Neural Comput & Applic (2004) 13: 101–111
DOI 10.1007/s00521-004-0412-5

ORIGINAL ARTICLE

J. M. Górriz · C. G. Puntonet · M. Salmerón
J. J. G. de la Rosa

A new model for time-series forecasting using radial basis functions and exogenous data

Received: 7 April 2004 / Accepted: 20 April 2004 / Published online: 25 May 2004

Abstract In this paper, we present a new model for time-series forecasting using radial basis functions (RBFs) as a unit of artificial neural networks (ANNs), which allows the inclusion of exogenous information (EI) without additional pre-processing. We begin by summarizing the most well-known EI techniques used ad hoc, i.e., principal component analysis (PCA) and independent component analysis (ICA). We analyze the advantages and disadvantages of these techniques in time-series forecasting using Spanish bank and company stocks. Then, we describe a new hybrid model for time-series forecasting which combines ANNs with genetic algorithms (GAs). We also describe the possibilities when implementing the model on parallel processing systems.

1 Introduction

Several techniques have been developed to forecast time series using stock value data. There also exist numerous forecasting applications, as analyzed in [1]: signal statistics pre-processing and communications, industrial control processing, econometrics, meteorology, physics, biology, medicine, oceanography, seismology, astronomy, and psychology.

A possible solution to this problem was described by Box et al. [2], who developed a time-series forecasting analysis technique based on linear systems. Basically, the procedure consisted of suppressing the non-seasonality of the series by parameter analysis, which measures time-series data correlation, and models the selection which best fits the data obtained (a specific-order ARIMA model). But, in real systems, non-linear and stochastic phenomena occur, and so, the series' dynamics cannot be described exactly by classical models. Artificial neural networks (ANNs) have improved forecasting results, detecting the non-linear nature of the data. ANNs based on radial basis functions (RBFs) allow a better forecasting adjustment; they implement local approximations to non-linear functions, minimizing the mean square error to achieve the adjustment of neural parameters. Platt's algorithm [3], resource allocating network (RAN), consisted of controlling the size of the neural network, thus, reducing the computational cost associated with the calculus of the optimum weights in perceptron networks.

Matrix decomposition techniques have been used as an improvement of Platt's model [4]; these take the most relevant data in the input space, to avoid processing non-relevant information. The model incorporating these techniques, "neural model with automatic parameter adjustment for prediction" (NAPA-PRED), also includes neural pruning [5]. An improved version, INAPA-PRED, based on support vector machine (SVM) philosophy, is presented in Sect 6.2 in the discussion of endogenous time-series forecasting.

The next step was to include exogenous information in these models. Principal component analysis (PCA) is a well-established tool in finance. It has been shown that prediction results can be improved by using this technique [4]. Although both methods linearly transform the observed signal into components, the difference is that, in PCA, the goal is to obtain principal components which are uncorrelated (features), giving projections of the data in the direction of the maximum variance [6]. PCA algorithms use only second-order statistical information. On the other hand, in [7], we can discover interesting structures in finance using the new signal-processing tool, independent component analysis (ICA), which finds statistically independent components by using higher-order statistical information to separate the

J. M. Górriz (✉) · J. J. G. de la Rosa
Department of Systems Engineering and Automation,
Electronic Technology and Electronics,
University of Cádiz, Spain
E-mail: juanmanuel.gorriz@uca.es

C. G. Puntonet (✉) · M. Salmerón
Department of Computer Architecture and Computer Technology,
University of Granada, Spain
E-mail: carlos@atc.ugr.es

signals [8, 9]. This new technique may use entropy [10], contrast functions based on information theory [11], mutual information [12], or geometric considerations in data distribution spaces [13-17], etc. Forecasting and analyzing financial time series using ICA can contribute to a better understanding and more accurate prediction of financial markets [18, 19]. Nevertheless, in this paper, we seek to exclude pre-processing techniques which may contaminate raw data.

2 Forecasting model (cross prediction model)

The cross prediction model (CPM) is shown in Eq. 1. We consider a data set consisting of correlated signals from a stock exchange and seek to build a forecasting function, **P**, for one of the set of signals, {series$_1$,...,series$_S$}, which allows us to include exogenous data from the other series. If we consider just one series [4], the individual forecasting function can be expressed in terms of RBFs as [20]:

$$\mathbf{F}(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x}) = \sum_{i=1}^{N} h_i \exp\left\{ -\frac{\|\mathbf{x} - c_i\|}{r_i^2} \right\} \tag{1}$$

where **x** is a p-dimensional input vector at time t, N is the number of neurons (RBFs), f_i is the output for each i-th neuron, c_i is the center of the i-th neuron which controls the local space situation of this cell, and r_i is the radius of the i-th neuron. The global output is a linear combination of the individual outputs for each neuron with weight h_i. Thus, we are using a method for moving beyond linearity, where the core idea is to augment/replace the input vector, **x**, with additional variables, which are transformations of **x**, and then use linear models in this new space of derived input features. RBFs are one of the most popular kernel methods for regression over the domain $R^n$and consists of fitting a different, but simple model, at each query point, c_i, using those observations close to this target point in order to obtain a smoothed function. This localization is achieved via a weighting function or kernel f_i.

We apply/extend this regularization concept to extra series, including a row of neurons (Eq. 1), for each series and weighting these values with a factor, b_{ij} (the neural resources of ANNs are updated using an on-line SVM algorithm, INAPA-PRED, see Sect. 6.2). Finally, the global smoothed function for the stock j is defined as:

$$\mathbf{P}_j(\mathbf{x}) = \sum_{i=1}^{S} b_{ij} F_i(\mathbf{x}, j) \tag{2}$$

where F_i is the smoothed function of each series, S is the number of input series, and b_{ij} are the weights for j-stock forecasting. Obviously, one of these weight factors must be relevant in this linear fit (b_{jj}~1, or auto-weight factor).

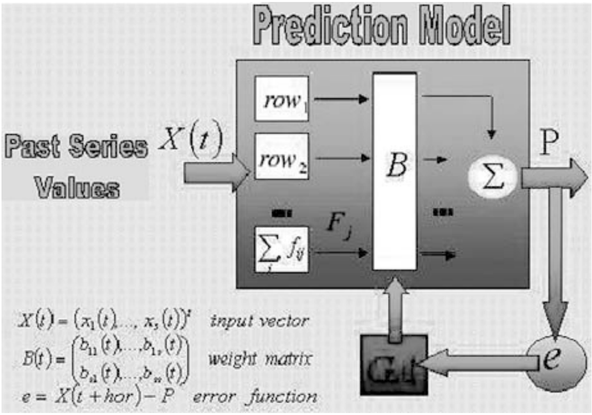
Prediction Model
Past Series Values
$X(t)$
row_1
row_2
$\sum_j f_{ij}$
F_j
B
Σ
P
e
GA
$X(t) = (x_1(t),\ldots,x_s(t))^t$ input vector
$B(t) = \begin{pmatrix} b_{11}(t),\ldots,b_{1s}(t) \\ b_{s1}(t),\ldots,b_{ss}(t) \end{pmatrix}$ weight matrix
$e = X(t+hor) - P$ error function

Fig. 1 Schematic representation of CPM with adaptive radii, centers, and input space ANNs (RAN + NAPAPRED + Reg [18]). This improvement consists of neural parameter adaptation when input space increases, i.e., RBF centers and radii are statistically updated when dynamic series change takes place

We can use matrix notation to include the set of forecasts in an S-dimensional vector, **P** (**B** in Fig. 1):

$$\mathbf{P}(\mathbf{x}) = \operatorname{diag}(\mathbf{B} \cdot \mathbf{F}(\mathbf{x})) \tag{3}$$

where $\mathbf{F} = (F_1,...,F_S)$ is an $S{\times}S$ matrix with $F_I \in R^S$and **B** is an $S{\times}S$ weight matrix. The operator diag extracts the main diagonal.

To test this model, we can choose a set of values for the weight factors as functions of the correlation factors between the series, and so, Eq. 2 can be expressed as:

$$\mathbf{P}(\mathbf{x}) = \left(1 - \sum_{i \neq j}^{S} \rho_i\right) F_j + \sum_{i \neq j}^{S} \rho_i F_i \tag{4}$$

where **P** is the forecasting function for the desired stock, j, and ρ_I is the correlation factor with the exogenous series, i.

We can include Eq. 4in the generalized additive models for regression proposed in supervised learning [21]:

$$\mathbf{E}\{Y|X_1,\ldots,X_n\} = \alpha + f_1(X_1) + \ldots + f_n(X_n) \tag{5}$$

where the X_is usually represent predictors and Y represents the system output; f_js are non-specific smooth ("non-parametric") functions. Thus, we can fit this model by minimizing the mean square error function, or by the other methods presented in [21].

3 Forecasting model and genetic algorithms

CPM uses a genetic algorithm for b_I parameter fitting (see Sect. 6.1 for further discussion). A canonical GA is constructed by operations of parameter encoding, population initialization, crossover, mutation, mate selection, population replacement, etc. Our encoding parametric system consists of the codification into genes and chromosomes or individuals as a string of binary digits using the complement representation, although other encoding methods are also possible, i.e., [22-25], where the value of each parameter is a gene and an individual is encoded by a string of real numbers instead of binary ones. In the initial population generation step, we assume that the parameters lie in a bounded region [0,1] (at the edge of this region, we can reconstruct the model without exogenous data) and Nindividuals are generated randomly. After the initial population, N, is generated, the fitness of each chromosome I_i is determined by the function:

$$\aleph(I_i) = \frac{1}{e(I_i)} \tag{6}$$

To overcome the problem of convergence in the optimal solution, we add a positive constant to the denominator. Another important question in the canonical GA is the definition of the selection operator. New generations for mating are chosen with their fitness function values determined by roulette wheel selection. Once the new individuals have been selected, we apply crossover ($\mathbf{P}_c$) to generate two offspring. In the next step, the mutation operator ($\mathbf{P}_m$) is applied to these offspring to prevent premature convergence. In order to improve the convergence speed of the algorithm, we included mechanisms such as elitist strategy, in which the best individual in the current generation always survives into the next.

The GA used in the forecasting function (Eq. 2) has an absolute error value start criterion. Once it has started, it uses the values (or individual) found to be optimal (elite) the last time and applies a local search around this elite individual. Thus, an efficient search is performed around an individual (set of b_is) in which one parameter is more relevant than the others.

The computational time depends on the encoding length and the number of individuals and genes. Because of the probabilistic nature of the GA-based method, the proposed method almost converges to a global optimal solution on average. Our simulation did not produce any non-convergent cases. Figure 2 shows the iterative procedure implemented for the global prediction system including GA.

4 Simulations

With the aim of assessing the performance of the CP model, we used indexes of Spanish banks and companies for a given period, with particular attention to the IBEX35 index, which we consider is the most representative sample of Spanish stock movements.

We considered the simplest case, which consists of two time series corresponding to the companies' ACS (series$_1$) and BBVA (series$_2$). The former is the target of the forecasting process and the latter is introduced as external information. The period studied was July-October 2000. Each time series includes 200 points corresponding to selling days (quoting days).

The horizon of the forecasting process (*hor*) was set at 8; the weight function of the forecasting function was a correlation function between the two time series for series$_2$ (in particular, we chose its square) and the

Fig. 2 Pseudo-code of CPM + GA

```
Step 1:Initialization Parameters
W = size of prediction window; M = input series maximum length;
Hor = forecast horizon; $N_{ind}$ = $n^o$ individuals of GA;
Epsilon = neural increase; delta= distance between neurons;
uga = activation of GA;Error = forecast error;
Matrix $N_{inps}$ = number of neural inputs of each series;
Matrix $n_{RBFs}$ = number of neurons of each series;
Matrix B = Matrix Weight Vector;
$M_{neuron}$ = Neurons Parameters Matrix, radius, centres, etc.of each series
$Vect_{inp}$ = Input Vector Matrix; $Vect_{Out}$ =Predicted values for each series
Target = Real Data in (t+hor); P = forecast function

Step 2: Modelling Input Space.
Toeplizt $\mathbf{A}$ Matrix in $t_o$
Relevant data series determination in $\mathbf{A}$(Des. SVD,QR)

Step 3: Iteration.
FOR $i = 1 \to Max - Iter$
    P(t) = BT*Output; Error = Target(t+hor) - P(t)
    (Seek for vector B) IF (error > uga)
        Execute GA (Selection, Crossover, Mutation, Elitism...)
    ENDIF
    (Neural parametrers) IF ($error > epsilon$ and $dist(Vect_{inp}, radius) > delta$)
        Add neuron centered in $Vect_{inp}$
    ELSE
        (Evolution of neural networks) Execute pruning.
        Update $M_{neuron}$ (Gradient Descend Method).
    ENDIFELSE
    (Input Space Fit) IF ($error >> epsilon$) Modelling Input Space.
        Update $M_{neuron}$ and $Vect_{inp}$.
    ENDIF
ENDFOR
```

difference to one for series$_1$. We established a 10-day forecasting window (W), and the maximum lag number was set at $2W$, in order to achieve a 10×20 Toeplitz matrix. The first time point of the algorithm was set at 50. Figure 3 shows the forecasting results from lag 50 to 200, corresponding to series$_1$.

Note the instability of the system in the very first iterations, until it reaches an acceptable convergence. The most interesting feature of the result is shown in Table 1. From this table, it is easy to deduce that, if we move one of the two series horizontally, the correlation decreases dramatically. This is how we avoid the delay problem encountered with certain networks, where the information introduced into the system is non-relevant. This problem is due to the increased level of information, together with the fact that we have enclosed only one additional time series (series$_2$), despite the increased neuron resources. At the end of the process, we used 20 neurons for net 1 and 21 for net 2. Although the forecasting is acceptable, we expect a better performance with more data point series.

The next step consisted of using the general algorithm including the GA. A four-individual (N_{ind}) population of dimension 2×1 was used; this is sufficient because the searching space was bounded. The GA was run four times before convergence was reached. The individuals were encoded with 34 bits (17 bits for each parameter). In this case, convergence is defined in terms of the adjustment function; other authors use other GA parameters, such as the absence of change in the individuals after a certain number of generations. We observed a considerable improvement in the forecasting results and evidence of the disappearance of the delay problem, as shown in Fig. 3.

The number of neurons at the end of the process is the same as in the former case, since we have only modified the weight of each series during the forecasting process. The dynamics and values for the weights are shown in Table 2.

Error behavior is shown in Table 2. Note that:

- We can bound the error by appropriate selection of the b_1 parameters, when the dynamics of the series

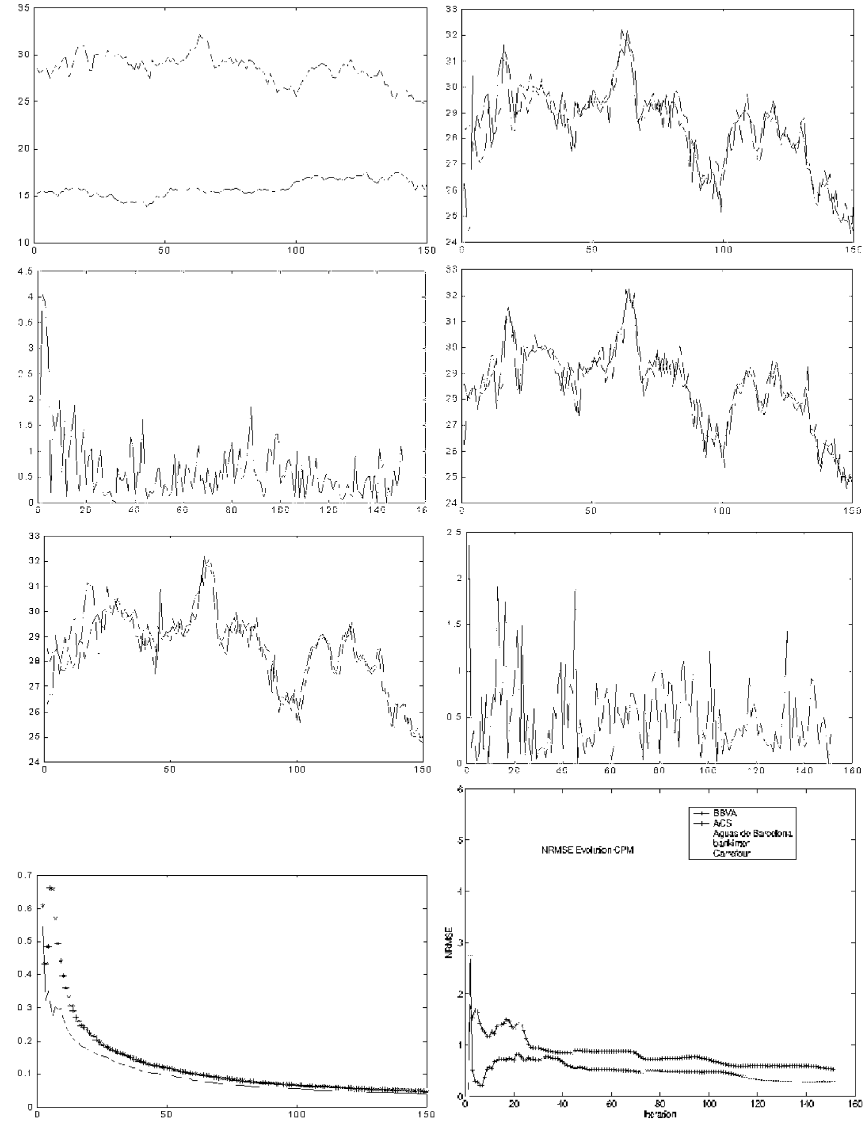

Fig. 3a-h Simulation results. **a** *top*: real series ACS; *bottom*: real series BBVA. **b** Real series and predicted ACS series with CPM. **c** Absolute value of the error with CPM. **d** Real series and predicted ACS series with CPM + GA. **e** Real series and predicted ACS series without exogenous data. **f** Absolute value of the error with CPM + GA. **g** NRMSE evolution for CPM (*dots*) and CPM + GA (*line*). **h** NRMSE evolution for selected stock indexes

Table 1 Correlation coefficients between real and predicted signals with different lags

Lag	ρ	Lag	ρ
0	0.89	0	0.89
+1	0.79	−1	0.88
+2	0.73	−2	0.88
+3	0.68	−3	0.82
+4	0.63	−4	0.76
+5	0.59	−5	0.71
+6	0.55	−6	0.66
+7	0.49	−7	0.63
+8	0.45	−8	0.61
+9	0.45	−9	0.58
+10	0.44	−10	0.51

Table 2 Dynamics and values of the weights for the GA

b_{series}	T_1	T_2	T_3	T_4
b_1	0.8924	0.8846	0.8723	0.8760
b_2	0.2770	0.2359	0.2860	0.2634

is coherent (avoiding large fluctuations in stock values)

- The algorithm converges faster, as shown at the very beginning of the graph
- The forecasting results are better using GA, as shown in Fig. 3, which describes the evolution of the normalized mean square error

Finally, we carried out a simulation with nine indexes and computed the prediction function for five series, obtaining similar results, as presented in Table 2. In the complete model, we limited the input space dimension to three for the extra series and to five for the target series. The NRMSE depends on each series (data set) and target series (evolution). In this table, we also compare the ICA method versus CPM for a limited data set (70 iterations). The NRMSE of the indexes increases at the beginning of the process using the ICA method and converges to CPM NRMSE values when the data set is enlarged (estimators approach higher-order statistics). This effect is also observed when the dynamics of the series change suddenly, due to a new, independent event.

Due to the symmetric character of our forecasting model, it is possible to implement parallel programming languages (like PVM) to build a more general forecasting model for a set of series. We would launch the same number for "child" processes and banks; these would run forecasting vectors, which would be weighted by a square matrix with dimension equal to the number of series, **B**. The "parent" process would have the results of the forecasting process for the calculation of the error vector, in order to update the neuron resources. Therefore, we would utilize the computational cost of a forecasting function to calculate the rest of the series.

5 Conclusions

In addition to the above ideas, we conclude that our new forecasting model for time series is characterized by:

- The enclosing of external information. We avoid preprocessing and data contamination by applying ICA and PCA. Series are enclosed in the net directly.
- Forecasting results are improved by means of hybrid techniques using well known techniques like GA.
- The possibility of implementation in parallel programming languages (e.g., "parallel virtual machine" (PVM)), and the better performance and lower computational time achieved by using a neuronal matrix architecture.

6 Appendix: theoretical background

The purpose of this section is twofold: on the one hand, we discuss in detail the GA implemented for parameter fitting. We model the GA as an inhomogeneous Markov chain [26] exhibiting the features and properties of our genetic operators. On the other hand, we discuss further the endogenous model used as a "kernel" for time-series forecasting.

6.1 Description of GA

As noted in Sect. 3, CPM uses a GA for b_I parameter fitting. A GA can be modeled by means of a *time inhomogeneous Markov*chain [26], obtaining interesting properties related with weak and strong ergodicity, convergence, and the distribution probability of the process [27]. In the latter reference, a canonical GA is constituted operations of parameter encoding, population initialization, crossover, mutation, mate selection, population replacement, fitness scaling, etc., proving that, with these simple operators, a GA does not converge to a population containing only optimal members. However, some GAs converge to the optimum, e.g., the *elitist GA* [28] and those which introduce *reduction operators* [29].

We have borrowed the notation mainly from [30], where the model for GAs is an inhomogeneous Markov chain model of probability distributions (**S**) over the set of all possible populations of a fixed finite size. Let **C** be the set of all possible creatures in a given world (vectors of dimension equal to the number of extra series) and a function f:**C** $\rightarrow R^{+}$. The task of the GAs is to find an element, $c\in$**C**, for which $f(c)$ is maximal. We encode creatures into genes and chromosomes or individuals as strings of length ℓ of binary digits (size of alphabet A is $a=2$) using one-complement representation; other encoding methods are also possible [22-24] or [25], where the value of each parameter is a gene and an individual is encoded by a string of real numbers instead of binary ones.

In the initial population generation step (choosing randomly $p \in \wp_N$, where $\wp_N$ is the set of populations, i.e., the set of N-tuples of creatures containing $a^{L \equiv \; N\cdot\ell}$ elements), we assume that creatures lie in a bounded region [0,1] (at the edge of this region, we can reconstruct the model without exogenous data). After the initial population, p, has been generated, the fitness of each chromosome, c_i, is determined via the function:

$$f(c_i) = \frac{1}{e(c_i)} \tag{7}$$

where e is an error function (i.e., the square error sum in a set of neural outputs, solving the convergence problem in the optimal solution by adding a positive constant to the denominator).

The next step in the canonical GA is to define the selection operator. New generations for mating are selected, depending on their fitness function values, using *roulette wheel selection.* Let $p=(c_1,...,c_N) \in\wp_N$, $n\in\mathscr{N}$, and f be the fitness function acting in each component of p. Scaled fitness selection of p is a lottery for every position $1 \le I \le N$ in population p, such that creature c_j is selected with probability:

$$\frac{f_n(p,j)}{\sum_{i=1}^{N} f_n(p,i)} \tag{8}$$

thus, proportional fitness selection can be described by column stochastic matrices $\mathbf{F}_n$, $n\in\mathscr{N}$, with components:

$$\langle q, \mathbf{F}_n p\rangle = \prod_{i=1}^{N} \frac{n(q_i) f_n(p,q_i)}{\sum_{j=1}^{N} f_n(p,j)} \tag{9}$$

where $p,q\in\wp_N$ so $p_i,q_i \in\mathbf{C}$, $\langle\cdots\rangle$ denotes the standard inner product, and $n(d_i)$ is the number of occurrences of q_i in p.

Once the two individuals have been selected, an elementary crossover operator $\mathbf{C}(K,P_c)$ is applied (setting the crossover rate at a value, i.e., $P_c \to 0$, which implies children similar to parent individuals), which is given (assuming even N) by:

$$\mathbf{C}(K,P_c) = \prod_{i=1}^{N/2} ((1-P_c)\mathscr{I} + P_c\mathbf{C}(2i-1,2i,k_I)) \tag{10}$$

where $\mathbf{C}(2i-1,2i,k_I)$ denotes the elementary crossover operation of c_I, c_j creatures at position $1 \le k \le \ell$ and $\mathscr{I}$ is the identity matrix, to generate two offspring (see [27] for further properties of the crossover operator).

The mutation operator, $\mathbf{M}_{\mathbf{P}m}$, is applied (with probability $\mathbf{P}_m$) independently at each bit in a population $p\in\wp_N$, to avoid premature convergence (see [22] for further discussion). The multi-bit mutation operator with change probability following a *simulated annealing* law with respect to the position $1 \le i \le L$ in $p\in\wp_N$:

$$\mathbf{P}_{m(i)} = \mu\cdot\exp\left(\frac{-\operatorname{mod}\left\{\frac{i-1}{N}\right\}}{\emptyset}\right) \tag{11}$$

where $\varnothing$ is a normalization constant and μ, the change probability at the beginning of each creature p_i in population p, can be described as a positive stochastic matrix in the form:

$$\langle q,\mathbf{M}_{\mathbf{P}_m}p\rangle = \mu^{\Delta(p,q)}\exp\left(-\sum_{\operatorname{dif}(i)}^{\Delta(p,q)} \frac{\operatorname{mod}\left\{\frac{i-1}{N}\right\}}{\emptyset}\right)\cdot \prod_{\operatorname{equ}(i)}^{L-\Delta(p,q)} \times\left[1-\mu\cdot\exp\left(\frac{\operatorname{mod}\left\{\frac{i-1}{N}\right\}}{\emptyset}\right)\right] \tag{12}$$

where $\Delta(p,q)$ is the Hamming distance between p and $q\in\wp_N$, dif(i), resp. equ(i) is the set of indexes where p and q are different resp. equal. From Eq. 12 and observing how the matrices act on populations, we can write:

$$\mathbf{M}_{\mathbf{P}_m} = \prod_{\lambda=1}^{N}\left(\left[1-\mu\cdot\exp\left(\frac{\operatorname{mod}\left\{\frac{\lambda-1}{N}\right\}}{\emptyset}\right)\right]1 + \mu\cdot\exp\left(\frac{\operatorname{mod}\left\{\frac{\lambda-1}{N}\right\}}{\emptyset}\right)\hat{m}^1(\lambda)\right) \tag{13}$$

where $\hat{m}^1(\lambda)=\mathbf{1}\otimes\mathbf{1}\ldots\otimes\overbrace{\hat{m}^1}^{\lambda}\otimes\ldots\mathbf{1}$ is a linear operator on $V_\wp$, the free vector space over A^L and $\hat{m}^1$ is the linear 1-bit mutation operator on V_1, the free vector space over A. The latter operator is defined acting on the alphabet as:

$$\langle\hat{a}(\tau'),\hat{m}^1\hat{a}(\tau)\rangle = (a-1)^{-1}, \quad 0\leqslant\tau'\neq\tau\leqslant a-1 \tag{14}$$

i.e., the probability of changing a letter in the alphabet once mutation occurs, with probability equal to $L\mu$.

The spectrum of $\mathbf{M}_{\mathbf{P}m}$ can be evaluated according to the following expression:

$$\operatorname{sp}(\mathbf{M}_{\mathbf{P}_m}) = \left\{\left(1-\frac{\mu(\lambda)}{a-1}\right)^{\lambda};\quad \lambda\in[0,L]\right\}$$

$$\text{where } \mu(\lambda) = \exp\left(\frac{-\operatorname{mod}\frac{\lambda-1}{N}}{\emptyset}\right). \tag{15}$$

The operator presented in Eq. 13 has similar properties to the constant multi-bit mutation operator, $\mathbf{M}_\mu$, presented in [27]. $\mathbf{M}_\mu$ is a contracting map in the sense presented in [30]. It is easy to prove that $\mathbf{M}_{\mathbf{P}m}$ is a contracting map too, by using the Corollary B.2 in [27] and the eigenvalues of this operator (Eq. 15).

We can also compare the coefficients of ergodicity:

$$\tau_r(\mathbf{M}_{\mathbf{P}_m})<\tau_r(\mathbf{M}_\mu) \tag{16}$$

where $\tau_r(\mathbf{X}) = \max\left\{\|\mathbf{X}_v\|_r : v\in\mathscr{R}^n,\ v\perp e \text{ and } \|v\|_r=1\right\}$.

Mutation is more likely at the beginning of the string of binary digits (“small neighborhood philosophy”). In order to improve the speed of convergence of the algorithm, we have included mechanisms such as elitist strategy (reduction operator [31]) in which the best individual in the current generation always survives into the next (a further discussion of the reduction operator, $\mathbf{P}_R$, can be found in [32]).

Finally, the GA is modeled, at each step, as the stochastic matrix product acting on probability distributions over the populations:

Fig. 4 Pseudo-code of GA

```
Initialize Population
i=0
while not stop do
  do N/2 times
    Select two mates from $\mathbf{p_i}$
    Generate two offspring using crossover operator
    Mutate the two children
    Include children in new generation $\mathbf{p_{new}}$
  end do
  Build population $\hat{\mathbf{p}}_\mathbf{i} = \mathbf{p_i} \cup \mathbf{p_{new}}$
  Apply Reduction Operators (Elitist Strategies) to get $\mathbf{p_{i+1}}$
  i=i+1
end
```

$$\mathbf{S}_{\mathbf{P}^n_m,\mathbf{P}^n_c} = \mathbf{P}^n_R \cdot \mathbf{F}_n \cdot \mathbf{C}^k_{\mathbf{P}^n_c} \cdot \mathbf{M}_{\mathbf{P}^n_m} \tag{17}$$

As shown above, the GA used in the forecasting function (Eq. 2) has an absolute error value start criterion (i.e., $error > uga = 1.5$). Once it starts, it uses the values (or individual) found to be optimal (elite) the last time, and applies a local search (using the selected mutation and crossover operators) around this elite individual. Thus, we perform an efficient search around an individual (set of b_Is) in which one parameter is more relevant than the others.

The computational time depends on the encoding length and the number of individuals and genes. Because of the probabilistic nature of the GA-based method, the proposed method almost converges to a global optimal solution on average. In our simulation, no non-convergent case was found. Figure 4 shows the GA-pseudocode and Fig. 1 completes the iterative procedure implemented for the overall prediction system including GA.

6.2 Endogenous learning machine

The purpose of this section is to introduce the foundations of SVMs [33] and their connection with RT [34] in order to show the new on-line algorithm for time-series forecasting.

SVMs are learning algorithms based on the structural risk minimization principle [35] (SRM), characterized by the use of the expansion of SV "admissible" kernels and the sparsity of the solution. They have been proposed as a technique in time-series forecasting [36, 37] and have addressed the overfitting problem present in classical neural networks, thanks to their high capacity for generalization. The solution for SVM prediction is achieved by solving the constrained quadratic programming problem. SV machines, thus, are non-parametric techniques, i.e., the number of base functions is unknown beforehand. The solution of this complex problem in *real-time applications* can be extremely complicated because of high computational time demands.

SVMs are essentially regularization networks (RN) with the kernels being Green's function of the corresponding regularization operators [38]. Using this connection, with a suitable choice of regularization operator (based on SVM philosophy), we should obtain a parametric model that is very resistant to the overfitting problem. Our parametric model is a RAN [3] characterized by the control of neural resources and by the use of matrix decompositions, i.e., singular value decomposition (SVD) and QR decomposition to input selection and neural pruning [4].

The following text is organized, thus: in the next subsection, we give a brief overview of the basic VC theory. The SV algorithm and its connection to RT theory is then presented and, finally, the new on-line algorithm is described.

6.2.1 Foundations of VC theory

A general notion of functional approximation problems[1] can be described as follows:

Let $\Delta \equiv \{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$ be a set of independent and identically distributed training samples with unknown probability distribution function, $F(x,y)$. The learning problem is that of choosing, from the given set of functions, $f(x,\alpha_\ell)$, $\alpha \in \Lambda$, where Λ is a set of parameters, the one that best approximates the output, y, of the system. Thus, the selection of the desired function must be based on the training set, Δ, i.e., applying the empirical risk minimization principle:

$$\begin{aligned} R(\alpha_\ell) &\equiv \int L(y, f(x,\alpha))\mathrm{d}F(x,y) \\ &\simeq \frac{1}{\ell}\sum_{i=1}^{\ell}(y_i - f(x_i,\alpha))^2 = R_{\mathrm{emp}}(\alpha_\ell) \end{aligned} \tag{18}$$

where we substitute the loss function, $L(y,f(x,\alpha))$, measuring the discrepancy between the response, y, to a given input, x, and the solution, $f(x,\alpha)$, for a specific loss which forms the least squares method. Under certain conditions, the functional empirical risk converges towards the expected risk and, hence, the approximation in Eq. 18 holds, i.e., $\ell \to \infty$. However, under small sample sizes, non-convergence may occur and the overfitting problem could arise [39]. This is avoided by introducing a regularization term [34] to limit the com-

[1] Before discussing this problem, we prove the existence of an exact representation for the continuous function in terms of simpler functions using Kolmogorov's theorem.

plexity of the loss function class arising from the problem of model selection [39].

The theory on controlling the generalization ability of learning machines is devoted to constructing a new, inductive principle to minimize the functional risk and to control the complexity of the loss function class. This is a major task, as noted above, whenever a small sample of training instances "ℓ" is used [33]. To construct learning methods, we use the bounds found by Vapnik:

$$R\left(\alpha_{\ell}^{k}\right) \leqslant R_{\mathrm{emp}}\left(\alpha_{\ell}^{k}\right) + \Re\left(\frac{\ell}{h_k}\right) \tag{19}$$

where R is the actual risk, R_{emp} is the empirical risk depending on the samples, $\Re$ is the confidence interval, α_ℓ^{k}[2] is the set of selected parameters that defines the class of approximation functions, and $h$is the VC dimension[3]. In order to minimize the right-hand side of inequality 19, we apply the SRM principle as follows:

Let $\mathscr{L}_1 \subset \mathscr{L}_2 \subset \cdots \subset \mathscr{L}_{\mathrm{k}} \cdots$ be a nested "admissible"[4] family of loss function classes with a finite VC dimension denoted by "h_i" with $i = 1,...,k$, for a given set of observations, Δ. The SRM principle chooses the suitable class, $\mathscr{L}_k$(and the function $L(x,\alpha_\ell^k)$, minimizing the guaranteed risk (right-hand side of inequality 19. In other words, the higher the complexity in the class function, the lower the empirical risk with the higher confidence interval (the second term in the bounds of the expected risk).

6.2.2 *Support vector machines and regularization theory*

The SV algorithm is a non-linear generalization of the generalized portrait developed in the 1960s by Vapnik and Lerner [40]. The basic idea in SVM for regression and function estimation is to use a mapping function, **Φ**, from the input space, $\mathscr{F}$, into a high-dimensional feature space, $\mathscr{F}$, and then to apply a linear regression. Thus, the standard linear regression transforms into:

$$f(x) = \langle \omega \cdot \Phi(x) \rangle + b \tag{20}$$

where $\Phi : \chi \rightarrow \mathscr{F}$, $b$is a bias or threshold, and $\omega \in \mathscr{F}$is a vector defining the function class. The target is to determine ω, i.e., the set of parameters in the neural network, minimizing the regularized risk expressed as:

$$R_{\mathrm{reg}}[f] = R_{\mathrm{emp}}[f] + \lambda \|\omega\|^2 \tag{21}$$

thus, we are enforcing "flatness" in the feature space, that is, we seek small ω. Note that Eq. 21 is very common in RN with a certain second term.

The SVM algorithm is a suitable way of solving the minimization of Eq. 21, which can be expressed as a quadratic programming problem using the formulation stated in [33]:

$$\text{minimize} \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{\ell}\left(\xi_i + \xi_i^*\right), \tag{22}$$

given a suitable loss function $L(\cdot)$ [5], a constant $C \geq 0$ and with variables ξ_I, $\xi_I^* \geq 0$. The optimization problem is solved by constructing a Lagrange function, introducing dual variables, using Eq. 22 and the selected loss function.

Once it is uniquely solved, we can write the vector ω in terms of the data points as follows:

$$\omega = \sum_{i=1}^{\ell}\left(\alpha_i - \alpha_i^*\right)\Phi(x_i) \tag{23}$$

where α_i and α_i^* are the solutions of the above-mentioned quadratic problem. Once this problem, with high computational demands[6], is solved, we introduce Eq. 23 into Eq. 20 and obtain the solution in terms of dot products:

$$f(x) = \sum_{i=1}^{\ell}\left(\alpha_i - \alpha_i^*\right)\langle \Phi(x_i) \cdot \Phi(x) \rangle + b \tag{24}$$

At this point, we use a strategy to avoid computing the dot product in the high-dimensional feature space in Eq. 24, replacing it with a kernel function that satisfies Mercer's condition. Mercer's theorem guarantees the existence of this kernel function:

$$f(x) = \sum_{i=1}^{\ell} h_I \cdot k(x_I, x) + b \tag{25}$$

where $h_i \equiv \left(\alpha_i - \alpha_i^*\right)$ and $k(x_i, x) = \langle \Phi(x_i) \cdot \Phi(x) \rangle$. Finally, we note, regarding the sparsity of the SV expansion (Eq. 24), that only the elements satisfying $|f(x_i) - y_i| \geq \epsilon$, where ϵ is the standard deviation of $f(x_i)$ from y_i (see selected loss function), have non-zero Lagrange multipliers, α_i and α_i^*. This can be proved by applying Karush-Kuhn-Tucher (KKT) conditions [41] to the SV dual optimization problem.

RT appears in the methods described for solving *ill-posed problems* [34]. In RN, we minimize an expression similar to Eq. 21. However, the search criterion is to enforce smoothness (instead of flatness) for the function in the input space (instead of the feature space). Thus, we have:

$$R_{\mathrm{reg}}|f| = R_{\mathrm{emp}}|f| + \frac{\lambda}{2}\left\|\hat{P}f\right\|^2 \tag{26}$$

[2] The subindex k is related to the structure or subset of loss functions used in the approximation.

[3] Roughly speaking, the VC dimension, h, measures how many training points can be separated for all possible labeling using functions of the class.

[4] In the strict sense presented in [33], that is, they are bounded functions or satisfy a certain inequality.

[5] For example, Vapnik's ϵ insensitive loss function [33]: $L(f(x) - y) = \begin{cases} |f(x) - y| - \varepsilon & \text{for } |f(x) - y| \geqslant \varepsilon \\ 0 & \text{otherwise} \end{cases}$

[6] This calculation must be performed several times during the process.

where $\hat{P}$ denotes a regularization operator in the sense of [34], mapping from the Hilbert space, H, of functions to a dot product space, D, such that $\langle f, g\rangle \; \forall f, g \in H$ is well defined. Applying Frèchet's differential[7] to Eq. 26 and the concept of Green's function of $\hat{P} * \hat{P}$, we have:

$$\hat{P} * \hat{P} \cdot G\left(x_i, x_j\right) = \delta\left(x_i - x_j\right) \tag{27}$$

(here, δdenotes Dirac's δ, that is $\langle f,\ \delta(x_i)\rangle = f(x_i)$), and, hence [4]:

$$f(x) = \lambda \sum_{i=1}^{\ell} \left[y_i - f(x_i)\right]_{\varepsilon} \cdot G(x, x_i) \tag{28}$$

The correspondence between SVM and RN (Eqs. 25 and 28) is proved if and only if Green's function, G, is an "admissible" kernel in the terms of Mercer's theorem, i.e., if we can write G as:

$$G\left(x_i, x_j\right) = \left\langle \Phi(x_i), \Phi\left(x_j\right)\right\rangle \text{ with } \Phi : x_i \rightarrow \left(\hat{P}G\right)(x_i, .) \tag{29}$$

A similar proof of this connection can be found in [38]. Hence, given a regularization operator, we can find an admissible kernel such that the SV machine using it will enforce flatness in the feature space and minimize Eq. 26. Moreover, given an SV kernel, we can find a regularization operator such that the SVM can be seen as an RN.

6.2.3 *On-line algorithm using regularization operators*

In this section, we describe a new on-line RN based on "RAN" algorithms[8] [3], which consist of a network using RBFs, a strategy for:

- Allocating new units (RBFs), using a two-part novelty condition [3]
- Input space selection and neural pruning using matrix decompositions such as SVD and QR with pivoting [4] together with a learning rule based on SRM, as discussed in the previous sections. Our network has one layer, as stated in Eq. 25. In terms of RBFs, the latter equation can be expressed as:

$$f(x) = \sum_{i=1}^{N(t)} h_i \cdot \exp\left(-\frac{\left\|x(t) - x_i(t)\right\|^2}{2\sigma_i^2(t)}\right) + b \tag{30}$$

where $N(t)$ is the number of neurons, $x_i(t)$ is the center of the neurons, and $\sigma_i(t)$ is their radius at time t. In order to minimize Eq. 26, we propose a regularization operator based on SVM philosophy. We enforce flatness in the feature space, as described in Sect 6.2, using the regularization operator, $\left\|\hat{P}f\right\|^2 \equiv \|\omega\|^2$, and, thus:

$$R_{\text{reg}}|f| = R_{\text{emp}}|f| + \frac{\lambda}{2} \sum_{i,j=1}^{N(t)} h_i h_j k\left(x_i, x_j\right) \tag{31}$$

We assume that $R_{\text{emp}} = (y - f(x))^2$ and minimize Eq. 31 by adjusting the centers and radii (gradient descend method, $\Delta\chi = -\eta\partial R|f|/\partial\chi$ with simulated annealing):

$$\Delta x_i = -2\frac{\eta}{\sigma_i}(x - x_i)h_i(f(x) - y)k(x, x_i) + \alpha \sum_{i,j=1}^{N(t)} h_i h_j k\left(x_i, x_j\right)\left(x_i - x_j\right) \tag{32}$$

and

$$\Delta h_i = \tilde{\alpha}(t) f(x_i) - \eta(f(x) - y)k(x, x_i) \tag{33}$$

where $\alpha(t)$ and $\tilde{\alpha}(t)$ are scalar-valued "adaptation gains", related to a similar gain used in the stochastic approximation processes; as in these methods, it should decrease in time. The second summand in Eq. 32 can be evaluated in several regions, inspired by the "divide-and-conquer" principle and used in unsupervised learning, i.e., competitive learning in self-organizing maps [42] or in expert SVMs [43]. This is necessary because of the volatile nature of time series, i.e., the dynamics of stock returns vary between different regions, leading to gradual changes in the dependence between the input and output variables. Thus, the super-index in the latter equation is redefined as:

$$N_c(t) = \left\{s_i(t) : \|x(t) - x_i(t)\| \leqslant \rho\right\} \tag{34}$$

the set of neurons close to the current input. The structure of the algorithm is shown below as pseudocode, including the set of initial parameters:

```
program online-algorithm
(Note: to denotes current iteration; k denotes
prediction horizon);
Initialize parameters and variables
Build input Toeplitz matrix A using (3W-1)
input values
Input space selection: determine Np relevant
lags L using SVD and QR_wp [8]
Determine input vector: x = x(to-k-L(1))
while (true)
if (n_rbfs > 0)
Compute f(x)
Find nearest RBF:|x-x_dmin|
else
f(x) = x(to-k-1)
Calculate error: e = |f(x)-x(to)|
if (e > epsilon and |x-x_ d_min| > delta) [7]Add RBF
 with parameters:
x_ I = x, sigma_i = kappa*|x-c_dmin}|, h = e
else
Execute pruning (SVD & QR_wp to neural activations) [8]
Update parameters minimizing actual risk
(15}(16}
if (e > theta*epsilon and n_inpsmax_inps)
n_inps = n_inps + 1
Determine new lags: L = [L_1,L_2,...,L_Np]
Add rbf_add RBFs
to = to + 1end
```

[7] Generalized differentiation of a function: $\mathrm{d}R|f| = [(\mathrm{d}/\mathrm{d}\rho)R[f + \rho h]]$, where $h \in H$.

[8] The principal feature of these algorithms is the sequential adaptation of neural resources.

Table 3 Evolution of normalized root mean square error (NRMSE). Xth-S denotes the Xth-step prediction NRMSE on the test set (noisy Mackey-Glass with delay changing operation mode

Method	1st-S	25th-S	50th-S	75th-S	100th-S
NAPA_PRED	1.1982	0.98346	0.97866	0.91567	0.90985
Standard SVM	0.7005	0.7134	0.7106	0.7212	0.7216
SVM_online	0.7182	0.71525	0.71522	0.72094	0.7127

6.2.4 *Experiments*

The application goal of our network is to predict complex time series. We chose the high-dimensional chaotic system generated by the Mackey-Glass delay differential equation:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = -b \cdot x(t) + a \cdot \frac{x(t-\tau)}{1+x^{10}(t-\tau)} \qquad (35)$$

with $b=0.1$, $a=0.2$, and delay $t_d=17$. This equation was originally presented as a model of blood regulation and became popular in modeling time series benchmarks. We added two modifications to Eq. 35:

- Zero-mean Gaussian noise with standard deviation equal to 1/4 of the standard deviation of the original series
- Random dynamics changes in terms of delay (between 100 and 300 time steps) t_d= 17,23,30 We integrated the chaotic model using MatLab software on a Pentium III 850 MHz computer, obtaining 2,000 patterns. For our comparison, we used 100 prediction results from SVM_online (presented in this paper), standard SVM (with ϵ-insensitive loss), and NAPA_PRED (RAN algorithm using matrix decompositions being one of the best on-line algorithms available to date [4]). Clearly, there is a remarkable difference between the above on-line algorithm and SVM philosophy. Standard SVM and SVM_online achieve similar results for this set of data at the beginning of the process. In addition, there is a noticeable improvement in the last iterations because of the volatile nature of the series. The change in time delay, t_d, leads to gradual changes in the dependence between the input and output variables and, in general, it is hard for a single model including SVMs to capture such a dynamic input-output relationship inherent in the data. Focusing our attention on the on-line algorithm, we observe the better performance of the new algorithm, such as a lower number of neurons ("sparsity"), and improved input space dimension and forecasting results (Table 3).

References

1. Pollock DSG (1999) A handbook of time series analysis, signal processing and dynamics. Academic Press, San Diego, California
2. Box GEP, Jenkins GM, Reinsel GC (1994) Time series analysis: forecasting and control, 3rd edn. Prentice Hall, Englewood Cliffs, New Jersey
3. Platt J (1991) A resource-allocating network for function interpolation. Neural Comput 3(2):213-225
4. Salmerón-Campos M (2001) Predicción de series temporales can redes neuronales de funciones radiales y técnicas de descomposición matricial. PhD thesis, Departamento de Arquitectura y Technología de Computadores, University of Granada
5. Moisés Salmerón, Julio Ortega, Carlos G. Puntonet, Alberto Prieto (2001) Improved RAN sequential prediction using orthogonal techniques. Neurocomputing 41:153-172
6. Masters T (1995) Neural, novel and hybrid algorithms for time series prediction. Wiley, New York
7. Back AD, Weigend AS (1997) Discovering structure in finance using independent component analysis. In: Proceedings of the 5th international conference on neural networks in the capital markets (Computational finance 1997), London, December 1997
8. Back AD, Trappenberg TP (2001) Selecting inputs for modelling using normalized higher order statistics and independent component analysis. IEEE Trans Neural Networ 12:(3):612-617
9. Hyvarinen A, Oja E (2000) Independent component analysis: algorithms and applications. Neural Networks 13:411-430
10. Bell AJ, Sejnowski TJ (1995) An information-maximization approach to blind separation and blind deconvolution. Neural Comput 7:1129-1159
11. Comon P (1994) Independent component analysis: a new concept. Signal Process 36:287-314
12. Amari S, Cichocki A, Yang HH (1996) A new learning algorithm for blind source separation. In: Advances in neural information processing systems 8. MIT Press, Cambridge, Massachusetts, pp 757-763
13. Puntonet CG (1994) Nuevos algoritmos de separación de fuentes en medios lineales. PhD thesis, Departamento de Arquitectura y Tecnología de Computadores, University of Granada
14. Theis FJ, Jung A, Puntonet C, Lang EW (2003) Linear geometric ICA: fundamentals and algorithms. Neural Comput 15(2):419-439
15. Puntonet CG, Mansour A, Ohnishi N (2002) Blind multiuser separation of instantaneous mixture algorithm based on geometrical concepts. Signal Process 82(8):1155-1175
16. Puntonet CG, Ali Mansour (2001) Blind separation of sources using density estimation and simulated annealing. IEICE Trans Fund Electr E84-A:2539-2547
17. Rodríguez-Álvarez M, Puntonet CG, Rojas I (2001) Separation of sources based on the partitioning of the space of observations. Lect Notes Comput Sci 2085:762-769
18. Górriz Sáez JM (2003) Algoritmos híbridos la modelización de series temporales con técnicas ar-ica. PhD thesis, Departamento de Ing de Sistemas y Aut Tec Eleectrónica y Electrónica , University of Cádiz
19. Back AD, Weigend AS (1997) Discovering structure in finance using independent component analysis. In: Proceedings of the 5th international conference on neural networks in the capital markets (Computational finance 1997), London, December 1997
20. Moody J, Darken CJ (1989) Fast learning in networks of locally-tuned processing units. Neural Comput 1:284-294
21. Hastie T, Tibshirani R, Friedman V (2000) The elements of statistical learning. Springer, Berlin Heidelberg New York
22. Michalewicz Z (1992) Genetic algorithms + data structures = evolution programs. Springer, Berlin Heidelberg New York
23. Szapiro T, Matwin S, Haigh K (1991) Genetic algorithms approach to a negotiation support system. IEEE Trans Syst Man Cybern 21:102-114
24. Chen S, Wu Y (1997) Genetic algorithm optimization for blind channel identification with higher order cumulant fitting. IEEE Trans Evolut Comput 1:259-264
25. Chao L, Sethares W (1994) Nonlinear parameter estimation via the genetic algorithm. IEEE Trans Signal Proces 42:927-935

26. Olle Haggstrom (1998) Finite Markov chains and algorithmic applications. Cambridge University Press, Cambridge, UK
27. Schmitt LM, Nehaniv CL, Fujii RH (1998) Linear analysis of genetic algorithms. Theor Comput Sci 200:101-134
28. Suzuki J (1995) A markov chain analysis on simple genetic algorithms. IEEE Trans Syst Man Cybern 25:655-659
29. Eiben AE, Aarts EHL, Van Hee KM (1991) Global convergence of genetic algorithms: a markov chain analysis, parallel problem solving from nature. Lect Notes Comput Sci 496:4-12
30. Schmitt LM (2001) Theory of genetic algorithms. Theoret Comput Sci 259:1-61
31. Lozano JA, Larrañaga P, Graña M, Albizuri FX (1999) Genetic algorithms: bridging the convergence gap. Theoret Comput Sci 229:11-22
32. Rudolph G (1994) Convergence analysis of canonical genetic algorithms. IEEE Trans Neural Networ 5:96-101
33. Vapnik V (1998) Statistical learning theory. Wiley, New York
34. Tikhonov AN, Arsenin VY (1997) Solutions of ill-posed problems. Winston, Washington, pp 415-438
35. Vapnik V, Chervonenkis A (1974) Theory of pattern recognition (in Russian). Nauka, Moscow
36. Muller KR, Smola AJ, Ratsch G, Scholkopf B, Kohlmorgen J (1999) Using support vector machines for time series prediction. In: Scholkopf B, Burges CJC, Smola AJ (eds) Advances in kernel methods-support vector learning. MIT Press, Cambridge, Massachusetts, pp 243-254
37. Muller KR, Smola AJ, Ratsch G, Scholkopf B, Kohlmorgen J, Vapnik V (1997) Predicting time series with support vector machines. In: Proceedings of the 7th international conference on artificial neural networks (CANN'97), Lausanne, Switzerland, May 1997, pp 999-1004
38. Smola AJ, Scholkopf B, Muller KR (1998) The connection between regularization operators and support vector kernels. Neural Networks 11:637-649
39. Muller KR, Mika S, Ratsch G, Tsuda K, Scholkopf B (2001) An introduction to kernel-based learning algorithms. IEEE Trans Neural Networ 12(2):181-201
40. Vapnik V, Lerner A (1963) Pattern recognition using generalized portrait method. Automat Rem Contr 24:774-780
41. Kuhn HW, Tucker AW (1951) Nonlinear programming. In: Proceedings of the 2nd Berkeley symposium on mathematical statistics and probabilistics. University of California Press, pp 481-492
42. Kohonen T (1990) The self-organizing map. P IEEE 78(9):1464-1480
43. Cao L (2003) Support vector machines experts for time series forecasting. Neurocomputing 51:321-339