ARMA and NN fitness function
% Source pure CMA-ES: [37], N. Hansen, The CMA Evolution Strategy: A Tutorial

% --------------------------- Initialization -------------------------

% input parameters:
% y_measured_training - training set, matrix (#,1)
% y_measured_test - test set, matrix (#,1)
% nr_records_part - number of records for the plot of a part of the test data
% nr_records_total - number of records for the plot of all test data

% User defined input parameters (need to be edited)

strfitnessfct1 = 'cmaes_nn';               % name of objective/fitness function
N_NN = 4;                                  % for NN model architecture: number of hidden nodes S1, sigmaNN for
                                           % training of the model, number of additional input nodes Add, coefficient
                                           % for NN model)
S1 = randint(1,1,[1,10]);                  % objective variables initial point; [0,10] - limits for S1
SigmaNN = 1e-5 + (0.8-1e-5) * rand(1);     % [1e-5, 0.8] - limits for sigma strfitnessfct1
Add = randint(1,1,[0,5]);                  % objective variables initial point; [0,5] - limits for additional input
                                           % nodes
coeffNN = sqrt(2) * randn(1);              % coefficient for NN model: normally distributed with variance 2

strfitnessfct2 = 'cmaes_arma';             % name of objective/fitness function
N_ARMA = 4;                                % for ARMA model architecture: number of p for autoregressive operator
                                           % AR(p), number of q for moving average operator MA(q), sigmaARMA for
                                           % training of the model, coefficient for ARMA model
AR_and_MA = randint(N_ARMA-2,1,[1,10]);    % objective variables initial point; [0,10] - limits for AR and MA
                                           % operator respectively
SigmaARMA = 1e-5 + (0.8-1e-5) * rand(1);   % [1e-5, 0.8] - limits for sigma strfitnessfct2
coeffARMA = sqrt(2) * randn(1);            % coefficient for ARMA model: normally distributed with variance 2

N = N_NN + N_ARMA;
xmean = [S1; SigmaNN; Add; coeffNN; AR_and_MA; SigmaARMA; coeffARMA];

if N ~= size(xmean,1) error('dimension of parameter vector xmean is not correct'); end

sigma = 0.4;                    % coordinate wise standard deviation (step size)

stopfitness = -15000;           % stop if fitness < stopfitness (minimalization)
stopeval = 1000;                % stop after stopeval number of function evaluations

% Strategy parameter setting: Selection
lambda = 4+floor(3*log(N));     % population size, offspring number
mu = floor(lambda/2);           % number of parents/points for recombination
weights = log(mu+1)-log(1:mu)'; % muXone array for weighted recombination
weights = weights/sum(weights); % normalize recombination weights array
mueff = sum(weights)^2/sum(weights.^2); % variance-effective size of mu

% Strategy parameter setting: Adaptation
cc = 4/(N+4);                   % time constant for cumulation for covariance matrix
cs = (mueff+2)/(N+mueff+3);     % t-const for cumulation for sigma control
mucov = mueff;                  % size of mu used for calculaying learning rate ccov
ccov = (1/mucov) * 2/(N+1.4)^2 + (1-1/mucov) * ...  % learning rate for covariance matrix
        ((2*mueff-1)/((N+2)^2+2*mueff));
damps = 1+2*max(0,sqrt((mueff-1)/(N+1))-1) + cs;    % damping for sigma

% Initialize dynamic (internal) strategy parameters and constants
pc = zeros(N,1);                % evolution path for C
ps = zeros(N,1);                % evolution path for sigma
B = eye(N);                     % B defines the coordinate system
D = eye(N);                     % diagonal matrix D defines the scaling
C = B*D*(B*D);                  % covariance matrix
eigeneval = 0;                  % B and D updated at counteval == 0
chiN = N^0.5*(1-1/(4*N)+1/(21*N^2));    % expectation of ||N(0,I)|| == norm(randn(N,1))

% --------------------------- Generation Loop -------------------------

filename = strcat('c:\cmaes\log_cmaes_meta.txt');
fid = fopen(filename,'w');
fprintf(fid,'\ny_measured_test = %s',mat2str(y_measured_test));

format long;

counteval = 0;
BIC_values = [];
sigma_values = [];
S1_values = [];
sigma_NN_values = [];
Add_values = [];
NN_coeff_value = [];
AR_values = [];
MA_values = [];
sigma_ARMA_values = [];
ARMA_coeff_value = [];
```

```
while counteval < stopeval

    % Generate and evaluate lambda offspring
    model_coef_parameters = cell(5,lambda);
    for k = 1:lambda,
        arz(:,k) = randn(N,1);                          % standard normally distributed vector
        arx(:,k) = xmean + sigma * (B*D*arz(:,k));      % add mutation  % Eq. 34

        % S1 and Add have to be integers:
        if arx(1,k) < 1
            arx(1,k) = 1;
        end
        if arx(3,k) < 0
            arx(3,k) = 0;
        end
        arx1Input = round(arx(1,k));
        arx3Input = round(arx(3,k));

        % p and q have to be integers:
        if arx(5,k) < 0
            arx(5,k) = 0;
        end
        if arx(6,k) < 0
            arx(6,k) = 0;
        end
        while (arx(5,k) < 1 && arx(6,k) < 1)
            arx(5,k) = randint(1,1,[0,1]);
            arx(6,k) = randint(1,1,[0,1]);
        end
        arx5Input = round(arx(5,k));
        arx6Input = round(arx(6,k));

        % objective function call ARMA part:
        [y_cmaesNN_training(:,k),W1,W2,b1,b2,BICmodelNN(k),RMSEmodelNN(k),bestnumberNN_parameter(k)] = ...
              feval(strfitnessfct1, y_measured_training, y_measured_test, arx1Input, arx(2,k), arx3Input);

        % objective function call NN part:
        [y_cmaesARMA_training(:,k),coef,BICmodelARMA(k),RMSEmodelARMA(k),bestnumberARMA_parameter(k)] = ...
              feval(strfitnessfct2, y_measured_training,y_measured_test, arx5Input, arx6Input, arx(7,k));

        y_cmaes_training(:,k) = arx(4,k) * y_cmaesNN_training(:,k) + arx(8,k) * y_cmaesARMA_training(:,k);

        SSE = (y_measured_training(1:size(y_measured_training,1),1) - y_cmaes_training(:,k)).^2;
        SSE = sum(SSE);
        RMSEmodel(k) = sqrt(SSE/(size(y_measured_training,1)));

        parameter = bestnumberNN_parameter(k)+ bestnumberARMA_parameter(k) + 2;
                        % +2 are the 2 coefficients for the respective part (ARMA or NN) of the model:
                         % arx(4) and arx(8)
        BIC = size(y_measured_training,1) * log(SSE/size(y_measured_training,1)) + parameter * ...
              log(size(y_measured_training,1));
        if arx(1,k) > 10
            BIC = 2*BIC;
        end
        if arx(3,k) > 5
            BIC = 2*BIC;
        end
        if arx(5,k) > 10
            BIC = 2*BIC;
        end
        if arx(6,k) > 10
            BIC = 2*BIC;
        end

        arfitness(k) = BIC;
        disp(sprintf('cmaes_meta inside lambda loop =====> BIC: %6.4f',arfitness(k)));
        model_coef_parameters(1,k) = {W1};
        model_coef_parameters(2,k) = {W2};
        model_coef_parameters(3,k) = {b1};
        model_coef_parameters(4,k) = {b2};
        model_coef_parameters(5,k) = {coef};

        counteval = counteval+1;
    end

    % Sort by fitness and compute weighted mean into xmean
    [arfitness, arindex] = sort(arfitness); % minimization

    xmean = arx(:,arindex(1:mu))*weights;   % recombination             % Eq. 35
    zmean = arz(:,arindex(1:mu))*weights;   % == sigma^-1*D^-1*B'*(xmean-xold)

    % Cumulation: Update evolution paths
    ps = (1-cs)*ps + (sqrt(cs*(2-cs)*mueff)) * (B*zmean);               % Eq. 36
    hsig = norm(ps)/sqrt(1-(1-cs)^(2*counteval/lambda))/chiN < 1.5+1/(N-0.5);
    pc = (1-cc)*pc + hsig * sqrt(cc*(2-cc)*mueff) * (B*D*zmean);        % Eq. 38

    % Adapt covariance matrix C
    C = (1-ccov) * C ...                    % regard old matrix         % Eq. 39
        + ccov * (1/mucov) * (pc*pc' ...    % plus rank one update
                              + (1-hsig) * cc*(2-cc) * C) ...
        + ccov * (1-1/mucov) ...            % plus rank mu update
            * (B*D*arz(:,arindex(1:mu))) ...
            * diag(weights) * (B*D*arz(:,arindex(1:mu)))';

    % Adapt step size sigma
    sigma = sigma * exp((cs/damps)*(norm(ps)/chiN - 1))
    current_xmean = xmean(1:8)
```

```
    % Update B and D from C
    if counteval - eigeneval > lambda/ccov/N/10         % to achieve O(N^2)
        eigeneval = counteval;
        C=triu(C)+triu(C,1)';       % enforce symmetry
        [B,D] = eig(C);             % eigen decomposition, B==normalized eigenvectors
        D = diag(sqrt(diag(D)));    % D contains standard deviations now
    end

    % Break, if fitness is good enough
    if arfitness(1) <= stopfitness
        break;
    end

    % Escape flat fitness
    if arfitness(1) == arfitness(ceil(0.7*lambda))
        sigma = sigma * exp(0.2+cs/damps);
    end

    %disp([num2str(counteval) ': ' num2str(arfitness(1))]);
    BIC_values = [BIC_values arfitness(1)];
    sigma_values = [sigma_values sigma];
    S1_values = [S1_values xmean(1)];
    sigma_NN_values = [sigma_NN_values xmean(2)];
    Add_values = [Add_values xmean(3)];
    NN_coeff_value = [NN_coeff_value xmean(4)];
    AR_values = [AR_values xmean(5)];
    MA_values = [MA_values xmean(6)];
    sigma_ARMA_values = [sigma_ARMA_values xmean(7)];
    ARMA_coeff_value = [ARMA_coeff_value xmean(8)];

    fprintf(fid,'\ncurrent counteval: %6.0f/%6.0f, BIC: %6.4f',counteval,stopeval,arfitness(1));
    disp(sprintf('cmaes_meta =====> current counteval: %6.0f/%6.0f, BIC: %6.4f',counteval,stopeval,arfitness(1)));

end % while, end generation loop

% ---------------------------- Ending Message -------------------------

xmin = arx(:, arindex(1));  % Return best point of last generation.
                            % Notice that xmean is expected to be even
                            % better.

y_cmaes_besttraining = y_cmaes_training(:,arindex(1));
bestRMSEmodel = RMSEmodel(1,arindex(1));
bestW1 = model_coef_parameters{1,arindex(1)};
bestW2 = model_coef_parameters{2,arindex(1)};
bestb1 = model_coef_parameters{3,arindex(1)};
bestb2 = model_coef_parameters{4,arindex(1)};
bestcoef = model_coef_parameters{5,arindex(1)};
bestAR = arx(5, arindex(1));
bestNNparam = bestnumberNN_parameter(1,arindex(1));
bestARMAparam = bestnumberARMA_parameter(1,arindex(1));

% creating entries in log-files:
fprintf(fid,'\n==> best model W1 = %s',mat2str(bestW1));
fprintf(fid,'\n==> best model W2 = %s',mat2str(bestW2));
fprintf(fid,'\n==> best model b1 = %s',mat2str(bestb1));
fprintf(fid,'\n==> best model b2 = %s',mat2str(bestb2));
fprintf(fid,'\n==> best model coefficients = %s',mat2str(bestcoef));
fprintf(fid,'\n==> best model AR = %s',mat2str(bestAR));
fprintf(fid,'\n==> best model number NN parameters = %s',mat2str(bestNNparam));
fprintf(fid,'\n==> best model number ARMA parameters = %s',mat2str(bestARMAparam));
fprintf(fid,'\n==> best model y_training %s',mat2str(y_cmaes_besttraining));
fprintf(fid,'\n==> best model RMSE %s',mat2str(bestRMSEmodel));

% apply the new model to test data
% first the NN part:
Add = size(bestW1,2)-size(bestb1,1)-1;
P = y_measured_test';
for i = 1:1:Add
    P = [P;delaysig(y_measured_test',i,i)];
end
Pseq = con2seq(P);
R0 = [min(y_measured_test) max(y_measured_test)];
R = R0;
for i=1:1:Add
    R =[R;R0];
end

NN_model = nnt2elm(R,bestW1,bestb1,bestW2,bestb2);
Y = sim(NN_model,Pseq);
z = seq2con(Y);
y_NN_test = z{1,1}';

% second the ARMA part:
ARMA_model = idpoly([1 bestcoef(1:round(bestAR))'],[],[1 bestcoef(round(bestAR)+1:size(bestcoef))']);
y_ARMA_test = predict(ARMA_model,y_measured_test,1);

y_cmaes_test = xmin(4) * y_NN_test + xmin(8) * y_ARMA_test;
NNcoeff = xmin(4);
ARMAcoeff = xmin(8);

fprintf(fid,'\ny_NN_test = %s',mat2str(y_NN_test));
fprintf(fid,'\ny_ARMA_test = %s',mat2str(y_ARMA_test));
fprintf(fid,'\ny_cmaes_test = %s',mat2str(y_cmaes_test));

SSE_test_data = sum ((y_measured_test(1:nr_records_part,1)-y_cmaes_test(1:nr_records_part,1)).^2);
RMSE_test_data = sqrt(SSE_test_data/nr_records_part);
```

```
disp(sprintf('\n=====> RMSE test data %i points ahead: %6.4f',nr_records_part,RMSE_test_data));
disp(sprintf('=====> MSE test data %i points ahead: %6.4f',nr_records_part,RMSE_test_data.^2));
fprintf(fid,'\n=====> RMSE test data %i points ahead: %6.4f',nr_records_part,RMSE_test_data);
fprintf(fid,'\n=====> MSE test data %i points ahead: %6.4f',nr_records_part,RMSE_test_data.^2);

% plot results:
yplot = [y_measured_test(1:nr_records_part,1) y_cmaes_test(1:nr_records_part,1)];
subplot(3, 1, 1);
plot(1:1:nr_records_part,yplot);
title(['Measured test data vs. CMA-ES meta level forecast, ', num2str(nr_records_part),' points ahead']);
legend('y measured test','y cmaes test',-1);

SSE_test_data = sum ((y_measured_test(1:nr_records_total,1)-y_cmaes_test(1:nr_records_total,1)).^2);
RMSE_test_data = sqrt(SSE_test_data/nr_records_total);
disp(sprintf('\n=====> RMSE test data %i points ahead: %6.4f',nr_records_total,RMSE_test_data));
disp(sprintf('=====> MSE test data %i points ahead: %6.4f',nr_records_total,RMSE_test_data.^2));
disp(sprintf('\n'));
fprintf(fid,'\n=====> RMSE test data %i points ahead: %6.4f',nr_records_total,RMSE_test_data);
fprintf(fid,'\n=====> MSE test data %i points ahead: %6.4f',nr_records_total,RMSE_test_data.^2);
fprintf(fid,'\n');

yplot = [y_measured_test(1:nr_records_total,1) y_cmaes_test(1:nr_records_total,1)];
subplot(3, 1, 2);
plot(1:1:nr_records_total,yplot);
title(['Measured test data vs. CMA-ES meta level forecast, ', num2str(nr_records_total),' points ahead']);
legend('y measured test','y cmaes test',-1);

subplot(3, 1, 3);
plot(1:lambda:lambda*size(BIC_values,2),BIC_values);
title('BIC values');
legend('Search process',-1);

fprintf(fid,'\nARMAcoeff = %s',mat2str(ARMAcoeff));
fprintf(fid,'\nNNcoeff = %s',mat2str(NNcoeff));

fprintf(fid,'\nBIC_values = %s',mat2str(BIC_values));
fprintf(fid,'\nsigma_values = %s',mat2str(sigma_values));
fprintf(fid,'\nS1_values = %s',mat2str(S1_values));
fprintf(fid,'\nsigma_NN_values = %s',mat2str(sigma_NN_values));
fprintf(fid,'\nAdd_values = %s',mat2str(Add_values));
fprintf(fid,'\nNN_coeff_value = %s',mat2str(NN_coeff_value));
fprintf(fid,'\nAR_values = %s',mat2str(AR_values));
fprintf(fid,'\nsigma_ARMA_values = %s',mat2str(sigma_ARMA_values));
fprintf(fid,'\nMA_values = %s',mat2str(MA_values));
fprintf(fid,'\nARMA_coeff_value = %s',mat2str(ARMA_coeff_value));

fclose(fid);

% --------------------------------------------------------------------
```

MATLAB code, ARMA part

```
function [y_cmaesARMA_training,coefficients,BICmodel,RMSEmodel,number_parameter]= cmaes_arma(y_measured_training, ...
    y_measured_test, AR, MA, sigma, nr_records_part, nr_records_total)

% ARMA part
% Source pure CMA-ES: [37], N. Hansen, The CMA Evolution Strategy: A Tutorial

% ---------------------------- Initialization -------------------------

% input parameters:
% y_measured_training - training set, matrix (#,1)
% y_measured_test - test set, matrix (#,1)
% AR - order p of autoregressive model
% MA - order q of moving average model
% sigma - coordinate wise standard deviation (step size)
% nr_records_part - number of records for the plot of a part of the test data
% nr_records_total - number of records for the plot of all test data

% User defined input parameters (need to be edited)

strfitnessfct = 'BIC';      % name of objective/fitness function
N = AR + MA;                % number of objective variables/problem dimension
xmean = rand(N,1);          % objective variables initial point
stopfitness = -15000;       % stop if fitness < stopfitness (minimalization)
stopeval = 25*N^2;          % stop after stopeval number of function evaluations

% Strategy parameter setting: Selection
lambda = 4+floor(3*log(N));     % population size, offspring number
mu = floor(lambda/2);           % number of parents/points for recombination
weights = log(mu+1)-log(1:mu)'; % muXone array for weighted recombination
weights = weights/sum(weights); % normalize recombination weights array
mueff = sum(weights)^2/sum(weights.^2); % variance-effective size of mu

% Strategy parameter setting: Adaptation
cc = 4/(N+4);                   % time constant for cumulation for covariance matrix
cs = (mueff+2)/(N+mueff+3);     % t-const for cumulation for sigma control
mucov = mueff;                  % size of mu used for calculaying learning rate ccov
ccov = (1/mucov) * 2/(N+1.4)^2 + (1-1/mucov) * ...  % learning rate for covariance matrix
        ((2*mueff-1)/((N+2)^2+2*mueff));
damps = 1+2*max(0,sqrt((mueff-1)/(N+1))-1) + cs;    % damping for sigma

% Initialize dynamic (internal) strategy parameters and constants
pc = zeros(N,1);                % evolution path for C
ps = zeros(N,1);                % evolution path for sigma
B = eye(N);                     % B defines the coordinate system
D = eye(N);                     % diagonal matrix D defines the scaling
C = B*D*(B*D);                  % covariance matrix
eigeneval = 0;                  % B and D updated at counteval == 0
chiN = N^0.5*(1-1/(4*N)+1/(21*N^2));    % expectation of ||N(0,I)|| == norm(randn(N,1))

% ---------------------------- Generation Loop -------------------------

filename = strcat('c:\cmaes\log_cmaes_arma_',int2str(AR),'_',int2str(MA),'_',num2str(sigma),'_.txt');

fid = fopen(filename,'w');
format long;

counteval = 0;
BIC_values = [];
while counteval < stopeval

    % Generate and evaluate lambda offspring
    for k = 1:lambda,
        arfitness(k) = 1e20;
        while arfitness(k) == 1e20
            arz(:,k) = randn(N,1);  % standard normally distributed vector
            arx(:,k) = xmean + sigma * (B*D*arz(:,k));      % add mutation  % Eq. 34
            % objective function call:
            [arfitness(k),ARMA_parameter(k)] = feval(strfitnessfct, arx(:,k), y_measured_training, AR, MA);
        end
        counteval = counteval+1;
    end

    % Sort by fitness and compute weighted mean into xmean
    [arfitness, arindex] = sort(arfitness); % minimization
    xmean = arx(:,arindex(1:mu))*weights;   % recombination            % Eq. 35
    zmean = arz(:,arindex(1:mu))*weights;   % == sigma^-1*D^-1*B'*(xmean-xold)

    % Cumulation: Update evolution paths
    ps = (1-cs)*ps + (sqrt(cs*(2-cs)*mueff)) * (B*zmean);               % Eq. 36
    hsig = norm(ps)/sqrt(1-(1-cs)^(2*counteval/lambda))/chiN < 1.5+1/(N-0.5);
    pc = (1-cc)*pc + hsig * sqrt(cc*(2-cc)*mueff) * (B*D*zmean);        % Eq. 38

    % Adapt covariance matrix C
    C = (1-ccov) * C ...                    % regard old matrix         % Eq. 39
        + ccov * (1/mucov) * (pc*pc' ...    % plus rank one update
                              + (1-hsig) * cc*(2-cc) * C) ...
        + ccov * (1-1/mucov) ...            % plus rank mu update
            * (B*D*arz(:,arindex(1:mu))) ...
            * diag(weights) * (B*D*arz(:,arindex(1:mu)))';

    % Adapt step size sigma
    sigma = sigma * exp((cs/damps)*(norm(ps)/chiN - 1));
```

```
    % Update B and D from C
    if counteval - eigeneval > lambda/ccov/N/10         % to achieve O(N^2)
        eigeneval = counteval;
        C=triu(C)+triu(C,1)';   % enforce symmetry
        [B,D] = eig(C);         % eigen decomposition, B==normalized eigenvectors
        D = diag(sqrt(diag(D))); % D contains standard deviations now
    end

    % Break, if fitness is good enough
    if arfitness(1) <= stopfitness
        break;
    end

    % Escape flat fitness
    if arfitness(1) == arfitness(ceil(0.7*lambda))
        sigma = sigma * exp(0.2+cs/damps);
    end

    BIC_values = [BIC_values arfitness(1)];

end % while, end generation loop

% ---------------------------- Ending Message ------------------------

number_parameter = ARMA_parameter(arindex(1));
xmin = arx(:, arindex(1));  % Return best point of last generation.
                            % Notice that xmean is expected to be even
                            % better.

ARMA_model = idpoly([1 xmin(1:AR)'],[],[1 xmin(AR+1:size(xmin))']);
y_ARMA_training = predict(ARMA_model,y_measured_training,1);
SSE = sum((y_measured_training(max(AR,MA)+1:size(y_measured_training,1),1)-
y_ARMA_training(max(AR,MA)+1:size(y_measured_training,1),1)).^2);
RMSE = sqrt(SSE/(size(y_measured_training,1)-(AR+MA)));
str_coefficients = mat2str(xmin);
fprintf(fid,'\n=====> coefficients: %s',str_coefficients);
fprintf(fid,'\n=====> best model ARMA level1, BIC: %6.4f, RMSE: %6.4f',arfitness(1),RMSE);
fprintf(fid,'\n=====> y_ARMA_training: %s',mat2str(y_ARMA_training));

fclose(fid);

y_cmaesARMA_training = y_ARMA_training;
coefficients = xmin;
BICmodel = arfitness(1);
RMSEmodel = RMSE;

% ---------------------------------------------------------------------

function [BIC,ARMA_parameter] = BIC(x, y_measured_training, AR, MA)

ARMA_model = idpoly([1 x(1:AR)'],[],[1 x(AR+1:size(x,1))']);
    theta = ARMA_model;
    if ~iscell(y_measured_training)
        ze = {y_measured_training};
    else
        ze = y_measured_training;
    end

    try % check the stability of the model at all
        [dum,X0] = pe(ze,theta,'e');
        if ~(X0(1,:)==zeros(size(X0,1)))
            try
            y_ARMA_training = predict(ARMA_model,y_measured_training,1);
            SSE = sum((y_measured_training((max(AR,MA)+1):size(y_measured_training,1),1)- ...
                       y_ARMA_training((max(AR,MA)+1):size(y_measured_training,1),1)).^2);
            BIC = size(y_measured_training,1) * log(SSE/size(y_measured_training,1)) + (AR+MA+1) * ...
                       log(size(y_measured_training,1));
            ARMA_parameter = AR+MA+1;
            catch
            BIC = 1e20;
            ARMA_parameter = AR+MA+1;
            end
        else
            BIC = 1e20;
            ARMA_parameter = AR+MA+1;
        end
    catch
        BIC = 1e20;
        ARMA_parameter = AR+MA+1;
    end
```

MATLAB code, NN part

```
function [y_cmaesNN_training,W1,W2,b1,b2,BICmodel,RMSEmodel,number_parameter]= cmaes_nn(y_measured_training, ...
    y_measured_test, S1, sigma, add_input_nodes, nr_records_part, nr_records_total)

% NN part
% Source pure CMA-ES: [37], N. Hansen, The CMA Evolution Strategy: A Tutorial

% --------------------------- Initialization -------------------------

% input parameters:
% y_measured_training - training set, matrix (#,1)
% y_measured_test - test set, matrix (#,1)
% S1 - number of hidden nodes
% sigma - coordinate wise standard deviation (step size)
% add_input_nodes - number of additional input nodes, standard = 0
% nr_records_part - number of records for the plot of a part of the test data
% nr_records_total - number of records for the plot of all test data

% User defined input parameters (need to be edited)
strfitnessfct = 'BIC';      % name of objective/fitness function
Rsize=1+add_input_nodes;

N = (S1*(S1+Rsize)) + 2*S1 +1;      % number of objective variables/problem dimension
xmean = randn(N,1);                 % objective variables initial point
stopfitness = -15000;               % stop if fitness < stopfitness (minimalization)
stopeval = 500;                     % stop after stopeval number of function evaluations

% Strategy parameter setting: Selection
lambda = 4+floor(3*log(N));     % population size, offspring number
mu = floor(lambda/2);           % number of parents/points for recombination
weights = log(mu+1)-log(1:mu)'; % muXone array for weighted recombination
weights = weights/sum(weights); % normalize recombination weights array
mueff = sum(weights)^2/sum(weights.^2); % variance-effective size of mu

% Strategy parameter setting: Adaptation
cc = 4/(N+4);                   % time constant for cumulation for covariance matrix
cs = (mueff+2)/(N+mueff+3);     % t-const for cumulation for sigma control
mucov = mueff;                  % size of mu used for calculaying learning rate ccov
ccov = (1/mucov) * 2/(N+1.4)^2 + (1-1/mucov) * ...  % learning rate for covariance matrix
        ((2*mueff-1)/((N+2)^2+2*mueff));
damps = 1+2*max(0,sqrt((mueff-1)/(N+1))-1) + cs;    % damping for sigma

% Initialize dynamic (internal) strategy parameters and constants
pc = zeros(N,1);                % evolution path for C
ps = zeros(N,1);                % evolution path for sigma
B = eye(N);                     % B defines the coordinate system
D = eye(N);                     % diagonal matrix D defines the scaling
C = B*D*(B*D);                  % covariance matrix
eigeneval = 0;                  % B and D updated at counteval == 0
chiN = N^0.5*(1-1/(4*N)+1/(21*N^2));    % expectation of ||N(0,I)|| == norm(randn(N,1))

% --------------------------- Generation Loop -------------------------

filename = strcat('c:\cmaes\log_cmaes_nn_',int2str(S1),'_',num2str(sigma),'_',num2str(add_input_nodes),'_.txt');

fid = fopen(filename,'w');
format long;
nntwarn off

counteval = 0;
BIC_values = [];

P = y_measured_training';
for i = 1:1:add_input_nodes
    P = [P;delaysig(y_measured_training',i,i)];
end
Pseq = con2seq(P);
R0 = [min(y_measured_training) max(y_measured_training)];
R = R0;
for i=1:1:add_input_nodes
    R =[R;R0];
end

while counteval < stopeval

    % Generate and evaluate lambda offspring
    for k = 1:lambda,
        arz(:,k) = randn(N,1);  % standard normally distributed vector
        arx(:,k) = xmean + sigma * (B*D*arz(:,k));      % add mutation  % Eq. 34
        % objective function call
        [arfitness(k),NN_parameter(k)] = ...
             feval(strfitnessfct,arx(:,k),y_measured_training,S1,add_input_nodes,Rsize,R,Pseq);
        counteval = counteval+1;
    end

    % Sort by fitness and compute weighted mean into xmean
    [arfitness, arindex] = sort(arfitness); % minimization
    xmean = arx(:,arindex(1:mu))*weights;   % recombination             % Eq. 35
    zmean = arz(:,arindex(1:mu))*weights;   % == sigma^-1*D^-1*B'*(xmean-xold)

    % Cumulation: Update evolution paths
    ps = (1-cs)*ps + (sqrt(cs*(2-cs)*mueff)) * (B*zmean);               % Eq. 36
    hsig = norm(ps)/sqrt(1-(1-cs)^(2*counteval/lambda))/chiN < 1.5+1/(N-0.5);
```

```
    pc = (1-cc)*pc + hsig * sqrt(cc*(2-cc)*mueff) * (B*D*zmean);        % Eq. 38

    % Adapt covariance matrix C
    C = (1-ccov) * C ...                    % regard old matrix         % Eq. 39
        + ccov * (1/mucov) * (pc*pc' ...    % plus rank one update
                              + (1-hsig) * cc*(2-cc) * C) ...
        + ccov * (1-1/mucov) ...            % plus rank mu update
            * (B*D*arz(:,arindex(1:mu))) ...
            * diag(weights) * (B*D*arz(:,arindex(1:mu)))';

    % Adapt step size sigma
    sigma = sigma * exp((cs/damps)*(norm(ps)/chiN - 1));

    % Update B and D from C
    if counteval - eigeneval > lambda/ccov/N/10         % to achieve O(N^2)
        eigeneval = counteval;
        C=triu(C)+triu(C,1)';   % enforce symmetry
        [B,D] = eig(C);         % eigen decomposition, B==normalized eigenvectors
        D = diag(sqrt(diag(D))); % D contains standard deviations now
    end

    % Break, if fitness is good enough
    if arfitness(1) <= stopfitness
        break;
    end

    % Escape flat fitness
    if arfitness(1) == arfitness(ceil(0.7*lambda))
        sigma = sigma * exp(0.2+cs/damps);
    end

    BIC_values = [BIC_values arfitness(1)];

end % while, end generation loop

% ---------------------------- Ending Message ------------------------

number_parameter = NN_parameter(arindex(1));
xmin = arx(:, arindex(1));  % Return best point of last generation.
                            % Notice that xmean is expected to be even
                            % better.

% weights of the new model:
W1 = [];
for i = 0:1:(S1+Rsize-1)
    W1 = [W1 xmin((S1*i+1):S1*(i+1))];
end
b1 = xmin((S1*(S1+Rsize))+1:(S1*(S1+Rsize))+ S1,1);
W2 = xmin((S1*(S1+Rsize))+S1+1:(S1*(S1+Rsize))+ 2*S1,1)';
b2 = xmin(size(xmin,1),1);
fprintf(fid,'\n=====> W1: %s',mat2str(W1));
fprintf(fid,'\n=====> b1: %s',mat2str(b1));
fprintf(fid,'\n=====> W2: %s',mat2str(W2));
fprintf(fid,'\n=====> b2: %s',mat2str(b2));

% apply the new model to training data:
NN_model = nnt2elm(R,W1,b1,W2,b2);
Y = sim(NN_model,Pseq);
z = seq2con(Y);
y_NN_training = z{1,1}';
SSE = sum((y_measured_training((add_input_nodes+2):size(y_measured_training,1),1)-
y_NN_training((add_input_nodes+2):size(y_measured_training,1),1)).^2);
RMSE = sqrt(SSE/(size(y_measured_training,1)-1-add_input_nodes));

fprintf(fid,'\n=====> best model NN level1, BIC: %6.4f, RMSE: %6.4f',arfitness(1),RMSE);
fprintf(fid,'\n=====> y_NN_training: %s',mat2str(y_NN_training));

fclose(fid);

y_cmaesNN_training = y_NN_training;
BICmodel = arfitness(1);
RMSEmodel = RMSE;

% ---------------------------------------------------------------------

function [BIC, NN_parameter] = BIC(x, y_measured_training,S1,add_input_nodes,Rsize,R,Pseq)

% initial weights and biases
W1 = [];
for i = 0:1:(S1+Rsize-1)
    W1 = [W1 x((S1*i+1):S1*(i+1))];
end
b1 = x((S1*(S1+Rsize))+1:(S1*(S1+Rsize))+ S1,1);
W2 = x((S1*(S1+Rsize))+S1+1:(S1*(S1+Rsize))+ 2*S1,1)';
b2 = x(size(x,1),1);

% create the model
NN_model = nnt2elm(R,W1,b1,W2,b2);

% apply the model to training data
Y = sim(NN_model,Pseq);
z = seq2con(Y);
y_NN_training = z{1,1}';
SSE = sum((y_measured_training((add_input_nodes+2):size(y_measured_training,1),1)- ...
             y_NN_training((add_input_nodes+2):size(y_measured_training,1),1)).^2);
NN_parameter = S1*(Rsize+2)+1;
BIC = size(y_measured_training,1) * log(SSE/size(y_measured_training,1)) + NN_parameter * ...
             log(size(y_measured_training,1));
```

Symbols

$_{ARMA}$	index for values contributed by ARMA model
b	bias for hidden units and/or output units in neural networks
C_x	denotes x^{th} context unit
c	coefficent, general
E	error of neural network
h	input of a neural network unit (weighted sum)
N_{ex}	number of training examples
N_f	number of forecasts
N_{par}	number of parameters
$_{NN}$	index for values contributed by neural network model
O_i	denotes i^{th} output unit
m	number of lags for neural networks, content of context units
p	number of lags for AR model, order of the process
q	number of lags for MA model, order of the process
$_{sum}$	index for values of a combined model
t	specific time where a time series value is recorded
m	number of lags for neural networks, content of context units
s	number of hidden units for neural networks
w_{ij}	weight of connections from hidden units to output units
w_{jk}	weight of connections from input units to hidden units
w_{jx}	weight of connections from context units to hidden units
y	time series
$\hat{y}_t$	actual time series value y at time t
y_t	modelled time series value y at time t
ε	white noise series
ε_t	actual white noise series value ε at time t
ϕ_m	m^{th} parameter of an AR model, m = [t-1..t-p]
η	learning rate
μ	number of patterns, presented to train a neural network
ϑ_n	n^{th} parameter of a MA model, n = [t-1..t-q]
σ	initial step size for ES, standard deviation
ξ_k	denotes k^{th} input unit

Bibliography

[1] Abraham, A., Grosan, C., Han, S.Y. & Gelbukh, A. (2005): Evolutionary Multiobjective Optimization Approach for Evolving Ensemble of Intelligent Paradigms for Stock Market Modeling. MICAI 2005: Advances in Artificial Intelligence, LNCS Volume 3789, 673-681.

[2] Liu, F., Ng, G.S. & Quek, C. (2007): RLDDE: A novel reinforcement learning-based dimension and delay estimator for neural networks in time series prediction. Neurocomputing, 70(7-9), 1331-1341.

[3] Manabe, Y. & Chakraborty, B. (2007): A novel approach for estimation of optimal embedding parameters of nonlinear time series by structural learning of neural network. Neurocomputing, 70(7-9), 1360-1371.

[4] Mandischer, M. (2002): A comparison of evolution strategies and backpropagation for neural network training. Neurocomputing, 42(1-4), 87-117.

[5] Zou, H.F., Xia, G.P., Yang, F.T. & Wang, H.Y. (2007): An investigation and comparison of artificial neural network and time series models for Chinese food grain price forecasting. Neurocomputing, 70(16-18), 2913-2923.

[6] Zhang, G.P. (2003): Time series forecasting using a hybrid ARIMA and neural network model. Neurocomputing, 50(2003), 159-175.

[7] Hansen, J.V. & Nelson, R.D. (2002): Data mining of time series using stacked generalizers. Neurocomputing, 43(1-4), 173-184.

[8] Kohzadi, N., Boyd, M.S., Kermanshahi, B. & Kaastra, I. (1996): A comparison of artificial neural network and time series models for forecasting commodity prices. Neurocomputing, 10(2), 169-181.

[9] Araujo, A.D., Madeiro, R., Sousa, F.D.R.P., Pessoa, L.F.C. & Ferreira, T.A.E. (2006): An Evolutionary Morphological Approach for Financial Time Series Forecasting. 2006 IEEE Congress on Evolutionary Computation, 2467-2474.

[10] Cortez, P., Rocha, M. & Neves, J. (2004): Evolving Time Series Forecasting ARMA Models. Journal of Heuristics, 10(4), 415-429.

[11] Ferreira, T.A.E., Vasconcelos, G.C. & Adeodato, P.J.L. (2005): A new evolutionary method for time series forecasting. Genetic and Evolutionary Computation - GECCO 2005, 2221-2222.

[12] Ferreira, T & Vasconcelos, G. & Adeodato, P. (2004): A Hybrid Intelligent System Approach for Improving the Prediction of Real World Time Series. 2004 IEEE Congress on Evolutionary Computation, 736-743.

[13] Arco-Calderón, C.L.D., Vinuela, P.I. & Castro, J.C.H. (2004): Forecasting Time Series by means of Evolutionary Algorithms. Parallel Problem Solving from Nature - PPSN VIII, LNCS Vol. 3242, 1061-1070.

[14] Cortez, P., Rocha, M. & Neves, J. (2001): Genetic and Evolutionary Algorithms for Time Series Forecasting. Engineering of Intelligent Systems. 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2001), LNCS Vol. 2070, 393-402.

[15] Santini, M. & Tettamanzi, A. (2001): Genetic Programming for Financial Time Series Prediction. Genetic Programming: 4th European Conference (EuroGP 2001), LNCS Vol. 2038, 361-370.

[16] Whigham, P.A. & Recknagel, F. (2001): An inductive approach to ecological time series modelling by evolutionary computation. Ecological Modelling, 146(1-3), 275-287.

[17] Orfila, A., Ballester, J.L., Oliver, R., Alvarez, A. & Tintoré, J. (2002): Forecasting the solar cycle with genetic algorithms. Astronomy and Astrophysics 386(1), 313-318.

[18] Hulthén, E. & Wahde, M. (2004): Improved time series prediction using evolutionary algorithms for the generation of feedback connections in neural networks. In: Computational Finance and its Applications, WIT Press, Southampton, Boston, 211 pp.

[19] Il Kang, H.; De-Shunag, H.; Xiao-Ping, Z.; Guang-Bin, H. (2005): A fuzzy time series prediction method using the evolutionary algorithm. Advances in Intelligent Computing, LNCS Vol. 3645, 530-537.

[20] Cai, X., Zhang, N., Venayagamoorthy, G.K. & Wunsch II, D.C. (2004): Time Series Prediction with Recurrent Neural Networks Using a Hybrid PSO-EA Algorithm. 2004 IEEE International Joint Conference on Neural Networks, Vol. 2, 1647-1652.

[21] Riley, J. & Ciesielski, V. (2002): Evolving Fuzzy Rules for Reactive Agents in Dynamic Environments. Fourth Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02), 124-130.

[22] Valdés, J.J. & Mateescu, G. (2002): Time Series Model Mining with Similarity-Based Neuro-Fuzzy Networks and Genetic Algorithms: A Parallel Implementation. Rough Sets and Current Trends in Computing: Third International Conference (RSCTC 2002), LNCS Vol. 2475, 279-288.

[23] Chen, Y., Yang, B., Dong, J. & Abraham, A. (2005): Time-series forecasting using flexible neural tree model. Information Sciences, 174(3-4), 219-235.

[24] Weinert, W.R. & Lopes, H.S. (2004): A gene-expression programming system for time-series modeling. Proceedings of XXV Iberian Latin American Congress on Computational Methods in Engineering (CILAMCE 2004).

[25] Weinert, W.R. & Lopes, H.S. (2004): EGIPSYS: an Enhanced Gene Expression Programming Approach for Symbolic Regression Problems. International Journal of Applied Mathematics and Computer Science, 14(3), 375–384.

[26] Park, J.-H. & Choi, Y.-K. (1996): An On-line PID Control Scheme for Unknown Nonlinear Dynamic Systems using Evolution Strategy. Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96), 759-763.

[27] Ghazali, R., Hussain, A. & El-Deredy, W. (2006): Application of Ridge Polynomial Neural Networks to Financial Time Series Prediction. 2006 International Joint Conference on Neural Networks (IJCNN'06), 913- 920.

[28] Tan, T.Z., Quek, C. & Ng, G.S. (2005): Brain-inspired Genetic Complementary Learning for Stock Market Prediction. 2005 IEEE Congress on Evolutionary Computation (CEC 2005), Vol. 3, 2653- 2660.

[29] Hassan, Md.R., Nath, B. & Kirley, M. (2006): HMM based Fuzzy Model for Time Series Prediction. 2006 IEEE International Conference on Fuzzy Systems, 2120-2126.

[30] Dhahri, H. & Alimi, A.M. (2006): The Modified Differential Evolution and the RBF (MDE-RBF) Neural Network for Time Series Prediction. 2006 International Joint Conference on Neural Networks (IJCNN'06), 5245-5250.

[31] Op 't Landt, F.W. (1997): Stock Price Prediction using Neural Networks. University Leiden 1997, MSc. Thesis.

[32] Singh, S. & McAtackney, P. (1998): Dynamic Time-Series Forecasting using Local Approximation. 10th IEEE International Conference on Tools with AI, 392-399.

[33] Igel, C., Hansen & N., Roth, S. (2007): Covariance Matrix Adaptation for Multi-objective Optimization. Evolutionary Computation, 15(1), 1-28.

[34] Hansen, N. & Ostermeier, A. (2001): Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary Computation, 9(2), 159-195.

[35] Spieth, C., Streichert, F., Speer, N. & Zell, A. (2004): Optimizing Topology and Parameters of Gene Regulatory Network Models from Time-Series Experiments. Genetic and Evolutionary Computation – GECCO 2004, LNCS Vol. 3102, 461-470.

[36] Hansen, N. & Ostermeier, A. (1996): Adapting arbitrary normal mutation distributions in evolution-strategies: the covariance matrix adaptation. Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96), 312-317.

[37] Hansen, N. (2005): The CMA Evolution Strategy: A Tutorial. http://lautaro.bionik.tu-berlin.de/user/niko/cmatutorial.pdf.

[38] Hansen, N. & Kern, S. (2004): Evaluating the CMA Evolution Strategy on Multimodal Test Functions. Parallel Problem Solving from Nature - PPSN VIII, LNCS Vol. 3242, 282-291.

[39] González, J., Rojas, I. & Pomares, H. (2002): Evolutive Identification of Fuzzy Systems for Time-Series Prediction. Parallel Problem Solving from Nature - PPSN VII, LNCS Vol. 2439, 517-526.

[40] Ture, M. & Kurt, I. (2006): Comparison of four different time series methods to forecast hepatitis A virus infection. Expert Systems with Applications, 31(1), 41-46.

[41] Balaguer, E., Palomares, A. , Soria, E. & Martín-Guerrero, J.D. (2008): Predicting service request in support centers based on nonlinear dynamics, ARMA modeling and neural networks. Expert Systems with Applications, 34(1), 665-672.

[42] Wang, T.-Y. & Huang, C.-Y. (2007): Applying optimized BPN to a chaotic time series problem. Expert Systems with Applications, 32(1), 193-200.

[43] Aznarte M., J.L., Benítez Sánchez, J.M., Nieto Lugilde, D., de Linares Fernández, C., de la Guardia, C.D. & Alba Sánchez, F. (2007): Forecasting airborne pollen concentration time series with neural and neuro-fuzzy models. Expert Systems with Applications, 32(4), 1218-1225.

[44] Hassan, Md.R., Nath, B. & Kirley, M. (2007): A fusion model of HMM, ANN and GA for stock market forecasting. Expert Systems with Applications, 33(1), 171-180.

[45] Yoshihara, I. & Aoyama, T. & Yasunaga, M. (2000): A Fast Model-Building Method for Time Series Using Genetic Programming. Genetic and Evolutionary Computation Conference (GECCO '00), Morgan Kaufmann 2000, 537.

[46] Yen, G.G. (2006): Multi-Objective Evolutionary Algorithm for Radial Basis Function Neural Network Design. In: Studies in Computational Intelligence, Vol. 16, 221-239.

[47] Baroglio, C., Giordana, A., Piola, R., Kaiser, M. & Nuttin, M. (1996): Learning Controllers for Industrial Robots. Machine Learning, 23(2-3), 221-249.

[48] Zheng, D., Wang, J. & Zhao, Y. (2006): Time Series Predictions Using Multi-scale Support Vector Regressions. Theory and Applications of Models of Computation, LNCS Vol. 3959, 474-481.

[49] Lewis, C.D. (1997): Demand forecasting and inventory control. John Wiley, New York, USA.

[50] Shumway, R.H. & Stoffer, D.S.: (2000): Time series analysis and its applications. Springer-Verlag, New York, USA.

[51] Wikipedia (2007): http://en.wikipedia.org.

[52] Bäck, T. (1996): Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York, USA.

[53] Bäck, T. & Shir, O. (2006): Evolutionary Algorithms. Lecture Handouts Leiden University, Fall, 2006.

[54] Bäck, T. (2004): Evolution Strategies for Industrial Applications. IPA Herfstdagen on Intelligent Algorithms, Institute for Programming research and Algorithmics, Eindhoven University of Technology.

[55] Hertz, J., Krogh, A. & Palmer, R. G. (1991): Introduction to the theory of neural computation. Westview Press, Boulder, CO, USA.

[56] Qi, M., Zhang, G. P. (2001): An investigation of model selection criteria for neural network time series forecasting. European Journal of Operational Research, 132, 666-680.

[57] BMW Group (2003): Die entscheidende Minute. BMW-Forscher stellen ein verbessertes Modell zur Prognose lokaler Verkehrsbewegungen vor. http://www.bmwgroup.com/d/0_0_www_bmwgroup_com/forschung_entwicklung/science_club/veroeffentlichte_artikel/2003/news200311.html.

[58] Coello, C.A.C. (2001): A Short Tutorial on Evolutionary Multiobjective Optimization. Evolutionary Multi-Criterion Optimization: First International Conference (EMO 2001), LNCS Vol. 1993, 21-40.

[59] Quchani, S.A. & Tahami, E. (2007): Comparison of MLP and Elman Neural Network for Blood Glucose Level Prediction in Type 1 Diabetics. 3rd International Conference on Biomedical Engineering (Biomed 06), IFMBE Proceedings Vol. 15, 54-58.

[60] Zhang, ZQ., Tang, Z. & Vairappan, C. (2007): A Novel Learning Method for Elman Neural Network Using Local Search. Neural Information Processing - Letters and Reviews, 11(8), 181-188.

[61] U. Brunelli, V. Piazza, L. Pignato, F. Sorbello & S. Vitabile (2006): Hourly Forecasting of SO_2 Pollutant Concentration Using an Elman Neural Network. Neural Nets: 16th Italian Workshop on Neural Nets (WIRN 2005)/International Workshop on Natural and Artificial Immune Systems (NAIS 2005), LNCS Vol. 3931, 65-69.

[62] Gao, X.Z. & Ovaska, S.J. (2002): Genetic algorithm training of Elman neural network in motor fault detection. Neural Computing & Applications, 11(1), 37–44.

[63] Daliakopoulos, I.N., Coulibaly, P. & Tsanis, I.K. (2005): Groundwater level forecasting using artificial neural networks. Journal of Hydrology, 309(1-4), 229–240.

[64] Delgado, M., Pegalajar, M.C. & Cuéllar, M.P. (2006): Memetic evolutionary training for recurrent neural networks: an application to time-series prediction. Expert Systems, 23(2), 99–115.