Time Series Prediction Using Evolving Polynomial Neural Networks

A dissertation submitted to the
University of Manchester Institute of Science and Technology
for the degree of MSc

1999

Amalia Foka

Control Systems Centre
Department of Electrical Engineering & Electronics

DECLARATION

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

ABSTRACT

Real world problems are described by non-linear and chaotic processes which makes them hard to model and predict. The aim of this dissertation is to determine the structure and weights of a polynomial neural network, using evolutionary computing methods, and apply it to the non-linear time series prediction problem.

This dissertation first develops a general framework of evolutionary computing methods. Genetic Algorithms, Niched Genetic Algorithms and Evolutionary Algorithms are introduced and their applicability to neural networks optimisation is examined.

Following, the problem of time series prediction is formulated. The time series prediction problem is formulated as a system identification problem, where the input to the system is the past values of a time series, and its desired output is the future values of a time series. Then, the Group Method of Data Handling (GMDH) algorithms are examined in detail. These algorithms use simple partial descriptions, usually polynomials, to gradually build complex models. The hybrid method of GMDH and GAs, Genetics-Based Self-Organising Network (GBSON), is also examined.

The method implemented for the time series prediction problem is based on the GBSON method. It uses a niched generic algorithm to determine the partial descriptions of the final model, as well as the structure of the neural network used to model the time series to be predicted. Finally, the results obtained with this method are compared with the results obtained by the GMDH algorithm.

ACKNOWLEDMENTS

I would like to express my sincere gratitude to my supervisor Mr. P. Liatsis, for his interest, help and invaluable guidance given throughout the work for this project.

I would like to thank all my friends for being there for me and making Manchester a nice place to live in. Also, I would like to thank Thodoros and Anthoula for using their laptop to run the simulations that enabled me to save invaluable time.

This dissertation is dedicated to my parents, Fotis Fokas and Lambrini Foka, for believing in me and making it possible for me to get here. This dissertation is also dedicated to my sister, Elpiniki, and my brother, Chrysanthos, for contributing in their own way to support my studies at UMIST.

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Neural Networks

Neural networks [7] attempt to simulate the structure and functions of the brain and the nervous system of human beings. They are massively parallel networks of simple processors called artificial neurons. The artificial neurons are interconnected, and each connection is weighted. The weights of the connections store the knowledge of the network.

Neural networks are widely used in many application areas because of their advantages. They are massively parallel and hence they can have high computation rates. Their topology and weights are adaptive; therefore they are able to learn, which makes neural networks well suited for problem solving like prediction, system identification, optimisation, classification and vision [71]. They can effectively model complex non-linear mappings and a broad class of problems because of their non-parametric nature. Another important feature of neural networks is their intrinsic fault tolerance. Unlike Von Neumann architectures even if some neurons fail, the network can perform well because knowledge is distributed across all the elements of the network.

Neural networks can be classified into a number of categories according to the type of connections between the neurons. In *feedforward* networks, connections between neurons are forward, going from the input layer to the output layer. In *recurrent* (or *feedback*) networks, there are feedback connections between different layers of the network. Neural networks can also be classified according to the order of the input send to neurons. *First order* networks send weighted sums of inputs through the transfer functions. *Higher order / polynomial* networks send weighted sums of products or functions of inputs through the transfer functions.

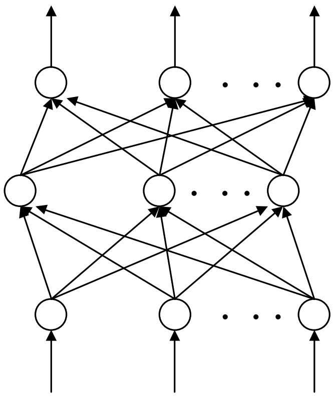


Figure 1.1 Feedforward neural network.

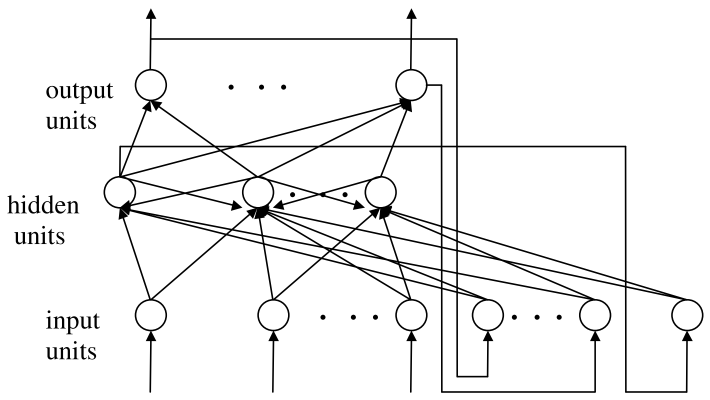


Figure 1.2 Recurrent neural network.

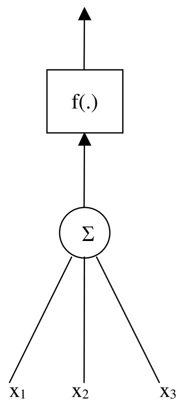


Figure 1.3 First order neural network.

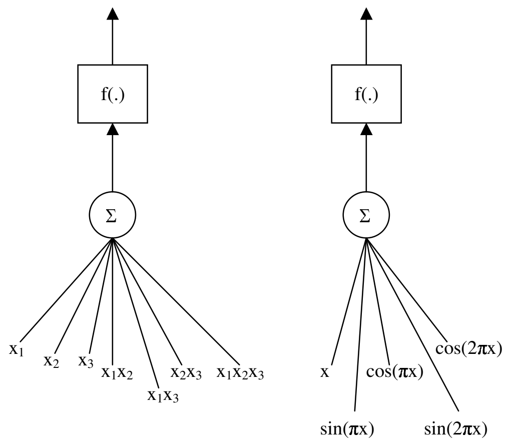


Figure 1.4 Higher order neural networks.

To find the optimal neural network structure for a solution to a particular problem, training algorithms have to be used. Training algorithms can be divided into two main categories: *supervised* and *unsupervised*. In supervised learning, a teacher is required that defines the desired output vector for each input vector. In unsupervised learning, the units are trained to discover statistically salient features of the inputs and learn to respond to clusters of patterns in the inputs.

1.2 Evolutionary Computing

Evolutionary computing methods try to mimic some of the processes observed in natural evolution. They are based on the Darwinian principle of natural evolution where the fittest survives.

Darwinian evolution [23] is a robust search and optimisation mechanism. Evolutionary computing methods can be applied to problems where heuristic solutions are not available or lead to unsatisfactory results. As a result, they may be used as optimisation algorithms for almost any purpose.

Evolutionary computing methods operate within a *population* of *individuals*, which are initially randomly selected. The individuals of a population represent potential solutions to a specific problem. The initial population evolves towards successively better solutions by the use of the processes of *reproduction*, *selection* and *mutation*. The *fitness* value of an individual gives a measure of its performance on the problem to be solved.

Evolutionary computing methods are currently divided into three main categories: *Genetic Algorithms*, *Evolutionary Algorithms* and *Genetic Programming*. There is a very thin line between Genetic Algorithms [32] and Evolutionary Computing [65], but because the original Genetic Algorithm introduced used only binary encoding of potential solutions, the classification of an algorithm is based on that. Thus, algorithms that use binary encoding are classified as Genetic Algorithms and those that use real valued vectors are classified as Evolutionary Computing algorithms. Genetic Programming [23] is a methodology for automatically generating computer programs.

1.3 Dissertation Aims and Objectives

The aim of this dissertation is to investigate the use of evolutionary computing methods in determining the topology and weights of polynomial neural networks, applied to the time series prediction problem.

Traditional training algorithms for neural networks, such as the backpropagation algorithm, are based on gradient descent. These algorithms follow the gradient of an error function to modify the network parameters. This method can often result to a suboptimal solution because these algorithms may stuck in local minima. The search for an optimal set of weights and topology of neural networks is a complex, combinatorial optimisation problem and evolutionary computing methods have been proved to outperform traditional training methods [2], [10], [11], [13], [14], [24], [49], [75].

Polynomial neural networks [14] are feedforward networks that use higher order correlations of their input components. This makes them attractive for use in system identification and modelling since they can perform non-linear mappings with only a single layer of units.

Polynomial neural networks architectures structured using evolutionary computing methods will be applied to the time series prediction problem. Traditional approaches to prediction were based either on finding a law underlying the given dynamic process or phenomenon or on discovering some strong empirical regularities in the observation of the time series. For the first case, if a law can be discovered and analytically described, e.g. by a set of differential equations, then by solving them we can predict the future if the initial conditions are completely known and satisfied. The problem with this approach is that usually information about dynamic processes is only partial and incomplete. In the second case, if there is a time series consisting of samples of a periodic process, it can be modelled by the superposition of sinusoids generated by a set of second order differential equations. In real-world problems though, regularities such as periodicity are masked by noise and some phenomena are described by chaotic time series in which the data seem random without apparent periodicities [14].

1.4 Dissertation Organisation

Chapter 2 deals with the basic concepts of *evolutionary computing methods*. It covers the topics of *Genetic Algorithms*, *Niched Genetic Algorithms* and *Evolutionary Algorithms*, and their basic operators. An overview of evolutionary computing methods in neural networks optimisation is also given.

Chapter 3 gives a definition of time series signals and some of their properties. The prediction of time series signals is formulated, and the procedure of prediction is presented. The most commonly used linear and non-linear models for time series are also summarised. Then, the Group Method of Data Handling (GMDH) is presented, along with its most common algorithms. Finally, the Genetics-Based Self-Organising Network (GBSON) method for time series prediction is outlined.

Chapter 4 presents how the time series prediction problem was implemented and the results of the simulations conducted on different time series. Following, the results of the simulations are compared with the results obtained using the GMDH algorithm.

Chapter 5 gives the conclusions of this dissertation and recommendations for future work in this area of research.

CHAPTER 2

EVOLUTIONARY COMPUTING TECHNIQUES & ALGORITHMS

2.1 Introduction

In this chapter, the various evolutionary computing techniques and algorithms will be introduced. First, the working principles of the Genetic Algorithm (GA) are outlined by presenting its operators and various strategies. Following, the Niched GA is presented and finally the Evolutionary Algorithm (EA) operators and strategies are reviewed.

2.2 Genetic Algorithms

Genetic algorithms are stochastic algorithms whose search methods are based on the mechanics of natural selection and natural genetics. John Holland [32] originally developed them. The aim of Holland's work was to develop a theory of adaptive systems that retain the mechanisms of natural systems. The features of natural systems of self-repair, self-guidance and reproduction intrigued early researchers in this field to be applied in problem solving.

Problem solving can be thought of as a search through a space of potential solutions. The desired output of such a search is the best solution. Thus, this task can be viewed as an optimisation process. Traditional optimisation methods such as *hill climbing* have been used in many applications but they require the existence of the derivative of an objective function and continuity over its domain. In real world problems, it is almost impossible to meet these requirements.

Random search optimisation schemes have also been used, but they lack efficiency. These conventional optimisation schemes are not robust for a broad field of problems. Genetic algorithms try to overcome the problem of robustness by being a directed search process using random choice as a tool.

2.2.1 Working principles

In a genetic algorithm, the first step is to define and code the problem to be solved. A typical single-variable optimisation problem can be outlined as

$$\text{Maximise } f(x) = x^2$$

$$\text{Variable bound: } x_{\min} \leq x \leq x_{\max}.$$

The problem is defined with the use of an objective function that indicates the *fitness* of any potential solution, and for the above problem is x^2. The decision variables are coded as a finite length string called *chromosome*, $A = a_l a_{l-1} \cdots a_1$, where l is the string length. The *alphabet* of a coding defines the possible values of the bit or *gene* a_i, i.e. in a binary coding the alphabet is $\{0, 1\}$. For example, if four-bit binary strings are used to code the variable x, the string ($0\ 0\ 0\ 0$) is decoded to the value x_{min}, the string ($1\ 1\ 1\ 1$) is decoded to the value x_{max}, and any other string is decoded to a value in the range (x_{min}, x_{max}), uniquely. In natural terminology, the values of the alphabet are called *alleles* and the position of the gene, indicated by i, is called *locus*. The choice of the string length l and the alphabet determine the accuracy of the solution and the computation time required to solve the problem [17]. The *principle of minimal alphabets* defines that the smallest alphabet that permits a natural expression of the problem should be selected.

Genetic Algorithms begin with a population of chromosomes created randomly. Following, the initial population is evaluated. Three main operators -*reproduction, crossover* and *mutation*- are used to evolve the initial population towards better solutions. The population is evaluated, and if the termination criteria are not met, the three main operators are applied again. One cycle of these operators and the evaluation procedure is known as a *generation* in GA terminology.

```
begin
     Choose a coding to represent variables;
     Initialise population;
     Evaluate population;
     repeat
          Reproduction;
          Crossover;
          Mutation;
          Evaluate population;
     until (termination criteria);
end.
```

Figure 2.1 Pseudocode for a simple genetic algorithm.

2.2.2 Operators

2.2.2.1 Reproduction

The reproduction operator is based on the Darwinian nature selection procedure, where the fittest survives. There exists a number of reproduction operators in GA literature [3], [28], [53], [55], but the essential idea in all reproduction operators is that above average chromosomes are picked from the current population and multiple copies of them are inserted in the mating pool.

The most common way of implementing the reproduction operator is by using *roulette wheel selection*. Each chromosome in the population is assigned a probability of reproduction, so that its likelihood of being selected is proportional to its fitness relative to the other chromosomes fitness values in the population. For a chromosome i with fitness f_i, in a population of n individuals, its probability of being selected for reproduction, p_i, is given by:

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i}.$$

The roulette wheel has a slot for each individual, sized according to its probability of being selected for reproduction. To create a new population, the roulette wheel is spun n times. The new population will have more copies of the individuals with high fitness values and less copies of the individuals with small fitness values.

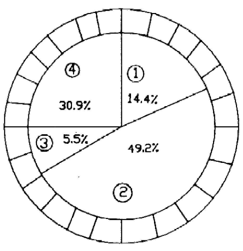


Figure 2.2 An example of the roulette wheel.

Another proposed implementation of the roulette wheel selection operator is with *Stochastic Universal Sampling* (*SUS*) [55]. SUS spins the wheel once but with *n* equally spaced pointers that are used to select the *n* parents.

The proportional selection methods described above are likely to cause premature convergence to a local optimum solution [53], [55]. This is due to the existence of super individuals whose fitness values are above average and have a large number of offspring in next generations. This prevents other individuals from contributing any offspring in next generations with the result of not exploring other regions of the search space. To address this problem, the method of *rank selection* [53], [55] has been proposed. In this method, the individuals in the population are ranked according to their fitness values and the selection is performed according to their rank, rather on their fitness value. A linear ranking selection method proposed by Baker [4] works by choosing the expected value *max* of the best individual in the population (rank = *n*), and setting the expected value *min* as the expected value of the worst individual (rank = 1). Then, the expected value of each individual *i* is given by

$$E(i) = min + (max - min)\frac{rank(i) - 1}{n}.$$

This function takes a linear curve through *max* such that the area under the curve equals the population size *n*.

Another common reproduction method is the *tournament selection* [27]. This method selects randomly some number *k* of individuals, and reproduces the best one from this set of *k* elements into the next generation. As the number *k* of individuals selected to compete increases, the selective pressure of this procedure increases.

Boltzmann tournament selection (*BTS*) [29] is motivated by the original tournament selection scheme and simulated annealing. BTS operates in such a way that samples over time become Boltzmann distributed. In this selection scheme, competition is performed between three members of the population. The first member *i*, is selected randomly and the second and third members, *j* and *k*, are selected in such a way that they differ by a constant *Θ* from the first member. Members *j* and *k* compete first, with member *j* probabilistically winning according to the function

$$\frac{1}{1+e^{\frac{f(k)-f(j)}{T}}}.$$

Then, the winner competes with member *i*. Member *i* wins with probability

$$\frac{1}{1+e^{\frac{f(i)-f(winner)}{T}}},$$

where *T* is BTS' s current temperature and $f(\cdot)$ is the fitness function.

In evolving rule-based systems, such as classifier systems, the method of *steady-state selection* [69] is used. In this method, only a few individuals are replaced in each generation. The fittest individual' s offspring of the current population replaces the least fit individuals, to form the new population in the next generation. The fraction of new individuals at each generation is referred as the *generation gap*.

2.2.2.2 Crossover

The main purpose of the crossover operator is to search the parameter space. The search needs to be performed in a way that the information stored in the parent strings is maximally preserved, because these parent strings are instances of good strings selected using the reproduction operator. Hence, the idea behind the crossover operator is to combine useful segments of different parents to form an *offspring* that benefits from advantageous bit combinations of both parents. The two parents are selected randomly from the population with probability p_c. The probability that an individual will be a parent for crossover, p_c, is usually between 0.6 and 0.95.

The simplest crossover operator is the one proposed by Holland, the *one-point crossover*. An integer position k along the chromosome is randomly selected, where $1 \le k \le l-1$ and l is the length of the chromosome. Exchanging all bits on the right side of the crossing point k, two new chromosomes are created, called offspring. So, if the two parent chromosomes are

$$A = 0\ 0\ 0 \mid 0\ 0$$
$$B = 1\ 1\ 1 \mid 1\ 1$$

and | denotes the randomly selected point k, the resulting offspring will be

$$A' = 0\ 0\ 0\ 1\ 1$$
$$B' = 1\ 1\ 1\ 0\ 0$$

The one-point crossover suffers from its *positional bias*, i.e. a strong dependence of the exchange probability on the bit positions, since as bit indices increase towards l, their exchange probability approaches one. Therefore, the search in the parameter space is not extensive, but the maximum information is preserved from parents to offspring. It has also been noted that the one-point crossover treats some loci preferentially, resulting to segments exchanged between the two parents always containing the endpoints of the strings.

To reduce positional bias and the endpoint effect, the *multi-point crossover* [18] was introduced. Multiple points are randomly chosen along the chromosome and each second segment between subsequent crossover points is exchanged. The most common multi-point crossover operator is with two points, shown schematically in Figure 2.3.

A further generalisation to the multi-point crossover is the *uniform crossover*. In uniform crossover, an exchange occurs at each bit position with probability p (typically $0.5 \le p \le 0.8$). In contrast to the one-point crossover, in the uniform crossover the search through the parameter space is very extensive but minimum information is preserved between parents and offspring.

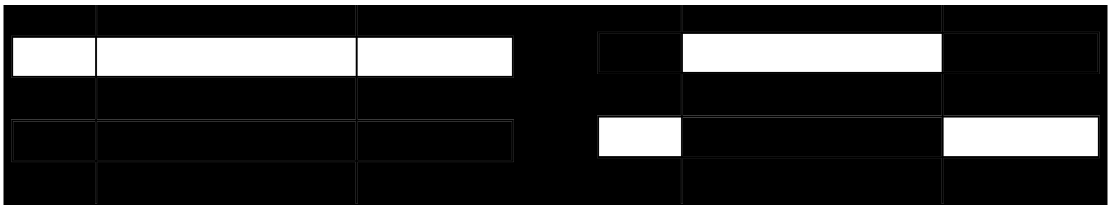


Figure 2.3 Two-point crossover.

Other proposed crossover operators are the *segmented crossover, shuffle crossover* and the *punctuated crossover.* The segmented crossover works like the multi-point crossover except that the number of the crossover points is replaced by a segment switch rate. This switch rate specifies the probability that a segment will end at any point in the string.

Shuffle crossover can be applied in conjunction with any other crossover operator. It first shuffles the bit positions of the parents randomly, then the strings are crossed, and finally unshuffles the strings. This operator has the ability of reducing the positional bias of standard crossover operators.

The punctuated crossover works also like the multi-point crossover but here the number and positions of the crossover points are self-adaptive. A number of bits are added to the chromosome that represent the number and position of the crossover points and thus the operator itself is subject to crossover and mutation.

More details about crossover schemes can be found in [3], [12], [15], [28], [53], [55].

2.2.2.3 Mutation

The mutation operator in genetic algorithms is a secondary operator that occasionally changes single bits of the chromosomes by inverting them. Its purpose is to keep diversity in the population. After the operators of reproduction and crossover have been applied, it is possible to lose genetic material that might be useful, i.e. a certain value at a certain bit position. The mutation operator protects against irrecoverable loss of useful genetic material.

The mutation probability per bit, p_m, is small and usually lies within the range [0.001, 0.01]. If there is a chromosome of length 10 and p_m is 0.1, then one bit of the chromosome will be mutated. So, if A = 1 0 0 0 1 1 0 0 1 0 and the bit to be mutated is the first, then after the mutation A′= 1 0 0 0 1 1 0 0 1 1.

More details about mutation schemes can be found in [3], [28], [53], [55].

2.2.3 Schemata

The theoretical foundations of genetic algorithms rely on a binary string representation of solutions and the *schemata* [28], [53]. A *schema* is a similarity template describing a subset of strings with similarities at certain bit positions.

To build a schema, a *don't care* symbol is introduced denoted by *. A string with fixed and variable symbols defines a schema. For example the schema {01**}, defined over the binary alphabet and the don't care symbol, describes the subset {0100, 0101, 0110, 0111} and the schema {011*} describes the subset {0110, 0111}. The schema {****} represents all strings of length 4.

It can be seen that a schema describes a number of strings. The number of strings a schema represents is given by 2^r, where r is the number of don' t care symbols, and each string of length l is matched by 2^l schemata. The *order* of a schema, denoted by $o(H)$, is defined as the number of the fixed positions in the schema and defines the speciality of a schema. The order of the schemata {010*} and {***1} is 1 and 3, respectively. The *defining length* of a schema, denoted by $\delta(H)$, is the distance between the first and the last fixed string positions and defines the compactness of information contained in a schema. The defining length of the schemata {0**1} and {*0**} is $\delta(H)$ = 4 - 1 = 3 and $\delta(H)$ = 2 - 2 = 0, respectively.

The use of schemata makes the search over a space of potential solutions directed. That is because when there is information available on a particular chromosome's fitness value, e.g. {1100}, there is also partial information about the schemata it can form, i.e. {1***}, {11**}, {1**0}, {***0} and so on. This characteristic is termed *implicit parallelism*, as it is through a single sample, information is gained with respect to many schemata.

The information gained from schemata can be useful in the formation of new generations after the operators of reproduction, crossover and mutation are applied. The reproduction operator simply ensures that schemata with high fitness have an increased number of samples in the next generation. If the number of strings in a population matched by a schema H at time t is denoted by $m(H, t)$, and its probability of being selected in the next generation is

$$p_H = \frac{f(H)}{\sum f(i)},$$

the expected number of strings matched by H in the next generation will be

$$m(H,t+1) = m(H,t) \cdot n \cdot f(H) \big/ \sum f(i).$$

By denoting the average fitness of the population as $\bar{f} = \sum f(i) \big/ n$, the above equation becomes

$$m(H,t+1) = m(H,t) \cdot f(H) \big/ \bar{f}.$$

From the above equation, it can be seen that schemata with above average fitness values will have more copies in the next generation, whereas schemata with below average fitness values will have fewer copies in the next generation.

The crossover operator has a different effect on schemata depending on their defining length. Schemata with long defining length are likely to be disrupted by the crossover operator, and on the contrary schemata with short defining length are most likely to remain unchanged. The probabilities of a schema to remain unchanged, $p_u(H)$, and to be disrupted, $p_d(H)$, are defined as [53]

$$p_u(H) = 1 - \frac{\delta(H)}{l-1} \text{ and } p_d(H) = \frac{\delta(H)}{l-1}.$$

When the mutation operator is applied with a low probability, it is unlikely to disrupt a schema.

These observations give rise to the *schema theorem* and the *building block hypothesis*.

Schema Theorem [28]: Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

Building Block Hypothesis [28]: A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high performance schemata, called building blocks.

2.2.4 A simple genetic algorithm

The problem of maximising the function $f(x) = x^2$, where x varies between 0 and 31, will be considered. To code this problem a binary string of length 5 is sufficient to represent the values from 0 to 31.

First, the initial population has to be generated. For a population of size 4, an unbiased coin is tossed 20 times, where heads equals 1 and tails equals 0. Each individual in the population has to be decoded and its fitness has to be evaluated. The initial population generated is shown in Table 2.1. In this example, the decoding is straightforward and for the third string its value is $0 \approx 2^0 + 0 \approx 2^1 + 0 \approx 2^2 + 0 \approx 2^3 + 1 \approx 2^4 = 16$ and its fitness value is $f(x) = 16^2 = 256$.

The iterative process of the genetic algorithm starts with reproduction that is performed with the roulette wheel selection. The roulette wheel is spun four times to give the four new members of the population. The probability of each individual to be selected and its expected count of times to be reproduced are shown in Table 2.1. The actual count from the roulette wheel is also shown in

Table 2.1. The result from the roulette wheel is the expected one where individuals with high fitness have more copies in the new population and individuals with low fitness die off.

String No.	Initial Population	x value	$f(x)=x^2$	Pselect $\frac{f_i}{\sum f}$	Expected Count $\frac{f_i}{\bar{f}}$	Actual Count
1	0 1 0 1 1	11	121	0.29	1.16	1
2	0 1 0 1 0	10	100	0.24	0.96	1
3	1 0 0 0 0	16	256	0.62	2.46	2
4	0 0 0 1 1	3	9	0.02	0.09	0
Sum			414	1.00	4.00	4.0
Average			104	0.25	1.00	1.0
Max			256	0.62	2.46	2.0

Table 2.1 The initial population generated.

The probability of crossover in this example is equal to 1.0, so all the members of the new population will be parents. The couples of parents and the points of crossover are chosen randomly. The result of crossover is shown in Table 2.2.

The probability of mutation in this example is assumed to be equal to 0.001. For the total number of bits in the population the expected mutations are equal to 5≈4≈0.001 = 0.02, thus no mutations are expected.

The results of a single generation are shown in Table 2.2. The average fitness of the members of the population has increased from 104 to 203 and the maximum fitness value has also increased from 256 to 361. Iterating this process will eventually result in a string with all ones that has maximum fitness value.

Mating Pool after Reproduction	Mate (randomly selected)	Crossover Site (randomly selected)	New Population	*x* value	$f(x)=x^2$
0 1 0 | 1 1	3	3	0 1 0 0 0	8	64
0 1 | 0 1 0	4	2	0 1 0 0 0	8	64
1 0 0 | 0 0	1	3	1 0 0 1 1	19	361
1 0 | 0 0 0	2	2	1 0 0 1 0	18	324
Sum					813
Average					203
Max					361

Table 2.2 The population after the first generation.

2.2.5 GA-based methods for neural networks optimisation

2.2.5.1 GA-simplex operator and granularity encoding

Maniezzo [49] proposed a method for evolving the topology and weight distribution of neural networks, based on the *GA-simplex* operator and *granularity* encoding that allows the algorithm to evolve the length of the coding string. The variable-length binary coding allows its minimisation through evolution, and thus minimisation of the search space.

This representation uses the network nodes as basic functional units and encodes all the information relevant for a node in nearby positions. The first byte of the string specifies the *granularity*; the number of bits according to which the weights of the present connections have been specified. Each of the next bytes represents a node in the network. The first bit is the *connectivity bit* that is present only if a connection exists. When a connectivity bit is present, it is followed by the binary encoding of the relative weight.

The operators applied in this method are the standard roulette wheel selection, single-point crossover, point mutation and the GA-simplex operator. The GA-simplex operator works on three individuals of the population x_1, x_2, and x_3 to generate a new individual x_4. It is implemented in four steps:

```
Step 1: Rank the three parents by fitness value; suppose
            $FF(x_1) \geq FF(x_2) \geq FF(x_3)$
Step 2: FOR each ith bit of the strings
Step 3:         IF $x_{1i} = x_{2i}$, THEN $x_{4i} = x_{1i}$
Step 4:         ELSE $x_{4i}$ = negate ($x_{3i}$)
```

The fourth step of the operator can be applied either deterministically or with a probability.

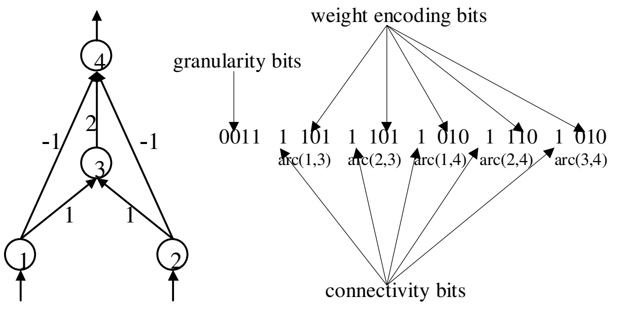


Figure 2.4 Granularity encoding.

2.2.5.2 Population-Based Incremental Learning

Population-Based Incremental Learning (PBIL) [5] is a modification of the standard genetic algorithm. The objective of PBIL is to create a real valued probability vector that specifies the probabilities of having ' 1' in each bit position of high evaluation solutions. The probability vector can be considered as a prototype for high evaluation vectors, for the solution space being explored.

Before the evolution of solution vectors starts, the probability vector is initialised with the value of 0.5 to all its bit positions, so the probability of generating 0 and 1 is equal. Then, the initial population of potential solutions is generated according to the probability vector. The population is evaluated, and the probability vector is updated towards the best solution as

$$probability_{i,t+1} = (probability_{i,t}(1.0 - LR)) + best\_member \cdot LR),$$

and away from the worst solution as

$$\begin{aligned} probability_{i,t+1} = (probability_{i,t}(1.0 - negative\_LR)) + \\ best\_member \cdot negative\_LR) \end{aligned},$$

where *LR* denotes the learning rate, how fast to exploit the search performed, and *negative_LR* denotes the negative learning rate, how much to learn from negative examples.

In this method, a crossover operator is not applied. Only the mutation operator is applied to the probability vector, to prevent it from converging to extreme values without performing an extensive search. The mutation operator is applied with probability p_m, at each point of the probability vector. Each vector position is shifted in a random position by $shift_m$, that defines the amount a mutation alters the value in the bit position. The probability vector after mutation will be

$$probability_{i,t+1} = probability_{i,t}(1.0 - shift_m) + direction \cdot shift_m,$$

where *direction* can take the value of 0 or 1.

The new population of potential solutions is formed according to the altered probability vector. The mechanism of *elitist selection* [53] is performed before the probability vector is altered again. During elitist selection the best member of the previous population replaces the worst member of the current population. This mechanism is used in case a better solution is not produced in the current population.

2.2.5.3 GA / Fuzzy approach

Carse and Fogarty [10], [11] have proposed a genetic algorithm / fuzzy logic based method for evolving Radial Basis Function neural networks. In their implementation, the 2N-tuple (C_{i1}, R_{i1},..., C_{ij}, R_{ij}, ..., C_{iN}, R_{iN}) represents potential solutions, where N is the number of inputs, and (C_{ij}, R_{ij}) is the centre and width of the Gaussian radial basis function of the hidden node, for the *j*th input variable.

A modified two-point crossover operator is applied, where the two crossover points X_{1j} and X_{2j} are chosen as

$$X_{1j} = MIN_j + (MAX_j - MIN_j) \approx Rd_1$$
$$X_{2j} = X_{1j} + (MAX_j - MIN_j) \approx Rd_2 \quad .$$

Rd_1 and Rd_2 are chosen randomly in the range [0,1] with uniform probability density. [MIN_j, MAX_j] is the allowed range of the *j*th-input variable. The selection of these points ensures that one member of the generated offspring will contain genes that satisfy the condition

$$\forall j,\ ((C_{ij} > X_{1j}) \text{ AND } (C_{ij} < X_{2j})) \text{ OR } ((C_{ij} + MAX_j - MIN_j) < X_{2j})$$

and the other member of the offspring will contain the remaining generated genes that do not satisfy the above condition.

The mutation operator used is the standard point mutation operator applied with probability p_m.

2.2.5.4 Hybrid of GA and Back Propagation (BP)

Fukumi and Akamatsu [25] proposed a method that combines GA with the Back Propagation training algorithm. After a randomly initialised population of potential solutions is generated, the networks are trained using a Back Propagation algorithm with forgetting link of weight (BPWF). The network is trained to achieve 100% classification accuracy. The BPWF uses as a criterion the value of *J*,

$$J = \sum_i (d_i - o_i)^2 + \varepsilon \sum_{i,j} | w_{i,j} | ,$$

and the weights are updated by $\Delta w_{i,j}$,

$$\Delta w_{i,j}(t) = \eta \Delta w_{i,j}(t) - \varepsilon \,\text{sgn}(w_{i,j}(t)) ,$$

where d_i is the desired output, o_i is the actual output and ε is the forgetting factor. The forgetting factor utilises the Deterministic Mutation operator proposed in this method. Deterministic Mutation is used to reduce the number of 1' s in chromosomes that represent connections to a node, and thus reduce their complexity.

The fitness value refers to the number of 1' s in the chromosome, on the condition that the classification accuracy for every sample is 100%. A small number of 1' s leads to high fitness value because it represents a small sized network. The individuals that do not have classification accuracy of 100% are discarded.

The selection of a pair of individuals is done in a stochastic manner based on their fitness value. The selected pairs of individuals are recombined with the crossover operator to generate the offspring. Then the generated offspring is trained with Back Propagation without a forgetting factor. Finally, before the procedure described is applied to the new population, a standard point mutation operator is applied to the generated offspring.

2.3 Niched Genetic Algorithms

Genetic Algorithms are effective at identifying and converging populations to a single global optimum. Many real-world problems contain multiple solutions that are optimal or near optimal. The domains of such problems require the identification of multiple optima. In a multi-modal domain, each peak can be thought of as a *niche* that can support a certain number of concepts. *Niched genetic algorithms* [70] were introduced to identify the niche locations and populate niches according to their fitness relative to the other peaks. Following, the mechanisms introduced to implement a niched genetic algorithm will be presented.

2.3.1 Fitness Sharing

Fitness sharing accomplishes niching by degrading the fitness value of an individual that has highly similar members within the population. This scheme causes population diversity pressure, which helps maintain population members at local optima.

There is an obvious need for a similarity metric. The similarity metric can be based on either phenotype or genotype similarity. A common genotype similarity metric is the Hamming distance, which is the number of different bits between two members of the population. A common phenotype similarity metric is the Euclidean distance in the parameter space. Phenotypic sharing gives slightly better results, due to the decreased noise in the decoded parameter space [47].

The shared fitness f_i' of a member of the population *i*, is defined by

$$f_i' = \frac{f_i}{m_i},$$

where m_i is the niche count calculated by summing a sharing function over all members of the population

$$m_i = \sum_{j=1}^{n} sh(d_{ij}),$$

where *n* is the population size, d_{ij} is the distance between *i* and *j* and $sh(d_{ij})$ is the sharing function. The sharing function measures the similarity between two population members. It returns one if there are two identical members, zero if their distance is higher than a dissimilarity threshold and an intermediate value at an intermediate level of dissimilarity. The most commonly used sharing functions are of the form

$$sh(d_{ij}) = \begin{cases} 1 - \left(\dfrac{d_{ij}}{\sigma_s}\right)^{\alpha} & \text{if } d_{ij} < \sigma_s \\ 0 & \text{otherwise} \end{cases},$$

where σ_s is the dissimilarity threshold referred to as the *niche radius* and α is a constant parameter that regulates the shape of the sharing function. Commonly, α is set to one resulting to the triangular sharing function.

A main drawback of fitness sharing is the difficulty of setting the niche radius σ_s. Deb [42] has presented a method that sets σ_s according to the number of peaks, q, that are expected in the search space as

$$\frac{1}{2^l} \sum_{i=0}^{\sigma_s} \binom{l}{i} = \frac{l}{q},$$

where *l* is the string length. The difficulty in estimating the niche radius with this method is that in most cases the expected number of peaks in the search space is not known.

Another approach to the problem of determining σ_s is to define it as a function [40]. The niche radius function, *r*, is defined as

$$r(i) = r(0)\beta^i, \beta \in (0,1) \quad .$$

r(0) is the maximum distance in the initial population and defines the diameter of the domain to be explored. β is constant that is determined by the equation

$$\frac{N}{r(0)Mv} = \sum_{i=1}^{STm} \frac{1}{u(r(0)\beta^i)},$$

where N is the number of function evaluations, M is the upper bound of the number of species and normally M = [population size / 4], v is a threshold value that determines how many species will receive sufficient function evaluations for crossing the whole space, and *STm* is the number of steps in the ' cooling' procedure. The function *u(x)* determines the speed with which species move in the search space to find the niche on which they are stable. For binary domains the speed function is given by

$$u(x) = \frac{3}{11}\sqrt{x} \; .$$

For fitness sharing to be effective, it has to be implemented with less biased selection methods [64]. Widely used methods are the Stochastic Remainder Selection (SRS) and Stochastic Universal Selection (SUS). The use of the Tournament Selection with fitness sharing has been criticised as a naive combination, because the detailed dynamics are chaotic and because the mean-field performance exhibits a steady decline in the number of niches the population can support [58]. Instead, the method of tournament selection with continuously updated sharing was introduced [58]. In this method, tournament selection is applied according to shared fitness values that have been continuously updated using only the individuals actually chosen to be members of the next generation.

In addition, fitness sharing must use low reproduction operators to promote stability of sub-populations [64]. Crossover between individuals of different niches often leads to poor individuals. To avoid this, mating restriction schemes have been introduced. A scheme like that [41] allows recombination between members whose distance is less than a dissimilarity threshold.

2.3.1.1 Clustering

Yin and Germay proposed that a clustering algorithm is implemented prior to sharing, in order to divide the population into niches [60]. The population is divided, by MacQueen' s adaptive KMEAN clustering algorithm, into k clusters of individuals that correspond to k niches. In this method the niche count is determined as

$$m_i = n_c\left(1 - \left(\frac{d_{ic}}{2 \cdot d_{\max}}\right)^{\alpha}\right),$$

where n_c is the number of individuals in the cluster c, α is a constant, d_{ic} is the distance between the individual i and the centre of its niche c and d_{max} is a threshold parameter. The advantage of this method is that the algorithm itself determines the number of clusters that the initial population is divided into, hence no a priori knowledge about the peaks is required. Two clusters are merged if the distance between their centres is smaller than a threshold parameter d_{min} and when an individual' s distance from all existing cluster centres is higher than d_{max}, a new cluster is formed with this individual as a member.

2.3.1.2 Dynamic Niche Sharing

The dynamic niche sharing method was developed to reduce the computational expense of fitness sharing [54]. It is assumed that the number of peaks, q, can be estimated and that the peaks are all at a minimum distance $2\sigma_s$ from each other. In this method, first the q peaks are identified and then all individuals are classified as either belonging to one of these dynamic niches or else belonging to the non-peak category. An individual i is considered to be within a dynamic niche j if its distance d_{ij} from peak j is less than σ_s. The niche count is now defined by

$$m_i' = \begin{cases} n_j & \text{if } i \text{ is within dynamic niche } j \\ m_i & \text{otherwise} \end{cases},$$

where m_i is niche count as defined in the original fitness sharing scheme and n_j is the niche population size of the jth dynamic niche.

2.3.1.3 Fitness Scaling

A method proposed to increase sharing efficiency is to use fitness scaling [64]. The fitness scaling scheme increases differentiation between optima and reduces deception. An implementation of fitness scaling is by using power scaling of the original fitness. Thus, the shared fitness is given by

$$f_i' = \frac{f_i^{\beta}}{m_i}.$$

The choice of the constant β determines the balance between exploration and exploitation and hence annealing the scaling power is proposed.

2.3.1.4 Niched Pareto Genetic Algorithm

This method proposed a modification to the tournament selection scheme for use with shared fitness [33]. Two candidates for selection are randomly picked from the population, and selecting also randomly t_{dom} (tournament size) individuals, forms a comparison set. Following, each candidate is compared against each individual in the comparison set. If one candidate is dominated by the comparison set, and the other is not, the latter is selected for reproduction. If neither or both are dominated by the comparison set, the candidate with the smallest niche count, m_i, is selected for reproduction.

2.3.1.5 Co-evolutionary Shared Niching

This method overcomes the problem of determining the locations and radii of niches by utilising two populations that evolve in parallel [62]. There is a population of businessmen and a population of customers. The locations of the businessmen correspond to niche locations, and the locations of customers correspond to solutions. On the customer population, the operators of selection and recombination are applied, whereas on the businessmen population, the operators of selection and mutation are applied.

In the customer population, the shared fitness of each individuals is evaluated as in the original fitness sharing scheme, but the niche count, m_i, is now determined without the need of a niche radius. All customers are compared to all businessmen and each customer is assigned to the closest businessman. The businessmen-customer counts are used as niche counts. Following generations are obtained by applying the operators of selection and crossover.

In the businessmen population, single-bit mutation is applied to each individual. The businessman is replaced by its offspring if there is an improvement over the original individual and if it has distance d_{min} at least from the other businessmen. If these conditions are not met the original businessman is retained and the process is repeated for a maximum of n_{limit} times ($n_{limit} \leq l$). If no candidate is found that meets the criterion, the next businessman is selected and the process is repeated.

2.3.2 Crowding

Crowding methods insert new elements in the population by replacing similar elements. Originally, crowding was introduced by De Jong [63]. In this method, only a proportion of the population, specified by the generation gap (G), is chosen to undergo crossover and mutation and dies each generation. A random sample of *CF* individuals is taken from the population, where *CF* is the *crowding factor.* An offspring replaces the most similar individual from the sample.

2.3.2.1 Deterministic Crowding

Mahfoud [47] proposed an improvisation of the standard crowding method of De Jong, by introducing competition between children and parents of identical niche. In this method, all population elements are grouped into *n/2* pairs. Following, these pairs are crossed and the offspring is optionally mutated. Each offspring competes in a tournament against one of the parents that produced it. The set of tournament that yields the closest competitions is held. Closeness is computed according to some appropriate distance measure, preferably phenotypic distance.

2.3.2.2 Probabilistic Crowding

Probabilistic crowding [51] is a variant of the deterministic crowding method described before. If *i, j* are the individuals picked to compete for inclusion in the next generation, they will compete in a probabilistic tournament. Individual *i* wins with probability

$$p(i) = \frac{f(i)}{f(i) + f(j)}.$$

There are three variants of this method where the selected individuals can undergo either mutation or crossover or both.

2.3.2.3 Restricted Tournament Selection

In this method [64], two elements are initially selected to undergo crossover and mutation. After recombination, a random sample of *CF* individuals is taken from the population as in standard crowding. Each offspring competes with the closest sample element and the winners are inserted in the population. This procedure is repeated *n/2* times.

2.3.3 Restricted Competition Selection

The restricted competition selection (RCS) method restricts competitions among dissimilar individuals during selection to reach stable sub-populations [43]. In this method, after the initial population of size *n* is created, the best *m* individuals of this population form the *elite set*. The parents to undergo crossover and mutation are selected randomly, and after the operators have been applied, the elite set is added to the population to produce the *competition set* having *n+m* individuals. The similar elements of the competition set, the ones that their distance is less than the niche radius, compete and the loser' s fitness is set to zero. After all similar elements compete, the new elite set is formed and selecting the best *n* elements from the competition set forms the new population. This process is repeated for a number of generations.

2.3.4 Clearing

The clearing procedure is applied after evaluating the fitness of individuals and before applying the selection operator [59]. The individual in a niche with the best fitness is the *dominant* individual. The *non-dominant* individuals that belong to a niche have a distance from the dominant individual less than the *clearing radius*. The clearing method preserves the fitness of the dominant individual while it resets the fitness of all other individuals of the same niche to zero. Thus, this method fully attributes the whole resource of a niche to a single individual, the winner.

2.4 Evolutionary Algorithms

Evolutionary algorithms are an alternative approach to simulate evolution. They emphasise the behavioural link between parents and offspring, rather than the genetic link. They work in a similar way to genetic algorithms; they maintain a population of potential solutions and make use of the selection principle of the survival of the fittest individual. However, there are many differences between genetic algorithms and evolutionary algorithms.

2.4.1 Working principles

Evolutionary algorithms were initially introduced as a mechanism with only one parent represented by a real valued vector x. New individuals evolved by a mutation operator. Mutation is applied to the parent by adding a zero mean Gaussian random variable with a preselected standard deviation σ,

$$x^{t+1} = x^t + N(0,\sigma) .$$

The resulting individual is evaluated and compared to its parent, and the best individual survives to become a parent in the next generation, while the other is discarded. This mechanism is termed as (1+1)-ES.

To incorporate the concept of populations in the (1+1)-ES method, the (μ+1)-ES was introduced. In the (μ+1)-ES method, μ parents recombine to generate an offspring. The offspring is mutated as in the (1+1)-ES method and replaces the worst parent to form the new population.

Schwefel [65] then introduced the ($\mu+\lambda$)-ES and (μ,λ)-ES methods. In the ($\mu+\lambda$)-ES method, μ parents are used to create λ offspring and all solutions compete for survival, with the best μ members being selected as parents in the next generation. In the (μ,λ)-ES method, μ parents are again used to create λ offspring, but only the λ offspring compete for survival. This leads to the condition of $\lambda > \mu$ required holding.

Although the initial evolutionary algorithm method of (1+1)-ES incorporated only the genetic operator of mutation, the latter methods of ($\mu+\lambda$)-ES and (μ,λ)-ES introduced the use of the operator of recombination or crossover.

In general, an evolutionary algorithm method starts by defining the problem to be solved as finding an l-dimensional real valued vector associated with the extremum of an objective function $f\colon 3^{l} \rightarrow 3$. Following, a random population of real valued vectors is initialised and evaluated. Then, until a termination condition is met, a repetitive process of the operators of crossover, mutation and selection is applied.

2.4.1.1 Crossover

The crossover operators in evolutionary algorithm methods are mainly subdivided into two categories: *discrete recombination* and *intermediate recombination.*

In discrete recombination, when two parents are selected, the offspring is generated by randomly copying the corresponding component of one of the parents. So, for two parents A and B the generated offspring's C components, x_i^C will be

$$x_i^C = x_i^A \, or \, x_i^B \text{, for all } i \in \{1,...,l\}.$$

In intermediate recombination, the offspring's components are obtained by calculating the arithmetic mean of the corresponding components of both parents.

$$x_i^C = x_i^A + 0.5 \cdot (x_i^B - x_i^A), \text{ for all } i \in \{1,\dots,l\}.$$

A generalisation to intermediate recombination was also proposed by Schwefel. Instead of calculating the mean of the two parent components, a weight factor α belonging to the interval of [0,1], is multiplied with the difference of the two parent components.

$$x_i^C = x_i^A + \alpha \cdot (x_i^B - x_i^A), \text{ for all } i \in \{1,\dots,l\}.$$

2.4.1.2 Mutation

In evolutionary algorithm methods, the mutation operator is applied by adding to the original vector a Gaussian random variable of zero mean and a predefined standard deviation σ.

$$x^{t+1} = x^t + N(0,\sigma)$$

Another approach proposed is to evolve the standard deviation σ, in parallel with the potential solution vector. In this case, the solution vector comprises from both the trial vector x of l dimensions, and the perturbation vector σ, which provides instructions on how to mutate x. Thus, the new solution vector (x^{t+1}, σ^{t+1}) will be

$$\sigma_i^{t+1} = \sigma_i^t \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$$

$$x_i^{t+1} = x_i^t + N(0, \sigma_i^{t+1})$$

where $i=1,\dots,l$, $N(0,1)$ represents a single standard Gaussian random variable, $N_i(0,1)$ represents the ith independent identically distributed standard Gaussian, and τ and τ' are operator set parameters, which define individual and global step-sizes, respectively. The global factor $\exp(\tau' \cdot N(0,1))$ allows for an overall change of the mutability and guarantees the preservation of all degrees of freedoms. The factor $\exp(\tau \cdot N(0,1))$, allows individual changes of the mean step sizes σ_i. Schwefel suggested that

$$\tau \propto \left(\sqrt{2\sqrt{l}} \right)^{-1},$$

$$\tau' \propto \left(\sqrt{2l}\right)^{-1},$$

and usually τ', τ are set equal to one.

Further extensions were made to the mutation operator to incorporate correlated mutations, so that the distribution of new trials could adapt to contours on the error surface. The surfaces of equal probability density to place an offspring by mutation are called *mutation ellipsoids.* Under independent Gaussian perturbations, mutation ellipsoids are aligned with the co-ordinate axes. By using correlated mutations, mutation ellipsoids have arbitrary orientation in the search space and individuals can adapt to any advantageous direction of search. For this purpose, the *rotation angles* $a_{ij} \in [-\pi, \pi]$ were introduced and they are defined by

$$\tan(2\alpha_{ij}) = \frac{2c_{ij}}{\sigma_i^2 - \sigma_j^2},$$

where c_{ij} is the corresponding element of the covariance matrix. The new solution vector (x^{t+1}, σ^{t+1}, α^{t+1}) now will be

$$\sigma_i^{t+1} = \sigma_i^{t} \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$$

$$\alpha_j^{t+1} = \alpha_j^{t} + \beta \cdot N_j(0,1)$$

$$x_i^{t+1} = x_i^{t} + N(0, C(\vec{\sigma}^{t+1}, \vec{\alpha}^{t+1}))$$

where C is the covariance matrix represented by the vectors $\vec{\sigma}$ of standard deviations and $\vec{\alpha}$ of rotation angles, for all $i \in \{1,...,l\}$ and for all $j \in \{1,...,l \cdot (l-1)/2\}$, and β is a constant and it was suggested by Schwefel to be set as

$$\beta \approx 0.0873.$$

2.4.1.3 Selection

The selection operator's function depends upon the evolutionary algorithm method used. When using the (μ+λ)-ES method, after the genetic operators of crossover and mutation are applied, the best μ members from both the parents and offspring are selected to form the new generation of population. In the (μ,λ)-ES method, the best μ members are again selected, but this time only out of the λ offspring.

Although the $(\mu+\lambda)$-ES method might seem at first more appropriate to use because it keeps the best individuals from both the parents and the offspring, this might not always be true. The (μ,λ)-ES method has the advantage, when applied to multi-modal distribution, that it can escape from local minima easily. In addition, in the case of changing environments, the $(\mu+\lambda)$-ES method preserves outdated solutions and is not able to follow the moving optimum.

2.5 EA-based methods for neural networks optimisation

2.5.1 EPNet

In [75] and [76], Yao and Liu have proposed an evolutionary algorithm that uses only the mutation operator to evolve the topology and weights of feedforward ANNs. Five different types of mutation are applied.

The algorithm starts by randomly initialising a population of potential solutions. Each network is partially trained using a modified BP (MBP) algorithm with adaptive learning rate. After training, the error of each network is evaluated and if it has not been significantly reduced through training, the network is marked with "failure". Otherwise the network is marked with "success".

Following, the networks are ranked from the best to worst according to their fitness value. In this method, the fitness value of each network is determined by the inverse of the error E obtained over a validation set containing T patterns.

$$E = 100 \cdot \frac{o_{\max} - o_{\min}}{T \cdot n} \sum_{t=1}^{T} \sum_{i=1}^{n} (y_i(t) - z_i(t))^2$$

o_{max} and o_{min} are the maximum and minimum values of output coefficients, n is the number of output nodes and $y_i(t)$ and $z_i(t)$ are the actual and desired output of node i for pattern t, respectively. A single parent j is selected out of M sorted individuals for mutation with probability p_j,

$$p(j) = \frac{j}{M}.$$

The selected parent is then altered by a hybrid training algorithm consisting of MBP and Simulated Annealing (SA). The network is first trained using the MBP, and each error E is checked every k epochs, where k is predefined. If the error decreases, the learning rate is increased. Otherwise, the learning rate is reduced and the new weights and error are discarded. When the MBP cannot improve the network anymore, the SA algorithm is applied. If after the SA algorithm has been applied, the network has not improved significantly, four different mutation operators are applied.

First, the *Hidden Node Deletion* operator is applied. A specified number of hidden nodes are randomly deleted from the network. The network is trained with MBP and its error value is evaluated. If this pruned network is better than the worst network of the current population, the pruned network replaces the worst one and no further mutations will be applied. Otherwise, the generated offspring is discarded and the *Connection Deletion* operator is applied.

The approximate importance of each connection in the network is calculated using

$$significance(w_{i,j}) = \frac{\sum_{t=1}^{T} \xi_{i,j}^{t}}{\sqrt{\sum_{t=1}^{T} (\xi_{i,j}^{t} - \bar{\xi}_{i,j})^2}},$$

where $\xi_{i,j}^{t} = w_{i,j} + \Delta w_{i,j}^{t}(w)$ and $\bar{\xi}_{i,j}$ is the average over the set of all $\xi_{i,j}^{t}$. A specified number of connections are deleted according to their significance. The resulting network is trained with MBP and its error values are evaluated. As before, if this network is better than the worst network of the current population, it will replace it. Otherwise, the generated offspring is discarded and the mutation operator of *Connection and Node Addition* will be applied.

A specified number of connections are added probabilistically according to the significance function described before. They are selected from those connections with zero weights. The generated offspring 1, is trained with MBP and its error value is determined. An offspring 2 is generated by node addition. New nodes are added by randomly selecting an existing node and splitting it. The two nodes obtained from splitting will have the same connections, and their weights are determined according to the following equations

$$w_{ki}^{1} = w_{ki}^{2} = w_{ki}, \quad \text{for } i \geq k$$
$$w_{ki}^{1} = (1+\alpha)w_{ki}\ , \quad \text{for } i < k\ ,$$
$$\mathrm{w}_{\mathrm{ki}}^{2} = -\alpha w_{ki}, \qquad \text{for } i < k$$

where α is a mutation parameter that can have either a fixed or random value. Offspring 2 is also trained with MBP and its error value is determined. Offspring 1 and offspring 2 compete for survival, and the winner replaces the worst network in the current population.

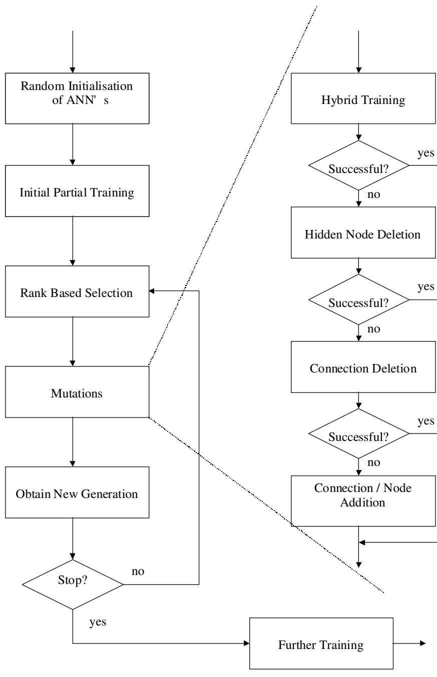


Figure 2.5 The EPNet

When the evolutionary process stops, the best member of the population is further trained with MBP on a combined training and validation set. Then, this network is tested on an unseen training set to evaluate its performance.

An alternative approach is to make use of all the members of the population evolved. Each individual is treated as a module and the linear combination of all members is the *ensemble* of the evolved ANNs (EANNs).

The simplest linear combination is *majority voting*. All individuals of the last population are treated equally and participate in voting. The output the most EANNs, will be the output of the ensemble.

In *Rank-Based Linear Combination*, a weight factor of each EANN is calculated using

$$w_i = \frac{\exp(\beta(N+1-i)}{\sum_{j=1}^{N} \exp(\beta j)},$$

where N is the population size and β is a scaling factor. The output of the ensemble is then determined by $O = \sum_{j=1}^{N} w_j o_j$, where o_j is the output of each EANN.

Other methods for constructing the NN ensemble, incorporate the use of the Recursive Least Squares (RLS) algorithm or the use of a GA, to determine a subset of the population that would be used as an ensemble.

2.5.2 Co-Evolutionary Learning System (CELS)

Yao and Liu have also introduced a co-evolutionary learning system (CELS) [45], to design NN ensembles. In this method, each NN is trained using negative correlation learning. It introduces a correlation penalty term into the error function of each NN, so that the individual NN can be trained co-operatively. Thus, the error function E_i for an individual i is defined as:

$$E_i = \frac{1}{N}\sum_{n=1}^{N} E_i(n) = \frac{1}{N}\sum_{n=1}^{N}\left[\frac{1}{2}\left(d(n) - F_i(n)\right)^2 + \lambda p_i(n)\right],$$

where N is the number of training patterns, $E_i(n)$ is the value of the error of individual NN i at presentation of the nth training pattern, $F_i(n)$ is the output of individual NN, and p_i is a correlation penalty function. The parameter $\lambda > 0$ is used to adjust the strength of the penalty. The function p_i can be

$$p_i = (F_i(n) - F(n)) \sum_{j \neq i} (F_j(n) - F(n)),$$

where *F(n)* is the average of the output of all individual NNs for the *n*th training pattern.

The standard mutation operator is applied by adding to the original weight vectors a Gaussian random variable of zero mean and standard deviation equal to 1.

The fitness value of each NN is determined using a reward scheme. For each training case, if there are $p > 0$ individual NNs that classify it, then each of these *p* individuals receives *1 / p* fitness reward, and the rest individuals receive zero fitness reward. To obtain the fitness value, the fitness reward for each individual is summed over all training cases.

At the end of the evolutionary process, in order to construct the ensembles, the methods of *majority voting, Rank-Based Linear Recombination* and *RLS* described before can be used.

2.5.3 Symbiotic Adaptive Neuro-Evolution (SANE)

In SANE [56], [63], two separate populations are maintained and evolved: a population of neurons and a population of neural network blueprints. Each individual in the neuron population specifies a set of connections to be made within a neural network. Each individual in the network blueprint population specifies a set of neurons to include in a neural network. The neuron evolution searches for effective partial networks, while the blueprint evolution searches for effective combinations of the partial networks.

In the neuron population, SANE evolves a large population of hidden neuron definitions for a three-layer feedforward network. A neuron is represented by a series of connection definitions that describe the weighted connections of the neuron from the input layer and to the output layer. A connection definition consists of a label and weight pair. The label is an integer value that specifies an input or output unit, and the weight is a floating-point number that specifies the weight.

L	W	L	W	L	W
4	0.1	6	-0.7	7	0.4
1	-0.8	5	-1.4	7	0.3
2	0.1	4	-0.6	7	-1.2

L: label
W: weight

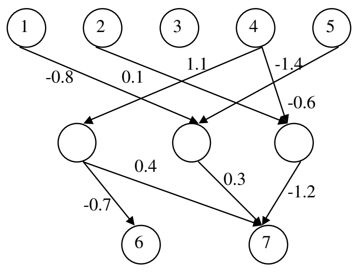


Figure 2.6 The neuron population is shown on the left and the corresponding network is shown on the right.

The blueprints are made up of a series of pointers to members of the neuron population and define an effective neural network from a previous generation.

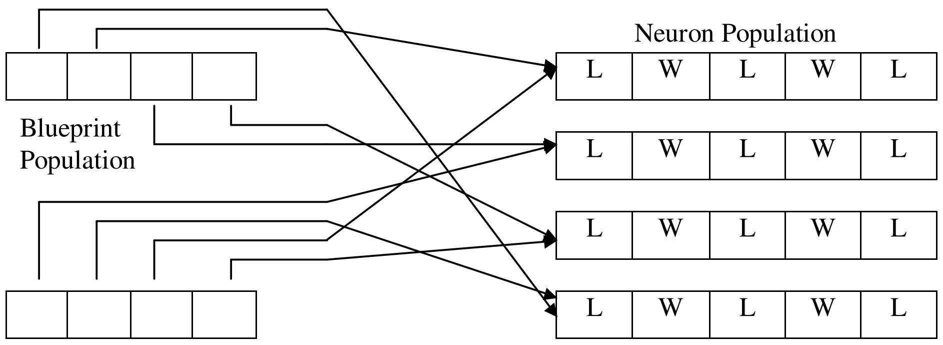


Figure 2.7 The network blueprint population and the pointers to neurons.

The evolution process in SANE is performed in two stages: the evaluation stage and the reproduction phase. In the evaluation stage, the blueprints and neurons are evaluated simultaneously. Every network blueprint is decoded and then its fitness value is determined. The fitness value of each network blueprint is added to the fitness value of each individual neuron contained in the network. After all the network blueprints have been evaluated, each neuron' s fitness is normalised by dividing its accumulated fitness value by the number of blueprints where it was a participant.

In the reproduction stage, the standard operators of reproduction, crossover and mutation are applied to generate the new populations of blueprints and neurons.

2.5.4 Hybrid of EA and Single Stochastic Search

In this method [50], once the initial population is randomly initialised and parents are selected according to their fitness value, three sets of offspring are generated. The first set of offspring is generated by the perturbation of the parent with a Gaussian random variable of zero mean and standard deviation σ, $N(0, \sigma)$. The standard deviation σ can be fixed or proportional to the corresponding height of the response surface or conditionally based on search performance. Blending the parameters of the randomly selected pair of parents generates the second set of offspring. This can be implemented with a standard crossover operator. The final set of offspring is generated using the Solis and Wets method [67].

The Solis and Wets method generates a Gaussian random variable $N(b, \sigma)$ where the variance and standard deviation parameters, b and σ, are determined using the following algorithm. In this algorithm *scnt* and *fcnt* denote the repeated number of successes and failures, respectively, in decreasing the objective function *f*. The contraction, *ct,* and expansion, *ex,* constants, as well as the lower, σ_l, and upper, σ_u, limits of the value that the standard deviation can take, are predefined.

1. Initialise search vector x_0 and bias vector b_0. Set k=0, scnt=*0*, *fcnt*=0, s_0=1.
2. Set $\sigma_k = \begin{cases} ex \cdot \sigma_{k-1} & \text{if } scnt > Scnt \\ ct \cdot \sigma_{k-1} & \text{if } fcnt > Fcnt \\ \sigma_u & \text{if } \sigma_{k-1} < \sigma_l \\ \sigma_{k-1} & \text{otherwise} \end{cases}$
3. Generate a multivariate Gaussian random variable $\xi_k \sim N(b_k, \sigma)$

4. If $f(x_k + \xi_k) < f(x_k)$, then set $x_{k+1} = x_k + \xi_k$ and $b_k = 0.4\xi_k + 0.2b_k$, *scnt=scnt+1, fcnt=0.*
5. Otherwise, if $f(x_k - \xi_k) < f(x_k) < f(x_k + \xi_k)$, then set $x_{k+1} = x_k - \xi_k$ and $b_k = b_k - 0.4\xi_k$, *scnt=scnt+1, fcnt=0.*
6. Otherwise, $x_{k+1} = x_k$ and $b_k = 0.5b_k$, *fcnt=fcnt+1, scnt=0.*
7. If *k*=maximum number of iterations stop, else go to step 2.

After the three sets of offspring have been generated, their fitness is evaluated and they compete for survival. The fittest survives and thus the new population is generated. McDonell and Waagen [50] have used the Akaike' s minimum information theoretical criterion (AIC) to determine the fitness value of each individual.

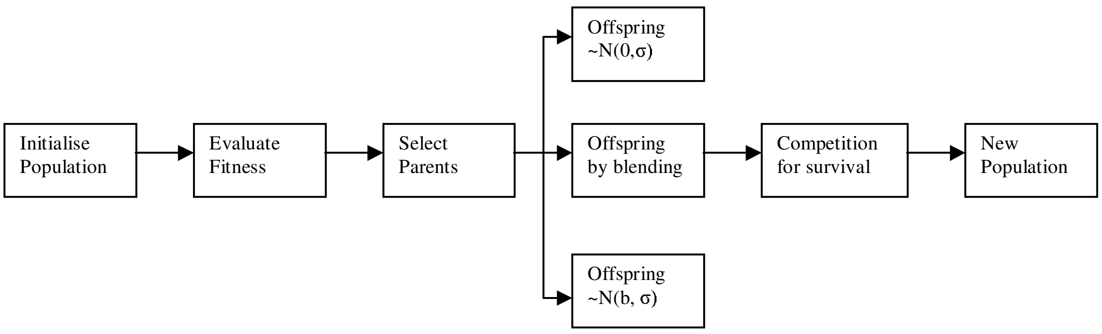


Figure 2.8 Hybrid of EA and Single Stochastic Search method.

2.5.5 Multi-path Network Architecture

In [13], Cheng and Guan present a multi-path network architecture to evolve neural networks. The multi-path architecture is used to accommodate for a possible undesired minimum resulting from an evolution process. When the evolution process is trapped in an undesired minimum with respect to one path, the method seeks an alternative path to carry out the evolution.

When networks are trained, it is in general required to minimise an energy function *E,*

$$E = X^T \mathbf{W} X$$

where X is the pattern space, and **W** is the connection matrix generated by the learning rule. In this method, **W** is decomposed into K subsets,

$$\mathbf{W} = W_1 \cup W_2 \cup \ldots \cup W_K$$

where

$$W_k = \{w_{ij}^k, i, j = 1,2,...,N\}, k = 1,2,...,K\ .$$

Each W_k represents a different path in the network for learning and evolution. By using K different rules to train each W_k, K distinct paths are obtained. A multi-level energy function **E** is defined by

$$\mathbf{E} = f(E_1, E_2, \ldots, E_K)$$

All K rules use the same training patterns and thus the desired minima will be at the same locations on the surfaces of their respective energy functions. However, the locations of the undesired minima will not be the same because different optimisation criteria have been used.

During the evolution of a neural network, a random path is chosen. The evolution process in this path continues until a minimum is reached. Then, a different path is chosen and the evolution process starts again until a minimum in this different path is reached. This procedure continues until the minimum obtained from all the possible paths is the same.

2.5 Summary

This chapter dealt with the basic concepts of the various evolutionary computing techniques; Genetic Algorithms, Niched Genetic Algorithms and Evolutionary Algorithms. Their basic operators were introduced and the main strategies for neural networks optimisation were outlined.

CHAPTER 3

PREDICTION OF TIME SERIES SIGNALS

3.1 Introduction

This chapter introduces time series signals and their basic properties. Following, the procedure for time series signals prediction is outlined. Finally, the Group Method of Data Handling (GMDH) algorithm and the Genetics-Based Self-Organising Network (GBSON) method for time series prediction are introduced.

3.2 Time Series Signals

A time series is a set of observations x_t, each one being recorded at a specific time t. A discrete time series is one where the set of times at which observations are made is a discrete set. Continuous time series are obtained by recording observations continuously over some time interval. An example of a discrete time series can be seen in Figure 3.1.

Analysing time series data led to the decomposition of time series into components. Each component is defined to be a major factor or force that can affect any time series. Three major components of time series have been identified. *Trend* refers to the long-term tendency of a time series to rise or fall.

Seasonality refers to the periodic behaviour of a time series within a specified period of time. The fluctuation in a time series after the trend and seasonal components have been removed, is termed as the *irregular* component.

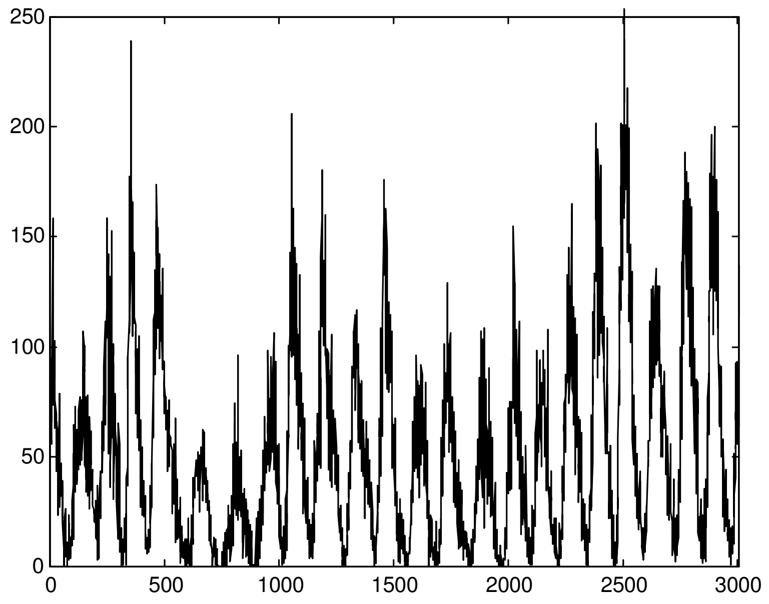


Figure 3.1 Monthly sunspot numbers, 01/1749-07/1999.

If a time series can be exactly predicted from past knowledge, it is termed as *deterministic*. Otherwise it is termed as *statistical*, where past knowledge can only indicate the probabilistic structure of future behaviour. A statistical series can be considered as a single realisation of some *stochastic process*. A stochastic process is a family of random variables defined on a probability space. A realisation of a stochastic process is a sample path of this process. Thus, if a stochastic process is defined by

$$X_t(\omega) = A(\omega)\cos(vt + \Theta(\omega)),$$

then the realisations of this stochastic process would be the functions of t, obtained by fixing ω, defined by the form

$$x(t) = a\cos(vt + \theta).$$

To gain insight into the dependence between the random variables of a time series, its *autocovariance function* has to be evaluated. The autocovariance function $\gamma_x(\cdot)$ of a time series *x(t)* is defined by [8]

$$\gamma_x(t_1, t_2) = E\langle x(t_1)x(t_2)\rangle\,,$$

where $E\langle\cdot\rangle$ is the expectation operator. If a stochastic process is such that

$$\gamma_x(t_1, t_2) = E\langle x(t_1+\tau)x(t_2+\tau)\rangle\,,$$

then its probabilistic structure does not change with time and the process is said to be *strictly stationary*. If the above condition holds only with the restriction

$$E\langle x(i)\rangle = \mu, \text{ for all } i$$

where μ is the mean value of the process, then the process is termed as *stationary*. Otherwise, the process is termed as *non-stationary*.

To measure the degree to which two processes *x(t)* and *y(t)* are related over the time axis, the *cross correlation* function is evaluated. The cross correlation function is defined as [9]

$$\gamma_{xy}(t_1, t_2) = E\langle x(t_1+\tau)y(t_2+\tau)\rangle\,.$$

3.3 The Procedure of Time Series Signals Prediction

The prediction of time series signals is based on their past values. Therefore, it is necessary to obtain a data record. When obtaining a data record, the objective is to have data that are maximally informative and an adequate number of records for prediction purposes. Hence, future values of a time series $x(t)$ can be predicted as a function of past values $x(t\text{-}1)$, $x(t\text{-}2)$, .., $x(t\text{-}\varphi)$.

$$x(t+\tau) = f(x(t\text{-}1), x(t\text{-}2), .., x(t\text{-}\varphi))$$

The problem of time series prediction now becomes a problem of system identification. The unknown system to be identified is the function $f(\cdot)$ with inputs the past values of the time series.

While observing a system there is a need for a concept that defines how its variables relate to each other. The relationship between observations of a system or the knowledge of its properties is termed as the *model* of the system. Models can be given in several different forms. A *mental model* does not involve any mathematical formalisation, but the system' s behaviour is summarised in a nonanalytical form in a person' s mind. A mental model is a driver' s perception of a car' s dynamics*Graphic models* make use of a graph or a table to summarise the properties of a system. *Mathematical models* are mathematic relationships among the system variables, often differential or difference equations. In system identification, a set of candidate models is specified, where the search for the most suitable one will be restricted.

The search for the most suitable model for a system is guided by an assessment criterion of the goodness of a model. In the prediction of time series, the assessment of the goodness of a model is based upon the prediction error of the specific model.

After the most suitable model of a system has been determined, it has to be validated. The validation step in the system identification procedure is very important because in the model identification step, the most suitable model obtained was chosen among the predefined candidate models set. This step will certify that the model obtained describes the true system. Usually, a different set of data than the one used during the identification of the model, the *validation set*, is used during this step.

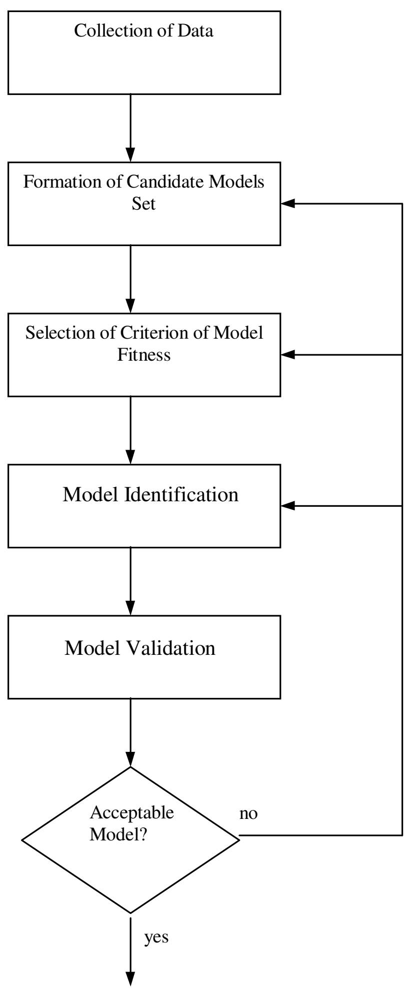


Figure 3.2 The Procedure of Time Series Signals Prediction.

3.3.1 Models

3.3.1.1 Linear Models

3.3.1.1.1 Autoregressive (AR)

The general form of the *autoregressive* (AR) model is given by the linear difference equation [31]

$$x(t) = a_1 x(t-1) + \ldots + a_{n_a} x(t-n_a) + e(t),$$

where the current value of the time series is expressed as a weighted sum of past values plus the white-noise term $e(t)$ with variance σ_e^2. Thus, $x(t)$ can be considered to be regressed on the n_a previous values of $x(\cdot)$.

If an exogenous variable $u(t)$ is added, the ARX model is obtained. The general form of this model is [46]

$$x(t) = a_1 x(t-1) + ... + a_{n_a} x(t-n_a) + b_1 u(t-1) + ... + b_{n_b} u(t-n_b) + e(t).$$

The adjustable parameters for this model are

$$\theta = \begin{bmatrix} -a_1 & -a_2 ... - a_{n_a} & b_1 & b_2 ... b_{n_b} \end{bmatrix}^T.$$

The two models can be rewritten as

$$\text{AR: } A(z^{-1})x(t) = e(t)$$

$$\text{ARX: } A(z^{-1})x(t) = B(z^{-1})u(t) + e(t)$$

where

$$A(z^{-1}) = 1 - a_1 z^{-1} - ... - a_{n_a} z^{-n_a}$$

and

$$B(z^{-1}) = b_1 z^{-1} + ... + b_{n_b} z^{-n_b}.$$

Now the model structure of AR and ARX can be shown schematically in Figures 3.3 and 3.4, respectively. It can be seen that the disadvantage of this model is that the white noise goes through the denominator dynamics of the systems before being added to the output.

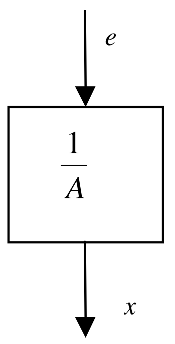


Figure 3.3 The AR model structure.

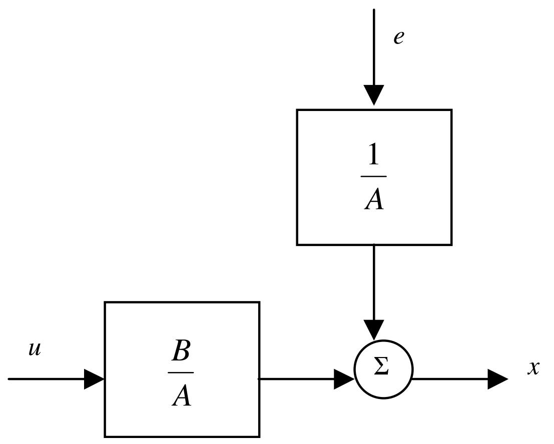


Figure 3.4 The ARX model structure.

3.3.1.1.2 Moving Average (MA)

AR and ARX models give limited freedom in describing the properties of the disturbance terms. The *moving average* (MA) model describes the time series as a moving average of white noise. The general form of the MA model is [74]

$$x(t) = e(t) + c_1 e(t-1) + ... + c_{n_c} e(t-n_c) .$$

The adjustable parameters now are

$$\theta = [c_1 \quad c_2 ... c_{n_c}]^T .$$

The model can be rewritten as

$$x(t) = C(z^{-1}) e(t) ,$$

with

$$C(z^{-1}) = 1 + c_1 z^{-1} + ... + c_{n_c} z^{-n_c} .$$

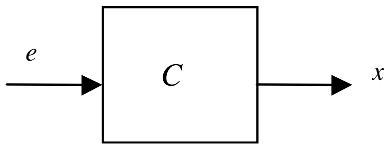


Figure 3.5 The MA model structure.

3.3.1.1.3 Mixed models of AR & MA

If the models AR and MA are combined, the model ARMA is obtained. The general form for this model is [8]

$$x(t) = a_1 x(t-1) + ... + a_{n_a} x(t-n_a) + e(t) + c_1 e(t-1) + ... + c_{n_c} e(t-n_c) .$$

In the same way the model ARMAX is obtained and has the form

$$\begin{aligned} x(t) = a_1 x(t-1) + ... + a_{n_a} x(t-n_a) + b_1 u(t-1) + ... + b_{n_b} u(t-n_b) + \\ e(t) + c_1 e(t-1) + ... + c_{n_c} e(t-n_c) \end{aligned} .$$

The two models can be rewritten as

$$\text{ARMA: } A(z^{-1}) x(t) = C(z^{-1}) e(t)$$

$$\text{ARMAX: } A(z^{-1}) x(t) = B(z^{-1}) u(t) + C(z^{-1}) e(t) ,$$

using the same notation as before.

If the equation error *e*(*t*) in the ARMAX model is described by an ARMA model, the model ARARMAX is obtained with the general form [46]

$$A(z^{-1})x(t) = B(z^{-1})u(t) + \frac{C(z^{-1})}{D(z^{-1})}e(t),$$

where

$$D(z^{-1}) = 1 + d_1 z^{-1} + \ldots + d_{n_d} z^{-n_d}.$$

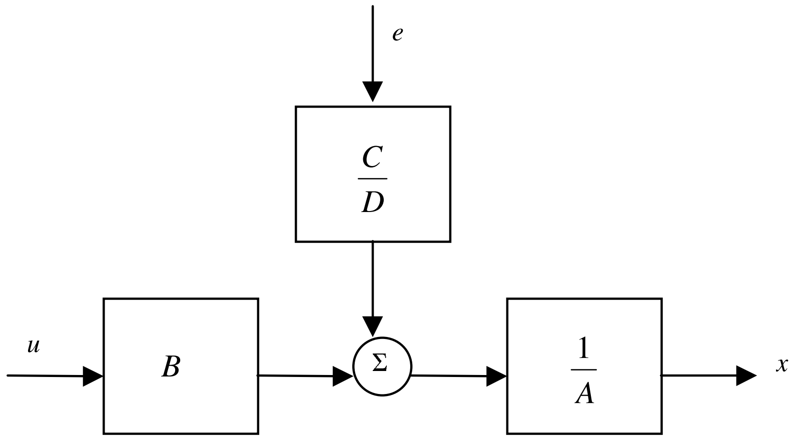


Figure 3.6 The ARARMAX model structure.

3.3.1.1.4 Integrated ARMA models (ARIMA)

Many observed non-stationary time series exhibit certain homogeneity and can by accounted for by a modification of the models described before. The *integrated* models ARIMA and ARIMAX are obtained by replacing *x*(*t*) and *u*(*t*) by their differences $\Delta x(t) = x(t) - x(t-1)$ and $\Delta u(t) = u(t) - u(t-1)$. Usually, the symbol ∇ is used to denote the difference operator. Thus, first differences are denoted as $\nabla x(t) = x(t) - x(t\text{-}1)$. Higher order differences are defined as $\nabla^d x(t)$, where *d* is the order, and they are calculated by consecutively taking differences of the differences. The models have now the general form [46]

$$\text{ARIMA: } A(z^{-1})\nabla^d x(t) = C(z^{-1})e(t)$$

$$\text{ARIMAX: } A(z^{-1})\nabla^d x(t) = B(z^{-1})\nabla^d u(t) + C(z^{-1})e(t).$$

3.3.1.1.5 Seasonal ARMA Models (SARMA)

When a time series exhibits seasonality, it is useful to try to exploit the correlation between the data at successive periods of time. To represent the seasonal models, the seasonal difference operator is introduced. The first-order seasonal difference with a *span* of *s* periods is defined as

$$\nabla_s x(t) = x(t) - x(t-s)\,.$$

The seasonal ARMA (SARMA) model has the general form [74]

$$\text{SARMA: } A(z^{-s})\nabla_s^D \nabla^d x(t) = C(z^{-s})e(t)\,,$$

where *D* represents the order of the seasonal difference operator, *s* the span, *d* is the order of the difference operator and the polynomials $A(z^{-s})$ and $C(z^{-s})$ are defined as

$$A(z^{-s}) = 1 - a_1 z^{-s} - \ldots - a_{n_a} z^{-n_a s}\,,$$

$$C(z^{-s}) = 1 + c_1 z^{-s} + \ldots + c_{n_c} z^{-n_c s}\,.$$

3.3.1.1.6 Output Error (OE)

This model assumes that the relation between input and undisturbed output *w* can be written as a linear difference equation, and that the disturbances consist of white measurement noise [46], thus

$$w(t) + f_1 w(t-1) + \ldots + f_{n_f} w(t-n_f) = b_1 u(t-1) + \ldots + b_{n_b} u(t-n_b)$$

and

$$x(t) = w(t) + e(t)\,.$$

The model can be written as

$$x(t) = \frac{B(z^{-1})}{F(z^{-1})} u(t) + e(t)\,,$$

where

$$F(z^{-1}) = 1 + f_1 z^{-1} + \ldots + f_{n_f} z^{-n_f}\,,$$

and the parameter vector to be determined is now

$$\theta = \begin{bmatrix} b_1 & b_2 \ldots b_{n_b} & f_1 & f_2 \ldots f_{n_f} \end{bmatrix}^T .$$

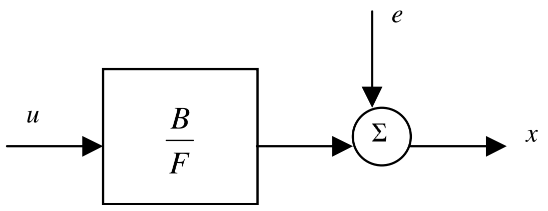


Figure 3.7 The OE model structure.

3.3.1.1.7 Box-Jenkins (BJ)

The Box-Jenkins model [46] is obtained by describing the output error of the OE model as an ARMA model. The BJ model' s general form is

$$x(t) = \frac{B(z^{-1})}{F(z^{-1})} u(t) + \frac{C(z^{-1})}{D(z^{-1})} e(t) .$$

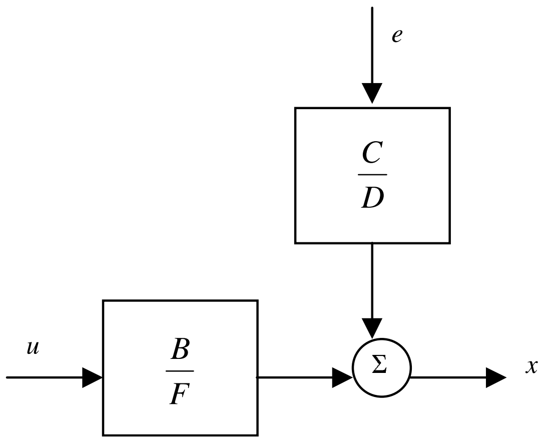


Figure 3.8 The BJ model structure.

3.3.1.1.8 General family of model structures

The models presented so far can be represented by a general model structure defined as

$$A(z^{-1})x(t) = \frac{B(z^{-1})}{F(z^{-1})}u(t) + \frac{C(z^{-1})}{D(z^{-1})}e(t),$$

depending on which of the five polynomials *A*, *B*, *C*, *D*, *E* and *F* are used. This general model can give rise to 32 different models.

3.3.1.2 Non-linear Models

3.3.1.2.1 Volterra Series Expansions

A general non-linear model has the form

$$h(x(t), x(t-1), ...) = e(t),$$

where *h* is a non-linear function and *e*(*t*) is a zero mean white process. It is assumed that the function *h* is invertible. Then the present value of the time series, *x*(*t*), can be expressed as a function of the past values of the zero mean white process. The general form of a non-linear model can now be written as

$$x(t) = h'(e(t), e(t-1), ...).$$

If it is assumed that h' is sufficiently well behaved so that it can be expanded in a Taylor series, then the *Volterra series expansion* is obtained. The Volterra series expansion, obtained by the Taylor series expansion about the point 0, is given by the equation [61]

$$\begin{aligned} x(t) = \mu + \sum_{u=0}^{\infty} g_u e(t-u) + \sum_{u=0}^{\infty}\sum_{v=0}^{\infty} g_{uv} e(t-u)e(t-v) \\ + \sum_{u=0}^{\infty}\sum_{v=0}^{\infty}\sum_{w=0}^{\infty} g_{uvw} e(t-u)e(t-v)e(t-w) + ... \end{aligned}.$$

The sequences g_u, g_{uv} and g_{uvw} are termed as the *kernels* of the Volterra series and are obtained by

$$g_u = \left(\frac{\partial h'}{\partial e(t-u)}\right)_0, \quad g_{uv} = \left(\frac{\partial^2 h'}{\partial e(t-u)\partial e(t-v)}\right)_0 \text{ and}$$

$$g_{uvw} = \left(\frac{\partial^3 h'}{\partial e(t-u)\partial e(t-v)\partial e(t-w)}\right)_0.$$

The constant term μ is obtained by $\mu = h'(0)$.

3.3.1.2.2 Wiener and Hammerstein Models

Wiener and Hammerstein models are used to describe systems that a linear model can describe their dynamics, but there are static non-linearities at the input or the output of the system. A Wiener model describes a system with static non-linearities at the output, while a Hammerstein model describes a system with non-linearities at the input. When there are non-linearities at both the input and output of the system, a Wiener-Hammerstein model is used. The structure of the Wiener and Hammerstein models is shown in Figures 3. 9 and 3.10, respectively. The static non-linear function $f(\cdot)$ can be parameterised either in terms of physical parameters or in black-box terms [46].

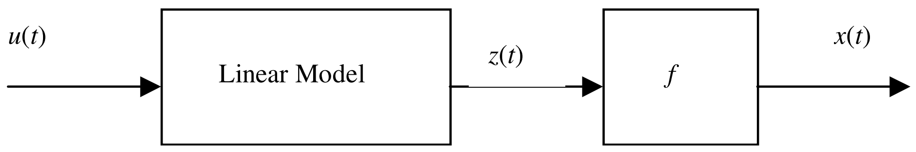


Figure 3.9 The Wiener model structure.

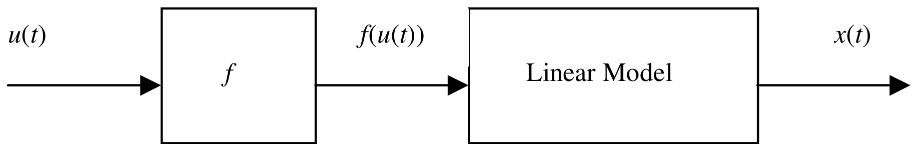


Figure 3.10 The Hammerstein model structure.

3.3.1.2.3 Bilinear Model

The bilinear model is an extension of the ARMA model. Its general form is given by the equation [61]

$$x(t)+\sum_{j=1}^{n_a} a_j x(t-j)=\sum_{j=0}^{n_c} c_j e(t-j)+\sum_{i=1}^{m}\sum_{j=1}^{k} b_{ij} x(t-i)e(t-j),$$

where $c_0 = 1$. It is apparent that if b_{ij} is set to zero, for all i, j, then the above model reduces to the standard ARMA model.

3.3.1.2.4 Threshold Autoregressive Model (TAR)

The threshold autoregressive model uses a number of linear AR models, each one with different parameters. The model used at each time is chosen according to a threshold value *d*. A first order TAR model has the form [61]

$$x(t) = \begin{cases} a^1 x(t-1) + e^1(t), & \text{if } x(t-1) < d \\ a^2 x(t-1) + e^2(t), & \text{if } x(t-1) \geq d \end{cases},$$

where a^1, a^2 are constants and $e^1(t)$, $e^2(t)$ are white noise processes.
A first order *l*-threshold model has the form

$$x(t) = a^i x(t-1) + e^i(t), \text{ if } x(t-1) \in R^i, \quad i = 1,...,l,$$

where R^i are given subsets of the real line, with R^1 denoting the interval $(-\infty, r_1]$ and R^l the interval (r_{l-1}, ∞).

The *k*-order TAR model can now be defined as

$$x(t) = a_0^i + a_1^i x(t-1) + ... + a_k^i x(t-k) + e^i(t).$$

3.3.1.2.5 Exponential Autoregressive Model (EAR)

The exponential autoregressive models were introduced in an attempt to construct time series models, which reproduce certain features of non-linear random vibrations theory. The EAR model is obtained by replacing the constants of the AR model with exponential functions of $x^2(t\text{-}1)$. Thus, now the parameters of the model a_i are given by the equation

$$a_i = \phi_i + \pi_i \exp(-\gamma x^2(t-1)),$$

where ϕ_i, π_i and γ are constants.

The general form of the EAR model is [61]

$$x(t) = a_1 x(t-1) + ... + a_k x(t-k) + e(t).$$

3.3.2 Measuring Prediction Accuracy

3.3.2.1 Mean Absolute Deviation (MAD)

The mean absolute deviation (MAD) measures the prediction accuracy by averaging the magnitudes of the prediction error for each record in the data set. If the actual value of a time series at time *t* is denoted by *x*(*t*) and its prediction is denoted as $\hat{x}(t)$, the error or residual of the prediction is given by $e(t) = x(t) - \hat{x}(t)$. Then, the formula for the mean absolute deviation is [20]

$$MAD = \frac{\sum | x(t) - \hat{x}(t) |}{N} = \frac{\sum | e(t) |}{N},$$

where *N* is the number of records in the data set.

3.3.2.2 Mean Square Error (MSE) & Root Mean Square Error (RMSE)

The mean square error measures the prediction accuracy in a similar way to MAD. It averages the sizes of prediction errors avoiding the cancelling of positive and negative terms. The MSE, instead of using the absolute value of the prediction errors, uses their square value. The advantage of using the square value of the prediction errors is that it gives more weight to large prediction errors than MAD. The formula for the mean square error is

$$MSE = \frac{\sum (y(t) - \hat{y}(t))^2}{N} = \frac{\sum e(t)^2}{N},$$

where *N* is the number of records in the data set.

The MSE is measured in the squares of the units of the original series, which makes it harder to be interpreted. For this reason, the root mean square error can be evaluated, that is given simply by the equation

$$RMSE = \sqrt{MSE},$$

and is measured in the same units as the original time series.

3.3.2.3 Mean Absolute Percent Error (MAPE)

The mean absolute percent error (MAPE) is a unit-free evaluation measurement. This allows the comparison of the accuracy of the same or different models on different time series. It is evaluated by expressing each prediction error as a percentage according to actual value of the time series according to the formula [20]

$$MAPE = \frac{\sum \left| \frac{e(t)}{x(t)} \right|}{N} \cdot 100\% ,$$

where *N* is the number of records in the data set.

3.3.2.4 Pearson' s Coefficient of Correlation (r)

Often the prediction accuracy of a model is determined with the use of plots. A plot such as that is the diagnostic plot of $x(t)$ versus $\hat{x}(t)$. The criterion for a good model when using this plot is that the plotted points fall close to the 45° line. To assist the evaluation of the goodness of model using this diagnostic plot, the Pearson' s coefficient of correlation is introduced. The correlation coefficient is evaluated by the formula [20]

$$r = \frac{SS_{x\hat{x}}}{\sqrt{SS_{xx} SS_{\hat{x}\hat{x}}}} ,$$

where

$$SS_{xx} = \sum (x - \bar{x})^2$$

$$SS_{\hat{x}\hat{x}} = \sum (\hat{x} - \bar{\hat{x}})^2$$

$$SS_{x\hat{x}} = \sum (x - \bar{x})(\hat{x} - \bar{\hat{x}})$$

and $\bar{x}$, $\bar{\hat{x}}$ are the mean actual and predicted values of the time series respectively. The correlation coefficient *r* takes values in the (-1, 1). To make the assessment of the prediction accuracy easier, the coefficient of determination r^2 takes values in the range (0, 1). The closer r^2 is to 1, the closer the plotted points come to a straight line. Precaution has to be taken to make sure that the line where the points come close is actually the perfect prediction (45°) line.

3.3.2.5 Theil's Inequality Coefficient (U)

Theil's inequality coefficient (*U*) measures the prediction accuracy of a model in relation to the "no-change" model. The "no-change" model assumes that the values of a time series are relatively unchanging from period to period. Then, the current value of the time series is used as the forecast for the next period, i.e. $x(t+1) = x(t)$.

Theil's inequality coefficient (*U*) can be evaluated using three different formulas [20]

$$U = \frac{\sqrt{\sum e(t)^2}}{\sqrt{\sum (x(t) - x(t-1))^2}},$$

$$U = \frac{\sqrt{MSE(\text{model})}}{\sqrt{MSE(\text{"no - change" model})}}$$

and

$$U = \frac{\sqrt{\sum (A(t) - P(t))^2}}{\sqrt{\sum A(t)^2}},$$

where $P(t)$ is the predicted change in period *t*, $P(t) = \hat{x}(t) - x(t-1)$, and $A(t)$ is the actual change in period *t*, $A(t) = x(t) - x(t-1)$.

The first formula is calculated from raw data. The second formula shows clearly how the comparison to the "no-change" model is made and finally the third formula is most relevant when the objective is to predict changes in a time series.

If *U* is evaluated according to the second formula, the model predicts perfectly if *U*=0, since its MSE will also be zero. If the model predicts about as well as the "no-change" model then *U* will be 1. If *U* is less than 1 then the model predicts better than the "no-change" model, and if *U* is greater than 1 the model predicts worse than the "no-change" model.

3.3.2.6 Theil' s Decomposition of MSE

The MSE can be decomposed as

$$MSE = (\bar{\hat{x}} - \bar{x})^2 + (s'_{\hat{x}} - rs'_x)^2 + (1 - r^2)s'^{\,2}_x ,$$

where $\bar{x}$, $\bar{\hat{x}}$ are the mean actual and predicted values of the time series respectively, r is the Pearson' s coefficient of correlation and

$$s'_x = \sqrt{\frac{\sum (x(t) - \bar{x}(t))^2}{N}} ,$$

$$s'_{\hat{x}} = \sqrt{\frac{\sum (\hat{x}(t) - \bar{\hat{x}}(t))^2}{N}} .$$

If both sides of the decomposed form of the MSE are divided by the *MSE*, the decomposition form becomes

$$1 = \underset{U_M}{\left[\frac{(\bar{\hat{x}} - \bar{x})^2}{MSE}\right]} + \underset{U_R}{\left[\frac{(s'_{\hat{x}} - rs'_x)^2}{MSE}\right]} + \underset{U_D}{\left[\frac{(1 - r^2)s'^{\,2}_x}{MSE}\right]} .$$

Each of the three components U_M, U_R, U_D can now be interpreted as a proportion or percentage of the MSE. U_M measures the proportion of the MSE that is caused by bias in the prediction model. U_R measures how much of the MSE is due to the regression line deviating from the 45° line. Finally, U_D measures the proportion of the MSE that is caused by random disturbances that cannot be controlled.

3.3.3 Model Structure Selection

In previous sections, various types of models were presented that are possible to represent a system. When choosing a model to represent a system, its type is not the only consideration. Ideally a selected model would be the simplest possible with the smallest possible prediction error. It is obvious that there will be a compromise between the complexity of the model and its prediction accuracy.

One of the most common criteria for the trade off between complexity and accuracy is the Akaike Information Criterion (AIC) that is given by [72]

$$AIC = N \log p(x \mid \hat{\theta}) + 2k$$

for a sequence of observations $x_1, x_2, ..., x_N$ from a random variable *x*, which is characterised by the probability density function $p_x(\theta)$. *N* is the number of records in the data set and *k* is the number of parameters in the model evaluated.

Another common criterion is the Minimum Description Length (MDL), which is defined by the equation [72]

$$MDL = -\log p(x \mid \hat{\theta}) + 0.5k \log N ,$$

using the same notation as before.

3.3.4 Model Validation

The last step in the system identification procedure is to validate the identified model. Validation of a model can be performed in a number of ways.

A possible validating method is to check whether the model satisfies its purpose. For example, in a prediction problem, the model can be validated by evaluating the prediction error over a validation data set that is different from the data set used during the identification of the model.

For a model that is parameterised in terms of physical parameters, a validation is to confront the estimated values and their estimated variances with what is reasonable from prior knowledge.

A very useful technique for validation of a model is the residual analysis. The residuals of a model are defined as

$$\varepsilon(t) = x(t) - \hat{x}(t) ,$$

and carry information about the quality of the model. Various tests can be performed on the residuals to validate the model. For example if the model is of type ARMAX, three possible correlation tests for the residuals are:

- $\varepsilon(t)$ is zero mean white noise, $E\langle\varepsilon(t)\varepsilon(s)\rangle = 0,$ for all t, s
- $\varepsilon(t)$ is independent of past inputs, $E\langle\varepsilon(t)u(s)\rangle = 0,$ for $t > s$
- $\varepsilon(t)$ is independent of all inputs, $E\langle\varepsilon(t)u(s)\rangle = 0,$ for all t, s

3.4 Application of Neural Networks in Time Series Prediction

The advantage of using neural network models is that they can approximate or reconstruct any non-linear continuous function. The learning process of a neural network can be regarded as producing a multi-dimensional surface composed of a set of simpler non-linear functions that fit the data in some best sense. Different neural network architectures can be used in time series prediction.

3.4.1 Multi-layer Perceptron (MLP)

The past values of the time series are applied to the input of the network. The hidden layer of the MLP network performs the weighting summation of the inputs and the non-linear transformation is performed by the sigmoid function. The log-sigmoid function is

$$f(x) = \frac{1}{1+\exp(-x)},$$

and the tan-sigmoid function is

$$f(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}.$$

The output layer of the network performs a linear weighting summation of the outputs of all the hidden units, producing the predicted value of the time series as [14]

$$\hat{x}(t) = w_0 + \sum_{j=1}^{h} w_j f_j\left(\sum_{i=1}^{n} w_{ji}x(k-i) + w_{j0}\right),$$

where h is the number of hidden units, n is the number of input units, w_{ji} are the weights between the input and hidden layer, w_j are weights between the hidden and output layer and $f_j(\cdot)$ is the sigmoid activation function at the *j*th hidden unit. The weights are adjustable and are determined during the training of the network.

The number of hidden layers and hidden units has to be determined before the training of the network is performed. It has been suggested that for a training set with p samples, a network with one hidden layer with (p-1) hidden units can exactly implement the training set [14]. However, this is only guidance and the number of hidden layers and units is problem specific. In addition, according to the problem, other activation functions than the sigmoid can be used. A two-layer MLP can exactly represent any Boolean function. A two-layer MLP with log-sigmoids in the hidden layer and linear functions in the output layer can approximate with arbitrarily small error any continuous function. A three-layer MLP with the same transfer functions as before, can approximate non-linear functions to arbitrary accuracy.

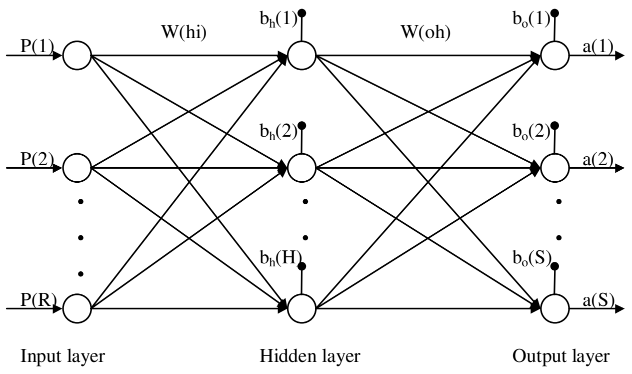


Figure 3.11 The Multi-layer Perceptron.

3.4.2 Radial Basis Function Networks (RBF)

The Radial Basis Function networks are two-layered structures. RBF networks have only one hidden layer with radial basis activation functions, and linear activation functions at the output layer. Typical choices for radial basis functions $\varphi(\mathbf{x}) = \Phi(\|\mathbf{x}-\mathbf{c}\|)$ are

- piecewise linear approximations: $\Phi(r) = r$,

- cubic approximation: $\Phi(r) = r^3$,
- Gaussian function: $\Phi(r) = \exp(-r^2 / \sigma^2)$,
- thin plate splines: $\Phi(r) = r^2 \log(r)$,
- multi-quadratic function: $\Phi(r) = \sqrt{r^2 + \sigma^2}$,
- inverse multi-quadratic function: $\Phi(r) = 1 \big/ \sqrt{r^2 + \sigma^2}$,

where σ is a parameter termed as the width or scaling parameter. The centres and widths of each radial basis function are determined during an initial training stage. The layer weights are determined in a different training stage.

The output of the network is a linear combination of the radial basis functions, and is given by [14]

$$x(t) = w_0 + \sum_{i=1}^{h} w_i \Phi\left(\left\|\mathbf{x}(t) - \mathbf{c}_i\right\|\right),$$

where $\mathbf{x}(t) = [x(t-1), x(t-2), ..., x(t-n)]^{\mathrm{T}}$.

RBF networks have the advantage that they have a simpler architecture than MLPs. In addition, they have localised basis functions, which reduces the possibility of getting stuck to local minima.

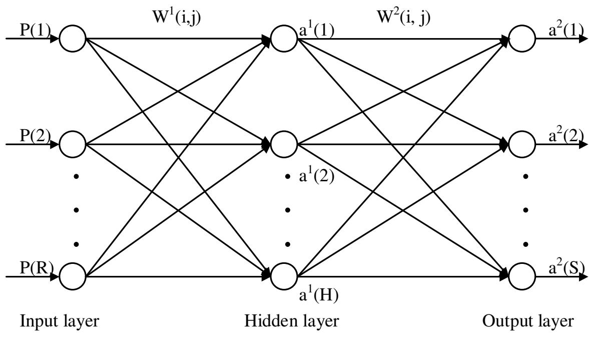


Figure 3.12 The Radial Basis Function network.

3.4.3 Sigma-pi and Pi-sigma Networks

Higher order or polynomial neural networks send weighted sums of products or functions of inputs through the transfer functions of the output layer. The aim of higher order neural networks is to replace the hidden neurons found in first order neural networks and thus reduce the complexity of their structure.

The sigma-pi network is a feedforward network with a single "hidden" layer. The output of the "hidden" layer is the product of the input terms and the output of the network is the sum of these products. They have only one layer of adaptive weights that results in fast training. The output of the network is given by

$$\hat{x}(t) = w_0 + \sum_{i=1}^{h} w_i \varphi_i(v_i) ,$$

where

$$v_i = \prod_{j=1}^{n} a_{ij} x_j ,$$

φ_i is the activation function at the "hidden" layer, a_{ij} are the fixed weights (usually set to 1) and w_i are the adjustable weights.

The pi-sigma network has a very similar structure to the sigma-pi network. Their difference is that the output of the hidden layer is the sum of the input terms and the output of the network is the product of these terms. They also have a single layer of adaptive weights, but in these networks the adaptive weights are in the first layer. The output of the network is

$$\hat{x}(t) = w_0 + \prod_{i=1}^{h} a_i \varphi_i(v_i) ,$$

where

$$v_i = \sum_{j=1}^{n} w_{ij} x_j ,$$

in the same notation as before.

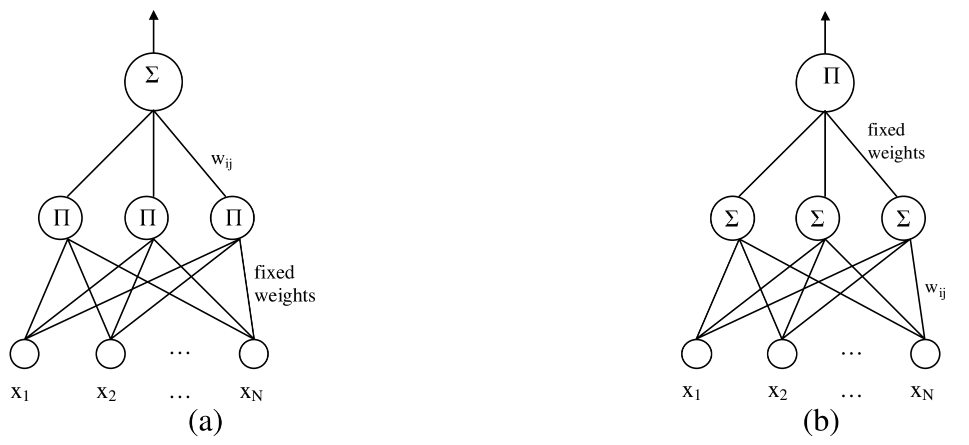


Figure 3.12 (a) The sigma-pi network, (b) The pi-sigma network.

3.4.4 The Rigde Polynomial network

The Ridge Polynomial network [66] is a generalisation of the pi-sigma network. It uses pi-sigma networks as basic building blocks. The hidden layer of the network consists of pi-sigma networks and their output is summed to give the output of the network. It also has only one layer of adjustable weights in the first layer.

Ridge Polynomial networks maintain the fast learning property of pi-sigma networks and have the capability of representing any multivariate polynomial. The Chui and Li' s representation theorem and the Weierstrass polynomial approximation theorem prove this property of ridge polynomial neural networks [26]. More details about Ridge Polynomial Neural Networks can be found in [26] and [66].

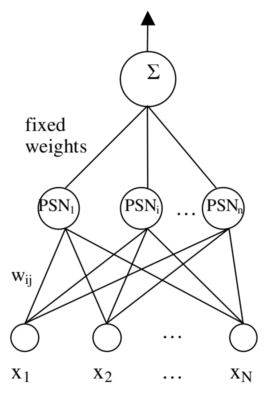


Figure 3.13 The Ridge Polynomial network

3.4 Group Method of Data Handling (GMDH)

The Group Method Data Handling (GMDH) [19] is a self-organising method that was initially proposed by Ivakhnenko to produce mathematical models of complex systems by handling data samples of observations. It is based on the sorting-out procedure, i.e. consequent testing of increasingly complex models, chosen from a set of models-candidates, in accordance with a given external criterion on a separate part of data samples. Thus, GMDH algorithms solve the argument

$$\tilde{g} = \arg \min_{g \subset G} CR(g),$$

where G is the set of candidate models and $CR(g)$ is an external criterion of model' g quality.

Most GMDH algorithms use polynomial reference functions to form the set of candidate models. The Kolmogorov-Gabor theorem shows that any function $y = f(\vec{x})$ can be represented as

$$y = a_0 + \sum_i a_i x_i + \sum_i \sum_j a_{ij} x_i x_j + \sum_i \sum_j \sum_k a_{ijk} x_i x_j x_k + \ldots,$$

where x_i is the independent variable in the input variable vector $\vec{x}$ and $\vec{a}$ is the coefficient vector. Other non-linear reference functions such as difference, logistic and harmonic can also be used. GMDH algorithms are used to determine the coefficients and terms of the reference functions used to partially describe a system. GMDH algorithms are multi-layered, and at each layer the partial description is simple and it is conveyed to the next layers to gradually obtain the final model of a complex system.

It has been proven that GMDH algorithms converge and that a non-physical model obtained by GMDH is better than a full physical model on error criterion [79]. A special feature of the GMDH algorithms is that the model to be selected is evaluated on a new data set, different from the one used to estimate its parameters.

3.4.1 Combinatorial GMDH algorithm (COMBI)

This is the simplest GMDH algorithm. First *n* observations of regression-type data are taken. These observations are divided into two sets: the training set and the validating set.

	Y	X - independent variables			
	Y_1	x_{11}	x_{12}	. . .	x_{1m}
Training	Y_2	x_{21}	x_{22}	. . .	x_{2m}
Observations					
	.	.	.	. . .	.
	.	.	.	. . .	.
	.	.	.	. . .	.
	Y_{nt}	$x_{nt,1}$	$x_{nt,2}$	. . .	$X_{nt,m}$
	.	.	.	. . .	.
Validation	.	.	.	. . .	.
Observations	.	.	.	. . .	.
	Y_n	x_n	x_{n2}	. . .	X_{nm}

The COMBI algorithm is multi-layered; at each layer, it obtains a candidate model of the system and once the models of each layer are obtained, the best one is chosen to be the output model.

The first layer model is obtained by using the information contained in every column of the training sample of observations. The candidate models for the first layer have the form

$$y = a_0 + a_1 x_i, i = 1,2,...,m .$$

To obtain the values of the coefficients a_0 and a_1 for each of the m models, a system of Gauss normal equations is solved. In the case of the first layer, the system of Gauss normal equation for the *i*th model will be

$$\begin{bmatrix} nt & \sum_{k=1}^{nt} x_{ki} \\ \sum_{k=1}^{nt} x_{ki} & \sum_{k=1}^{nt} x_{ki}^2 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{nt} y_k \\ \sum_{k=1}^{nt} x_{ki} y_k \end{bmatrix},$$

where *nt* is the number of observations in the training set.

After all possible models from this layer have been formed, the one with the minimum *regularity criterion AR*(*s*) [68] is chosen. The regularity criterion is defined by the formula

$$AR(s) = \frac{1}{nv} \sum_{i=nt+1}^{n} (y_i - \hat{y}_i)^2 ,$$

where *nv* is the number of observations in the validation set, *n* is the total number of observations, $\hat{y}$ is the estimated output value and *s* is the model whose fitness is evaluated.

A small number of variables that give the best results in the first layer, are allowed to form second layer candidate models of the form

$$y = a_0 + a_1 x_i + a_2 x_j, \quad i, j = 1,2,...,m .$$

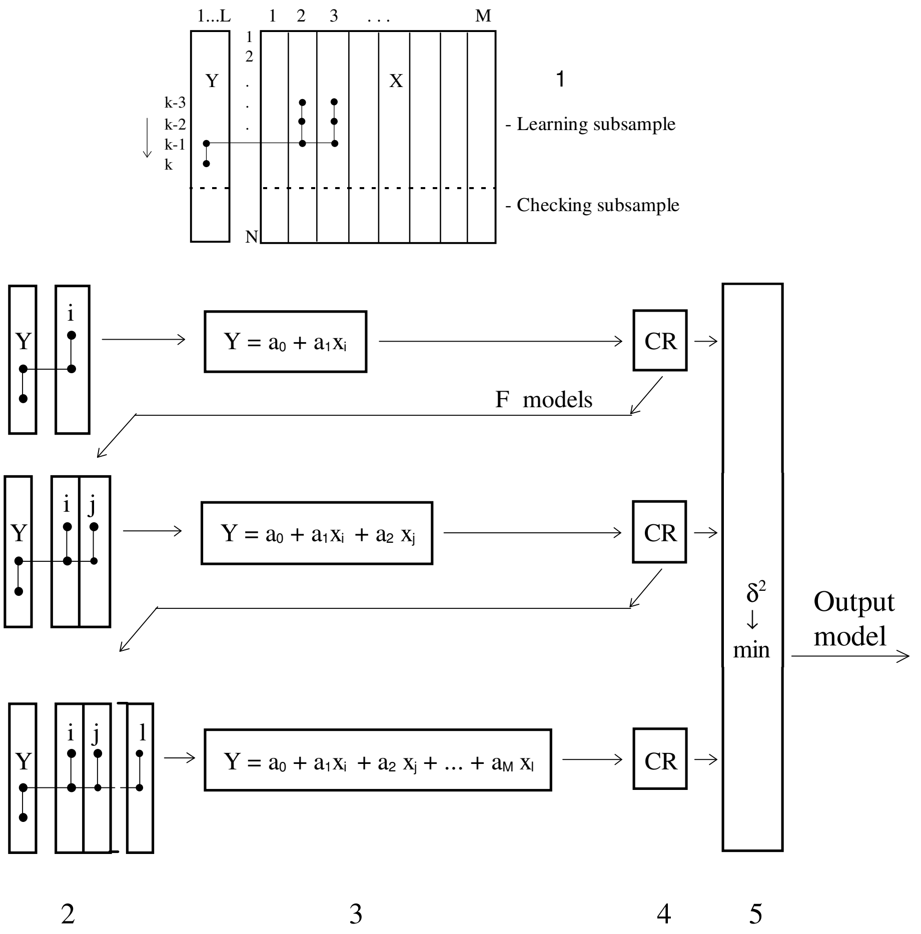
Models of the second layer are evaluated for compliance with the criterion, and again the variables that give best results will proceed to form third layer candidate models. This procedure is carried out as long as the criterion decreases in value, and candidate models at the *m*th layer will have the form

$$y = a_0 + a_1 x_i + a_2 x_j + ... + a_m x_l, \quad i, j, l = 1,2,...,m .$$

After the best models of each layer have been selected, the output model is selected by the *discriminating criterion* termed as δ^2. A possible discriminating criterion is the *variation criterion* $RR(s)$ defined by [6]

$$RR(s) = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2},$$

where $\bar{y}$ is the mean output value and s is the model whose fitness is evaluated. The model with the minimum value of the variation criterion $RR(s)$ is selected as the output model. Other discriminating criteria can be used that make a compromise between the accuracy and complexity of a model.

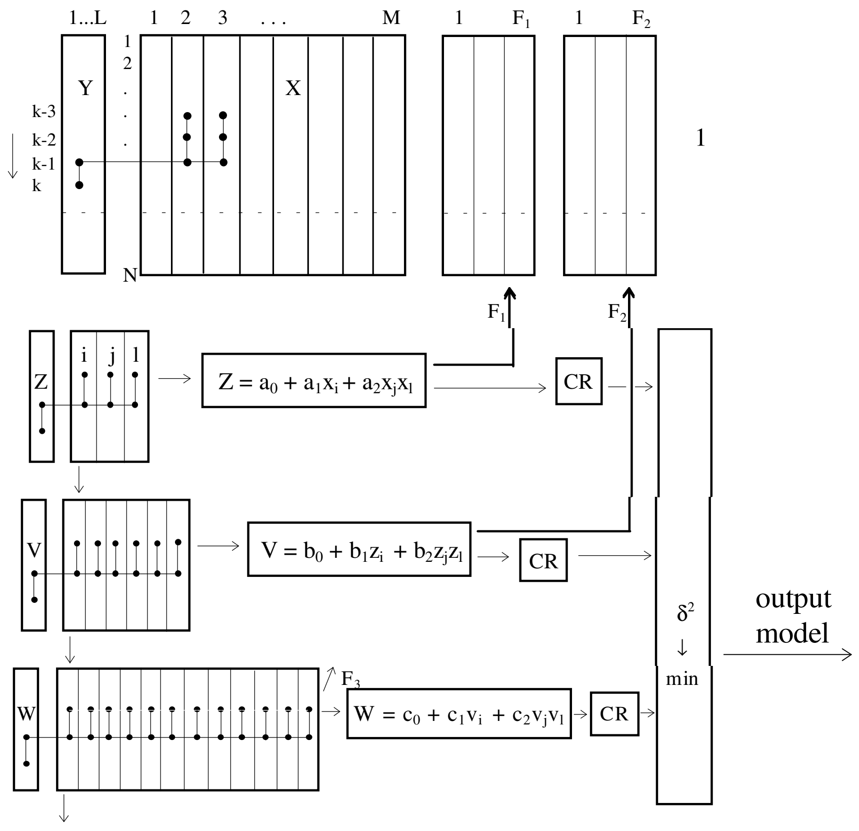


1 - data sampling,
2 - layers of partial descriptions complexing,
3 - form of partial descriptions,
4 - choice of optimal models,
5 - additional model definition by discriminating criterion.

Figure 3.9 The Combinatorial GMDH algorithm (COMBI).

3.4.2 Multi-layered Iterative GMDH algorithm (MIA)

The MIA algorithm [35] works in a similar way to the COMBI algorithm. The candidate models at the first layer are derived from the information contained in any two columns of the independent variables. The second layer uses information from four columns, the third from eight columns and so on. The optimal model at each layer is chosen in the same manner as in the COMBI algorithm, by the external criterion *CR*. In this algorithm though, a specified number of the best models in each layer are used to extend the input data sample. The output model is chosen again by a discriminating criterion.



Output model: $Y_{k+1} = d_0 + d_1x_{1k} + d_2\, x_{2k} + ... + d_m\, x_{M\,k}\, x_{M-1\,k}$

1 - data sampling
2 - layers of partial descriptions complexing
3 - form of partial descriptions
4 - choice of optimal models
5 - additional model definition by discriminating criterion
F1 and F2 - number of variables for data sampling extension.

Figure 3.11 The MIA algorithm.

3.4.3 Objective Systems Analysis Algorithm (OSA)

The OSA algorithm [39] realises candidate models as systems of equations between the independent variables of the observation data. It uses implicit templates, whose form is shown in Figure 3.10, to form the candidate models. The gradual increase in the complexity of candidate models corresponds to the increase in the complexity of the implicit templates. The output variables are not specified; instead any of the regressors can be an output variable.

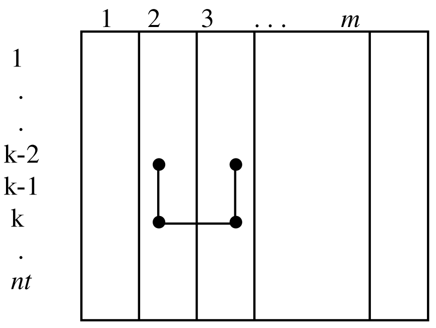


Figure 3.10 An implicit template.

At the first step of the OSA algorithm, candidate models are formed using information contained in each column of the independent variables matrix and have the form

$$x_{i(k)} = a_0 + a_1 x_{i(k-1)} + a_2 x_{i(k-2)},$$

where $\vec{x}, \vec{a}$ are the independent variables and coefficients vectors, *i* is the index of the column selected and *k* is the index of the observation point taken as an output.

At the second step, the information contained in two columns of the independent variables matrix are used, to form models represented by two-equation systems of the form

$$x_{i(k)} = a_0 + a_1 x_{i(k-1)} + a_2 x_{i(k-2)} + b_0 x_{j(k)} + b_1 x_{j(k-1)} + b_2 x_{j(k-2)}$$

$$x_j(k) = c_0 + c_1 x_{j(k-1)} + c_2 x_{j(k-2)} + d_0 x_{i(k)} + d_1 x_{i(k-1)} + d_2 x_{i(k-2)},$$

where *i*, *j* are the selected columns of independent variables, *k* is the point of observation in the training data set taken as the output and $\vec{a}$, $\vec{b}$, $\vec{c}$, $\vec{d}$ are the vectors of coefficients.

Models for the following steps are obtained in the same manner. The algorithm terminates when the *system criterion* stops decreasing. The system criterion is defined as

$$CR_{system} = \frac{1}{S}\sqrt{CR_1^2 + CR_2^2 + \ldots + CR_S^2} \ ,$$

where S is the number of equations in the system, and CR_i is the external criterion for each of the equations in the system. The output model is chosen as the system with the minimum system criterion.

The advantage of OSA is that the number of regressors is increased and in consequence, the information embedded in the data sample is utilised better. A disadvantage is that it requires a large amount of calculations to solve the system of equations and a greater number of models have to be searched.

3.4.4 Analogues Complexing Algorithm

The state of a complex object can be described by a vector of characteristic variables represented in the sample of observation data. To make the prediction, it is sufficient to find one or more close analogues of its state in previous history and to see what happened next. The model of the object is no longer needed since its role is played by the object itself. The Analogues Complexing algorithm [34] is non-parametric and is used when the dispersion of noise is too big. It searches for analogues from the given data sample which are equivalent to the physical model. Predictions are not calculated but selected from the observations data set.

The algorithm finds for the last part of the behaviour trajectory (output pattern), one or more analogous parts in the past (analogous pattern). The set of possible patterns $P_{i,k+1}$ for the output pattern $P_k^A = P_{n-k,k+1}$ is generated with the use of a sliding window, where $k+1$ is the width of the window, n is the total number of observations and a pattern is defined as $P_{i,k+1} = (\vec{x}_i, \vec{x}_{i+1}, \ldots, \vec{x}_{i+k})$.

The most similar patterns have to be selected from all possible patterns. The selection task of the most similar patterns is a four-dimensional problem with the following dimensions:

- variables included in the pattern,
- number of analogues selected,
- width of the patterns,
- values of weight coefficients with which the patterns are complexed.

If these patterns are found the prediction can be achieved by applying the known continuation of these analogous patterns.

3.4.5 Objective Computer Clusterisation (OCC) Algorithm

Clustering a data sample is its division into clusters. The number and points of the sample clusters are selected in such a way so that each cluster includes a compact group of closely situated sample points. Each point is present in only one cluster. Clustering can be regarded as a discrete description of a system.

OCC [38] is a non-parametric algorithm that finds optimal clusterisations of input data samples among all possible clusterisations. The use of data sample clusterisations as the description of complex systems is more effective than the use of parametric models, especially when the system is ill defined.

Construction of a hierarchical tree of clusterings orders and reduces the sorting while the clustering optimum according to a criterion is not lost. The optimal clusterisation can be found by the criterion of balance of clusterings. To compute the criterion, the sample is divided into two equal parts A, B. A clustering tree is constructed on each sub-sample and the criterion of balance is computed at each step with the same number of clusters. The criterion requires finding the clustering where both the number and co-ordinates of the centres (middle points) of corresponding clusters coincide and therefore the balance criterion BL is minimised. The balance criterion is defined as

$$BL = \frac{1}{MK}\sum_{j=1}^{M}\sum_{i=1}^{K}(x_{oA} - x_{oB})^2 -$$

where K is the number of clusters at a given step of the tree construction, M is the number of co-ordinates, x_{oA} are the co-ordinates of the centres of clusters constructed within part A, and x_{0B} are the co-ordinates of the centres of clusters within part B.

3.4.6 Probabilistic Algorithm based on the Multi-layered Theory of Statistical Decisions (MTSD)

In this algorithm, the optimal values of output variables are found according to the probabilities of the input variables. The single and pair empirical probabilities of the input variables in the data sample are calculated, and the optimal output value is found as the one with the maximum sum of probabilities defined as

$$\sum P = \sum P_{\text{single}} + \sum P_{\text{pair}} ,$$

where

$$P_{\text{single}} = \frac{E_{\text{single}}}{N},\ P_{\text{pair}} = \frac{E_{\text{pair}}}{N},\ E_{\text{pair}}(i,j) = E_{\text{single}}(i)\cdot E_{\text{single}}(j),$$

E is the expectation operator and N is the number of possible events. More details can be found in [37].

3.5 Twice Multi-layered Neural Nets (TMNN)

Self-organising modelling is based on *statistical learning networks*. These networks model complex systems by subdividing them into manageable pieces and then applying regression techniques to solve each of these problems. The disadvantage of the statistical networks is that they require a priori knowledge of the system to be modelled. The number of layers and nodes and their transfer functions have to be predefined. The GMDH overcomes this problem by adapting the architecture of the network while it builds the model of a system.

In twice multi-layered neural nets [36], the neurons comprising the network are also multi-layered. The self-organisation of a neural network is performed with the use of a GMDH algorithm to determine the number of neuron layers and the sets of input and output variables for each neuron.

The number of neurons in the first layer is equal to the number of independent variables given in the initial data set. The output variables of each layer of neurons are used as the input variables for the next layer. It is also possible to extend the regression area by allowing the input and output variables of a layer to be used as the input variables for the next layer. The extension of the regression area is normally accompanied by a threshold value for the number of variables that can be passed on from one layer to another. This is done to limit the computation time required.

3.6 Genetics-Based Self-Organising Network (GBSON)

In [42], Kargupta and Smith proposed a method for system identification using evolving polynomial networks. This approach was motivated from the work of Ivakhnenko who introduced the Group Method of Data Handling (GMDH).

The method introduced by Kargupta and Smith is the Genetics-Based Self-Organising Network (GBSON). It is a hybrid method of the GMDH and Genetic Algorithms. The GBSON method was introduced to overcome the drawbacks of the original GMDH algorithms, since they use local search techniques to obtain an optimal solution.

The GBSON uses polynomial neural networks to represent the model of the system to be identified. Each layer of the polynomial neural network is regarded as a separate optimisation problem. The input to the first layer of the network is the independent variables of the data sample. The output of each layer is the peak nodes obtained by the use of a multi-modal Genetic Algorithm. The peak nodes selected to be the output of a layer are also the inputs for the next layer.

The population members of the GA are network nodes represented by an eight-field bit string. The two first fields are used to represent the nodes from the previous layer connected to the present node. The other six fields are used to represent the coefficients of a quadratic function that determines the output of the node *y*,

$$y = a + bz_1 + cz_2 + dz_1z_2 + ez_1^2 + fz_2^2,$$

where z_1 and z_2 are the outputs of the connected nodes in the previous layer.

The fitness measure of a node is given by calculating its description length. The description length gives a trade off between the accuracy of the prediction and the complexity of the network. The equation used by Kargupta and Smith for calculating the description length is

$$I = 0.5n \log D_n^2 + 0.5m \log n,$$

where D_n^2 is the mean-square error, *m* is the number of coefficients in the model selected and *n* is the number of observations used to determine the mean -square error.

The multi-modal GA used in GBSON incorporates the fitness-sharing scheme, where the shared fitness is given by

$$f_i' = \frac{f_i}{m_i}.$$

f_i is the original fitness of the node and m_i is the niche count defined by

$$m_i = \sum_{j=1}^{N} sh(d_{ij}),$$

where

$$sh(d_{ij}) = \begin{cases} 1 - \left(\dfrac{d_{ij}}{\sigma_s}\right)^{\alpha} & \text{if } d_{ij} < \sigma_s \\ 0 & \text{otherwise} \end{cases},$$

N is the population size and d_{ij} is the Hamming distance between the members of the population *i* and *j*. The niche radius σ_s is determined by the equation

$$\frac{1}{2^l}\sum_{i=0}^{\sigma_s}\binom{l}{i}=\frac{l}{q},$$

where l is the string length and q is the number of nodes in the previous network layer.

New populations are obtained after applying the genetic operators of tournament selection, single-point crossover and point mutation. A mating restriction is also applied on the members to be crossed. If a member i is to be crossed, its mate j is selected such that $d_{ij} < \sigma_s$. If no such mate can be found then j is selected randomly.

The GBSON procedure continues until the GA converges to a layer with a single node.

3.7 Summary

In this chapter it was defined what a time series signal is and what are its basic properties. The procedure of time series signal prediction was presented, with the most commonly linear and non-linear models being stated. Methods for measuring the prediction were also outlined. The criteria for model structure selection and model validation were also stated. Following, the most commonly used neural networks in time series prediction were identified. Finally, the GMDH algorithm and GBSON method for time series prediction were analysed.

CHAPTER 4

RESULTS

4.1 Introduction

In this chapter, the method used to solve the time series prediction problem is firstly presented. Following, the results obtained with the implemented method are presented for various time series data. Finally, the performance of the implemented method is compared with the performance of the GMDH algorithm COMBI.

4.2 Implementation

The method used to solve the time series prediction problem is based on the GBSON method introduced by Kargupta and Smith [42]. This method is a hybrid of the GMDH and GA.

As with the original GMDH algorithm, initially the observed data table is formed. This table contains the past values of the time series to be predicted as m independent variables, x_i. The desired output values y_i are the dependent variables of the data table.

The independent variables of the data table form the input nodes of the first layer of the polynomial neural network used to model the time series. In the original GMDH algorithm, all possible combinations of pairs of the independent variables are formed to determine the parameters of the function used to evaluate the output of each node. The function used to evaluate the output of a node is

$$y = a + bx_i + cx_j + dx_i x_j + ex_i^2 + fx_j^2 .$$

The parameters of this function *a, b, c, d, e* and *f* are determined by solving a system of normal Gauss equations for each pair of independent variables. In the GBSON method, the pairs of independent variables and the parameters of the function used to evaluate the output of each node, are determined using a niched genetic algorithm.

The niched genetic algorithm is applied at each layer to determine the number of nodes in the layer, as well as the layer weights of the polynomial neural network. The output values of each layer are taken as the input values for the next layer to be determined. The construction of the polynomial neural network is terminated when the niched genetic algorithm constructs a layer with a single node.

In the following sections, the niched implemented genetic algorithm is described.

4.2.1 Representation

The potential network nodes are represented using an eight-field bit string. The first two fields represent the input of the current node, i.e. the nodes of the previous layer that it is connected with. The last six fields represent the parameters of the function used to determine the output of the current node. The schematic representation of a potential network node is shown in Figure 4.1. The size of the fields that represent the parameters of the function was set to 16 bits. The size of fields that represent the nodes of the previous layer the current node is connected to was set according to the number of independent variables of the previous layer.

		95 ... 80	79 ... 64	63 ... 48	47... 32	31 ... 16	15 ... 0
j	*i*	*f*	*e*	*d*	*c*	*b*	*a*

Figure 4.1 Schematic representation of potential network nodes.

4.2.2 Fitness Evaluation

A node' s fitness is evaluated according to its prediction error. The percent square error for a node is used as the node' s fitness in the GA. Thus, the fitness of a node is given by the formula

$$f_i = \frac{\sum_{t=1}^{nv} (x(t) - z_i(t))^2}{\sum_{t=1}^{nv} x(t)^2},$$

where *nv* is the number of points in the validation set, *x*(*t*) is the actual value of the time series at time *t* and $z_i(t)$ is the output of the *i*th node at time *t*.

4.2.3 Selection

The search for the optimal nodes in a layer of the network is a multi-modal problem. Therefore, a niched genetic algorithm is used and the selection is performed according to the shared fitness of a node. The shared fitness is given by

$$f_i' = \frac{f_i}{m_i}.$$

f_i is the original fitness of the node and m_i is the niche count defined by

$$m_i = \sum_{j=1}^{N} sh(d_{ij}),$$

where

$$sh(d_{ij}) = \begin{cases} 1-\left(\dfrac{d_{ij}}{\sigma_s}\right)^{\alpha} & \text{if } d_{ij} < \sigma_s \,, \\ 0 & \text{otherwise} \end{cases}$$

N is the population size and d_{ij} is the Hamming distance between the members of the population i and j. The constant a is set to one, to obtain the triangular sharing function. The niche radius σ_s is determined according to the method introduced by Jelasity [40], where the niche radius is defined as a function r. The function r is defined by the equation

$$r(i) = r(0)\beta^{i}, \beta \in (0,1) \quad ,$$

where i represents the layer at which the search is performed. The method for determining the constant β is described in section 2.3.1. This function reduces at each layer, and at the final layer the niche radius is equal to one. In the problems tested during this implementation, the niche radius became too small after the third layer of the network, and the genetic algorithm was not able to reach the optimal solution. Therefore, the niche radius is reduced only once for the second layer, and is kept constant for subsequent layers.

The selection of members of the population to be reproduced in the following generation is performed with the tournament selection operator. The tournament selection operator has been criticised when used with the fitness sharing scheme. However, when the tournament selection operator is used with a relatively high selection pressure (k = 6), the results obtained are satisfactory. The performance of the niched genetic algorithm was also tested with the use of the roulette wheel selection operator, implemented with stochastic universal sampling (SUS) as suggested in [64]. The tournament selection operator outperformed the SUS selection method in all cases.

4.2.4 Crossover & Mutation

Crossover is implemented using the 2-point crossover operator. It has been suggested to use mating restriction schemes, to avoid the formation of lethal offspring [42], [64]. A mating scheme proposed by Deb and used in [42] allows recombination between members whose distance is less than a dissimilarity threshold. This mating restriction scheme was tested in the implementation of the niched genetic algorithm, but crossover between randomly selected parents gave better results. The standard bit mutation operator was implemented.

4.2.5 Elitism

It is possible for a GA, after the crossover and mutation operators are applied, to result to worse solutions. To prevent this, the elitist operator was implemented. The elitist operator checks if the best peak value detected in the current population is better or worse than the best peak value of the previous population. If it is worse than the best peak value in the previous generation, then the worst members of the current population are replaced by the peak values of the previous population.

4.3 Simulation results

4.3.1 Sunspot Series

The first set of experiments was conducted on monthly sunspot numbers, recorded by the Sunspot Index Data Center (SIDC), from January 1749 to July 1999. These numbers are indicative of the average relative number of sunspots observed every day of the month. The solar energy output of the sun ionises the particles in the ionosphere. In result, the solar energy output of the sun determines which frequencies will pass through the ionosphere and which frequencies will bounce back to the earth. The prediction of the solar activities is therefore essential to organisations planning long-range high frequency communication links and space-related activities. The sunspot time series has been classified as quasi-periodic, and it has been found that the period varies between 7 to 16 years with irregular amplitudes, making the time series hard to predict.

The objective of the experiment is to generate a single-step prediction based on past observations. The data were normalised to take values from zero to one, before using them as input data to the polynomial neural networks. The input pattern was assigned as ($x(t$-1), $x(t$-2), $x(t$-3)) and the desired output was

$$x(t) = f((x(t-1), x(t-2), x(t-3))\,.$$

From the 3000 available data points, 500 points (2000 to 2500) were used for the validation of potential models. The experiments were run with a population size of 100 for 500 generations, with tournament size 6, probability of crossover 0.9 and probability of mutation 0.01.

GBSON resulted to a network with four layers to model the sunspot series. The polynomial neural network constructed by GBSON is shown in Figure 4.2. The network nodes at each layer are shown in ascending order, according to their PSE. Thus, nodes at the top are the ones with the smallest PSE for each layer. The most significant term in the partial descriptions,

$$y = a + bx_i + cx_j + dx_i x_j + ex_i^2 + fx_j^2\,,$$

of the model was the term x_j and the less significant term was the constant term. The past values of the sunspot series, ($x(t$-1), $x(t$-2), $x(t$-3)), contributed equally to obtain the final model.

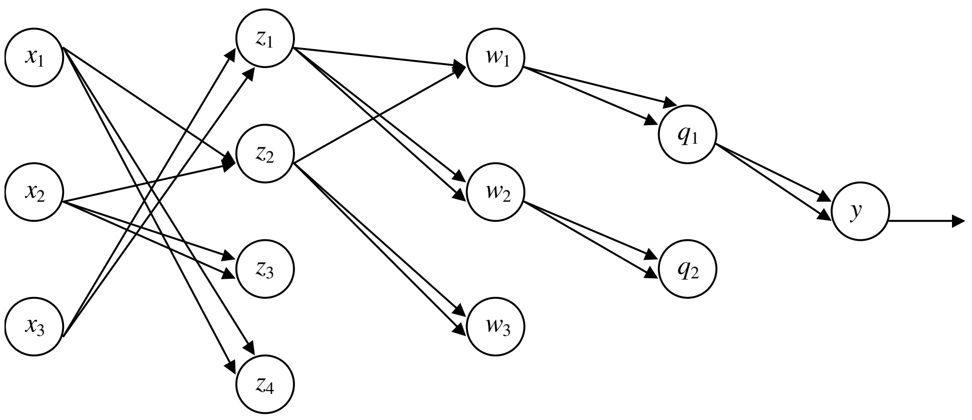


Figure 4.2 Network obtained for the sunspot series by GBSON.

The results of the prediction can be seen in Figure 4.3. The actual error of the prediction is shown in Figure 4.4. The percent square error (PSE) over the whole data set is 0.057589 and the root mean square error (RMSE) is 0.004167. The PSE over the validation data set is 0.052777. The difference of the PSE over the whole data set and the validation data set is small (0.004812), and thus the model obtained performs with approximately the same accuracy in data points that have not been used in any part of the modelling process.

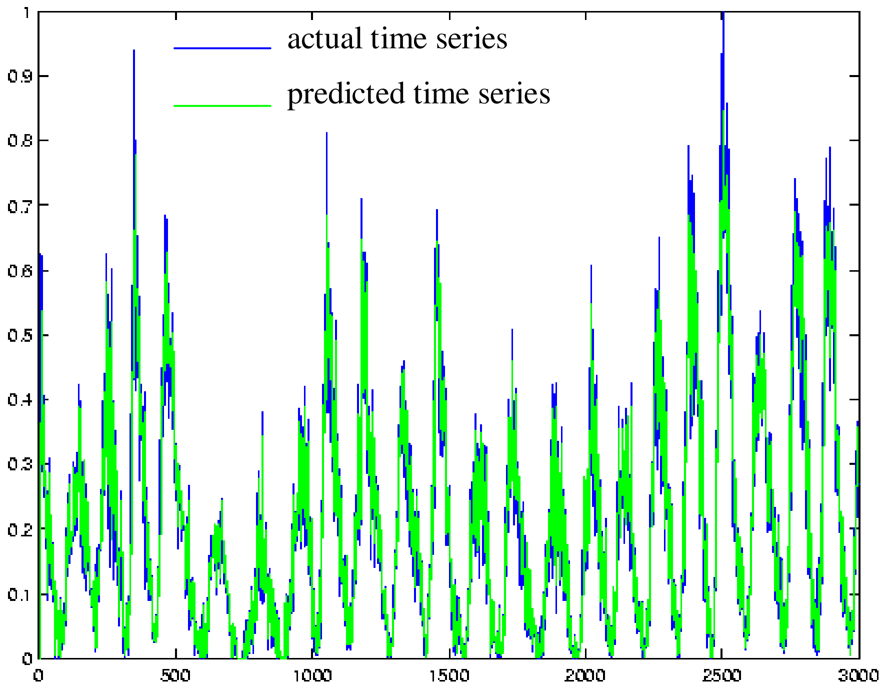


Figure 4.3 The actual and predicted with GBSON sunspot time series.

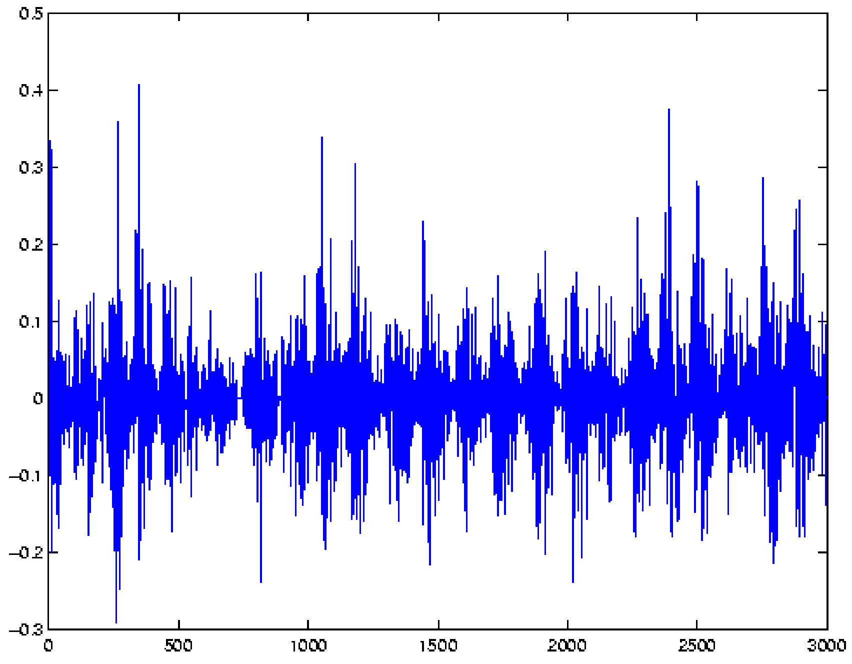


Figure 4.4 The actual error for each point of the prediction with GBSON of the sunspot time series.

4.3.2 Lorentz Attractor

Edward Lorentz obtained the Lorentz attractor system, in his attempt to model how an air current rises and falls while it is heated by the sun. The Lorentz attractor system is defined by the following three ordinary differential equations.

$$\frac{dx(t)}{dt} = \sigma x(t) - \sigma y(t)$$
$$\frac{dy(t)}{dt} = -y(t) + rx(t) - x(t)y(t)$$
$$\frac{dz(t)}{dt} = -bz(t) + x(t)y(t)$$

The Lorentz attractor system has also been used to model a far-infrared NH_3 laser that generates chaotic intensity fluctuations [41]. The far-infrared NH_3 laser is described by exactly the same equations, only the variables and constants have different physical meaning.

The time series used in this experiment, is the *x-component* in the Lorentz equations. The data were generated by solving the system of differential equations, that describe the Lorenz attractor, with the initial conditions of $\sigma = 10$, $r = 50$ and $b = 8/3$. The data were again normalised to take values from zero to one, before they were used as inputs to the polynomial neural networks.

The objective is to make one-step ahead prediction. The prediction is based on four past values ($x(t\text{-}1)$, $x(t\text{-}2)$, $x(t\text{-}3)$, $x(t\text{-}4)$) and thus the output pattern is

$$x(t) = f((x(t-1), x(t-2), x(t-3), x(t-4)).$$

The experiments were performed with 100 members in each population for 500 generations, with tournament size 6, probability of crossover 0.95 and probability of mutation 0.03. The data points 2000 to 2500 were used for model validation.

The network constructed by the GBSON method to model the Lorentz attractor has eight layers, and it is shown in Figure 4.5. The network nodes at each layer are shown in ascending order, according to their PSE. Thus, nodes with the smallest PSE for each layer are at the left, and nodes with the highest PSE for each layer are at the right. The most significant term in the partial descriptions,

$$y = a + bx_i + cx_j + dx_i x_j + ex_i^2 + fx_j^2,$$

of the model was the term x_i^2 and the less significant term was the constant term. The input variables x_3 and x_4, were the most significant variables in the model.

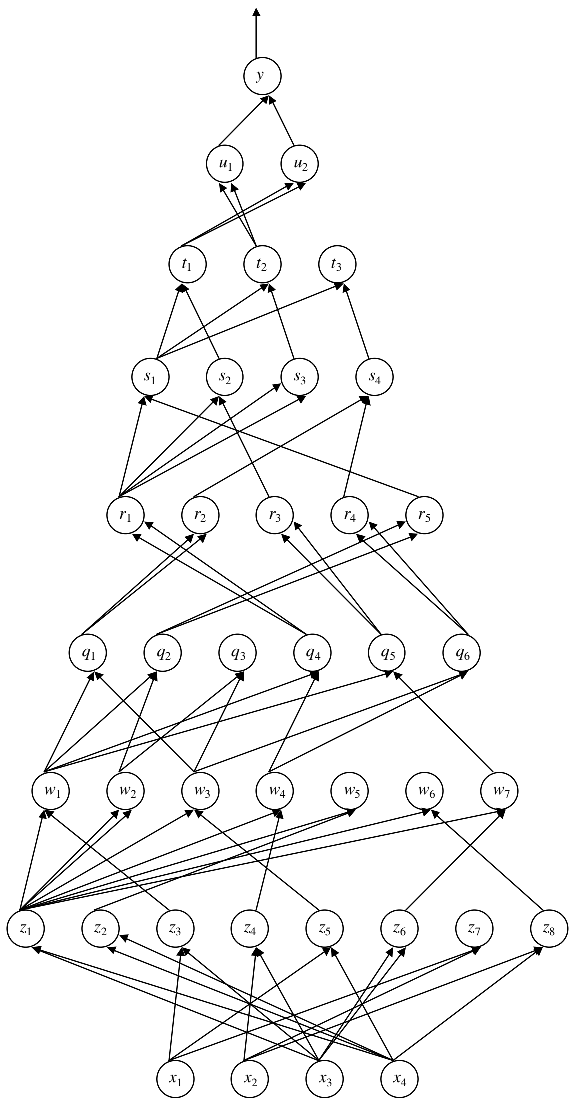


Figure 4.5 Network obtained for the Lorentz attractor by GBSON.

The results of the prediction and the actual system can be seen in Figure 4.6. The actual error of the prediction for each data point is shown in Figure 4.7. The PSE over the whole data set is 0.000244 and the RMSE is 0.000050. The PSE over the validation data set is 0.000231. As with the sunspot series, the difference of the PSE over the whole data set and the validation data set is small, and thus the generalisation of the network is very good.

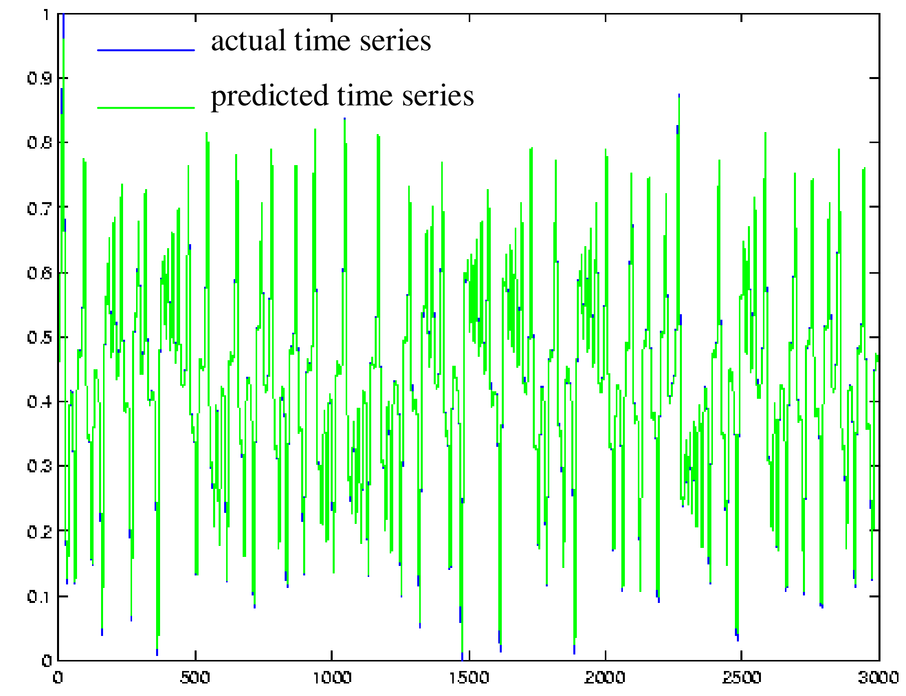


Figure 4.6 The predicted with GBSON and actual Lorentz attractor system.

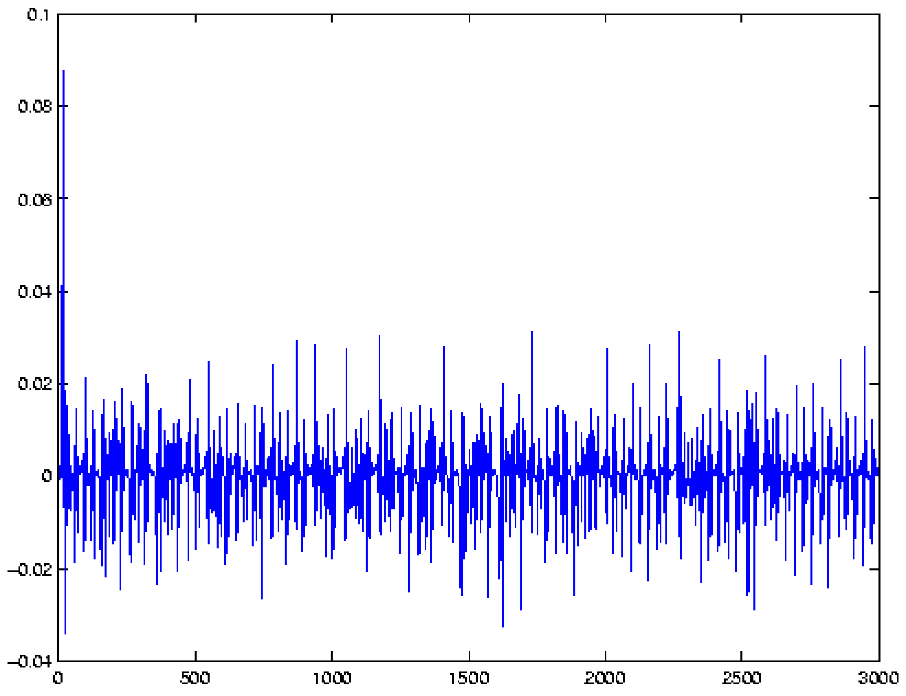


Figure 4.7 The actual error for each data point obtained from the prediction of the Lorentz attractor system with GBSON.

4.3.3 Exchange Rates

The last set of data to test the GBSON method, is the exchange rates for the British pound, the Canadian dollar, the Deutsche mark, the Japanese yen and the Swiss franc against the U.S. dollar. The data used are the daily observed exchange rates for all the above mentioned currencies from the 1st of June 1973 to the 21st of May 1987. The data were again normalised to take values from zero to one, before they were used as inputs to the polynomial neural networks.

The objective is to make one-step ahead prediction. The three past values were used as an input pattern, and thus the output pattern is

$$x(t) = f((x(t-1), x(t-2), x(t-3)).$$

The simulations were run with a population of size 100 for 500 generations with size of tournament 6, probability of crossover 0.95 and probability of mutation 0.03. The data points from 0 to 2500 were used for model validation.

The networks constructed by GBSON to model the exchange rates of all the currencies against the U.S. dollar, have one layer, except for the network used to model the Deutsche mark against the U.S. dollar that has three layers. The network used to model the exchange rates of the Deutsche mark against the U.S. dollar is shown in Figure 4.8. The models for the exchange rates of each currency are given below.

Britsh pound against U.S. dollar:

$$y = -0.0079 + 2.0192x_2 - 0.9976x_2 - 1.1946x_2x_2 + 1.0784x_2^2 + 0.1015x_2^2$$

Canadian dollar against U.S. dollar:

$$y = 0.0776 + 1.1060x_2 - 0.2842x_2 + 0.2953x_2x_2 + 0.6248x_2^2 - 0.8187x_2^2$$

Deutsche mark against U.S. dollar:

$$z_1 = 0.2041 + 0.5891x_3 - 0.0928x_2 + 0.2973x_3x_2 + 0.0688x_3^2 - 0.0605x_2^2$$

$$z_2 = 0.2041 + 0.5891x_3 - 0.0928x_3 + 0.2973x_3x_3 + 0.0696x_3^2 - 0.0589x_3^2$$

$$q_1 = 0.0242 - 0.6176z_1 + 1.5584z_2 + 0.6958z_1z_2 + 0.09380z_1^2 - 0.7571z_2^2$$

$$y = -0.3239 - 1.0011q_1 + 2.7956q_1 + 0.4945q_1q_1 - 1.0095q_1^2 + 0.0363q_1^2$$

Japanese yen against U.S. dollar:

$$y = -0.0463 + 2.6224x_2 - 1.4567x_2 - 0.6805x_2x_2 + 1.3770x_2^2 - 0.8416x_2^2$$

Swiss franc against U.S. dollar:

$$y = 0.1055 + 0.8191x_2 - 0.1243x_2 - 2.3707x_2x_2 + 1.1410x_2^2 + 1.4425x_2^2$$

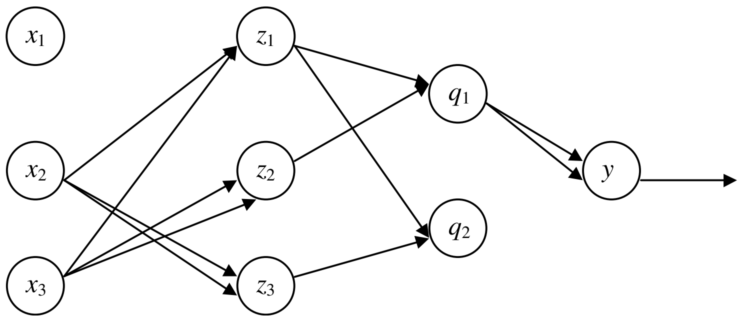


Figure 4.8 Network obtained for the exchange rates of the Deutsche mark against the U.S. dollar by GBSON.

The prediction results for all the currencies along with the actual time series, and the actual error of the prediction for each data point are shown in Figures 4.9 to 4.18. The PSE for the whole data set for the British pound is 0.000035, for the Canadian dollar it is 0.000005, for the Deutsche mark it is 0.000052, for the Japanese yen it is 0.000043 and for the Swiss franc it is 0.000102. The corresponding RMSE for each currency is 0.000020, 0.000004, 0.000029, 0.000014 and 0.000048, respectively. The generalisation of the models obtained was not as good as in the previous experiments. The models for the British pound, the Canadian dollar and the Deutsche mark, gave rise to a PSE for the new data set, double of the one for the validation set. The models for the Japanese yen and the Swiss franc generalised well over the new data set, and gave rise to a smaller PSE for this set than the one for the validation set.

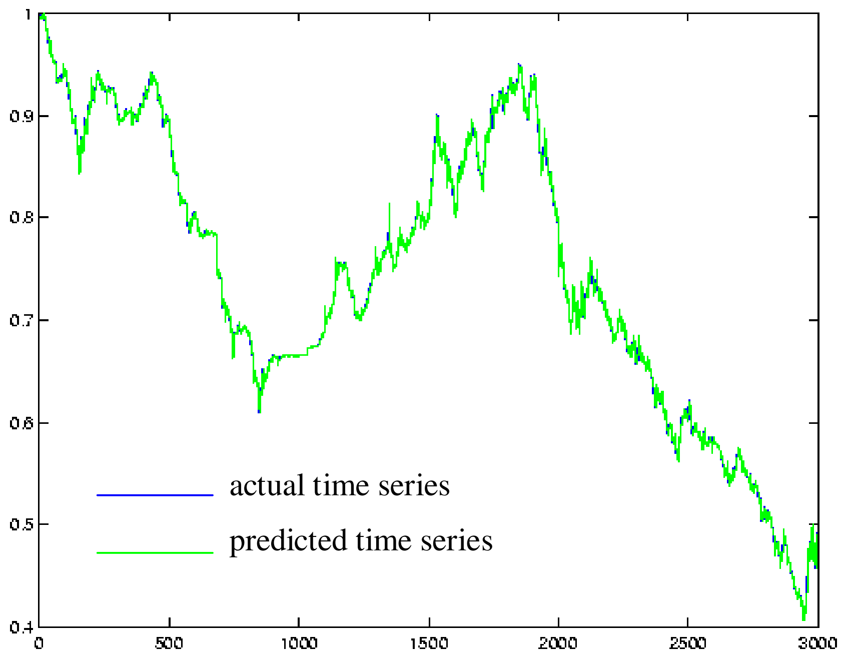


Figure 4.9 The actual and predicted with GBSON exchange rates for the British pound against the U.S. dollar.

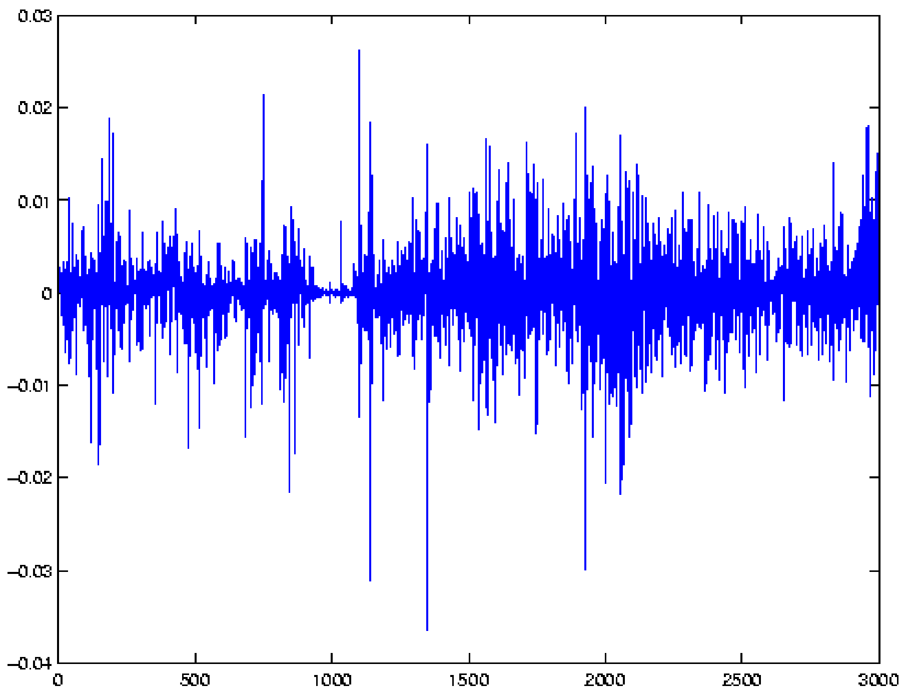


Figure 4.10 The actual error for each point of the exchange rates of the British pound against the U.S. dollar predicted with the GBSON.

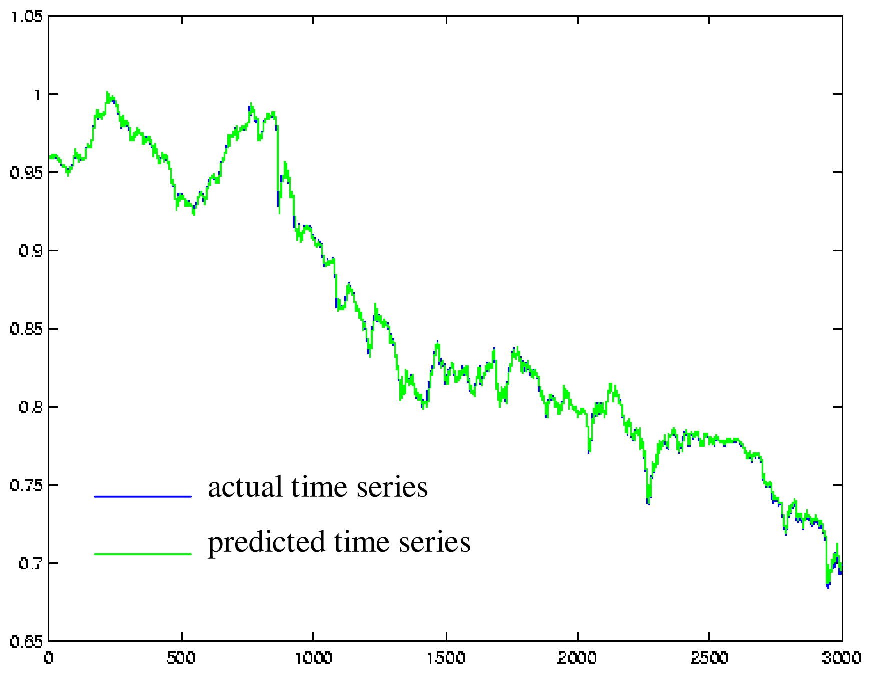


Figure 4.11 The actual and predicted with GBSON exchange rates for the Canadian dollar against the U.S. dollar.

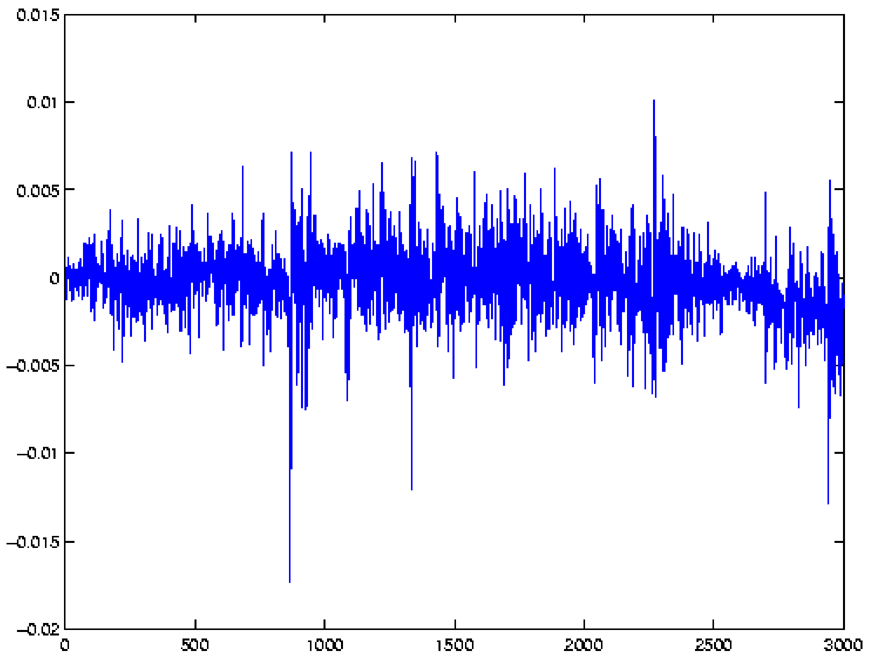


Figure 4.12 The actual error for each point of the exchange rates of the Canadian dollar against the U.S. dollar predicted with the GBSON.

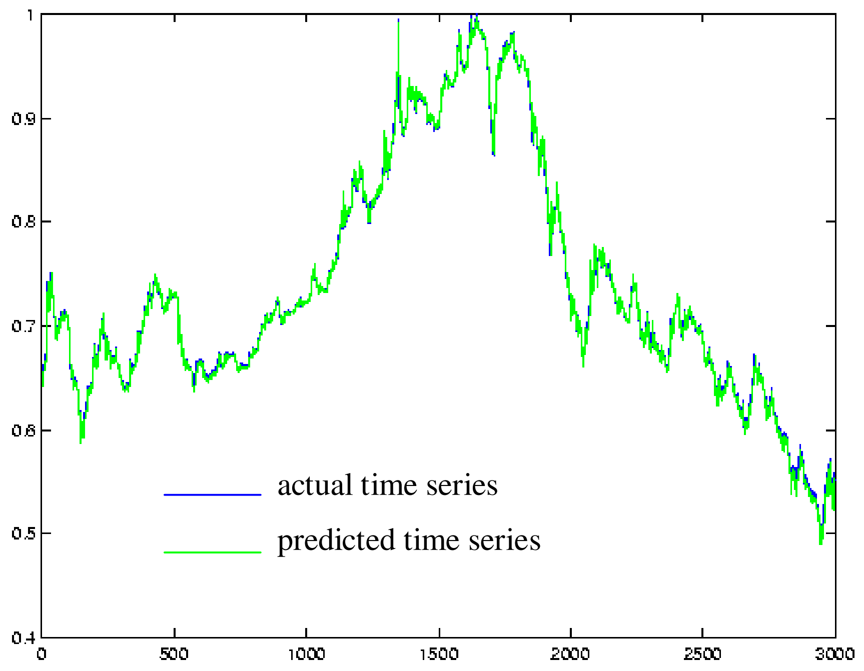


Figure 4.13 The actual and predicted with GBSON exchange rates for the Deutsche mark against the U.S. dollar.

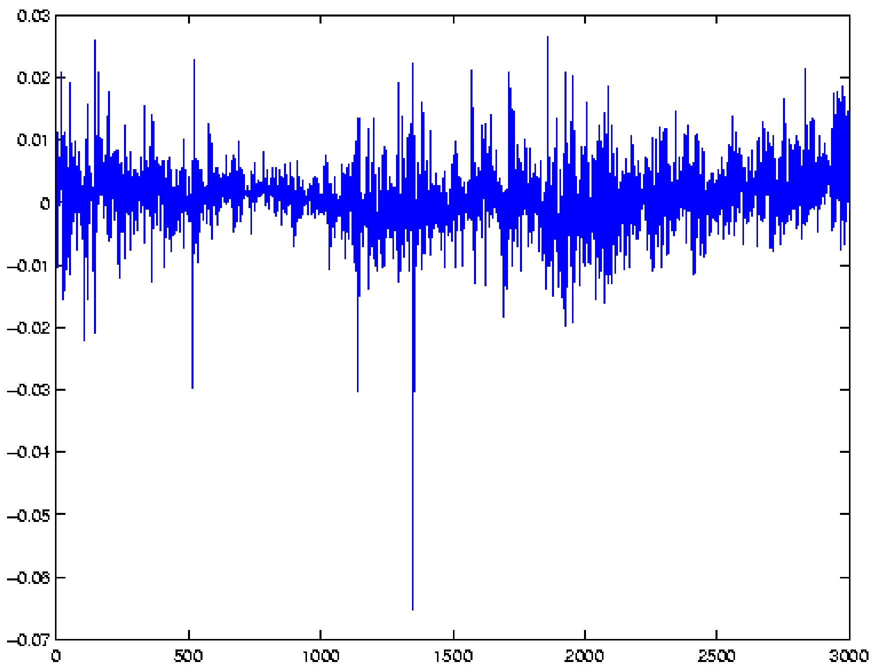


Figure 4.14 The actual error for each point of the exchange rates of the Deutsche mark against the U.S. dollar predicted with the GBSON.

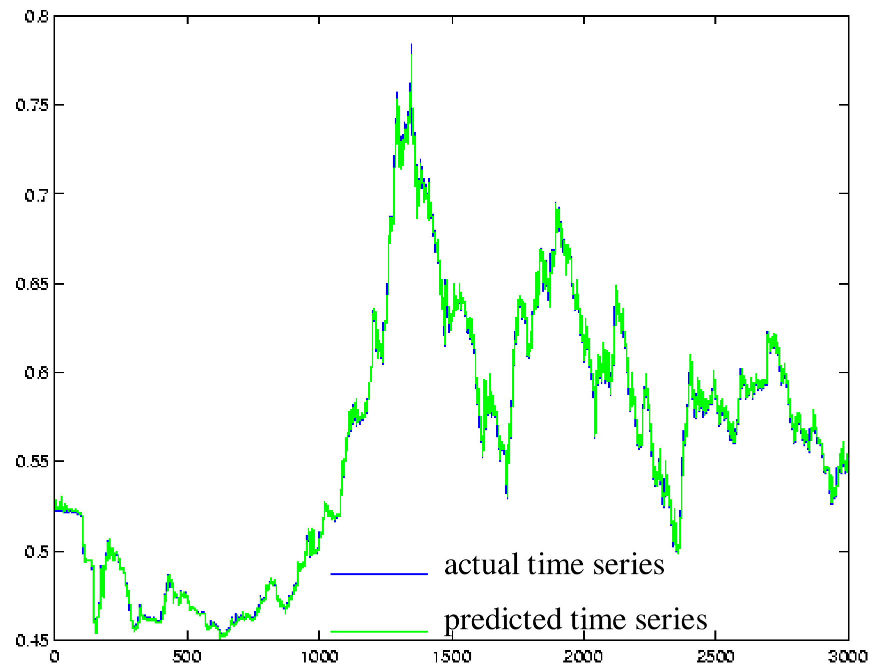


Figure 4.15 The actual and predicted with GBSON exchange rates for the Japanese yen against the U.S. dollar.

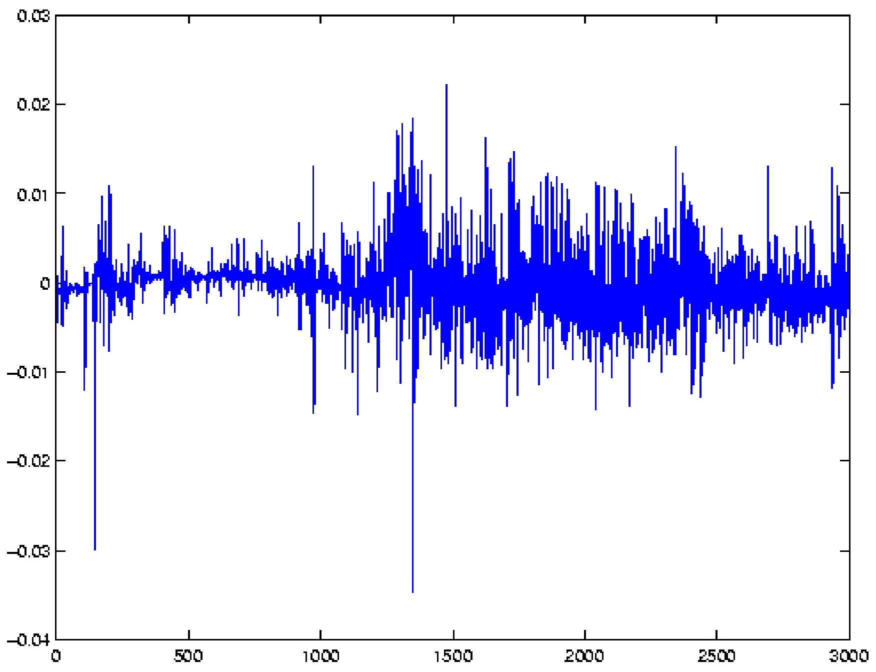


Figure 4.16 The actual error for each point of the exchange rates of the Japanese yen against the U.S. dollar predicted with the GBSON.

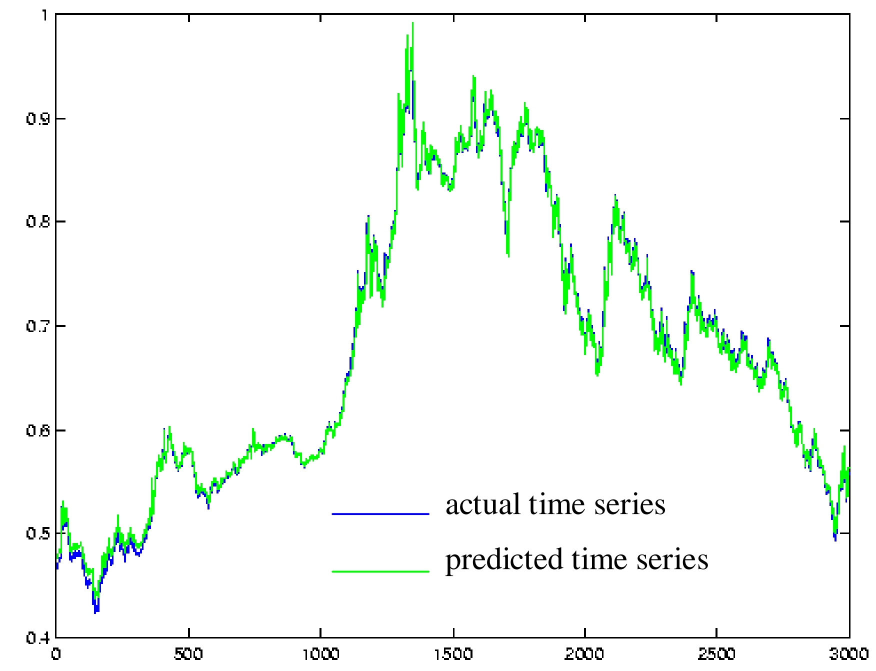


Figure 4.17 The actual and predicted with GBSON exchange rates for the Swiss franc against the U.S. dollar.

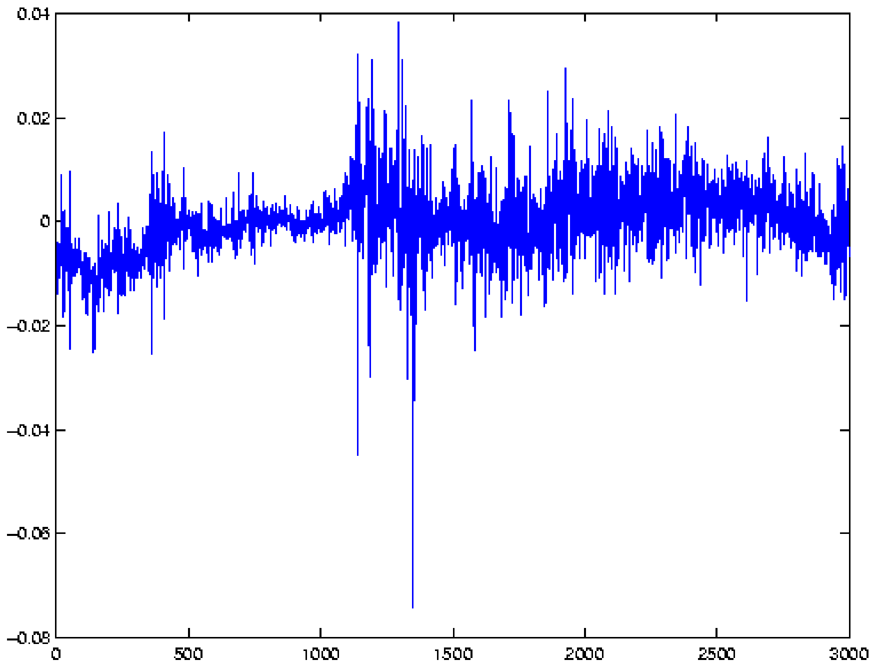


Figure 4.18 The actual error for each point of the exchange rates of the Swiss franc against the U.S. dollar predicted with the GBSON.

4.4 Comparison Of Results With The GMDH Algorithm

The simulation results presented in the previous section are compared with the results obtained using the GMDH COMBI algorithm described in section 3.4.1. To allow better comparison of the results obtained using the GBSON and COMBI algorithms, the same number of data was used for training and validation. Therefore, in all time series tested, the first 2000 points are used for training, the following 500 points are used for validation and the last 500 points are used for testing the model obtained in data that have not been used in any part of the modelling process. The model' s fitness is based on the percent square error as in the GBSON method.

4.4.1 Sunspot Series

The input pattern was assigned as ($x(t$-1), $x(t$-2), $x(t$-3)) and thus the output pattern is

$$x(t) = f((x(t-1), x(t-2), x(t-3)),$$

as in the GBSON method.

The algorithm resulted to a network with two layers, and its structure can be seen in Figure 4.19. The partial descriptions of the model are

$$z_1 = 0.0032 + 0.3736x_1 + 0.6676x_3 - 0.0458x_1x_3 + 0.1128x_1^2 - 0.2732x_3^2,$$

$$z_2 = 0.0033 + 0.4368x_2 + 0.6144x_3 - 0.268x_2x_3 - 0.0070x_2^2 + 0.0507x_3^2,$$

and the output of the network is given by

$$y = -0.0007 + 0.6962z_1 + 0.2959z_2 + 0.0092z_1z_2 - 0.4411z_1^2 + 0.4760z_2^2.$$

The percent square error (PSE) and the root mean square error (RMSE) over the whole data set are 0.061066 and 0.004419, respectively. The PSE and RMSE for each of the data sets for the COMBI and GBSON algorithms, are summarised in Table 4.1. The actual time series as well as the output generated by the network constructed by the COMBI algorithm is shown in Figure 4.20. The actual error for each point in the data set is shown in Figure 4.21.

	PSE	PSE	PSE	PSE	RMSE
	whole set	Training set	Validation set	new data set	Whole set
COMBI	0.061066	0.068587	0.063761	0.051060	0.004419
GBSON	0.057589	0.071203	0.052777	0.038802	0.004167

Table 4.1 Comparison of the results for the sunspot time series.

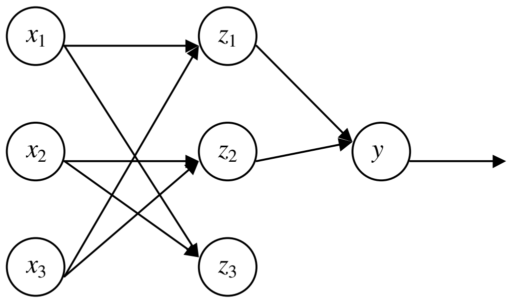


Figure 4.19 Network obtained for the sunspot series by COMBI.

The network obtained with the COMBI algorithm is less complex than the one obtained with the GBSON method. Nevertheless, the results obtained with the GBSON method are approximately 6% better for all the data points from the results obtained with the COMBI. In addition, the prediction over the new data set is 24% better. The only data set that the COMBI predicted with a smaller error, is the training set. This set though, is a new data set for the GBSON method, since the parameters are determined with a GA, and there is no training set for the GBSON. The GBSON method used only the points in the validation set to determine the fitness of solutions obtained by the GA. As a result, the GBSON algorithm generalises better than the COMBI algorithm.

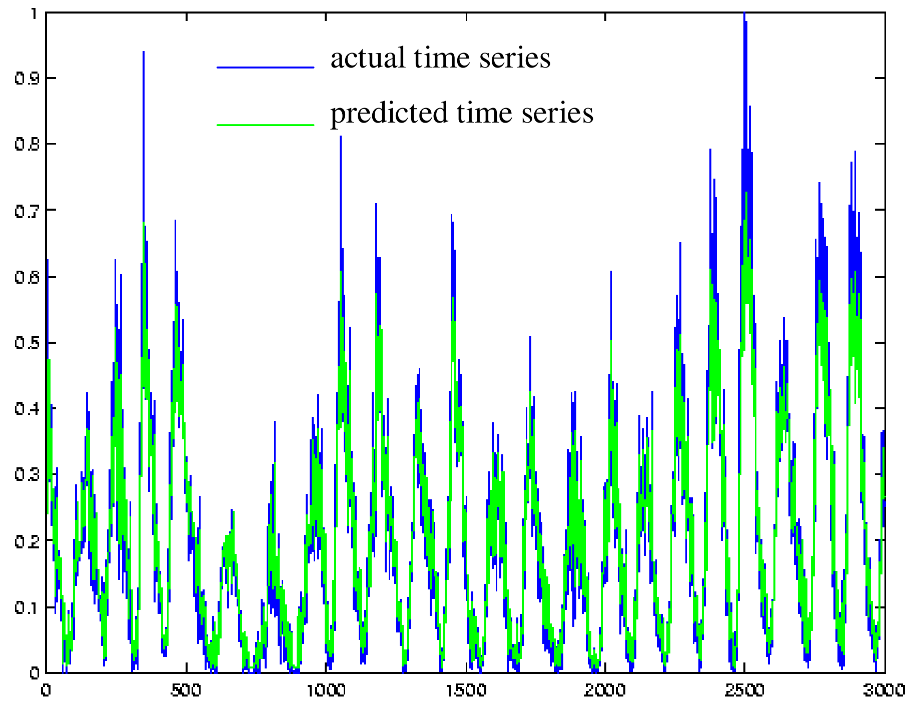


Figure 4.20 The actual sunspot time series and the prediction with COMBI.

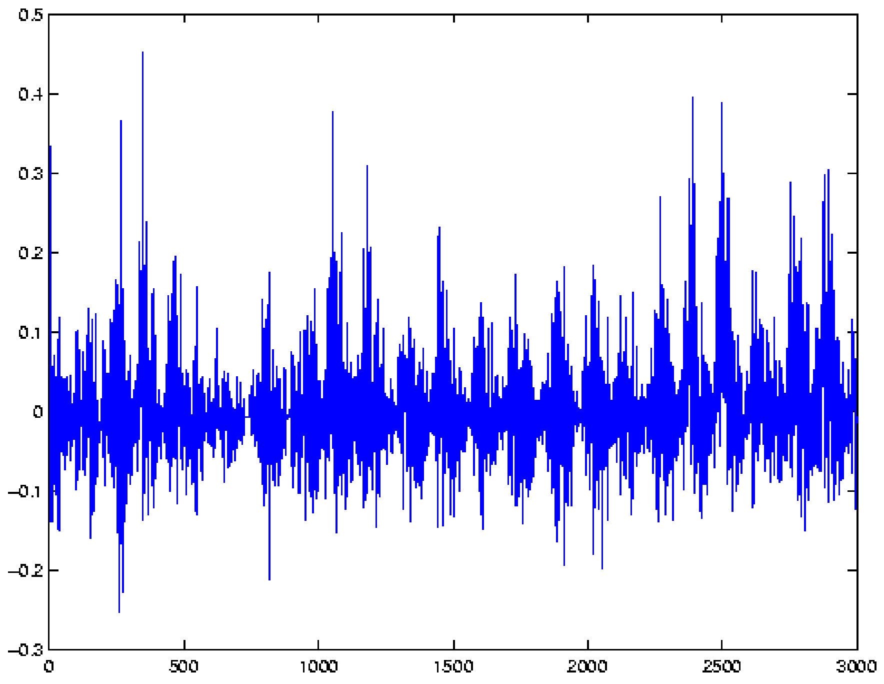


Figure 4.21 The actual error for each point of the sunspot time series predicted with COMBI.

4.4.2 Lorenz Attractor

For the Lorentz attractor the input pattern was ($x(t$-1), $x(t$-2), $x(t$-3), $x(t$-4)) and thus the output pattern is

$$x(t) = f((x(t-1), x(t-2), x(t-3), x(t-4)),$$

as in the GBSON method.

The COMBI algorithm converged to a network with two layers, shown in Figure 4.22. The partial descriptions of the model are

$$z_1 = 0.0127 - 0.9372x_3 + 1.9106x_4 + 0.1629x_3x_4 + 0.1971x_3^2 - 0.3664x_4^2,$$

$$z_3 = 0.0368 - 1.7343x_2 + 2.6571x_3 + 0.3694x_2x_3 + 0.4472x_2^2 - 0.8348x_3^2,$$

and the output of the network is given by

$$y = 0.0023 + 1.3140z_1 - 0.3197z_3 - 0.1995z_1z_3 - 0.2160z_1^2 + 0.4166z_3^2.$$

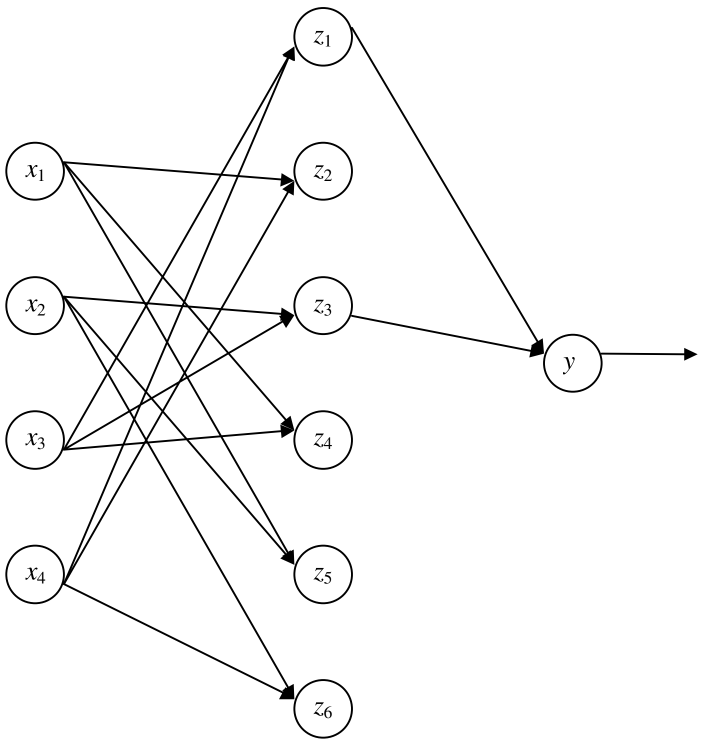


Figure 4.22 Network obtained for the Lorenz attractor by COMBI.

This network predicted the Lorentz attractor system with a PSE over the whole data set of 0.006652. The RMSE for the whole data set again was 0.001377. The actual system and its prediction are shown in Figure 4.23. The actual error of the prediction can be seen in Figure 4.24.

The comparison between the prediction achieved with the COMBI and GBSON algorithms is summarised in Table 4.2. The complexity of the model obtained with the GBSON method has increased considerably, it has six more layers in the network, but the results obtained are approximately 95% better for all data sets compared to the ones obtained with COMBI.

	PSE	PSE	PSE	PSE	RMSE
	whole set	Training set	validation set	New data set	whole set
COMBI	0.006652	0.006535	0.007448	0.006471	0.001377
GBSON	0.000244	0.000255	0.000231	0.000207	0.000050

Table 4.2 Comparison of the results for the Lorenz attractor system.

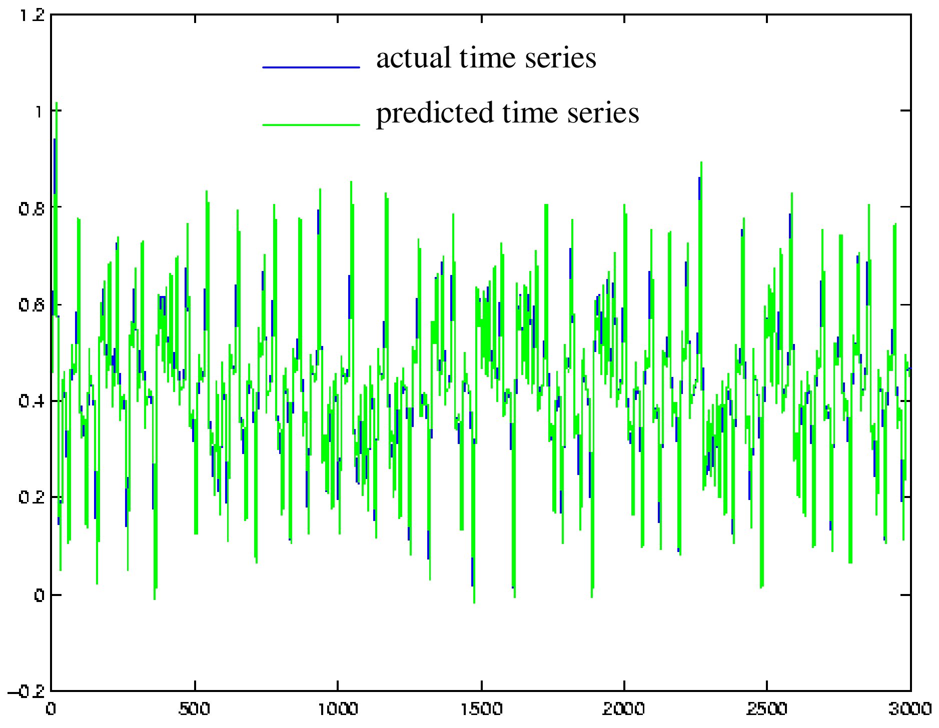


Figure 4.23 The actual and predicted with COMBI Lorentz attractor system.

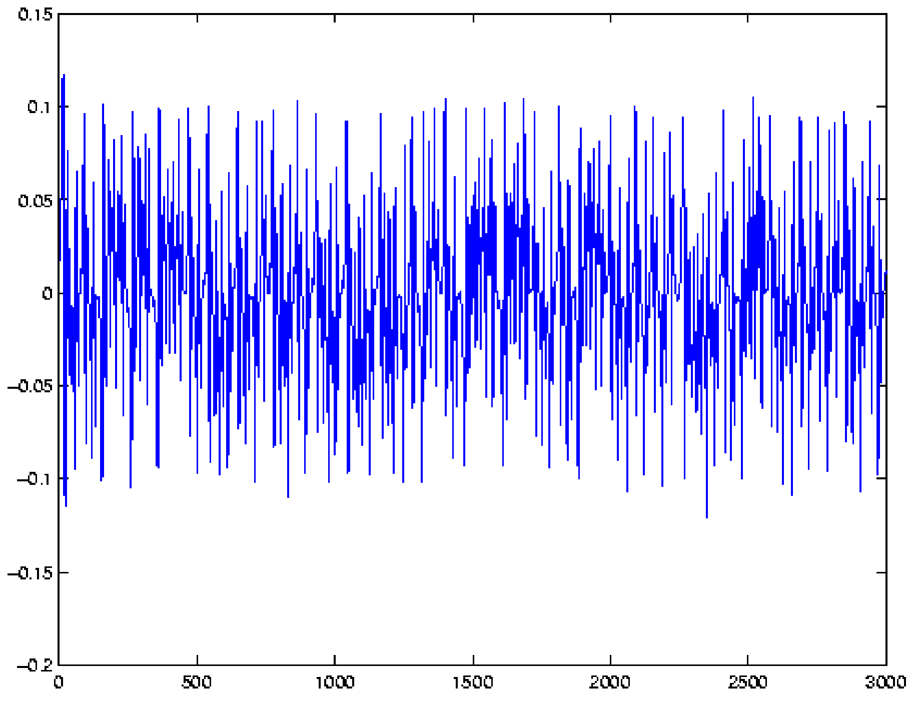


Figure 4.24 The actual error for each point of the Lorentz attractor predicted with COMBI.

4.4.3 Exchange Rates

The prediction of the exchange rates of the British pound, the Canadian dollar, the Deutsche mark, the Japanese yen and the Swiss franc against the U.S. dollar, was based on three past values. Thus, the output pattern is

$$x(t) = f((x(t-1), x(t-2), x(t-3)).$$

The COMBI algorithm converged to networks with only one layer, to model the exchange rates of all the currencies against the U.S. dollar. The models for the exchange rates of each currency are given below.

British pound against U.S. dollar:

$$y = -0.0076 + 0.1417x_2 + 0.8790x_3 - 1.7255x_2x_3 - 1.5967x_2^2 + 3.3084x_3^2$$

Canadian dollar against U.S. dollar:

$$y = 0.0199 + 0.5052x_1 + 0.4500x_3 - 4.0526x_1x_3 - 3.4242x_1^2 + 7.5017x_3^2$$

Deutsche mark against U.S. dollar:

$$y = -0.0088 + 0.1908x_2 + 0.8330x_3 - 1.4806x_2x_3 - 1.3031x_2^2 + 2.7683x_3^2$$

Japanese Yen against U.S. dollar:

$$y = -0.0082 + 0.2104x_2 + 0.8193x_3 - 2.1607x_2x_3 - 1.9295x_2^2 + 4.0644x_3^2$$

Swiss franc against U.S. dollar:

$$y = -0.0083 + 0.2532x_2 + 0.7735x_3 - 1.6010x_2x_3 - 1.3356x_2^2 + 2.9169x_3^2$$

The results of the prediction of the exchange rates along with the actual values, and the actual error of the prediction, are shown in Figures 4.25 to 4.34, for all the currencies. The comparison of these results with the results obtained with the GBSON method is summarised in Tables 4.3 to 4.7.

	PSE Whole set	PSE training set	PSE validation set	PSE new data set	RMSE whole set
COMBI	0.000035	0.000028	0.000057	0.000060	0.000020
GBSON	0.000035	0.000028	0.000059	0.000060	0.000020

Table 4.3 Comparison of the results for the exchange rates for the British pound against the U.S. dollar.

	PSE Whole set	PSE training set	PSE Validation set	PSE new data set	RMSE whole set
COMBI	0.000005	0.000004	0.000007	0.000006	0.000003
GBSON	0.000005	0.000004	0.000007	0.000009	0.000004

Table 4.4 Comparison of the results for the exchange rates for the Canadian dollar against the U.S. dollar.

	PSE Whole set	PSE training set	PSE validation set	PSE new data set	RMSE whole set
COMBI	0.000045	0.000043	0.000051	0.000056	0.000025
GBSON	0.000052	0.000046	0.000053	0.000089	0.000029

Table 4.5 Comparison of the results for the exchange rates for the Deutsche mark against the U.S. dollar.

	PSE Whole set	PSE training set	PSE validation set	PSE new data set	RMSE whole set
COMBI	0.000041	0.000041	0.000055	0.000025	0.000013
GBSON	0.000043	0.000043	0.000057	0.000028	0.000014

Table 4.6 Comparison of the results for the exchange rates for the Japanese yen against the U.S. dollar.

	PSE	PSE	PSE	PSE	RMSE
	Whole set	training set	validation set	new data set	Whole set
COMBI	0.000069	0.000071	0.000070	0.000056	0.000032
GBSON	0.000102	0.000110	0.000094	0.000070	0.000048

Table 4.7 Comparison of the results for the exchange rates for the Swiss franc against the U.S. dollar.

The results obtained, with both methods, for the British pound exchange rates are approximately the same. Both methods could not generalise well, and there is a considerable increase in the PSE over the new data set. The results obtained for the Canadian dollar exchange rates, are also approximately the same, but the COMBI algorithm generalised better than the GBSON algorithm. The results obtained for the Japanese yen exchange rates with COMBI are slightly better than the ones obtained with GBSON, but both methods could not generalise well. The results obtained for the Deutsche mark and Swiss franc exchange rates with COMBI were considerably better than the ones with GBSON. In addition, the model obtained by COMBI for the Swiss franc exchange rates generalised well.

It should also be noted that the models obtained by the COMBI algorithm used the past variables x_2 and x_3, except for the model for the Canadian dollar exchange rates. The GBSON algorithm resulted to models that use only the past variable x_2, except for the model for the Deutsche mark exchange rates.

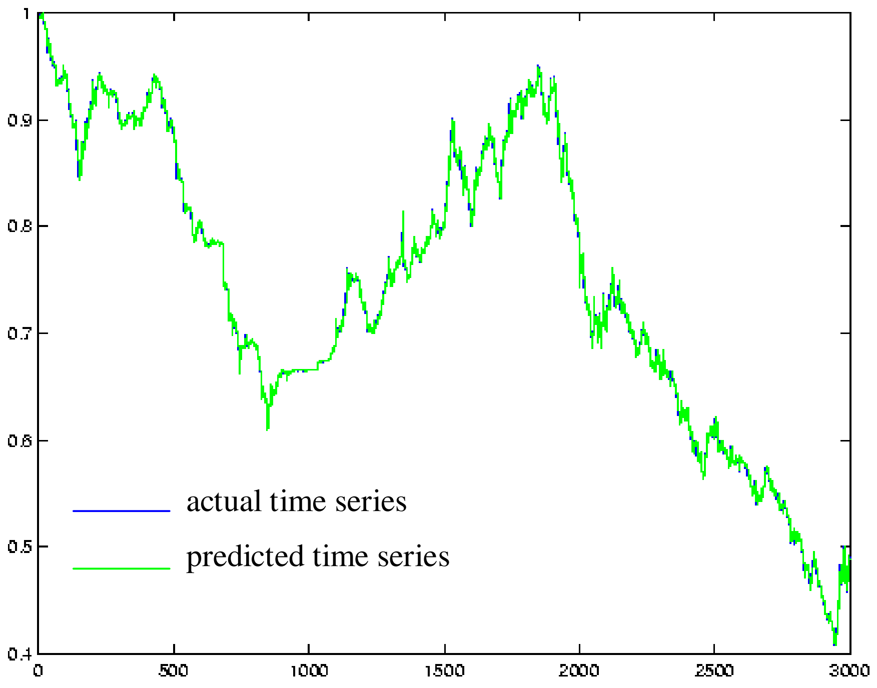


Figure 4.25 The actual and predicted with COMBI exchange rates for the British pound against the U.S. dollar.

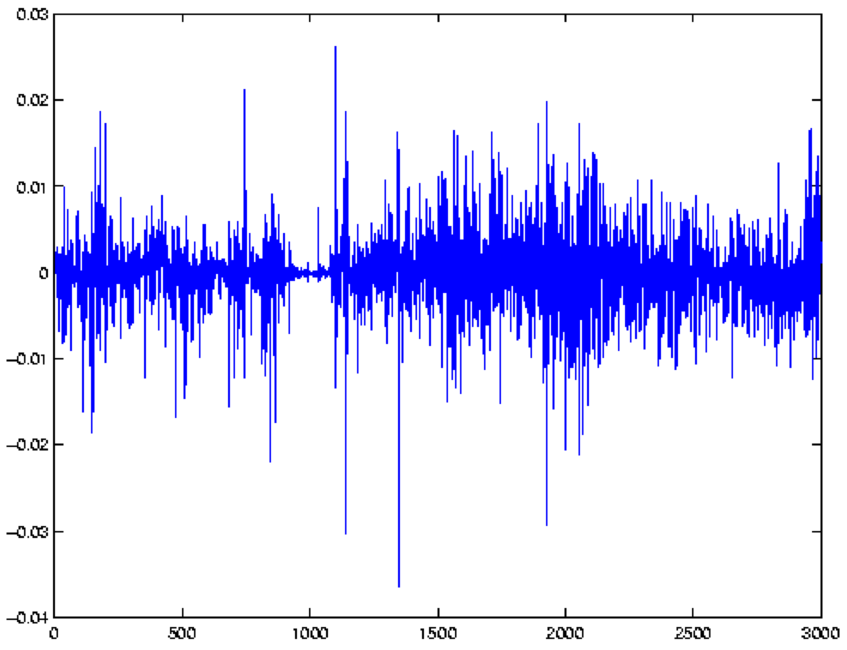


Figure 4.26 The actual error for each point of the exchange rates of the British pound against the U.S dollar predicted with the COMBI.

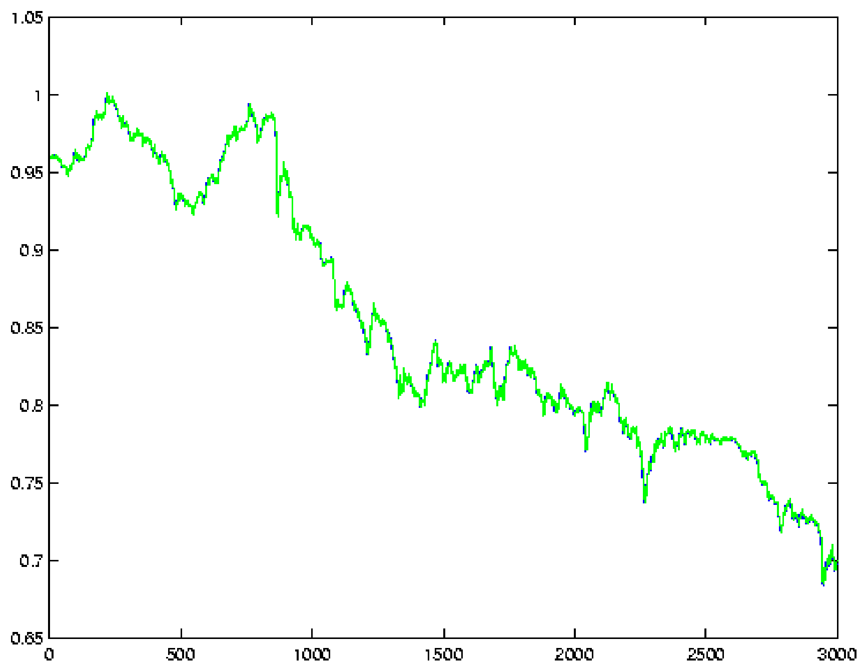


Figure 4.27 The actual and predicted with COMBI exchange rates for the Canadian dollar against the U.S. dollar.

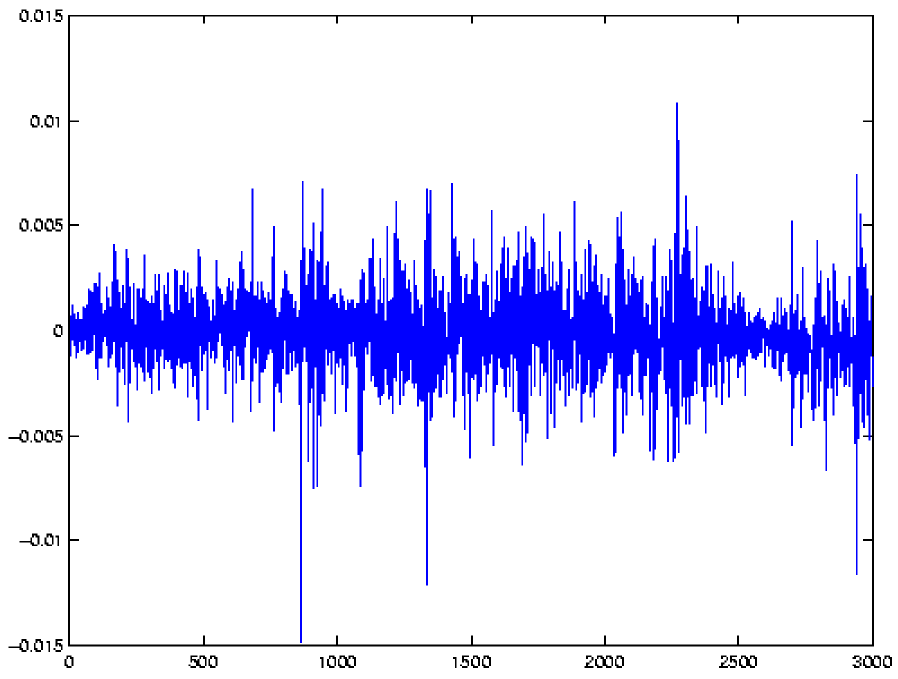


Figure 4.28 The actual error for each point of the exchange rates of the Canadian dollar against the U.S. dollar predicted with the COMBI.

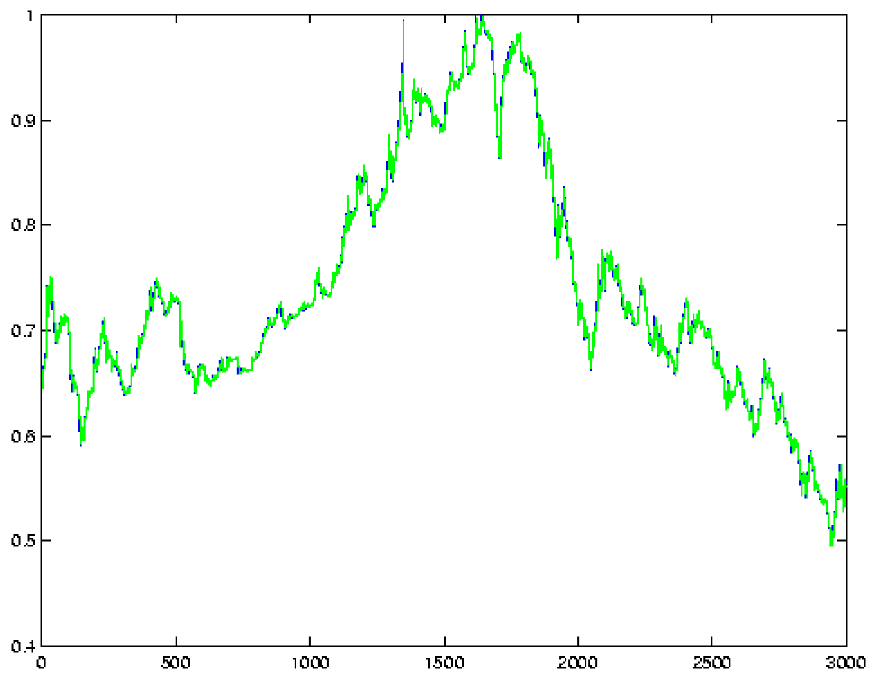


Figure 4.29 The actual and predicted with COMBI exchange rates for the Deutsche mark against the U.S. dollar.

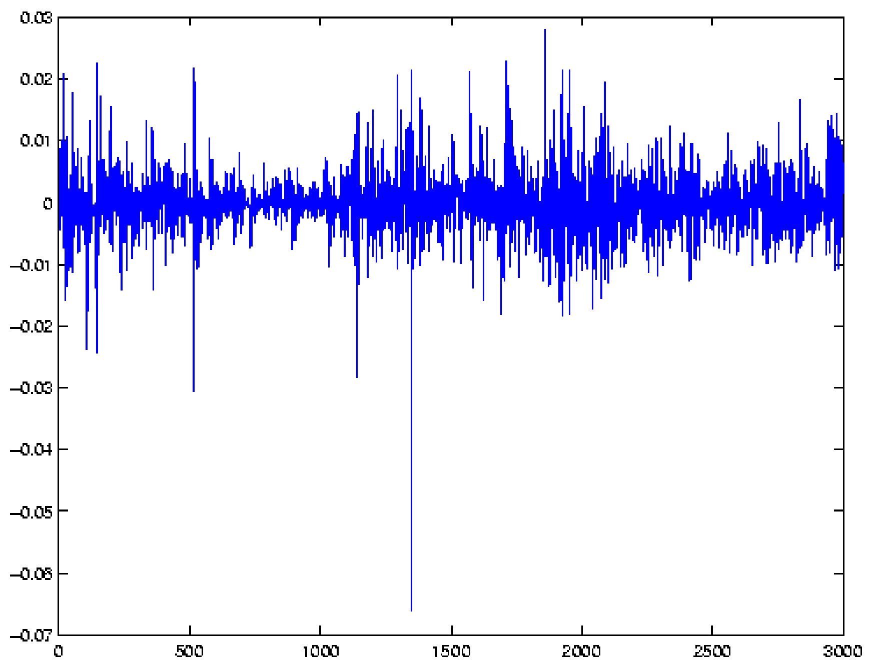


Figure 4.30 The actual error for each point of the exchange rates of the Deutsche mark against the U.S. dollar predicted with the COMBI.

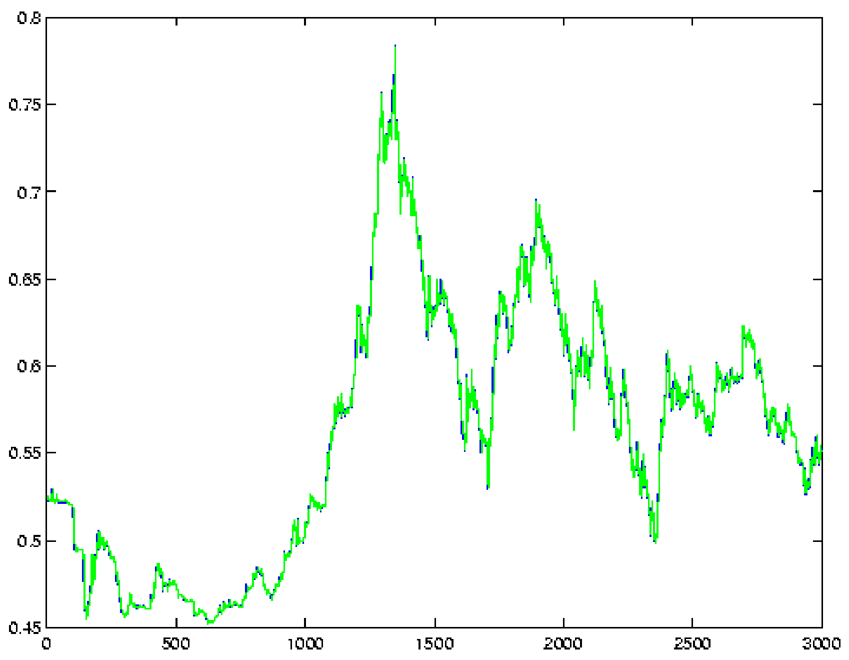


Figure 4.31 The actual and predicted with COMBI exchange rates for the Japanese yen against the U.S. dollar.

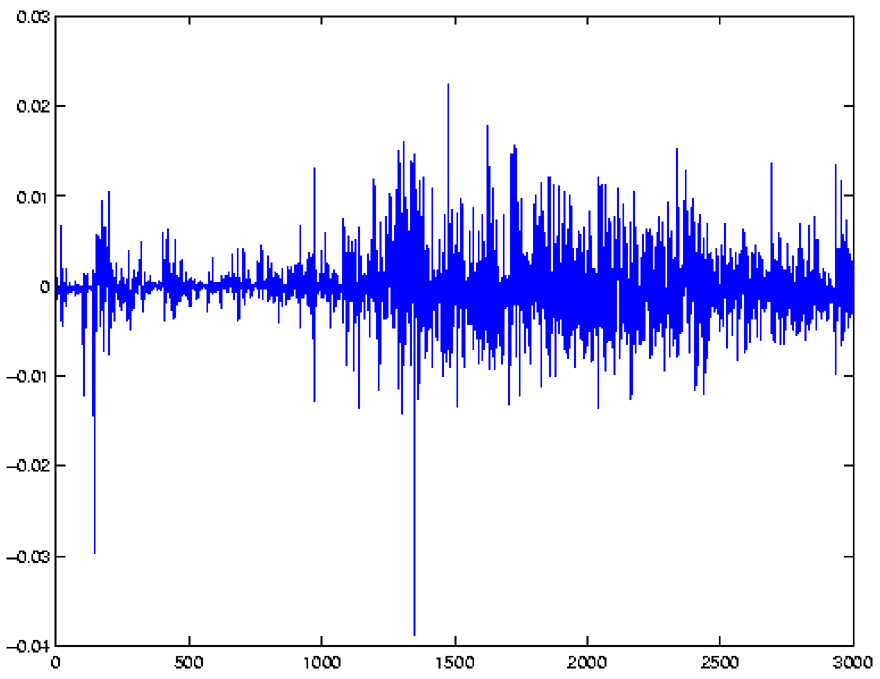


Figure 4.32 The actual error for each point of the exchange rates of the Japanese yen against the U.S. dollar predicted with the COMBI.

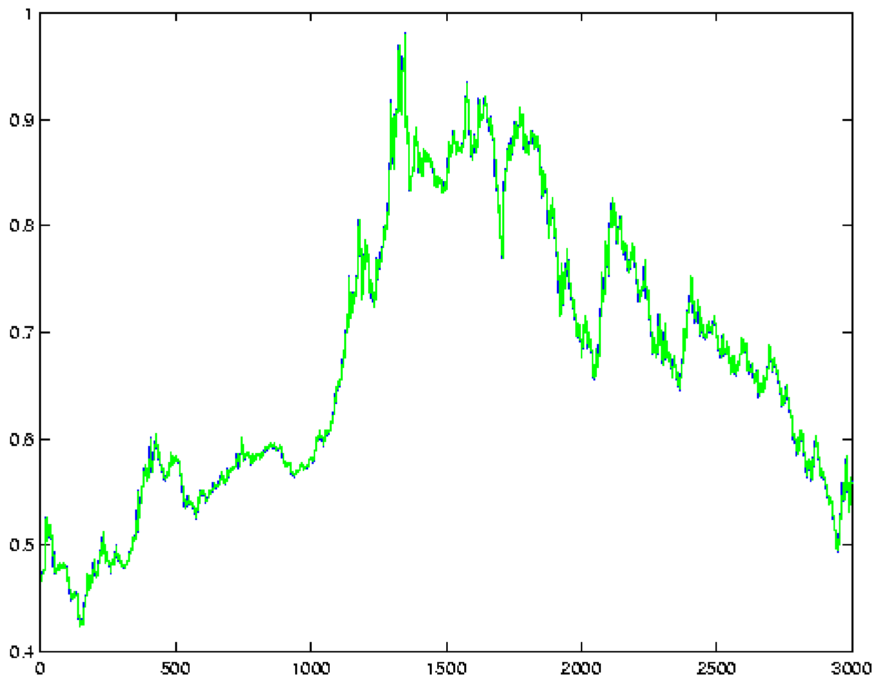


Figure 4.33 The actual and predicted with COMBI exchange rates for the Swiss franc against the U.S. dollar.

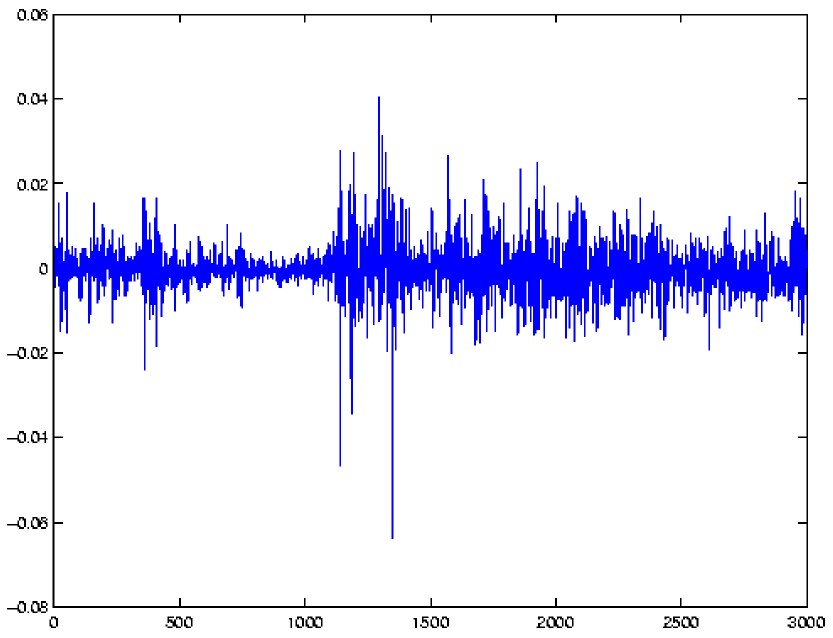


Figure 4.34 The actual error for each point of the exchange rates of the Swiss franc against the U.S. dollar predicted with the COMBI.

4.5 Summary

This chapter presented the method used, based on the GBSON method, to solve the time series prediction problem. Following, the results obtained with this method for the sunspot time series, the Lorentz attractor and the exchange rates for the British pound, the Canadian dollar, the Deutsche mark, the Japanese yen and the Swiss franc against the U.S. dollar, were presented. Finally, the results with the GBSON method were compared with the results obtained using the GMDH algorithm COMBI, for the same problems.

CHAPTER 5
CONCLUSIONS AND FURTHER WORK

5.1 Conclusions

This dissertation presented an overview of the work carried out for the project titled "*Time Series Prediction using Evolving Polynomial Neural Networks*". The aim of this dissertation project was to determine the structure and weights of a polynomial neural network using evolutionary computing methods and apply it to the time series prediction problem.

First, the various evolutionary computing methods were studied. These methods were classified into three categories: *Genetic Algorithms*, *Niched Genetic Algorithms* and *Evolutionary Algorithms*. The currently used evolutionary computing methods for neural networks optimisation were also investigated. Following, the procedure for time series prediction was outlined. The steps identified in this procedure were: collection of data, formation of candidate models set, selection of criterion of model fitness, model identification and finally model validation. The most commonly used linear, non-linear and neural network models were presented. Some methods for measuring the prediction accuracy were also identified. Then, the Group Method of Data Handling (GMDH) algorithms were analysed. Finally, a hybrid method of GMDH and GA, the Genetics-Based Self-Organising Network (GBSON) method for time series prediction was presented.

The method implemented for this dissertation project was based on the GBSON method. The main problems identified in this method by Kargupta and Smith [42] were credit assignment and co-operation versus competition. In the method implemented, it was attempted to overcome these problems. The main differences of the implemented method and the GBSON method introduced in [42] are:

- Determination of the niche radius;
- Fitness measure of potential solutions;
- Incorporating the mechanism of elitism;
- Selection mechanism;
- No use of mating restriction schemes;
- Crossover operator.

The implemented method was tested on the sunspot time series, the Lorentz attractor system and the exchange rates for various currencies. The results of this method, presented in section 4.3, were compared with the results obtained with the GMDH algorithm COMBI. In general, the COMBI algorithm produced simple models for the above mentioned problems. Although the results obtained with COMBI were good, its performance showed that it has not the ability to combine the partial descriptions of the model, obtained in the first layers of the network, to further reduce the prediction error. On the contrary, the GBSON algorithm showed that it has the ability to efficiently combine simple partial descriptions to produce complex models. This ability of the GBSON algorithm was best seen with the model obtained for the Lorentz attractor system. Another feature of the GBSON algorithm is that it exhibited good generalisation for the problems of the sunspot time series and the Lorentz attractor system. Although it used only 500 data points for validation purposes, it generalised better than the COMBI algorithm that used 2000 points for training and 500 points for validation. In conclusion, it can be said that the GBSON algorithm gave very encouraging results and has the potential for further improvements.

5.2 Further Work

The results obtained with the implemented method are very encouraging. In all problems tested there was a decrease in the prediction error compared to the GMDH algorithm COMBI. Although the results are encouraging, the problems of credit assignment and co-operation versus competition still remain.

One of the most important problems in the GBSON method, is the determination of the niche radius. Although, in the problems tested in this dissertation project, the way of determining the niche radius gave good results, it is likely that in other problems it wont work. It is proposed that a co-evolutionary method, where the niche radius is evolved along with the potential solutions is used. A scheme like that was implemented by Goldberg [62] and is described in section 2.3.1.3. It is believed that a scheme like that will solve the problem of determining the niche radius.

Another problem is the selection of the reproduction operator. The tournament selection scheme in this implementation gave satisfactory results, but it has been criticised in the literature when used with shared fitness schemes [58], [64]. Alternative selection schemes can be tested with the shared fitness scheme. The Restricted Competition Selection (RCS) [43] scheme described in section 2.3.3, produced stable sub-populations and gave good preliminary results.

An alternative approach would be to use a niched genetic algorithm that does not incorporate the fitness sharing scheme and thus the determination of a niche radius. Approaches like that are the crowding methods described in section 2.3.2 and the clearing method described in section 2.3.4.

REFERENCES

[1] Anderson O. D., Time series analysis and forecasting, Butterworths, 1976.

[2] Angeline P. J., Saunders G. M., Pollack J. B., An evolutionary algorithm that constructs recurrent neural networks, IEEE Transactions on Neural Networks, Vol. 5, No. 1, pp. 54-65,1994.

[3] Back T., Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms, Oxford University Press, 1996.

[4] Baker J. E., Adaptive selection methods for genetic algorithms, Proceedings of the First International Conference on Genetic Algorithms, pp. 101-111, 1985.

[5] Baluja S., Evolution of an artificial neural network based autonomous land vehicle controller, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 26, No.3, pp. 450-463,1996.

[6] Belogurov V. P., A criterion of model suitability for forecasting quantitative processes, Soviet Journal of Automation and Information Sciences, Vol. 23, No. 3, pp. 21-25, 1990.

[7] Bose N. K., Liang P., Neural network fundamentals with graphs, algorithms and applications, McGraw-Hill, 1996.

[8] Box G. E. P., Jenkins G. M., Time series analysis: forecasting and control, Holden-Day, 1976.

[9] Brockwell P. J., Davis R. A., Time series: theory and methods, Springer, 2nd Edition, 1991.

[10] Carse B., Fogarty T. C., Fast evolutionary learning of minimal radial basis function neural networks using a genetic algorithm, Lecture Notes in Computer Science, Vol. 1143, pp. 1-22, 1996.

[11] Carse B., Fogarty T. C., Tackling the "curse of dimensionality" of radial basis functional neural networks using a genetic algorithm, Lecture Note in Computer Science, Vol. 1141, pp. 710-719, 1996.

[12] Chen S., Smith S. F., Putting the "genetics" back into genetic algorithms, Foundations of Genetic Algorithms 5, Morgan Kaufmann, 1999.

[13] Cheng A. C. C., Guan L., A combined evolution method for associative memory networks, Neural Networks, Vol. 11, No. 5, pp. 785-792, 1998.

[14] Cichocki A., Unbehauen R., Neural networks for optimization and signal processing, Wiley, 1993

[15] Coli M., Gennuso G., Palarazzi P., A new crossover operator for genetic algorithms, Proceedings of 1996 IEEE International Conference on Evolutionary Computation (ICEC ' 96), pp. 201-206, 1996.

[16] De Falco I., Iazzetta A., Natale P., Tarantino E., Evolutionary neural networks for non-linear dynamics modeling, Lecture Notes in Computer Science, Vol. 1498, pp. 593-602, 1998.

[17] Deb, K., Genetic algorithms for function optimization, Genetic Algorithms and Soft Computing, pp. 3-29, 1996.

[18] Eshelman L. J., Caruana R. A., Schaffer J.D., Biases in the crossover landscape, Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, pp.10-19, 1989.

[19] Farlow S. J., The GMDH algorithm, Self-Organizing Methods in Modeling, pp. 1-24, 1984.

[20] Farnum N. R., Stanton L. W., Quantitative forecasting methods, PWS-KENT, 1989.

[21] Figueira-Pujol J. C., Poli R., Evolving neural networks using a dual representation with a combined crossover operator, Proceedings of the IEEE Conference on Evolutionary Computation, pp. 416-421, 1998.

[22] Figueira-Pujol J. C., Poli R., Evolving the topology and the weights of neural networks using a dual representation, Applied Intelligence, Vol. 8, No. 1, pp. 73-84, 1998.

[23] Fogel D. B., An introduction to simulated evolutionary optimization, IEEE Transactions on Neural Networks, Vol. 5, No. 1, pp. 3-14, 1994.

[24] Fogel D. B., Fogel L. J., Porto V. W., Evolving Neural Networks, Biological Cybernetics, Vol. 63, pp. 487-493, 1990.

[25] Fukumi M., Akamatsu N., Rule extraction from neural networks trained using evolutionary algorithms with deterministic mutation, Proceedings of the IEEE Conference on Evolutionary Computation, pp. 686-689, 1998.

[26] Fulcher G. E., Brown D. E., A polynomial network for predicting temperature distributions, IEEE Transactions on Neural Networks, Vol. 5, No. 3, pp. 372-379, 1994.

[27] Goldberg D. E., Deb K., A comparative analysis of selection schemes used in genetic algorithms, Foundations of Genetic Algorithms, Morgan Kaufmann, pp. 69-93, 1991.

[28] Goldberg D. E., Genetic algorithms in search, optimization, and machine learning, Addison-Wesley, 1989.

[29] Goldberg D.E., An analysis of Boltzmann tournament selection, IlliGAL Report No. 94007, Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory*,* 1994.

[30] Gomez-Ramirez E., Poznyak A. S., Structure adaptation of polynomial stochastic neural nets using learning automata technique, IEEE International Conference on Neural Networks-Conference Proceedings, Vol. 1, pp. 390-395,1998.

[31] Harvey A. C., Time series models, Philip Allan, 1981.

[32] Holland J. H., Adaptation in natural and artificial systems, University of Michigan Press, 1975.

[33] Horn J., The nature of niching: genetic algorithms and the evolution of optimal, cooperative populations, IlliGAL Report No. 97008, Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory*,* 1997.

[34] Ivakhnenko A. G, An inductive sorting method for the forecasting of multidimensional random processes and events with the help of analogues forecast complexing, Pattern Recognition and Image Analysis, Vol. 4, No. 2, pp. 177-188, 1994.

[35] Ivakhnenko A. G., Muller J. A., Parametric and non-parametric selection procedures in experimental system analysis, System Analysis Modeling and Simulation (SAMS), Vol. 9, pp. 157-175, 1992.

[36] Ivakhnenko A. G., Muller J. A., Self-organisation of nets of active neurons, System Analysis Modeling and Simulation (SAMS), Vol. 20, No. 1-2, pp.29-50, 1995.

[37] Ivakhnenko A. G., Osipenko V. V., Algorithms of transformation of probability characteristics into deterministic forecast, Soviet Journal of Automation and Information Sciences, Vol. 15, No. 2, pp. 7-15, 1982.

[38] Ivakhnenko A. G., Petukhova S.A., Objective choice of optimal clustering of data sampling under non-robust random disturbances compensation, Soviet Journal of Automation and Information Sciences, Vol. 26, No. 4, pp. 58-65, 1993.

[39] Ivakhnenko G. A., Objective system analysis algorithm and its perfection by parallel computing, Glushkov Institute of Cybernetics, Kiev, Ukraina, 1995.

[40] Jelasity M., Dombi J., GAS, a concept on modeling species in genetic algortihms, Artificial Intelligence, Vol. 99, No. 1, pp. 1-19, 1998.

[41] Kantz H., Schreiber T., Nonlinear time series analysis, Cambridge University Press, 1997.

[42] Kargupta H., Smith R. E., System identification with evolving polynomial networks, Proceedings of the 4th International Conference on Genetic Algorithms, pp. 370-376, 1991.

[43] Lee C.-G., Cho D.-H., Jung H.-K., Niching genetic algorithm with restricted competition selection for multimodal function optimization, IEEE Transactions on Magnetics, Vol. 35, No. 3, 1999.

[44] Leondes C. T., Optimization techniques, Academic Press, 1998.

[45] Liu Y., Yao X., Towards designing neural network ensembles by evolution, Lecture Notes in Computer Science, Vol. 1498, pp. 623-632,1998.

[46] Ljung L., System identification: theory for the user, Prentice Hall, 2nd Edition, 1999.

[47] Mahfoud S. W., Niching methods for genetic algorithms, IlliGAL Report No. 95001, Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory*,* 1995.

[48] Man K. F., Tang K. S., Kwong S., Halang W. A., Genetic algorithms for control and signal processing, Springer-Verlag, 1997.

[49] Maniezzo V., Genetic evolution of the topology and weight distribution of neural networks, IEEE Transactions on Neural Networks, Vol. 5, No. 1, pp. 39-53, 1994.

[50] McDonell J. R., Waagen D., Evolving recurrent perceptrons for time-series modeling, IEEE Transactions on Neural Networks, Vol. 5, No. 1, pp. 24-37,1994.

[51] Mengschoel O. J., Goldberg D. E., Probabilistic crowding: deterministic crowding with probabilistic replacement, IlliGAL Report No. 99004, Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, 1999.

[52] Michalewicz Z., A perspective on evolutionary computation, Lecture Notes in Artificial Intelligence, Vol. 956, pp. 73-89, 1995.

[53] Michalewicz Z., Genetic algorithms + data structures = evolution programs, 3rd edition, Springer-Verlag, 1996.

[54] Miller B. L., Shaw M. J., Genetic algorithms with dynamic niche sharing for multimodal function optimization, IlliGAL Report No. 95010, Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, 1995.

[55] Mitchell M., An introduction to genetic algorithms, MIT Press, 1999.

[56] Moriarty D. E., Miikkulainen R., Hierarchical evolution of neural networks, IEEE International Conference on Neural Networks-Conference Proceedings, Vol. 1, pp. 428-433, 1998.

[57] Norton J. P., An introduction to identification, Academic Press, 1986.

[58] Oei C. K., Goldberg D. E., Chang S. J., Tournament selection, niching, and the preservation of diversity, IlliGAL Report No. 91011, Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, 1991.

[59] Petrowski A., A clearing procedure as a niching method for a genetic algorithms, Proceeding of 1996 IEEE International Conference on Evolutionary Computation (ICEC ' 96), pp. 798-803, 1996.

[60] Pictet O. V., Dacorogna M. M., Dave R. D., Chopard B., Schirru R., Tomassini M., Genetic algorithms with collective sharing for robust optimization in financial applications, Neural Network World, Vol. 5, No. 4, pp. 573-587, 1995.

[61] Priestley M. B., Non-linear and non-stationary time series analysis, Academic Press, 1988.

[62] Quagliarella D., Periaux J., Poloni C., Winter G., Generic algorithms and evolution strategies in engineering and computer science, Wiley, 1998.

[63] Richards N., Moriarty D. E., Miikkulainen R., Evolving neural networks to play go, Applied Intelligence, Vol. 8, No. 1, pp. 85-96, 1998.

[64] Sareni B., Krahenbuhl L., Fitness sharing and niching methods revisited, IEEE Transactions on Evolutionary Computation, Vol. 2, No. 3, pp. 97-106, 1998.

[65] Schwefel H. P., Numerical optimization of computer models, Wiley, 1981.

[66] Shin Y., Ghosh J., Ridge polynomial networks, IEEE Transactions on Neural Networks, Vol. 6, No. 3, pp. 610-622,1995.

[67] Solis F. J., Wets R. J. B., Minimization by random search techniques, Mathematics of Operations Research, Vol. 6, No. 1, pp. 19-30, 1981.

[68] Stepashko V. S., Asymptotic properties of external criteria for model selection, Soviet Journal of Automation and Information Sciences, Vol. 21, No. 6, pp. 24-32, 1988.

[69] Syswerda G., A study of reproduction in generational and steady state genetic algorithms, Foundations of Genetic Algorithms, Morgan Kaufmann, pp.94-101, 1991.

[70] Tamaki H., Kita H., Kobayashi S., Multi-objective optimisation by genetic algorithms: a review, Proceedings of 1996 IEEE International Conference on Evolutionary Computation (SICE ' 96), pp. 517-522, 1996.

[71] Taylor M., Lisboa P., Techniques and applications of neural networks, Ellis Horwood, 1993.

[72] Tenorio M. F., Lee W.-T., Self-organizing network for optimum supervised learning, IEEE Transactions on Neural Networks, Vol. 1, No. 1, 1990.

[73] Teodorescu D., Time series - information and prediction, Biological Cybernetics, Vol. 63, No. 6, pp. 477-485, 1990.

[74] Vandaele W., Applied time series and Box-Jenkins models, Academic Press, 1983.

[75] Yao X., Liu Y., A new evolutionary system for evolving artificial neural networks, IEEE Transactions on Neural Networks, Vol. 8, No. 3, pp. 694-713,1997.

[76] Yao X., Liu Y., Making use of population information in evolutionary artificial neural networks, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 28, No. 3, pp. 417-425, 1998.

[77] Zhang B.-T., Ohm P., Muhlenbein, Evolutionary neural trees for modelling and predicting complex systems, Engineering Applications of Artificial Intelligence, Vol. 10, No. 5, pp. 473-483, 1997.

[78] Zhao Q., A general framework for cooperative co-evolutionary algorithms: a society model, IEEE International Conference on Neural Networks-Conference Proceedings, Vol.1, pp. 57-62,1998.

[79] http://www.inf.kiev.ua/GMDH-home/GMDH_res.htm