Title Time Series Prediction Using Supervised Learning and Tools from Chaos Theory

Name Andrew Nicola Edmonds

TIME SERIES PREDICTION USING SUPERVISED LEARNING AND TOOLS FROM CHAOS THEORY

ANDREW NICOLA EDMONDS

A thesis submitted to the Faculty of Science and Computing, University of Luton, in partial fulfilment of the requirements for the degree of Doctor of Philosophy

December, 1996

i. Abstract

In this work methods for performing time series prediction on complex real world time series are examined. In particular series exhibiting non-linear or chaotic behaviour are selected for analysis. A range of methodologies based on Takens' embedding theorem are considered and compared with more conventional methods. A novel combination of methods for determining the optimal embedding parameters are employed and tried out with multivariate financial time series data and with a complex series derived from an experiment in biotechnology. The results show that this combination of techniques provide accurate results while improving dramatically the time required to produce predictions and analyses, and eliminating a range of parameters that had hitherto been fixed empirically. The architecture and methodology of the prediction software developed is described along with design decisions and their justification. Sensitivity analyses are employed to justify the use of this combination of methods, and comparisons are made with more conventional predictive techniques and trivial predictors showing the superiority of the results generated by the work detailed in this thesis.

ii. Dedication

This work is dedicated to my family: Anneke, Alexander, Anna and our unborn child, my parents George and Patricia, and *Rex Quondam Rexque Futurus*.

iii. List of contents:

iv. Table of Figures

v. Preface

Elements of this work have been published in the proceedings of conferences and journals.

An overview of the financial time series results and methodology was published in the proceedings of the IEEE world conference in Computational Intelligence in Orlando, Florida (Edmonds 94a).

Another paper on the same subject with a more financial emphasis was published in the proceedings of "Neural networks in the capital markets" or NNCM94 (Edmonds 94b)

The biological time series work was published in Biosystems in 1996. (Davey et al. 96). Some of the diagrams are drawn from this paper.

The author performed all the mathematical analyses of the data in this paper except for the Fast Fourier transform and Auto Regressive Moving Average analyses.

vi. Acknowledgements

I would like to acknowledge the contribution of the following and thank them for their help:

Science in Finance Ltd. for funding this work and my college fees.

Diana Burkhardt and Osei Adjei of the University of Luton and Professor Reginald King of Cranfield University for their supervision, support and advice.

Reuters Limited for the provision of some of the data used.

Professor Douglas Kell of the University of Wales department of Biotechnology for the provision of some of the data used and the opportunity to co-operate in some interesting work.

John Deuss of Transworld Oil Research and Development Limited for his encouragement.

vii. Author's declaration

I declare that this thesis is my own unaided work. It is being submitted for the degree of Doctor of Philosophy at the University of Luton. It has not been submitted before for any degree or examination in any other University.

Andrew Nicola Edmonds

15th day of December 1996.

1. Introduction

This thesis describes work undertaken to investigate and develop methodologies for time series prediction of complex real world time series using principles from Chaos Theory and Computational Intelligence.

When this work was started various results had been announced using computational intelligence techniques to perform prediction of financial and biological series that had hitherto been regarded as intractable, and much interest had been generated in these techniques as a result. On closer inspection the methodologies used required that a range of parameters be found by manual or automated search since no analytical techniques were available. The results were obtained with considerable expenditure of computer time and human effort, and the empirical discovery of good values for parameters tended to undermine the validity of the results. The choices of parameter values so obtained could not be justified by any theory or heuristic, nor could the researcher claim without inordinate care that his results were truly the product of blind trials.

In this work various published techniques that had been used to solve other problems are bought together, modified and adapted and combined with new ideas to form a time series prediction methodology that eliminates the requirement to search for parameters and that requires

dramatically reduced computer resources. In the first version of the time series predictor there were seven parameters which required setting by experiment, In the final version the were none. The run time of the final version was 2 orders of magnitude faster than the first taking into account the improvements in hardware that had taken place over the period (see chapter 3). The accuracy of the results when tested on benchmark time series were at least as accurate as other published work (chapter 4). It is this successful adaptation and combination of techniques that is novel and represents the contribution to knowledge contained in this work.

The software was tested using financial time series data, specifically foreign exchange rates, because of the complexity of the data, its availability in vast quantities, and the interest there is in the predictions generated.

A further application arose during the course of this work to test the software on a problem in biotechnology, where chaos, and short term predictability had never before been identified, and only recently suspected. As will be seen the presence of both these characteristics were demonstrated using the techniques described in this work.

In Chapter 2 the types of time series and their characteristics that the techniques enshrined in this work attempt to predict are discussed along with a review of predictive and analytical techniques that have been used

on them. The function and characteristics of various modelling techniques drawn from Computational Intelligence are discussed. Finally simple predictive methods are described that are very useful in weeding out poor predictive results. Because of the many different fields from which this work is drawn the author has taken the decision to provide a more basic description of the technologies and ideas involved than would normally be found in a chapter which is intended to be a literature survey.

Chapter 3 is a description of the evolution of this work. It describes the construction of a time series predictor using Neural networks that represented the current state of knowledge in the field, and the problems associated with using it to predict financial time series. The large number of control parameters required are described, along with the paucity of techniques or heuristics to determine their proper value.
Various interim versions are described that first automate the search for parameters, then in successive stages reduce the requirements until in the final stage none of the original parameters survive.

Chapter 4 describes the final version of the time series predictor in greater detail. It explains the various new ideas and adaptations and improvements to existing algorithms that went into the time series predictor. The problems associated with handling financial time series data are described along with the solutions found. Methods for evaluating

the performance of the predictor are described along with the performance of the predictor on various artificial benchmark series.

Chapter 5 describes the performance of the predictor with financial time series. Sensitivity analyses are used to show that the system described finds good values for the various parameters that are required. The possible meaning and relevance of these predictions is discussed.

Chapter 6 describes the analysis of a time series drawn from a biotechnology experiment, and how both chaos and short term predictability were identified using these techniques, for the first time in this particular field.
The results and predictions generated are described, along with a short discussion of the potential use of this new knowledge.

Chapter 7 reviews the various conclusions reached in the work described in this thesis, and suggests useful areas for future work.

2. Background to work

In this section the key concepts, technologies and algorithms used either in the final version of the predictor or in interim versions will be described and their various merits discussed. Because of the many different fields from which this work is drawn the author has taken the decision to provide a more basic description of the technologies and ideas involved than would normally be found in a chapter which is intended to be a literature survey.

There has been a great deal of interest in recent years in the application of chaos theory to a variety of real world time dependant systems, and the ability that this new branch of mathematics promises to untangle and bring forth order from seeming disorder.

There has also been much attention given to the ability of computational intelligence tools such as Neural Networks, Genetic Programming, and genetic algorithms to model fuzzy or ill defined real world problems. There are many time series, notably in the natural sciences and finance, that had proved difficult to analyse with conventional linear methods, that are now beginning to be modelled using non-linear and non-parametric methods. (Tong, 90)

These "difficult" time series turn up in many, probably all, branches of science; from the analysis of the time variation in the population of rodents

in northern Scandinavia (Turchin, 93) to the analysis of the weather (Lorenz, 63)

Mathematical methods have been developed to characterise and to some extent analyse these time series, and these will be discussed in more detail later. It will be shown that the complexity of these series and the systems that underlie them, due to the presence of non-linearity, and the fact that they are represented by finite observations, justifies the use of empirical and non-parametric methods as described in this thesis.

2.1 Review of fundamental concepts

The ideas underpinning this work are derived from a number of different disciplines. Few areas of study can avoid reference to mathematics, but many of the techniques described were derived not by mathematicians but biologists or physicists trying to gain information from sampled time series that were both short and noisy. By chance, or perhaps by the nature of things, the same kinds of problems have been attacked using empirical methods in the study of Computational Intelligence.

2.1.1 Chaotic series

Many systems in the natural world are now known to exhibit chaos or non-linear behaviour, the complexity of which is so great that they were previously considered random. The unravelling of these systems has been aided by the discovery, mostly in this century, of mathematical

expressions that exhibit similar tendencies. By analysing these expressions techniques have been developed and applied to the real world The most famous and oldest of these is the logistic equation, originally conceived as a model of population growth.

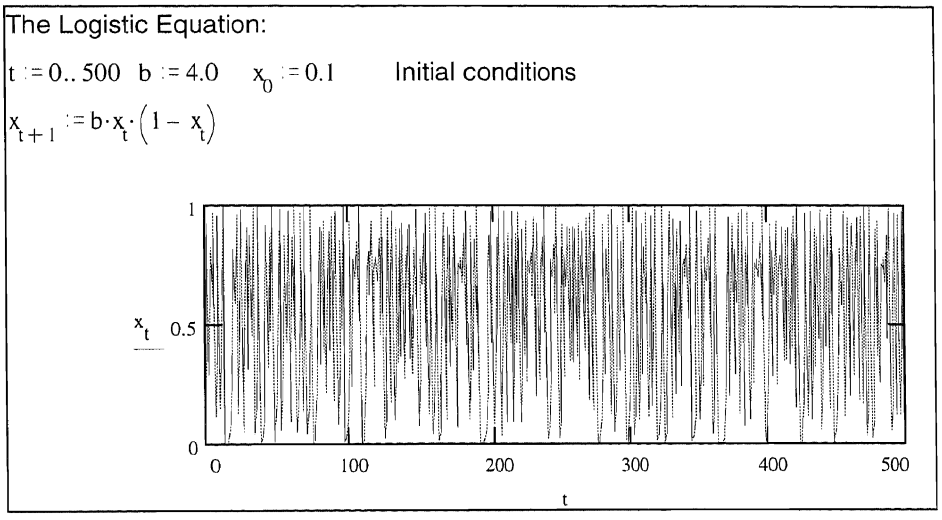


Figure 1, The first 500 points of the logistic Equation

Another simple example is the *Tent Map* :

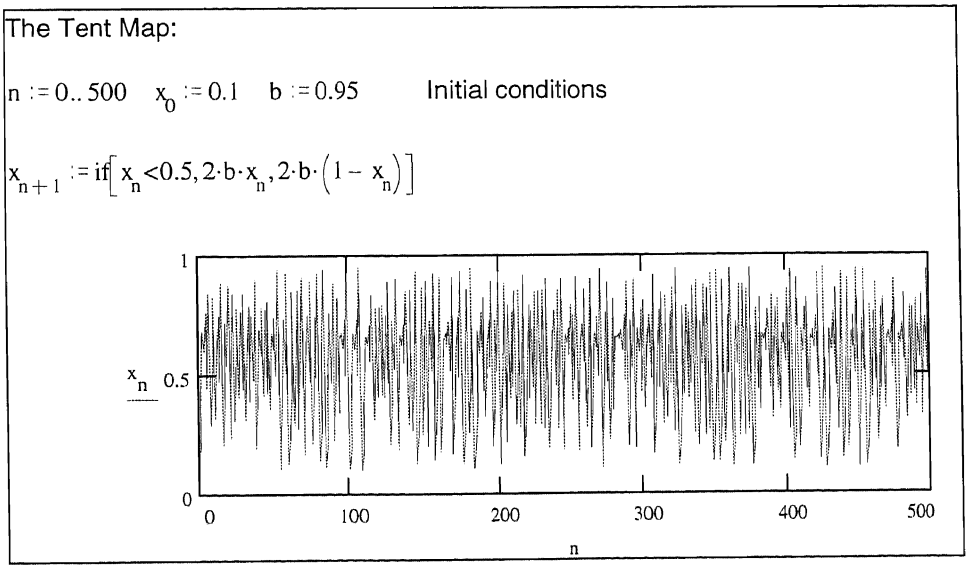


Figure 2, The first 500 points of the Tent Map

Yet another chaotic series is generated by evaluating the Mackey-Glass equation: (Mackey,77)

The Mackey-Glass Equation in its discrete form.

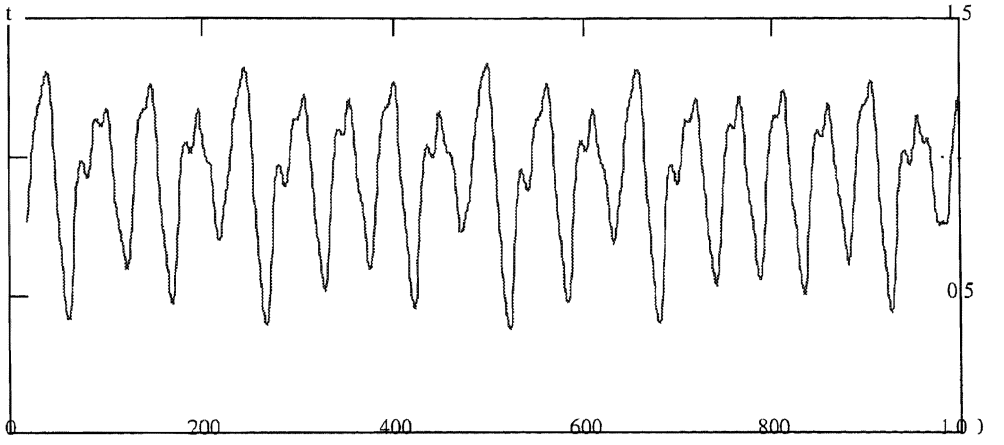
The following choice of variables and initial condition cause chaotic behavior.

a = 0.2, b = 0.9, c = 10, s = 18. X_0 to X_{18} = 0.7

The Equation:

$$x_t = bx_{t-1} + a\frac{x_{t-s}}{1 + x_{t-s}^c}$$

Time Evolution of the equation:



An embedded version of the equation:

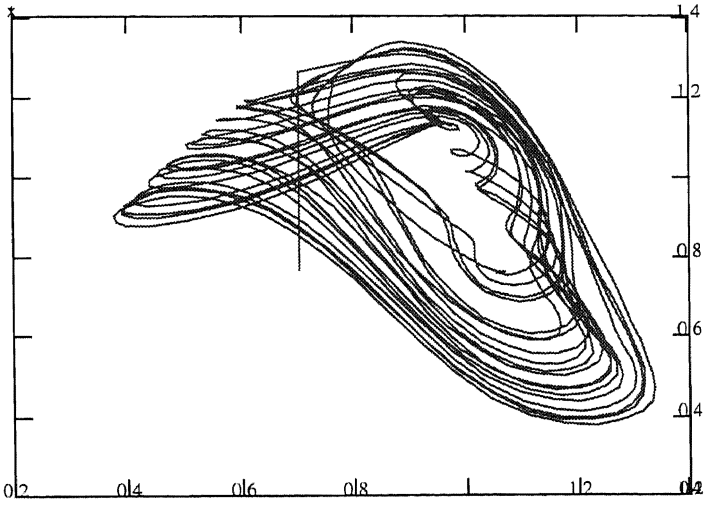


Figure 3 The first thousand points of the Mackey-Glass equation

And as the last example there is the system of equations first used by Lorenz (Lorenz, 63) in his attempts to model simple atmospheric interactions:

$$x_{t+1} = x_t + a(y_t - x_t)\delta$$
$$y_{t+1} = \delta(bx_t - y_t - x_t z_t) + y_t$$
$$z_{t+1} = \delta(x_t y_t - cz_t) + z_t$$

Where b = 28, a = 10, c = 8/3 and δ = 0.01

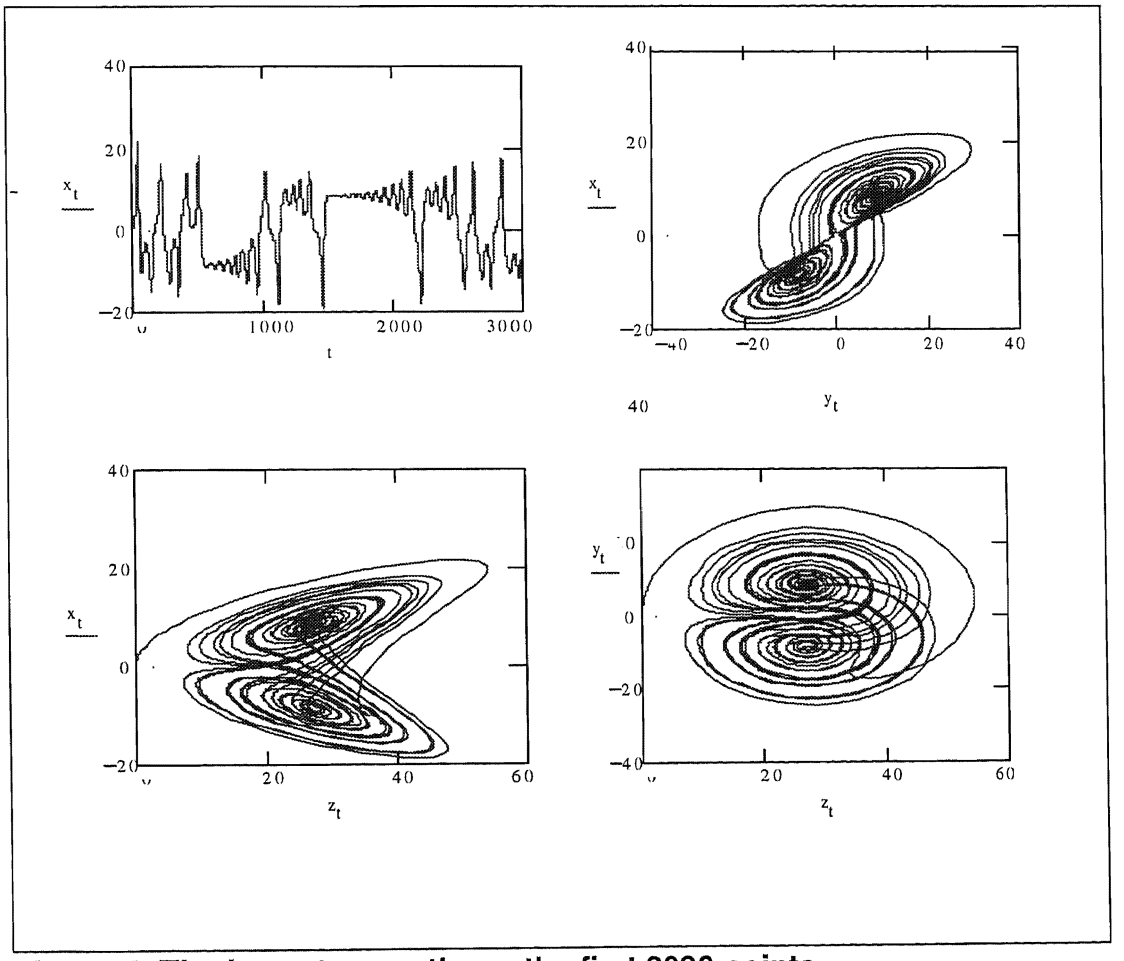


Figure 4, The Lorentz equations, the first 3000 points

The above series are introduced because they illustrate how complex behaviour can easily be produced by simple equations with non-linear elements and feedback, but also because they will later be used as benchmark series. Note that the path taken by the series, known as the *trajectory*, in figures 3 and 4 forms a complex shape beyond which the series never strays. This shape is known as the *attractor* of the function.

2.1.2 Embedding as a means of unravelling chaotic series

(Ruelle,81) and (Takens,81) first described a simple method for analysing chaotic series called *time series embedding*. It can be illustrated simply by observing the effects of plotting pairs of points $\mathbf{x_t}$ and $\mathbf{x_{t+1}}$ for both the Logistic equation and the Tent map as shown in Figure 5.

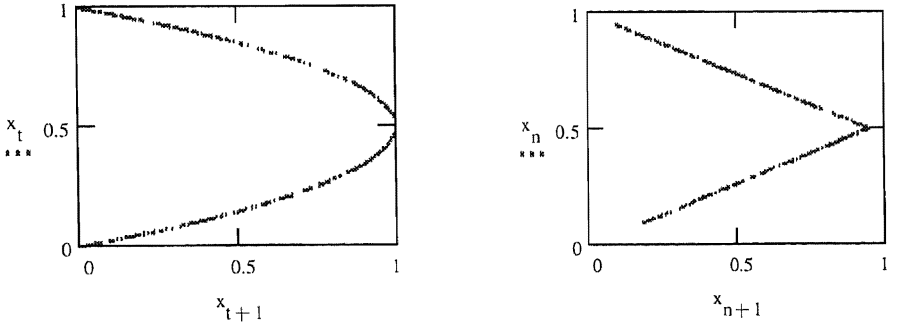


Figure 5, Embedded logistic equation and Tent map.

The above graphs show that in both cases plotting pairs of points from the time series has produced recognisable patterns. Given x_t we can make a very good estimate of x_{t+1} by interpolation. This principle extends to multiple dimensions, and in general can be written as:

$$\mathbf{X}_t = x_t, x_{d+t}, x_{2d+t}, x_{3d+t}, \ldots x_{nd+t} \qquad 2.1$$

Where **X** is the embedded vector, d is the *Separation*, and n the *embedding dimension*

Takens (Takens,81) showed that this principle generalises, and that given a chaotic series correctly embedded there existed a smooth function that would model it perfectly. Both the correct embedding dimension and the smooth function must however be discovered empirically. An extra complication is that the best predictions are not necessarily achieved using contiguous samples to form the embedding vector. In fact it is often better to separate the embedded samples by some gap. The gap is termed the embedding separation.

2.2 Chaos theory based measurements.

2.2.1 The Lyapunov exponent

An attribute of chaotic systems that has entered into legend as the butterfly effect is that of high sensitivity to initial conditions. It is considered proof of the presence of chaos if this attribute of the system under analysis can be demonstrated (Wolf 85). The standard method of identifying this attribute is by the calculation of the Lyapunov exponent (Wolf,85). This exponent measures to the base 2, therefore in bits, the rate at which neighbouring points on the attractor diverge as they are moved forward in time.

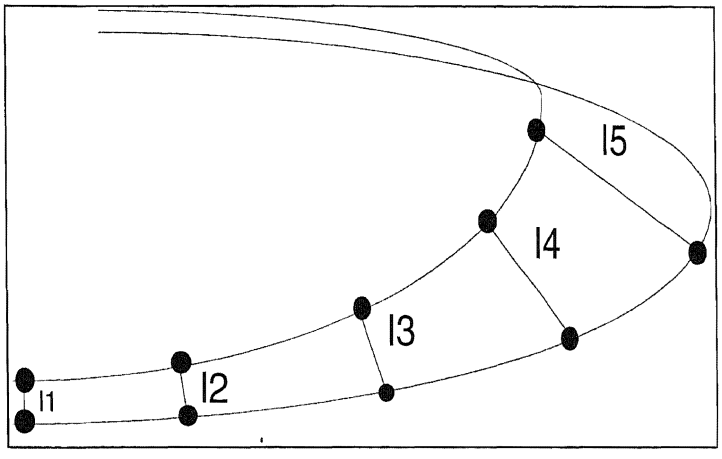


Figure 6, Two neighbouring trajectories on an attractor showing the characteristic divergence in time associated with chaos.

Given sufficient data we can talk about how a volume of space on the attractor dilates over time.

The trajectories on the attractor are embedded in a multi-dimensional space, and so the divergence is properly represented as the difference between 2 n-tuples. The dominant average Lyapunov exponent is defined as:

$$L = \log_2 \frac{\left(\sum_{n-1} \frac{l_{n+1}}{l_n} \right)}{n-1} \qquad 2.2$$

Where n indexes the samples, and l is the Euclidean distance between a trajectory and its nearest neighbour.

With very large amounts of data the density of points in a region of the attractor will permit the calculation of local Lyapunov exponents. That is to say that the rate of divergence will not necessarily be constant over the whole attractor, and so localised measurements can be made. In practice we are unlikely to be afforded this luxury when dealing with experimental data, and so the first exponent is the most that we can hope to measure.

The Lyapunov exponent has the units *Bits per sample time step.*

Positive Lyapunov exponents are considered evidence of chaos, Negative exponents of mean reverting behaviour, and the value zero is characteristic of cyclic behaviour. As an example of the latter, the attractor of a sinusoidal system such as an excited pendulum is a circle. If we consider the time evolution of two points one degree apart on the circle, the distance between them will remain constant, thus the ratio of two consecutive lengths will be unity, and thus the exponent will be zero.

2.2.2 Hurst Exponent

H.E. Hurst (Hurst,65) is responsible for a measure of predictability of time series that has interesting characteristics. The exponent is derived using so called R/S analysis. Given a time series **X** containing a number of points, n, and choosing an integer divisor p where for convenience: $10 <= p < n/2$, the data can be divided into n/p blocks.

For each block the average value is calculated, then the maximum range of each block and the standard deviation of each block.

The value (range)/(standard deviation) is calculated for each block and then averaged over the blocks.

This average value *rs* is related to the Hurst exponent by the following formula:

$$rs = \left(\frac{p}{2}\right)^{H} \qquad 2.3$$

where H is the Hurst exponent. In order to gain a more reliable estimate the value of rs is calculated for all the possible values of p, and the resulting tuples are logged and a linear regression is performed on them. It is the gradient of the regression line that is used as the Hurst exponent.

Hurst exponent values range between 0 and 1. A value of 0.5 indicates a true random walk, a value $0.5 < H < 1$ indicates so called persistent behaviour, in that one can expect with increasing certainty as the value moves towards one that whatever direction of change has been current will continue. A straight line with non zero gradient would have a Hurst exponent of 1.

Similarly, values $0 < H < 0.5$ indicates anti-persistent behaviour, in that one can expect that whatever direction of change is current is unlikely to continue. At the limit of 0 the time series must change direction every sample. This gives a clue to a relationship between one definition of the fractal dimension and the Hurst exponent. This is:

$$D = 2 - H \qquad 2.4$$

So a Hurst exponent of 1 gives a Fractal dimension of 1, as one would expect with a straight line. A Hurst exponent of 0 must belong to a time series that is so volatile as to fill the 2 dimensional space and thus have a dimension of 2.

2.3 Derivation of Embedding Parameters

In order to analyse and predict a chaotic time series we must embed it using the method in 2.1.2. The appropriate values of *d* and *s* are important to ensure accurate results. See 5.3.

2.3.1 Empirical methods

In the absence of theory to guide us empirical methods seem very attractive so long as the search space is not too large. One can define practical bounds for embedding parameters, and thus define a search space. Reasonable bounds might be:

1) we are looking for integer valued dimension and separation values
2) we are unlikely to be able to predict anything requiring an embedding dimension greater than 8
3) separation is unlikely to exceed 16 (based on experiment).

A brute force search of this space would require 128 trials to cover all possibilities. However without independent justification for a choice of embedding parameters one could only use the value of some derived metric such as prediction performance to determine the optimum. In order to perform prediction we must form a model of the attractor and thus require a supervised learning algorithm of some kind. In the early part of this work the only available supervised learning algorithm with which to

generate predictions, back-prop Neural Nets and variants, also had topological and functional parameters that were not independent of the embedding dimension.

Thus unfortunately the number of free variables was increased to 7, and the number of trials increased to 3,276,800. Some methodology to search this space more efficiently was required.

There are a variety of gradient and stochastic methods for optimising multi-parameter processes such as that discussed above. The one key problem in this work that constrains the selection of algorithm is the presence of a stochastic process, a back propagation neural net, in the process to be optimised. Neural nets, especially those with greater topological complexity than required for the problem in hand, will often find different solutions each time they are trained, or take different numbers of epochs to reach the same solution. The optimisation surface for a process containing a neural net is therefore very noisy. Gradient methods, such as hill climbing will not work well in the presence of noise since the gradient of the surface is forever changing. Two methods offer hope however: Simulated Annealing (Ingber,93) and Genetic Algorithms. The former was rejected because of a lack of specific evidence as to performance in the presence of noise. One has only to look around to see that Genetic Algorithms perform very well in a noisy environment.

2.3.1.1 Genetic Algorithm based selection of embedding parameters

Genetic Algorithms, (Gas) seek to copy the processes of natural selection (Darwin, 1859) and the strategies used by nature to modify species to ensure survival.

They were first invented by (Holland, 75) and his pupils in the mid 70s. They have found widest practical application in optimising a fixed number of parameters of some process interacting with an external environment. The main characteristics are:

- The representation of the parameters to be searched as strings of binary digits split into fields in imitation of chromosomes.

- The selection of good genetic material for reproduction based on fitness, an externally applied measure of its worth, and random chance.

- The use of cutting and splicing (crossover) to generate offspring from parents.

- The use of mutation to flip bits in the chromosomes during mating (normally with a low probability) as a way of adding "new blood" to the gene pool.

When designing a genetic algorithm search, the first thing to be decided is the format of the chromosomes. The parameters to be searched are encoded in binary and concatenated one to another to form strings of digits.

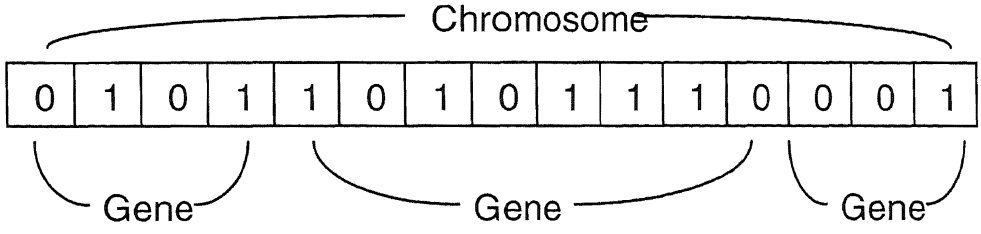


Figure 7, Genes and Chromosomes

The space allotted to a single parameter is called a Gene and consists of several binary digits. The order of genes within a chromosome is not important. Maximum and minimum values for each parameter need to be determined, and the numbers of bits in the gene determine the resolution to which this interval can be represented. The definition of these fields is external to the operation of the GA. This fact is important, the operators of crossover and mutation require no knowledge of the defined fields, and it is even possible to change the interpretation of the fields during processing.

This facility is the basis of Dynamic Parameter Encoding. (Schraudolph ,93). The next important decision is the population size. Typical values are 30-500. The larger the number the slower the optimisation, the smaller the number the greater the chance that some vital solution in a far corner of the parameter space will go unexplored.

The operation of a GA is fairly simple.(Davis,91. An initial population of chromosomes with a length decided by the sum of the gene lengths are initialised with random binary values. They thus represent randomly selected points in the parameter space.

In turn each chromosome is extracted from the population and its genes decoded to extract the parameter values. The parameters are tried out on the external process and a monotonic though not necessarily linear measure of performance must be generated. This measure called fitness is the associated with that chromosome.

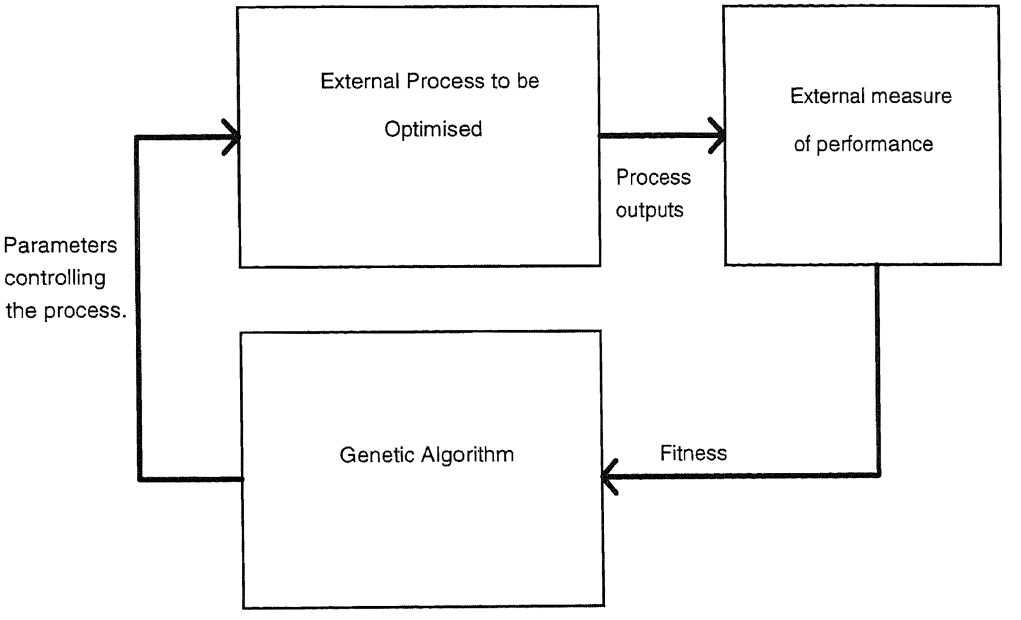


Figure 8, Genetic Algorithm processes

When every chromosome in the population has been tried out a new generation is created.

Individuals are selected in the most simple form of GA by using "roulette wheel" selection. The probability of selecting any particular chomosome n is set to:

$$p_t = \frac{f_t}{\sum_n f_n} \qquad 2.5$$

where f_t is the fitness of the chromosome, and n indexes the other chromosomes in the population.

Two mates are selected by the above process and by use of a biased random number generator with two states the partners are either selected for crossover or passed on to the next generation unchanged. The probability of crossover is generally set to ~0.8. If crossover is selected another biased random number generator is used to select a location on the chromosome strings with uniform probability. Both chromosomes are cut at this point and their respective halves swapped.

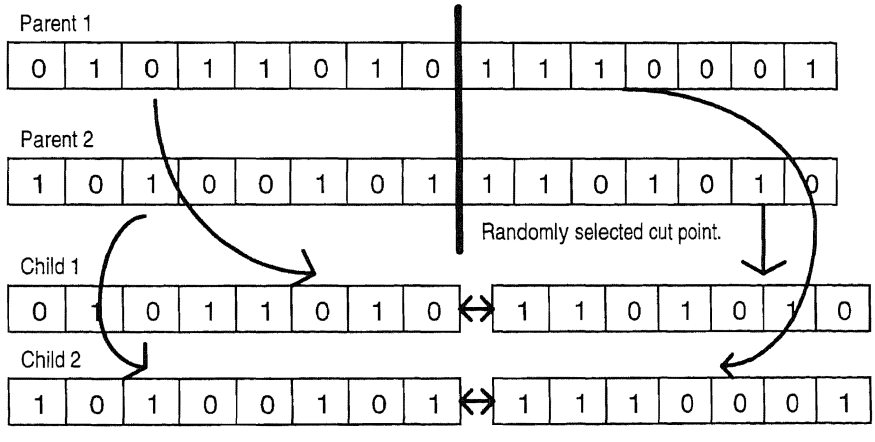


Figure 9, The Crossover operator

Whether or not crossover takes place, each bit of the two chromosomes is examined and based on a biased random number generator with two states arranged for low probability the bit is flipped, i.e. 0-1, 1-0. This

process is termed mutation, and the probability associated is the mutation rate. Typically this is set to ~0.01.

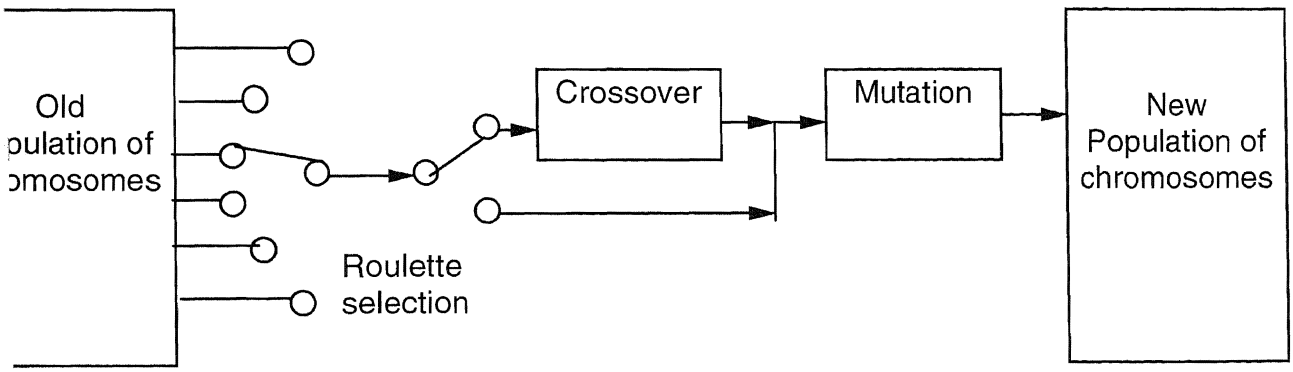


Figure 10, Generating a new generation

This selection, crossover and mutation process is repeated until a new population is formed, and then the old is destroyed. The process of evaluation is then performed and the whole process is repeated many times over.

When plotted over the generations, GAs exhibit increasing average fitness, and the production of a few super-individuals with high fitness. Commonly GAs are used to find several possible solutions to a problem, and the generation of super individuals too soon can prevent the searching of other possible solutions. This state of affairs is called premature convergence.

There are a few possible additions to the basic GA formulation that are appropriate to this application.

- Elitism is the process by which the individual chromosome with the highest fitness from each generation is passed on unmodified to

the next generation. The values represented in each gene can be read as binary coded numbers, or as Gray coded numbers. Gray code is an alternative to binary in which the representation for any two contiguous numbers changes by only one bit. Gray coded Genes are less prone to dramatic changes of value when mangled by the crossover and mutation operators. (Carmana,88)

- Fitness scaling is designed to handle the situation where the majority of the population have similar fitness scores. Under these circumstances the pressure for the reproduction process to select the best chromosomes is slight. The best has little more chance than the worst. This implementation of fitness scaling calculates the mean and the standard deviation of the population fitnesses. The transformation *fitness = fitness - (mean - standard deviation)* is applied to each chromosome's associated fitness value, and those with negative resultant fitnesses are set to 0. The variation in the set is now much larger and optimisation proceeds more quickly. (Grefenstette,86)

The parameters selected to optimise the Neural net based time series predictor were:

Name	Type of parameter	Number of Bits
Embedding dimension	integer 1-8	3
Embedding Separation	integer 1-16	4
Neural net 1st row hidden units	integer 1-16	4
Neural Net 2nd Row hidden units	integer 1-16	4
Hidden rows	integer 1-2	1
Learning rate	float look up table	4
Momentum	float look up table	4

The gene values were converted according to the description above.

2.3.2 Analytical methods

Takens (Takens,81) and Mañé (Mañé,81) determined the upper bounds for a successful embedding if one knew the fractal dimension of the attractor d_a. This is simply:

$$d_e \geq 2d_a + 1 \qquad 2.6$$

Where d_e is the embedding dimension, an integer value.

However, as Medio points out, (Medio,92) The fractal dimension of the Lorenz attractor (Lorenz, 63) is 2.06, thus Takens leads to an embedding dimension of 5, whereas it is known that the practical embedding dimension of this series is 3. When dealing with large amounts of

noiseless data, as can easily be obtained with the various series described in 2.1.1 an over-large choice of the dimension may not matter when calculating say Lyapunov exponents, but an over-large choice can be fatal where modelling is concerned. There is empirical evidence for this presented later in this thesis. See 5.3. Martin Casdagli (Casdagli,89) also showed this effect using a brute force search through a range embedding dimensions and constructing models of various types for each dimension value which he then tested on fresh data. The reasons for this effect are most likely to be parsimony, or the lack of it. Any model generated with more than the required number of inputs, and thus more than the required parameters or model complexity is less likely to perform well on out of sample data. See for justification of this concept 2.5.7.

The Takens upper bound still requires the calculation of the dimension of the attractor, and for experimental data where the equations underpinning the dynamics are not known this requires trial embeddings; so the whole process is somewhat circular. Some other method is therefore required to find the embedding dimension. There are several possible analytical methods for deriving a choice of embedding dimension. The one with the oldest lineage (Farmer, 89) is to look for the saturation of some system invariant, such as lyapunov exponent with increasing embedding dimension. Another is *the false nearest neighbours* algorithm and yet another is the analysis of the Kaplan-Glass statistic (Kaplan,91) with increasing embedding dimension. Of these the simplest to implement and

the seemingly most robust is the false nearest neighbours statistic. It has a geometrical formulation that sets it apart from the other two, which are concerned with local derivatives of the attractor, and thus gives a more independent view of the attractor dimension. The choice of a methodology for determining the separation is more complex. Conflicting advice is to be found on this subject in the literature. Medio claims the separation is irrelevant so long as it is small, Casdagli that it should be related to the mean orbital period, Abarbanel (Abarbanel,92) that the first minimum in the auto- correlation should be used, and Fraser & Swinney (Fraser,86) that the first minimum in the Auto mutual information should be used. The separation is clearly not irrelevant as evinced by 5.3. The mean orbital period is both hard to calculate for experimental data, and unfeasibly large. The concept of the first minimum of the auto-correlation is useful, but correlation as a measure is dependant on a Gaussian distribution, which we know not to hold for the data that we want to analyse. See 5.5. Mutual information makes no assumption about the distribution of the measured series, and is therefore the most attractive measure to hand.

2.3.2.1 False nearest neighbours

False nearest neighbours (Kennel 92) is a methodology for deriving a workable embedding dimension for a given time series.

A trial embedding is formed with separation given by whichever technique is chosen and embedding dimension n. The nearest neighbour vector for each embedded vector is found and the Euclidean distance between the two vectors measured. Then the embedding dimension is

increased by one for both the current vector and its neighbour by appending the appropriate data value. If the distance between these two new vectors is dramatically different after this change the neighbour is considered a "False Neighbour". The number of these is counted and expressed as a fraction of the number of rows. A curve of these values against n is plotted and an embedding dimension with a suitably low number of false neighbours selected.

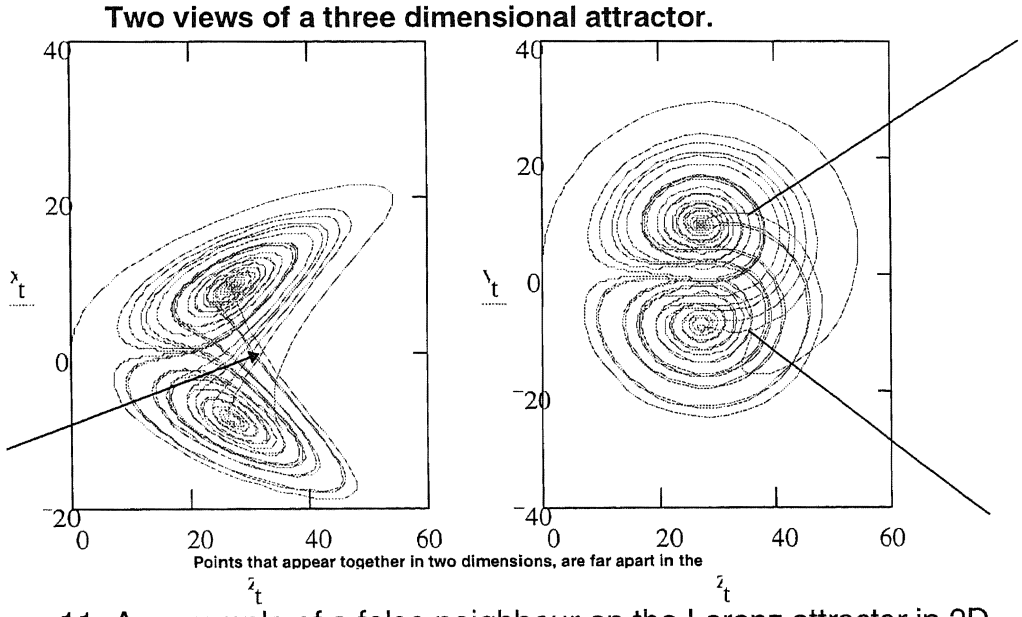


Figure 11, An example of a false neighbour on the Lorenz attractor in 2D

2.3.2.2 Mutual information

Mutual Information is a concept conceived by Claude Shannon (Shannon 49). Mutual Information attempts to measure in bits the amount of information that can be inferred about one series of symbols by another. Lafrance (Lafrance, 90) gives a derivation of this concept. In general given two series **x** and **y** with indexes *I* and *j* respectively, the average mutual information can be calculated as:

$$I(x, y) = \sum_{i,j} P(x_i, y_j) \log\left(\frac{P(x_i, y_j)}{P(x_i)P(y_j)}\right) \qquad 2.7$$

Note that Mutual information is positive and symmetrical, that is $I(x,y) > 0$ and $I(x,y) = I(y,x)$.

Auto mutual information is measured by using two copies of the same series, one delayed by some lag *t*. As *t* increases it is hoped that several minima will be observed in the resulting curve, and the value of *t* at the first minimum is selected (see Figure 33). The argument for this is somewhat intuitive. We want to choose an embedding separation so that each column in the embedding supplies the maximum amount of new information. This must imply the minimum of old information, and thus a minimum of mutual information.

With some series the mutual information values will just gradually decline without any clearly detectable minima. The advice of Abarbanel et al. is to take t=1 if this occurs.

2.4 Supervised Learning as non-parametric modelling

Regression analysis is an example of parametric modelling in that the form of the data series to be modelled has already been inferred or guessed at before the analysis begins. The purpose of the regression is to find parameters for the model that minimise the error between the modelled series and the series itself. The statistician must try increasingly complex models, and then use regression analysis on them to determine their parameters until the results are deemed acceptable.

In recent years, techniques have been developed that simplify the modelling process. Non-parametric techniques are those where both the form of the modelling function and the parameters to it can be modified during the analysis.

Normally these techniques are iterative, and since the modelled series must be accessed repeatedly such a technique can also be considered a form of learning. Typically the algorithm iteratively forms a better and better model as the data is presented, and so behaves as an analogue of human learning. Indeed practical algorithms have been trained, for instance, to learn the past tenses of common English verbs. Interestingly in the early stages of training, the same kinds of mistakes are made by the algorithm as a small child just learning his or her own language (McClelland,86).

The processes of learning can be split into three different categories for convenience: *Unsupervised learning*, *Supervised learning* and *Reinforcement learning*.

Unsupervised learning is where data is presented to a learning algorithm in the form of multiple consecutive n-tuples, and the algorithm is required to classify these n-tuples into meaningful groups without any external guidance. This is akin to learning a new language by immersion in the country where that language is spoken. Initially it is very hard to recognise the sounds of the language, or to separate the individual words. The ability can however be gained without the help of a teacher.

Supervised Learning is where both input data and a set of expected responses are presented to a learning algorithm. Typically both the input and response data are paired n-tuples. The algorithm should learn to associate the pairs of n-tuples, and correctly return the response n-tuple once training is completed when a data n-tuple is presented. The human analogue of such a process would be rote learning.

The usefulness of such an algorithm would be enhanced if, when an input n-tuple similar but not identical to one of the training input n-tuples were presented a response the same as or close to the expected response were generated.

This ability to respond rationally in the presence of noisy data is termed the "generalisation ability" of the algorithm.

As of yet there is no generally agreed measure of this quality. The quality depends on the ability of the algorithm to glean the general idea or function underlying the n-tuples to be learned, and to reject noise or false data points in the training set. Usually, as we shall see, algorithms learn

the gross features of a training set first and the finer detail last. Since we hope that the signal outweighs the noise in our data set, the noise is normally learned last, and various techniques, which will be discussed later, can be used to terminate training before this point.

Reinforcement learning, also known as learning with a critic, requires some external arbiter to permit learning. In this paradigm the algorithm is required to suggest a set of parameters for some external process, and is given a "correctness" score for that set of parameters. Rather like a game of twenty questions the algorithm is required to produce better and better parameter guesses as training continues. Some forms of Neural Network and genetic algorithms are examples of this form of learning.

Both Supervised and Reinforcement learning have been used in the work described in this thesis, though Reinforcement learning was dropped in the final version for reasons that will become clear. Supervised learning is used to model the smooth function generated by a suitable embedding of a system demonstrating non-linear behaviour, and thus to permit the generation of predictions.

2.5 Neural Networks

These are perhaps the most commonly used supervised learning algorithms. There are few subjects in computer science that have seen a greater growth than Neural Networks in the past ten years. A very large

volume of papers have been generated and several journals and numerous conferences are devoted to the subject. For all that the basic ideas underpinning this technology are fairly simple. In many cases Neural nets are the breach through which the world of computational intelligence is entered, and much of the work published under that banner is of more general application.

2.5.1 Simulated Neurons

Common to all implementations of neural nets, and there are many, is the concept that simple processing units can be used together in a way vaguely similar to the human brain to achieve some kind of group intelligence. The electro-chemical interactions of a living neuron with its neighbours are fairly well understood. Few implementations however give much regard to these models; they represent algorithms along the same theme that are more easily Implemented in software or silicon. One still hears the term biological plausibility used in the literature, mostly to remind experimenters of the attributes of a neural network that permit it to be called neural. These are that learning should be spread over a number of simple processing units, that learning should modify some attribute of the connections between the units or the units themselves, and that processing should be local. Local processing means that, ignoring synchronising signals, it should be possible for each neuron to process individually using the stimuli available to it, and to learn by modifying only itself or the attributes of connections directly attached to it. There are several successful algorithms that modify all the parameters of a net at

once in order to effect learning, and can only be formulated to act in this fashion. As useful as these algorithms may be they are not neural.

In back propagation neural nets a neuron has a simple structure. Synapses bring signal values to the input of the neuron. These values are real numbers and are summed at this node. The output is formed by applying a sigmoidal activation function to the sum. If we denote the sum of the input values as x, a typical choice of transfer function is:

$$y = \frac{1}{1 + e^{-x}} \qquad 2.8$$

2.5.2 Simulated Synapses

Synapses are the interconnections in a network, they carry signals from inputs to neurons and neuron to neuron in a single direction. The signals they carry are real numbers representing input values to the network or excitation values of neurons. A synapse has a weight associated with it that scales the signal passing through it. These weights are the only part of the model that is typically modified while training the network. A special type of synapse is commonly provided, termed a bias synapse. This has the value 1.0 permanently connected to the input.

2.5.3 Network topologies

In order to simplify the study and design of neural networks researchers tend to use simple topologies. Network designs fall into two types: feed forward and recurrent. A Feed forward net typically has a row of input terminals, 1 or more rows of hidden units, and one row of output units.

Interrogating a trained network involves signal propagating across the network in only one direction. There are no feedback paths. Typically to further simplify the topology each input terminal is connected to each hidden unit in the first row by a synapse. If more than one hidden row exists then each unit in that row is connected to each of the units in the next row and so on until the output row.
There are no connections that cross rows. Typically every hidden and output unit is also supplied with a bias unit.

Recurrent networks permit feedback across rows, and are capable of detecting and responding to sequences of input values. In this work reliance will be placed on the embedding process to disentangle time dependencies, and the promise from Takens that once we have embedded the signal we need only to model a smooth function.

2.5.4 The Error Surface

The most illuminating concept in understanding the various learning algorithms that can be applied to train neural nets is the error surface. Having in advance decided on a supervised learning training set consisting of input vectors and expected output vectors, and having decided on a topology for the neural net one has fixed the number of free variables, weights and bias weights in the network. Training the network will require selecting a particular set of values for these free variables. We can imagine therefore a high dimensional space where we devote one

dimension to each variable, and one extra dimension to the sum of the squares of the errors over the training vectors of the network.

Now for any choice of values for the variables, we can evaluate the network through the training set and arrive at a single value for the sum squared error. Thus there is a surface, the error surface, representing the performance of the network for different sets of weights and biases. Given enough time we could evaluate grids of points and build up a picture of this surface. This would indeed be one kind of learning algorithm.
If the network is capable of modelling the input data there will be one or more points on the surface where the error is, or is close to zero. The process of training a supervised learning algorithm is to incrementally move towards and find one of these minima.

2.5.5 Symmetry and multiple solutions

A trained fully connected network has many symmetries inherent in its design. In the example given below with four inputs, three hidden units, and one output, one could easily swap hidden units, say 1 for 3, and arrive at an identically performing but different net. We can thus be sure in this case that if there is one zero on the error surface there will be 2 more by symmetry.
We cannot be sure how many solutions exist for any given error surface, especially since the surface is unbounded. The presence of multiple solutions eases considerably the process of training. The weights of

Neural nets are normally seeded with small random numbers, thus placing the nets starting point somewhere close to the origin. Whichever learning algorithm is employed, the chances of finding a minima nearby are much enhanced by symmetry.

2.5.6 Practical Neural Learning Algorithms

Two will be considered, Back propagation (McClelland,86) and a guided random search algorithm by (Baba,89) based on work by Solis & Wets (Solis,81)

Back propagation is the most popular supervised learning algorithm and uses a gradient descent method to find the fastest downhill path on the error surface towards the solution.

The Baba algorithm is best explained as a series of steps:

1) The network is initialised with small random numbers for the weights and biases
2) The sum squared error is evaluated for the particular set of weights and biases.
3) If the error is below a predetermined level processing is terminated and the network deemed trained.
4) otherwise small random numbers with a Gaussian distribution are generated for each variable. These numbers are temporarily added each to their respective weight or bias, and the sum squared error is

evaluated over the training set. If this has improved the error the changes are incorporated. Go to 3.

Both algorithms have the effect of stepping across the error surface and following downward slopes.

BackProp can be caught in local minima, if bounded on every side, The Baba algorithm, though slower will not. Since the search is performed with Gaussian random numbers sooner or later a change of sufficient size to leave the minima will be generated.

2.5.7 Neural Network Minimisation

Even non parametric modelling techniques must have some parameters! As soon as Neural networks began to be used for practical applications where a net was trained on some subset of all the possible training vectors, and then required to predict others not in that subset, the problem of finding the appropriate network topology became apparent. Neural networks are considered universal approximators of all functions of practical interest (Hornik,89). This is based on the assumption that a neural network of unbounded size could be generated.

However the practical problem is finding the minimum network that will learn a given training set with a particular error rate. One can of course always generate a net that is too large.

The motivation for this is inspired both by practical necessity and theory; in this case computational Learning theory.

The philosopher William of Occam (1285-1349) is credited with the idea expressed *as "Non sunt multiplicanda entia praeter necessitatem"*. Literally "Entities should not be multiplied unnecessarily". This is known as Occam's razor and was used as a method for discarding elaborate and overblown philosophical theories. It can also be recast as the principle that the simplest model is most likely to be correct.

Cheeseman (Cheeseman 90) cast this problem of model minimisation in a Bayesian framework and showed that the most likely model in a Bayesian sense would be generated by minimising the sum of two terms: The complexity of the model, and the complexity of the residuals given the model. Complexity is measured here using the information theoretic concept of message length, and is the minimum number of bits required to encode the model or information. The principle is of great interest, but the practical implementation of such a minimisation system is difficult to achieve with non parametric modelling with the requirement to generate prior probabilities for each parameter.

The general principle is clear however. If the purpose of training a model on a subset of its possible input and output vectors (in sample data) is to produce the best possible response with data vectors not present in the training set (out of sample data), then one must trade off model complexity against training set error, and there must be an optimum combination.

The above discussion assumes a parsimonious model representation. In practice there are two elements to any excess of model definition, that

which is explained by the above and excess caused by lack of information about the required model complexity before the training process begins. An example might be useful here.

Let us assume that we generated a training set by taking some simple function, such as $x= y^2$ over the range 0,1 and sampling it at 0, 0.01, 0.02 etc. to build our training set. Let's assume also that small random numbers chosen to have Gaussian distribution were added to each training sample.

We could construct a net with perhaps three layers of Neurons and say five hidden units in each layer to learn the training set.

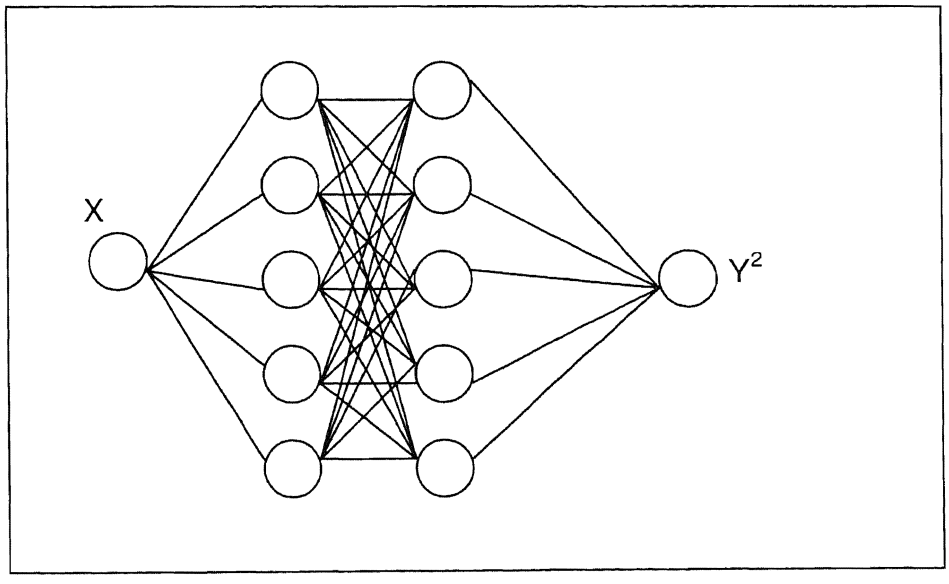


Figure 12, Example neural network illustrating sources of irrelevant complexity.

Using a non parametric modelling technique like Neural Networks there is nothing to guide our choice of topology before training begins. The above may well contain too many processing elements.

During training of the net we can expect the larger features of the training set to be learnt first, and for the net to continually refine its model. This is in the nature of gradient descent. At some point during training the finer features that are modelled are caused by the added noise. The network is oblivious to this and continues to construct and refine the model so as to explain the noise as well as the underlying function.

Now during the training process some of the hidden units may prove to be totally redundant, some of the weights may be set close to zero, or two or more units may share some modelling burden that could have been performed by one. We thus have a structural burden of dead wood. We also have inappropriate model refinement in that the model has learnt the perturbations of the input data due to the noise.

Removing the structural dead wood is the purpose of network minimisation. There are two ways to achieve this: growing a net from some small initial starting point by adding model complexity only as it is required, and pruning or skeletonising a baggy trained net.

Examples of the former are the Upstart Algorithm (Frean,90) and Cascade Correlation (Fahlman, 90). Examples of the latter are Optimal Brain Damage (OBD) (Le Cun,90) and Optimal Brain Surgeon (Hassibi,93) after training, and weight decay during training (Hinton, 86)

Since after training minimisation methods can be grafted on to existing back propagation algorithms they are the most attractive. The two examples above look at the second derivatives of the error surface of the

trained net and decide which weights can be deleted. In OBD the weights are ranked in order of magnitude of second derivative, and the lower n% are deleted. An iterative process is required to find the optimum deletion rate. A maximum acceptable error rate is chosen and small percentages of the weights are deleted. The net is briefly retrained and the process repeated until the chosen error rate is achieved.

2.5.8 Cross Validation

A non-parametric modelling technique such as Neural nets or Genetic Programming (Koza, 92) tends to learn gross features first and finer features as learning progresses. If the training set is both large and noiseless this will cause no problem, and we can permit the process of refinement to continue until some arbitrary level of accuracy on the training set is achieved. This is not the case with noisy and shorter data sets. Here the finer features are often noise rather than the underlying process to be modelled and one must stop training before these are learnt to get the best performance on out of sample data.

The following diagram is derived from Hecht-Nielsen (Hecht-Nielsen,90).

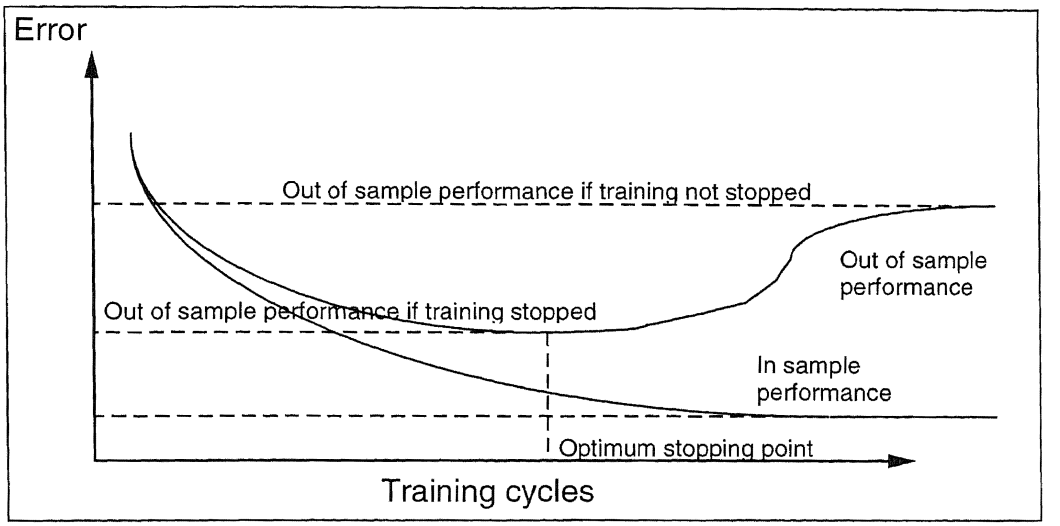


Figure 13, in and out of sample training performance

Cross validation is a method to prevent this over-training. A separate validation data set drawn from the same source is required and the performance is monitored on this set. Training is terminated on the minimum of this error. This requires hindsight, so the normal technique is to store copies of the model only when out of sample performance improves compared to the last training cycle, and to continue training for some suitably large period after the last model storage.

2.6 Surface modelling techniques

When a neural network is trained it represents a model of the function that generated the training set to a certain degree of error. The Neural network can be thought of as representing the data set, which can be reconstituted through it. Indeed one form of data compression relies on this principle (McClelland,86).

If this data compression is not considered a virtue for a given application there are other supervised learning algorithms that use the training data set as part of the model. These effectively require no training, instead what processing is required is transferred to the interrogation phase. These algorithms are basically interpolative. They rely on some interpolating function using existing points in the data set to give results. See 4.5.1 for a description of the local approximation algorithm used in this work.

2.7 Trivial Predictors

It is too easy, given the complexity of the tools at hand in time series analysis, and the many limitations on their use that may not be remembered, to see in the results of a predictive algorithm more than is really there. One of the most powerful methods for returning to sanity is the use of Trivial Predictors.

These are simple models that can be easily applied to the predicted data, and against which performance can be meaningfully and simply measured.

The first order trivial predictor simply predicts that a series sample will have the same value as the preceding sample, i.e.

$$x_t = x_{t-1} \qquad 2.9$$

The second order Trivial Predictor assumes the same change in the data as the previous sample, i.e.

$$x_t = 2x_{t-1} - x_{t-2} \qquad 2.10$$

The second order Trivial Predictor is a surprisingly powerful competitor, especially if the Hurst exponent of the time series in question is greater than 0.5 and tending towards 1. In the Biotechnology series analysed in 6.1 a 2nd order Trivial Predictor outperformed an Auto-Regressive Moving Average model (Box,76), considered a few years ago to be the state of the art in time series prediction..

2.8 Object Orientation of Design

Few who are aware of trends in the world of computers can have failed to notice Object Oriented Programming (OOP). The fundamental ideas behind OOP are concerned with increasing the "engineering virtue" of computer code. An entity possessing engineering virtue is some or all of: robust, cheap, easy to understand, easy to maintain and of general application. A 10mm bolt would be a good example of such an object. A more subtle form of engineering virtue might be called extensibility, both in practice, and of design. A Portacabin might be an example of such an object. If you require more space than your single Portacabin permits, you just park another one alongside, bolt them together and cut a door between the two.

Computer software written in high level languages such as Pascal or 'C' has traditionally had few of these virtues.

In recognition of these software shortcomings, the high cost of writing software, the high cost of maintenance and the high human cost of software faults, various methodologies have been developed. Out of the

vast possible space of working practices they represent methodologies that can be seen to have improved engineering virtue. Object Oriented Programming is the most successful of these methodologies. A programmer who adopts the methodology limits the range of responses he has to any problem, but automatically increases the virtue of the code he produces.

Virtue and Quality in this context are two different things, though the differences are subtle. Quality is about metrics, about faults and defects, about complexity and interrelatedness, about analyses that are made after the event, a fundamentally reductive process. There is no metric that measures how easily a design might be modified to encompass some new requirement in the future, a key element in engineering virtue.

OOP introduces the idea of *class* as a definition of an object that has interface functions known as *methods*. One designs a class to implement a particular software object, and decides on and implements the methods for the class. These methods act on and give access to the data members of the class that can usually be only accessed through the methods. Using this system the engineering virtue of 'data hiding' or 'need to know' can be achieved avoiding a whole class of bugs involving the spurious or unintended modification of a variable. Subsequently the methods may be found to be in error, in which case they can be changed, or inappropriate for some new purpose. If the latter a new class can be generated *inheriting* all or some of the methods and data members of the parent class. We now have two classes that have different behaviour but share

common code. This achieves another element of virtue, the re-use of software. Finally OOP offers a method for standardising interfaces that can make objects interchangeable at run time. This is called *Polymorphism*, and as its name suggests allows objects of disparate type to be treated similarly. If applied strictly by limiting methods and parameters to some pre-determined set we may have interchangeable objects that all perform the same function but by radically different methods.

Polymorphism is achieved through inheritance. Objects that inherit from other objects can be treated as if they are those parent objects for the purposes of reference.

The language of choice for OOP is C++, an extension of 'C' with object oriented additions. There is nothing about C++ that forces one to use object oriented design, one can still continue in the same bad ways of unstructured design, (and many do) but all the opportunities for virtue are there.

2.9 Overview

In this chapter an attempt has been made to review the literature associated with the work and to explain the underlying techniques and concepts used. Because the work covers several fields of knowledge the decision has been made to describe the techniques in more detail and at a more basic level than is customary for a thesis.

While the function of Neural networks and Genetic Algorithms may be known to many, the sections on derivation of embedding parameters, local

approximation and model parsimony contain information that is less widely known.

3. The evolution of the predictor - previous versions

This section describes the evolution of the time series predictor and the various algorithms and methodologies that were tried out. All the versions described here were designed and coded by the author.

3.1 Development and run time platform

Throughout the software generated was designed to run on IBM PCs and clones thereof using initially C and later C++ all under Microsoft Windows. Versions 1,2 & 3 were 16 bit implementations. Subsequent versions used 32 bit code.

The most recent version is configured as an OLE control and can be embedded easily into other applications.

The first version was tested on a 386 based computer, the most recent on a 120Mhz Pentium processor, since several years elapsed in the process of development. Some of the performance improvement seen is therefore due to improving hardware technology. Due to changes in operating systems and their processor support it would not be possible to test the most recent version on the same system on which the first version was developed, even if such a system could be found.

3.2 Reasons for the changes in design

The constituent parts of the predictor have changed dramatically since first conceived. The initial version represents the state of the art at the time the work began when viewed from the perspective of computational

intelligence. This is an important point since the intention was to make use of experience that the author had in various computational intelligence algorithms to produce a workable time series predictor. The fact that very little computational intelligence technology is left in the final version, and that this was the result purely of a desire to improve the predictor, rather than to use a particular technology whether it worked or not, is in itself of interest.

It became obvious in making use of the first version that processing was exceedingly slow, and that there were a variety of parameters that needed to be set. Rather like being presented with a television set with seven unmarked controls that interacted in subtle changing ways, occasional flashes of light were seen. The literature offered little, no or contradictory guidance on the proper values of these parameters, for example see . Experimentation showed that they were all mostly dependant on the time series under investigation, but the form of the dependency was unclear. The intention was formed to investigate methods for determining or eliminating these parameters with the hope of deriving a time series prediction and analysis system that required little or no user interaction and that performed its processing in as small a time as possible. All this was to be achieved without compromising the predictive performance that the first version occasionally showed when by chance the choice of parameters happened to be correct.

In the process of developing the predictor, driven both by practicality and inquisitiveness, a novel combination of technologies has been used to achieve the results shown at the end of this thesis.

The following table describes the various versions that were built.

3.3 Previous versions in detail

Version	Supervised learning algorithm	Embedding parameter selection	Typical run time
1	Neural nets	manual	20 min. per parameter choice
2	Neural nets	GA	48 hours
3	Neural nets & minimisation	GA	24 hours
4	Neural nets & minimisation	Analytical	30 minutes
5	Local approximation	Analytical	30 seconds

The reasons for the changes in each version are mostly concerned with performance and simplicity.

Version 1 required selection of Neural net topology and learning parameters, and manual selection of embedding parameters. The user thus had 7 parameters to set with no guidance as to the correct values.

Version 2 tried to optimise all of the parameters simultaneously, using genetic algorithms but required 48 hours typical run time on a PC on the kinds of data sets described in section 5.1. Each neural net evaluation would typically require 10 minutes, thus in this time 288 parameter evaluations would be performed. In reality this is far too few considering the search space of around 3 million possible parameter sets, but it was the maximum amount of time that could be allocated to training. Occasional good results were obtained, generating sufficient interest to justify continued research.

In version 3 Neural network minimisation was used to avoid having to choose the net topology parameters, thus reducing the search space dramatically. The search space was now very much smaller but runtimes were still in the order of 24 hours, albeit with a much better chance of finding optimal parameter sets.

Version 4 derived embedding parameters analytically, using False Nearest Neighbours and Auto Mutual Information and the run time of the predictor was dramatically shortened. Even so the neural net could take 10-30 minutes to train, and occasionally no solution, or a poor solution was found. The user was still required to run the predictor potentially multiple times until suitable validation performance was seen. In practice this meant that given two different training runs with similar performance the prediction for a given time period was not always the same in magnitude,

and sometimes in direction. This tended to undermine the users confidence in the tool in a financial application. For it to be wrong was acceptable, as long as it wasn't wrong too often, but for it to be seen to disagree with itself on the same data set on subsequent runs was considered unacceptable.

Finally in version 5 with the use of the Local approximation algorithm consistency of predictions were achieved. All the stochastic elements had been removed, and given a particular data set one would get the same results every time the program was run. As can be seen from the later sections the quality of the predictions is high. Run times for this version are in the order of thirty seconds, depending on the data, and the majority of this is taken up by the auto-mutual information calculations.

There was another important factor driving the evolution of the time series predictor, which was that of debugging and testing the application. For the purposes of academic research the run time is of secondary importance to the quality of predictions or insight into new techniques that might be gained, however debugging and testing any system requires that it be run repeatedly with slight changes of parameters or code. If the run time of that application is 48 hours then progress in fixing any problems is exceedingly slow. A reductionist approach can be taken, and the individual elements can be tested separately, but there are always some errors that will only be apparent when the completed system is run. Also since this

software was wholly speculative, and none of its elements had been run in that particular form before it was not possible to determine in advance the likely performance or capabilities of the individual elements. A certain amount of trial and error was therefore required in improving the predictor performance. Again this is difficult when each run requires 48 hours. These considerations were a powerful incentive to reduce the run time of the software.

4. Description of the time series predictor

In this section the time series predictor in its final version will be examined in detail. Each component part will be explained and finally some results on artificial chaotic series widely used as bench marks will be presented. Figure 14 shows the major components of the system and how they are connected.

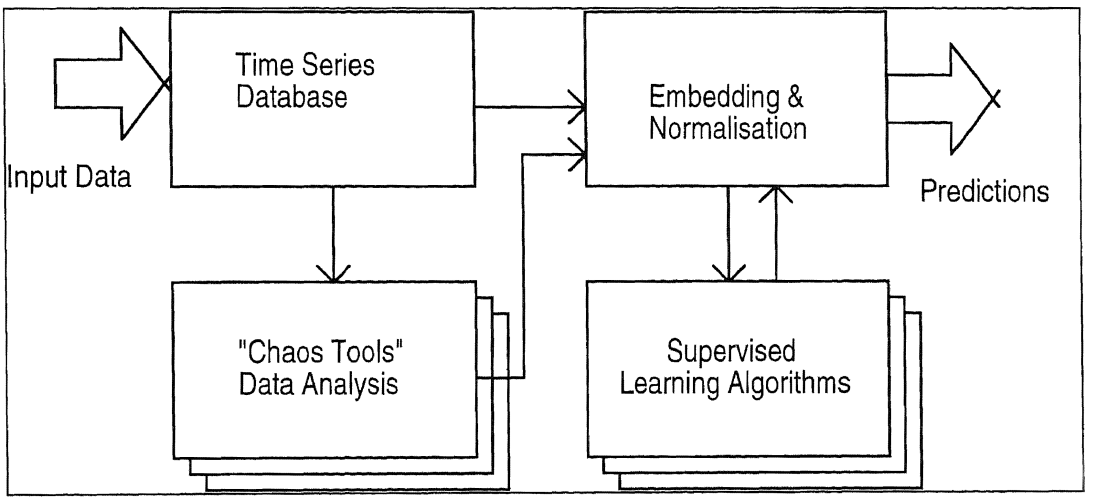


Figure 14 A block diagram of the Time Series Predictor

4.1 The time series database

The time series database was specifically designed to handle the demands of financial trading data. There are various characteristics of such data that make the database design unusual. This decision does not however prevent the database being used for simpler sampled data. Financial trading data represents different things in different markets, and is in fact a series of events loosely connected to a common theme. In

some markets such as commodities the prices provided by the exchanges are the actual values of deals struck. In Foreign Exchange (FOREX) the data supplied is not the price at which a deal was struck but the price at which a foreign exchange dealer is prepared to trade. The dealers constantly monitor the markets and post new prices electronically whenever their existing prices are out of date. In both cases the data represents discrete events that can occur at up to 30 times a second at peak times.

Some markets such as foreign exchange are traded round the clock, whereas commodity exchanges and stock exchanges have definite opening and closing times.

Different kinds of traders have different requirements as to the time scale of the data and predictions they require. Some "scalp" which is buying and selling over very short term intervals, perhaps seconds, and others look after pension funds that require investment horizons of years.

4.1.1 Formatting and storing input data

It is possible to perform chaos theory based time series prediction of financial data using data updates as the fundamental time-base of the predictions and analysis. The frequency of these updates varies enormously over a trading day. Alternatively, and more frequently one can sample the series at discrete intervals in time.

There are as yet no papers known to the author comparing the relative merits of the approach of using what might be called *update time* over elapsed time in financial series prediction. Because it was always

intended that this work should be of use outside the world of finance, see for instance section 6.1, the system described here was arranged to use elapsed time.

4.1.2 Re-sampling stored data

In the system described, input data are not considered to be samples equally spaced in time. They are however considered to be ordered in time. Each data point is time stamped and placed in a linked list in increasing time stamp order. Because the system described was designed to be able to handle multivariate data the entire database contains multiple linked lists indexed according to name.

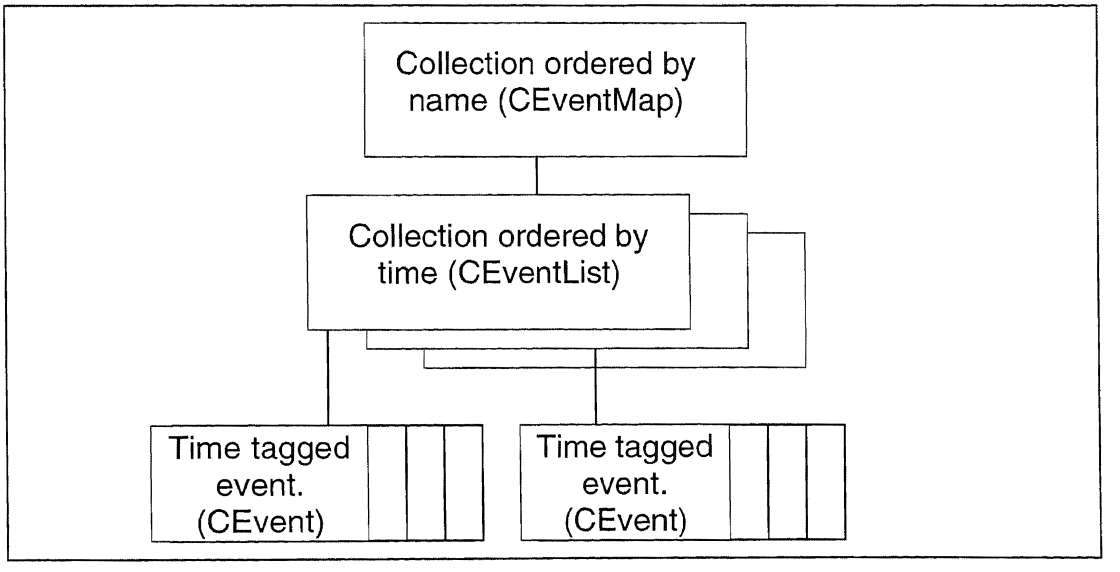


Figure 15 Organisation of the database

Figure 15 shows the organisation of the database. The fundamental organisation of the database is object oriented rather than relational.

The top level object is a collection indexing time series by name. When performing multivariate prediction each time series will have a different entry in this collection.

The next level down is another collection object, this time a linked list containing individual data objects ordered in increasing time stamp order. Finally the lowest level contains the sample objects, each sample contains a value and a time stamp corresponding to the time of collection or the end of the period summarised.

Training sets, validation sets, data for the chaos tools and for graphical display are all required from the database in a sampled format with regular time intervals between samples. In general we would expect the stored data to be present in the database at a higher frequency than the sample interval, but this will not always be the case. The sampled data values are generated by first determining start and end points for which data is available and which are an integer number of time periods apart, and then for each time sample scanning through the data for the last valid time stamped value before that sampling time. To save processing time when multiple sequential values are required the scanning process is resumed not from the beginning of the linked list, but from the last value used. This method handles properly the case where data values are sparser than sampled points by outputting repeated values where necessary. Where multivariate data is available each of the linked lists is sampled in the same manner, and at the same sampling times to produce synchronous series from probably asynchronous and irregular data. The

start and end times are chosen to be within the intersection of all of the linked lists.

4.1.3 Coping with gaps in the data due to market closure

Some markets close at the end of each working day, some run 24 hours. None run on Saturdays or Sundays. Even in 24 hour markets there are periods when trading is very light. The time series therefore always contains discontinuities. However trading after a break does not start from some brand new position, but always from around the last closing price. The simplest solution to generate a series with which one can make predictions is to concatenate the discontinuous data so as to make one continuous series.

4.1.4 Data normalisation

Many of the supervised learning algorithms generate output values only over some fixed interval. For instance back propagation neural networks with sigmoidal output layers typically produce values in the interval [0,1] or [-1,1] depending on the activation function. So in order to generate predictions in the actual range of the data, the results must be de-normalised. It is also common practice to normalise data into a supervised learning system. If for instance, one input were a share price of say 53.125, and another were a volume of 1,567,458 shares traded, any internal process that took into account both of these values inside a supervised learning algorithm would have to work hard to scale them accordingly. Most algorithms are adaptive and will achieve suitable scaling

in time, but learning will be severely slowed down. It is simpler, and learning proceeds more rapidly if input data is normalised. Normalisation consists of finding the highest and lowest levels in the data supplied, and calculating a scaling value and an offset value:

$$Scale = \frac{1+2\phi}{(\max - \min)} \qquad 4.1$$

$$Offset = \min(1-\phi)$$

where ϕ is a safety margin, typically 0.1.

The safety margin is added to ensure that we can safely represent a prediction outside of the prices historical range.

4.1.5 Quantization of price data

Financial data, for the sake of simplicity, is quoted in fixed intervals. Sometimes these intervals have been chosen historically and do not fit well with digital technology. For instance share prices in America are often quoted in eighths, sixteenths or thirty-seconds. Forex prices are quoted to five figure accuracy. Clearly a predictive product must only generate predictions consisting of the appropriate intervals.

4.1.6 Working with first differences

In many ways the rate of change of a financial time series is more important to financial traders than the series itself. Taking the first difference or generating a series consisting of the difference between

successive time samples in the price series is one way to accentuate the rate of change information.

In spot trading, where some quantity of a commodity is traded, the trader is less interested in the absolute value of the traded commodity than the difference in price from the time he or she bought it. This represents his or her profit or loss, assuming the trade was made purely for speculation, as the vast majority of trades are.

In futures markets this is even more true, where a contract to deliver a commodity is traded without any purchase other than a deposit of "margin money" to cover potential losses. In this case the value of the underlying commodity is of secondary importance.

It is distinctly possible that in modern markets driven by speculation the derivative series, measured as the discrete difference between successive time samples, contains more information than the absolute price series. To be able to take advantage of this possibility the database is configured to be able to return sample to sample differences rather than absolute values. Since we require that data be normalised before input to the supervised learning algorithm, we must save normalisation data as described in 4.1.4 for the differenced series as well.

4.2 Qualitative measurements performed on the data

Before attempting to predict a time series it is useful to have some idea of the predictability of the series, and the likely bounds in time on any predictions made. The following are members of the block "chaos tools" in Figure 14.

4.2.1 Lyapunov Exponent Implementation

This measure as described previously in section 2.2.1 describes how adjacent trajectories on the attractor under study diverge as they are evaluated in time. As a chaotic system evolves no trajectory will ever be the same as a preceding one, this is fundamental to the definition of a chaotic system, the most we can hope for is that a new trajectory behaves something like an adjacent one in a similar part of the phase space. By looking at pairs of trajectories and how they diverge we can estimate how rapidly we will cease to know the exact location of a projected trajectory, and thus the maximum achievable prediction accuracy. The measure calculated in practice is the average of these accuracies over the whole of the attractor, or at least our knowledge of it.

Calculation of the Lyapunov spectrum can be derived analytically where the equations of the process are known, Shimada & Nagashima (Shimada,79), Benettin, Galgani & Strelycin (Benettin,80) are examples. There are several published algorithms for performing this measurement on experimental data. The oldest and most tested algorithm is that by Wolf (Wolf,85), then there are estimates based on the generation of local Jacobian matrices from Eckmann et al. (Eckmann,86), Ellner et al. (Ellner,91) and Conte & Dubois (Conte,88), which have the benefit of calculating more than the first exponent, and finally there are attempts to go from the predictions themselves to calculation of the Lyapunov exponents as in D. J. Wales (Wales,91).

The algorithm of Wolf was selected for implementation because (1) only the first exponent was required, (2) Wolf makes available his FORTRAN code, (3) there is some debate as to the best method for implementing the Jacobian algorithm.

The approach of D. J. Wales was not taken because independent analysis of the major exponent was thought desirable, thus offering the chance of a cross check between the predictive algorithm and this value. As with all the other code used to support this work the language used for coding the Wolf algorithm was C++. The availability of another implementation, even in FORTRAN is invaluable however as a means of cross checking the method and results.

The run time of this algorithm was dramatically reduced by using a binary tree representation for the embedded data thus simplifying the search for neighbouring trajectories.

4.2.2 Hurst exponent implementation

A pseudocode explanation of this algorithm is contained in the appendices.

To form this measurement the data is divided up into equal sections, and the size of the section is decreased until some minimum number of samples per section is hit. In this implementation this is chosen as 10. For each choice of the number of sections starting with two the range and the standard deviation of each section data in each section is measured, and the ratio of range to standard deviation is averaged over the sections.

This value is stored along with the number of sections, the number of sections is incremented and the process repeated. The end result is a set of coordinate pairs. These are logged and linear regression is performed on them. The resulting gradient of the best fit line is the Hurst exponent,

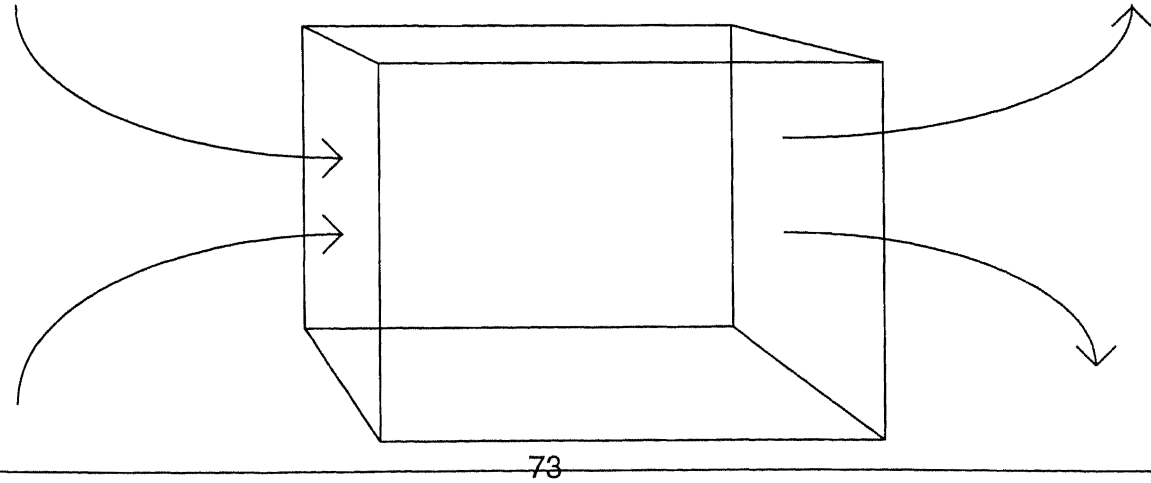
4.2.3 The effects of quantization on qualitative measures

Financial time series are by their nature quantized. There must be some minimum unit in all financial transactions. In every day transactions this unit is the Penny in the UK or the Cent in USA. Similarly share prices are quoted in minimum units, sometimes eighths or thirty-seconds.

Confusingly the term "tick" is used in the financial world both to describe the quantum size and to describe an update in the time series. A tick in this context is the smallest trade-able unit. Quantization is important since in a typical trading day a price may move only hundreds of ticks. During shorter intervals such as 5 minutes price movement may be counted in 5s or 10s.

This quantization is a source of uncertainty and noise to the predictive

Figure 16: Phase space is quantized into hypercubes



process. When financial data is embedded the effect is to divide the phase space up into hypercubes of permitted states.

The analysis of lyapunov exponents is similarly quantized and leads to nearby trajectories being indistinguishable. Since differenced data is used in the analysis the information we have about the price series can be approximated as:

$$Q = \log_2 \left(\frac{\max_{n=0}^{n=p-s}(x_{n+s} - x_n)}{inc} \right) \qquad 4.2$$

where **x** are the time series samples, *s* the chosen separation, *p* the number of samples and *inc* the granularity.

The result is in bits.

An upper bound of the accuracy of the predictions obtainable can be estimated as:

$$Acc = \frac{1}{2^{Q - l_n}} \qquad 4.3$$

where L is the measured lyapunov exponent and n indexes prediction steps.

We cannot directly infer from the above the effect of quantization on the Lyapunov exponent measurement itself, but the algorithm has been modified to keep track of these errors during calculation.

4.3 Embedding Analysis of the data

There are, at the highest level, two tasks involved in generating a time series prediction: finding a suitable model for the data, and finding a suitable method for presenting the data to the modelling algorithm. With the growth of non-parametric modelling techniques the latter problem has been much simplified, the former presents the greater difficulty.

Using the form of time series analysis presented in this thesis there are effectively 3 unknowns, The correct embedding dimension to use for a series, the correct sample separation to use for a series, and the number and type of "helper series" to use.

It is common to find in relevant published work, such as Refenes (Refenes,93) that the method used for finding these three unknowns is pure experiment. Very often the derivation of these parameters is not explained at all, the reader is presented with wonderful predictions but no means of estimating how many months of search were required to find the appropriate parameters. It is possible to automate the process using either a brute force search or a more efficient method such as Genetic Algorithms (Holland,1975). These methods are of necessity time consuming, though undoubtedly effective. Because they rely on no measurable quantity other than the goodness of fit of the final model it is

hard to have much confidence in their robustness. As an example if we were to perform linear regression on a series of data points where we knew the underlying function to be linear we would have high confidence in the results. Indeed confidence figures could be derived from the results. However in attempting time series prediction no real assumptions can be made about the underlying function driving the time series, and if we find a choice of parameters that give good predictions we may have only found some chance alignment of data values, one might call them Ley Lines in the data, that represent features that are not likely to occur again.
Since the model and parameters are effectively trained together it is not certain that the best solution found was not a combination of say a good model and mediocre embedding parameters or vice versa. An analytical method for determining the correct parameters without reference to a model would give the most confidence in the results.

4.3.1 Auto-Mutual Information

The implementation of the above requires the calculation of probabilities, and since we are dealing with real numbers as the symbols, rather than discrete patterns we cannot infer probabilities without some form of modelling and interpolation. If calculations were confined to financial time series with their quantised nature, see 4.1.5, this would perhaps not be necessary, especially if first differences were used. It was deemed desirable however to enable the time series predictor to be used with any time series.

The method used is due to Kennel, [Kennel 93] and rather than using a histogram based approach, which he shows to have drawbacks due to the problems associated with finding an adequate bin size, uses Kernel Density Estimators to calculate the probabilities. The kernel function used is the Epanechnikov rather than the Gaussian since the latter has infinite support and would obviate the efficiency gains from using binary trees. Each kernel function is scaled by a factor termed the bandwidth that scales the receptive area of the function. The algorithm performs an initial pass through the data to scale these bandwidths and thus ensure that the effective bin sizes are large where data is sparse and small where data is plentiful. Unlike the histogram approach, where a pre-defined number of bins chosen to be some factor smaller than the number of points are used, here there is one bin, or receptive field per data point, and the density and thus the probability is calculated by determining the distances to the nearest neighbours.

The implementation of this algorithm was performed by the author based on Kennel's published work. A more complete description in pseudocode form is given in the appendices.

4.3.1.1 Binary Trees

This algorithm and several others in this work use tree structures to represent the data points in a time series so that a search for the neighbours to a point can be performed efficiently. Without such a representation one could expect processing time to be proportional to N^2 where N is the number of data points in a series. With a binary tree

representation processing approximates to log(N). The purpose is to sort data into binary trees so that finding points and neighbours to points requires only a binary search. The algorithm is extended to cope with vectors as well as scalars so that multivariate data can be used. The implementation of these trees is derived from standard binary tree algorithms much modified by the author. The purpose is to sort multidimensional vectors into groups based on their distance from each other. Distance is measured using the Euclidean metric. Rather than moving data during the sort process an array of indices is generated. It is these that are sorted. The algorithm sets out to find cut points that approximately balance the number of points either side of the cut. This is done by calculating the mean and variance for each column of data. The column with the largest variance is selected as the *discriminator* and the mean of that column as the discrimination value.

A binary tree node is generated and the contents of the index array are rearranged so that those with values in the discriminator column greater than the discriminator value go to one side of a point in the array, and the rest the other. We have now two sections of the array and a new binary tree node is generated for each sub section selecting a new discriminator column and value. By this process the array is sub divided again and again till the members of a sub division of the array are fewer than some arbitrary number, say 20. The above can be formulated as a recursive procedure. When used with financial data which are traded in fixed quanta, identical points frequently occur, and the variance in a sub set can

be zero leading to numerical instability. The solution found by the author was to split the sub sets equally in this case.

This tree structure can be used easily to find a number of near neighbour points for a given vector.

Again using a recursive procedure the tree is followed, and at each node the discriminator column and value are used to determine on which branch the neighbour values lie. When finally a leaf node is found the Euclidean distance between each member to the given vector is calculated, and the indexes of the n closest placed in an array, where n is the number of neighbours required. If this exceeds the number of members of this leaf node then neighbouring leaves are searched also.

A pseudocode version of the above giving more detail may be found in the appendices.

4.3.1.2 Separation from AMI

Auto-Mutual information is performed by preparing two copies of a time series and rotating the data in one of them along by n samples. This introduces a discontinuity in the shifted data but should not be problematic if n is small compared to the number of data points.

The two vectors are applied to the Mutual information algorithm described above and the values of MI recorded for a range of trial values of n.

The separation is derived from the first minimum in the resulting curve of MI against n.

As can be seen from the results in 5.2 The curve of MI against n "bounces" with increasing n with a comb like structure. The first minimum is to be preferred to any other since it must result in the most succinct model formulation.

The algorithm derived to find this first minimum is as follows:

The auto mutual information for the given series at increasing offsets is calculated. At each stage the minimum and maximum values so far are calculated. If the current offset is the new low the maximum value is reset to that of the new low. When the processing is finished, that is that all offsets up to 17 have been tried, the point at which the low occurred is examined. If this is the last offset then the AMI graph was continually decreasing, and we can not find a minimum. If not the subsequent maximum is inspected. If this exceeds 1.1 times the minimum, then a winner is declared, and the offset at which that minimum occurred is selected as the separation distance, otherwise it is assumed that a minimum can not be found. Figure 28 is an example of a successful discovery of a valid separation value, Figure 32 is an example where this method failed to find such a value. Under these circumstances an exact choice of separation is not deemed to be critical and a separation of 1 is used.

4.3.2 False Nearest Neighbours

The implementation is fully described in pseudocode in the appendices, an overview of the algorithm will be given here. The idea is due to Kennel, but the implementation and development of the algorithm is due to the author.

The algorithm proceeds as follows:

A trial embedding with dimension 2, and separation as found in section 4.3.1.2 is performed on the data giving rise to a number of embedded vectors. These are applied to a binary tree as described in 4.3.1.1. For each vector the nearest neighbour is found and the distance between them noted. Then the embedding of both points is increased by one by finding the appropriate extra values from the database. The distance between the two extended embedding vectors is calculated and the ratio between the standard and extended vectors distances is calculated.

If the distance has increased by more than 5 a false neighbour is declared. The percentage of false neighbours for trial embeddings with increasing dimension is calculated, and the gradient between successive embedding dimension choices is calculated. As soon as this goes above -0.1 the preceding embedding dimension is selected.

The above method was developed by the author rather than using a simple threshold because the presence of noise in the time series, looking to all the world like higher dimensional chaos, results in a background level of false neighbours, thus pushing the curve up and

making the derivative a more sure indication of saturation. The constants given above are empirically derived, but looking at the graphs in Figure 29, Figure 31 and Figure 35 will show that the method is insensitive to an exact choice of these constants.

4.4 Training pattern generation

Takens' theorem (Takens 81) states that a chaotic process can be predicted by a smooth function if properly embedded. Supervised Learning algorithms are used in this work to generate these smooth functions from the time series. These learnt functions are mappings from input data provided as tuples to one or more predictions of the future value of the time series.

The process of embedding a single time series is as follows: The scalar series is converted to a series of vectors.

$$\mathbf{X}_t = x_t, x_{d+t}, x_{2d+t}, x_{3d+t}, \cdots x_{nd+t}$$

Where d is the *Separation*, and n the *embedding dimension.*

In practice, in order to perform predictions for a given time series, first the separation is calculated, then the embedding dimension (this is so that the separation calculated can be used in the optimal dimension calculations). Finally the embedded vectors are used as a training set to some example of a supervised learning algorithm, along with some future value of the time series representing the point to be predicted. (in the above nomenclature I use x_{nd+t+1}.)

The training set thus contains pairs of embedded vectors drawn from the past history of the series on the stimulus or input side, and subsequent values of the series on the target or output side.

A multivariate version of the above can be obtained by concatenating the input side, and separately the output side, of several suitably embedded time series. In this work a sample time was selected in order to give predictions of the required granularity and the database was required to supply samples with date stamps between two user selected times with the appropriate sampling interval. These were then converted into embedded vectors by inserting them into a First In First Out data structure tapped at the appropriate points.

The FIFO passes each scalar value input into the leftmost cell and the values contained in the cells are all moved along to the right. In the above example new sample scalar data must be presented 14 times before the fifo is full. Each subsequent insertion generates a new training vector and target scalar value. In general given p scalar values in the data base

p - d * n training vectors can be generated.

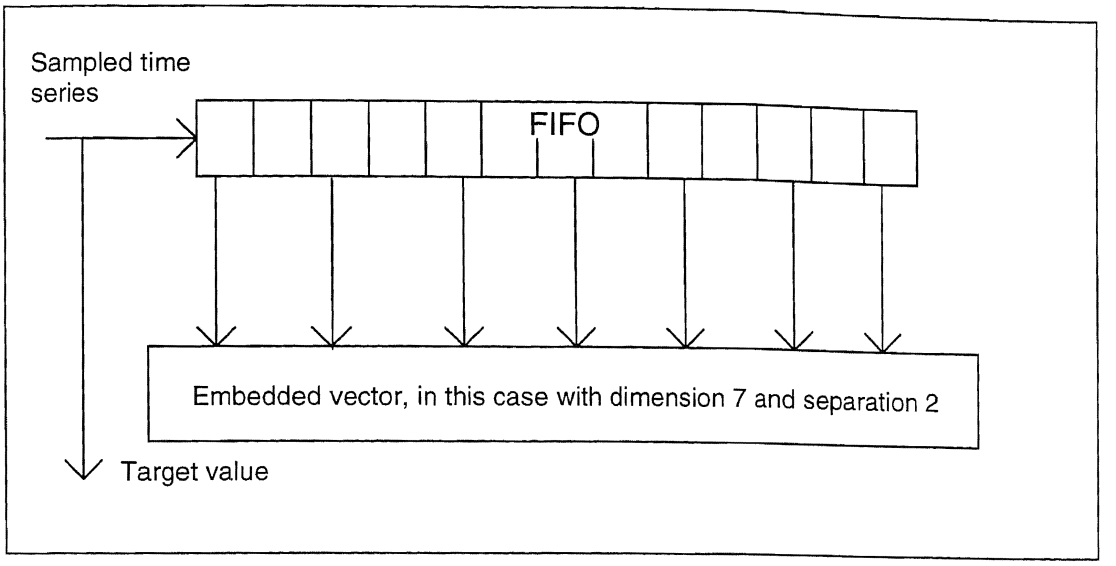


Figure 17, Embedding a time series using a FIFO

Multivariate prediction was obtained by maintaining separate databases for each input series, selecting embedding parameters for that series, embedding them appropriately and concatenating the input vectors and output scalars. Great care was taken to ensure that the sampling intervals were the same for each series and that the vectors were completely synchronised.

4.5 Supervised learning of the training patterns

In the supervised learning stage the learning algorithms were presented with the training set generated by the embedding process. The embedded data is normalised as described in 4.1.4 before being applied to the algorithm.

4.5.1 Local approximation algorithm

The algorithm used is inspired by an article by Sugihara and May (Sugihara,90), which did not define the precise algorithm they used to make predictions. The algorithm and implementation are therefore the work of the author.

The algorithm is relatively simple. The time series is embedded using the methods discussed in 4.3.1 and 4.3.2. The a target column vector of the next sample is also supplied. A binary tree is formed from the embedded data, and the tree is stored. This completes the learning phase of the algorithm. To make a prediction from a similarly embedded predictee vector the n+1 nearest neighbours are sought in the binary tree, where n is the selected embedding dimension. The idea behind this is that the simplest figure that can be created in n dimensions, a simplex, must have n + 1 vertices. The selected points will hopefully enclose the predictee, although there is no guarantee of this. An exponential interpolation scheme is used to weight the nearest neighbours according to Euclidean distance from the predictee. Finally these weights are used to interpolate between the various outcomes or target values corresponding to each of the nearest neighbour vectors.

The interpolation method is described by pseudocode in the appendices. This prediction method is fundamentally interpolative. It does not cope well with predictee points outside of the attractor. However if a good picture of the attractor has been built up these should be relatively rare.

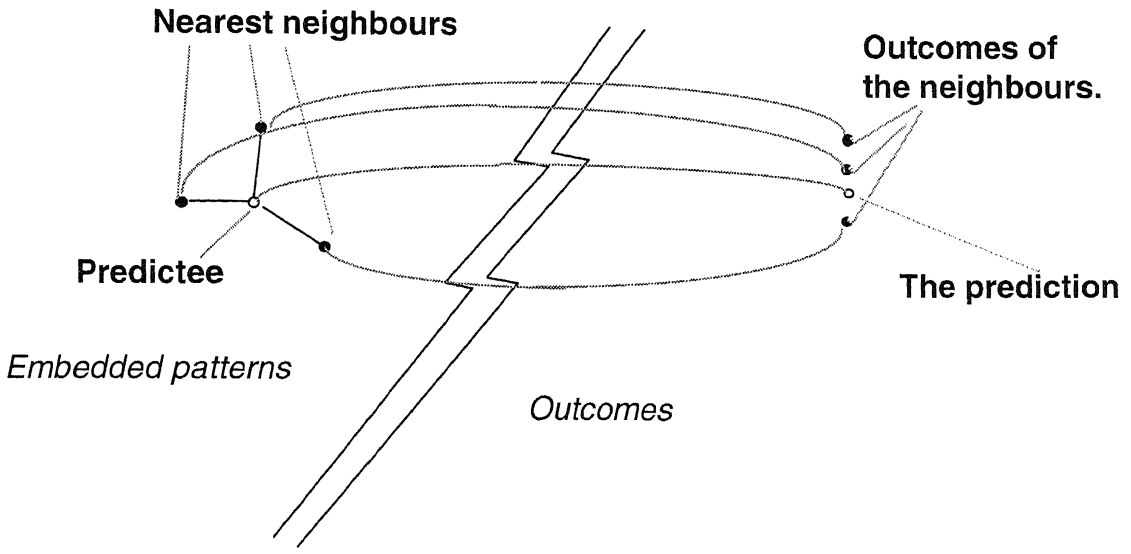


Figure 18, Local approximation interpolation

4.6 Conversion and formatting of predictions

Predictions are output by the supervised learning algorithm with the same scaling as that used for the target vector, and potentially in a differenced format. The predictions must be reconstructed by inverting the normalisation process, and if differencing was used the denormalised change must be added to the last in-sample value to achieve a prediction. If the prediction were of a quantized financial time series then the prediction would be rounded to the nearest quantum.

4.7 Iterated Predictions

If more than one prediction step is required, i.e. if a sequence of prediction steps are needed, there are two potential ways to achieve this. Either one could train several supervised learning algorithms with their target vectors suitably offset, or one can train a single supervised learning algorithm and use it repeatedly with some form of feedback.

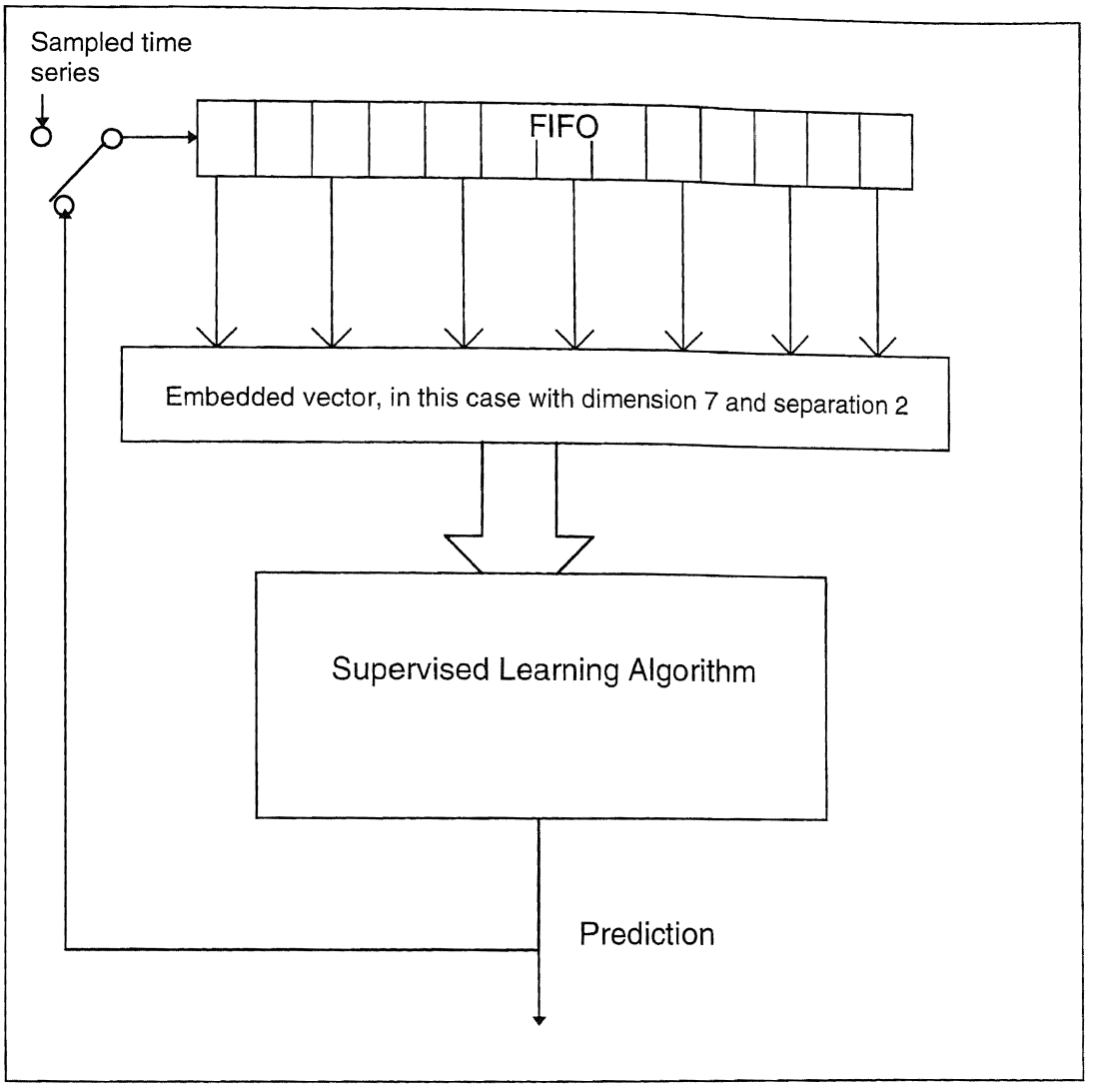


Figure 19, Achieving Iterated predictions using feedback.

Farmer (Farmer 88) showed that the latter method produced superior results. It is also easier to implement. As the iterated predictions are generated more and more of the contents of the FIFO are filled with predicted data till finally predictions are being generated solely on the basis of previous predictions. Impractical as this may sound in Figure 26 good predictions are still generated after 10 iterations given a FIFO length of 8.

4.8 Evaluation of predictions

Clearly a mechanism for determining the quality of the predictions is important. Largely the choice of metrics is dependant on the use to which the prediction is to be put. The following metrics were calculated by the program for each prediction run.

4.8.1 Validation data

The choice of validation data is fraught with problems. If the purpose of the use of the validation data is to prove general predictability, then a large validation period situated anywhere in the time series is adequate. If however one wants to make both useful predictions and prove the concept, as would be useful in a financial application, then the validation period must be chosen carefully. Validation data can not be training data, and thus if data is scarce a large validation set eats into a precious resource. If, as is believed with financial data, there is non-stationarity and mode changes occurring in the data, the best place for the validation set is using the most recent data. However using a large validation set at the end of the data implies that the model is trained on old rather than fresh data, and one may see the models best performance in the validation set, and poor performance in practical use thereafter. Using a small validation set risks using a model that has been evaluated on potentially atypical data. The general solution chosen by default in this time series predictor is to use a validation set that consists of 10% of the data situated between 80% and 90% of the training data. This was considered a good compromise.

4.8.2 Metrics

The most obvious measure of accuracy of predictions is sum squared error, evaluated as:

Sum over i of $(\text{Prediction}_i - \text{Actual}_i)^2$

over the validation set.

The correlation coefficient between the predicted and actual series is measured for each block of predictions, as is the hit rate, defined as the percentage of directional changes that were correctly predicted.

4.8.3 Similar day Information

As an interesting side effect of the local approximation algorithm one can graphically display the nearest neighbours used to form the simplex on a plot of the time series. This information is qualitative and anecdotal in the extreme, but it is interesting to see in the example below that four out of 5 of the neighbours illustrated occur just before a large drop in the market.

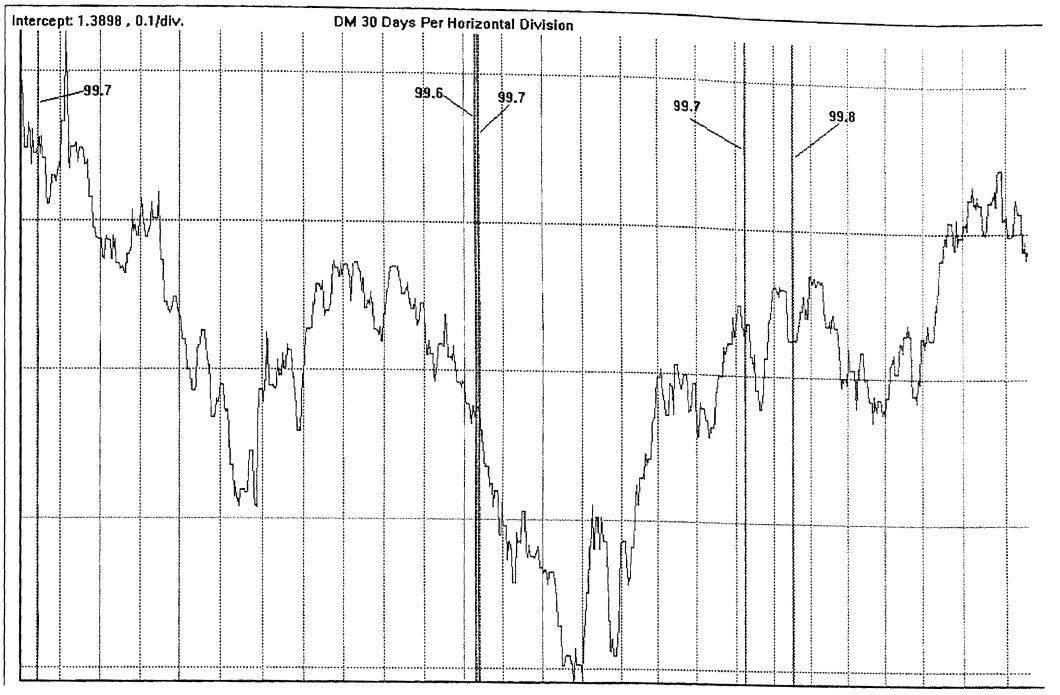


Figure 20, Graphic representation of similar day data

4.9 Performance of the predictor on artificial time series.

It is important to compare the performance of the time series predictor with other published work, yet this is surprisingly difficult to do. Such is the abundance of financial data that no two researchers seem to use the same series. Then there is the complication of sampling rate and start and end times to make it very hard to compare the performance of two predictors. The Biotechnological data that is analysed in chapter 6 has not yet been made available to other researchers, leaving only artificial time series as a method of performance comparison.

Of all the time series described in chapter 2 the most popular is the Mackey glass series. This has been extensively used as a bench mark by

Farmer & Sidorowich (Farmer, 87), Cadagli (Casdagli,89), Moody (Moody,89), Watkins et al (Watkins,94), and Junhong Nie (Nie,94) amongst others. While this is a commonly used bench mark the length of data used, and the offset to the predicted value, vary as do the parameters used in the series generation. The error measure used also varies, the two most popular being r.m.s. error and normalised r.m.s. Moody and Farmer achieved a normalised r.m.s of 0.012 using neural networks of different kinds. None of the other results seem to improve on this. As can be seen the result due to the techniques described in this work is 0.00109. Due to the concerns expressed above it is hard to be sure that like is being compared to like, but it can be fairly said that the performance of this predictor is comparable to if not better than the predictors described in the above papers.

In each case 2000 training and 1000 test points were generated. The predictor was set to predict the next value in the series. Hit Rate represents the percentage of correct predictions purely in terms of direction, and the normalised r.m.s. error being the r.m.s error divided by the standard deviation of the time series.

Series	Separation	Dimension	Hit-Rate	norm R.M.S.
Logistic	1	2	99.3	0.0053
Tent	1	2	99.4	0.0031
Mackey-Glass	11	3	93.8	0.0011

5. Performance of the Predictor In financial applications

In this chapter the experimental results are presented showing the performance of the predictor on financial data. In order to show that the embedding and separation values found analytically are near optimal sensitivity analysis is employed and results are presented of this analysis.

5.1 Foreign exchange data

Five time series were used to test the predictor, sampled at one minute intervals. They were the spot rates of the DM against the $, the £ against the $, and the 3 month deposit rate for the £,$ and DM.

The samples were taken over a single days trading. The third possibility, the DM against the £, is not included, since we can expect this to keep in line with the other two. These series are widely considered to have strong chaotic components (Medio 92). The graphs of each of these follow:

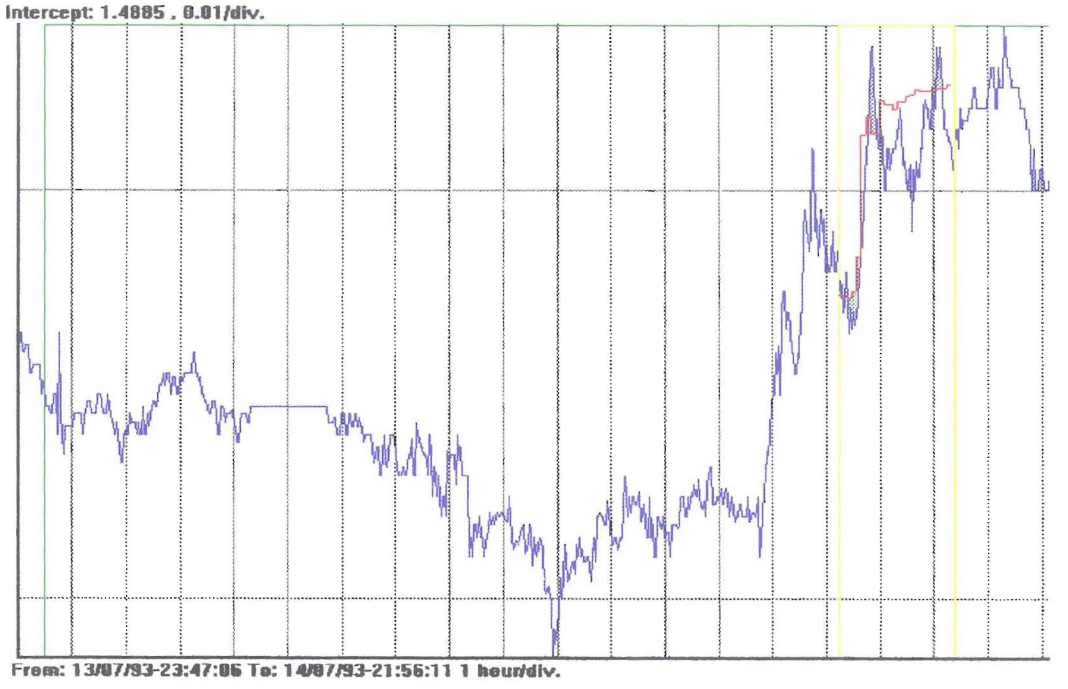


Figure 21: The DM/$

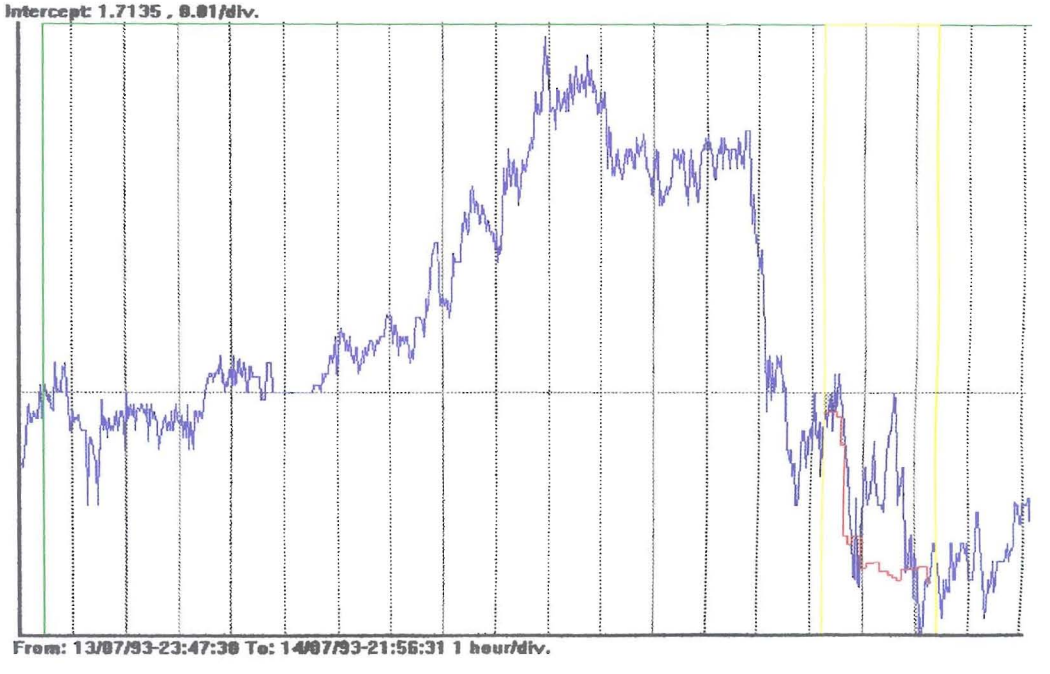


Figure 22: The £/$

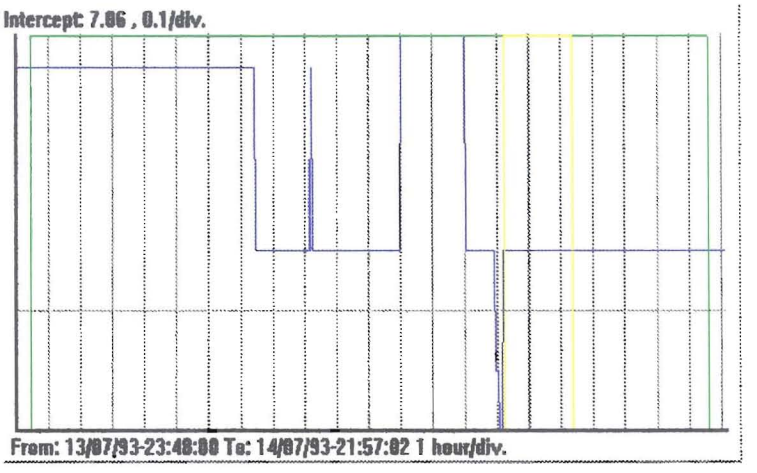


Figure 23, 3 Month Deposit Rate, DM

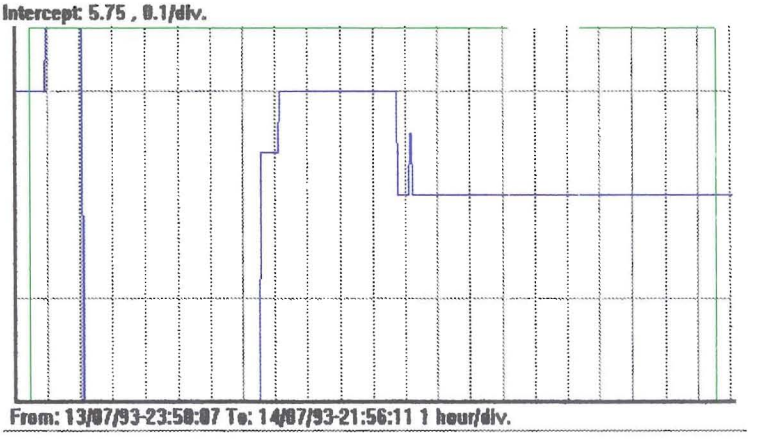


Figure 24, 3 month deposit rate, £

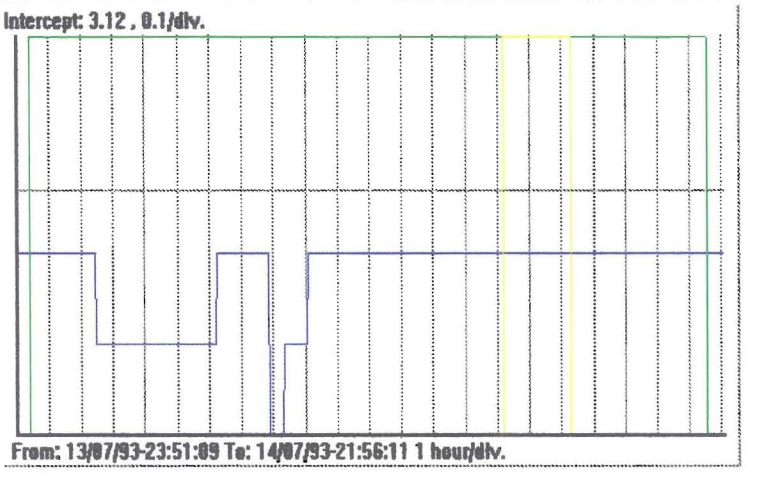


Figure 25, 3 Month Deposit Rate, $

The program used to make the evaluations was configured to predict both the £ and the DM in 5 minute intervals for a period of two hours. The predictions were iterated, that is the later predictions were based on the previous ones, not on the actual data. The graphs below show the predictions and the actual behaviour on the same graph. The predicted line is the more "steppy" of the two.

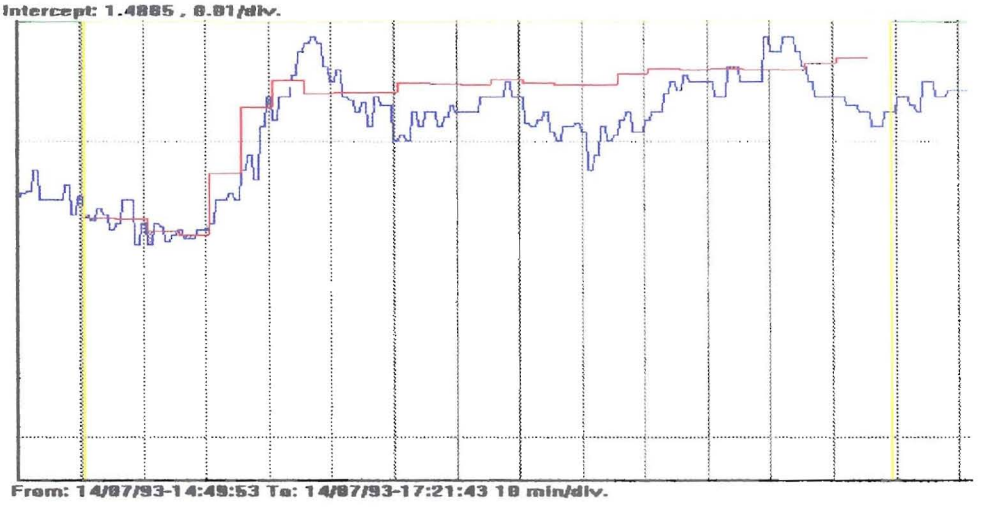


Figure 26, DM/$ Prediction for 2 Hours in 5 Minute increments

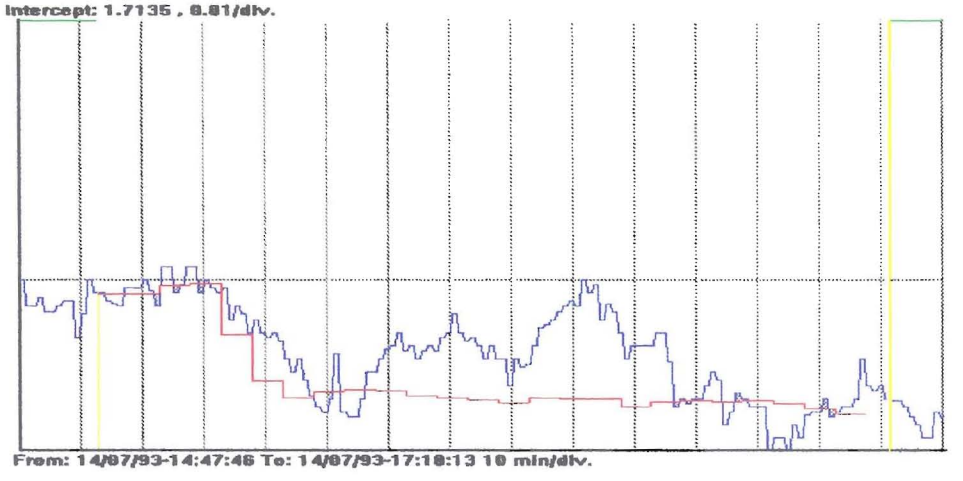


Figure 27, £/$ Prediction for 2 Hours in 5 Minute increments

5.2 False nearest neighbour and auto-Mutual Information responses

The following show the plots of FNN (false nearest neighbour percentage) and AMI (in bits) for different embedding or separation values for each time series:

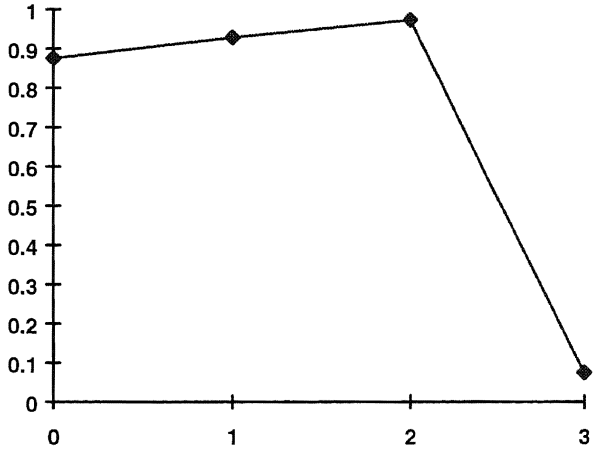


Figure 28, DM/$ Auto Mutual Information at different separations

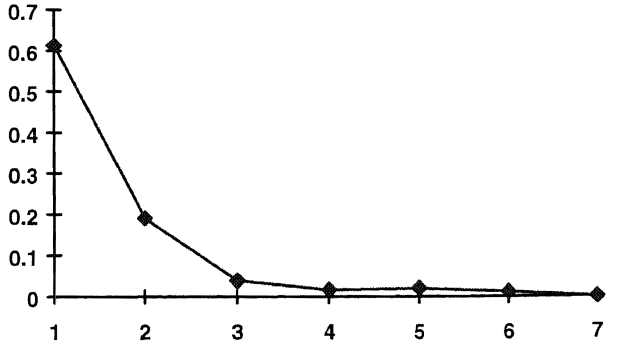


Figure 29, DM/$ False Nearest Neighbours at different dimensions

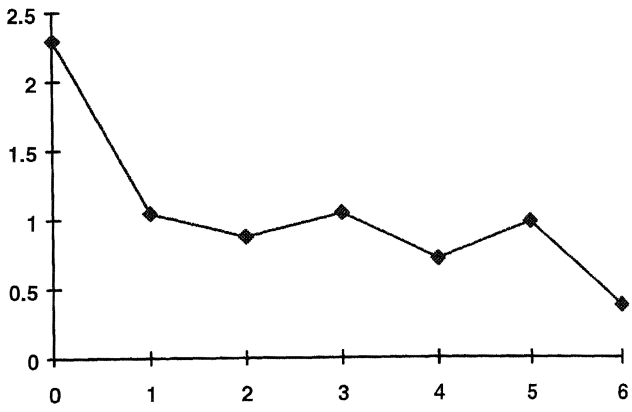


Figure 30, £/$ Auto Mutual Information at different separations

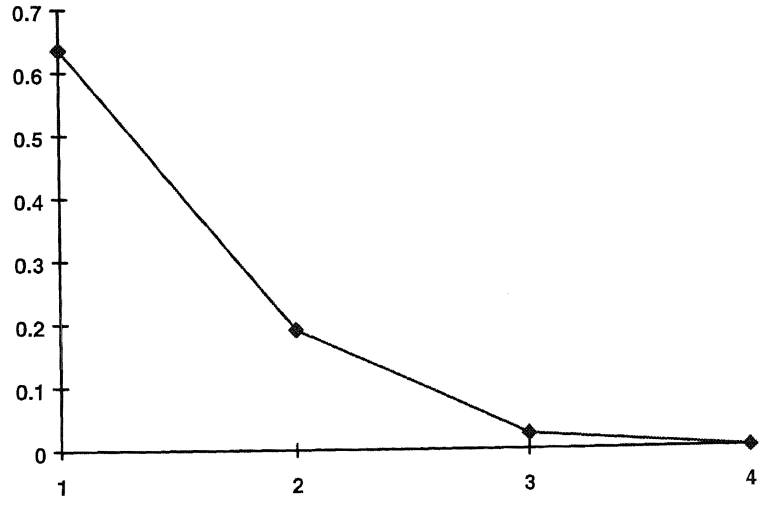


Figure 31, £/$ False nearest neighbours

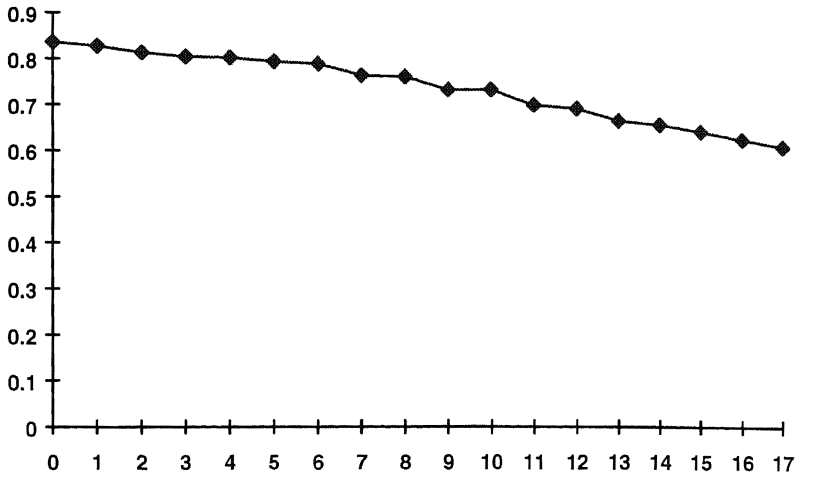


Figure 32, 3 month DM deposit Auto mutual information

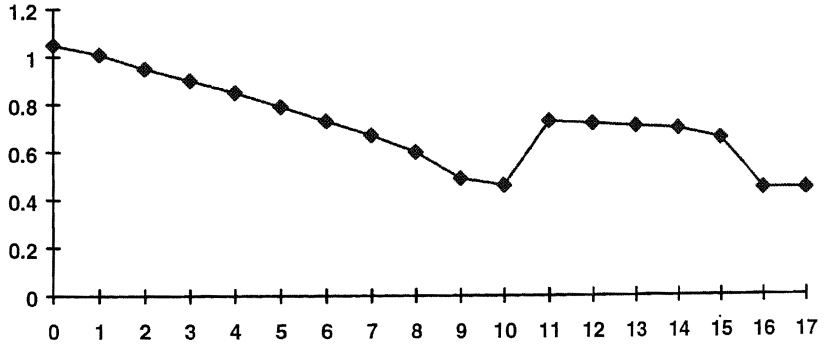


Figure 33, 3 month £ deposit Auto Mutual Information

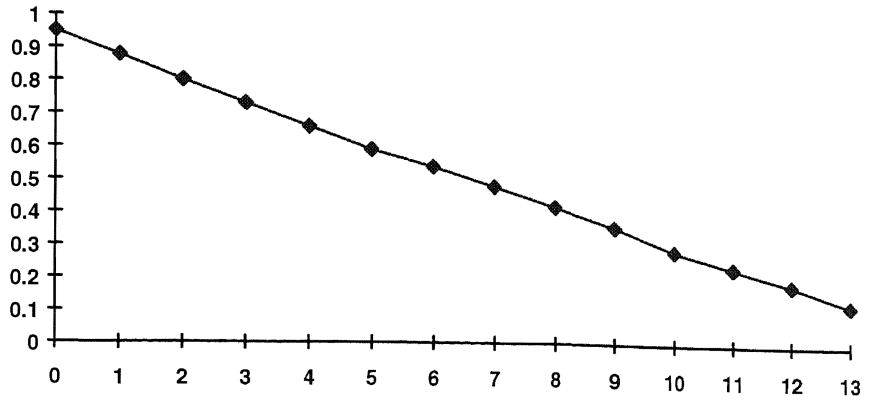


Figure 34,3 month $ deposit Auto Mutual Information

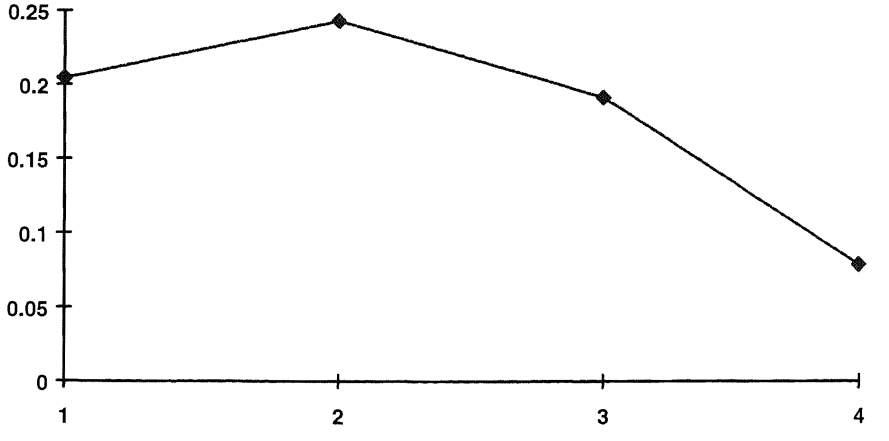


Figure 35, 3 month $ deposit False Nearest Neighbours

The difference between the frequently traded DM and £ and the deposit rates should be clear; the deposit rates have very little variation. The program used to generate AMI and FNN does not go on to perform FNN calculations unless the AMI value is resolved. In two out of the three

deposit series the AMI value did not dip sufficiently to be resolved. Looking at the series and the information in them, this is not surprising. Although predictable behaviour may be visible when viewed over days and weeks, at this resolution there is little variation in the data. This is not to say there are no dependencies between the spot rates and the deposit rates, merely that there is no need for embedding to represent these dependencies. The embedding dimension and separation for these series were set to 1 as recommended by Abarbanel.

5.3 Sensitivity Measurements for False Nearest Neighbours and Mutual Information

The software used to make these analyses, is limited to a maximum dimension and separation of 16. To exhaustively search for optimum embedding dimensions and separations in combination would require up to 256 trials. The somewhat simpler process of perturbing the optima obtained by our existing methods and observing the effects was used. The sequence was as follows:

1) The Auto Mutual Information and False Nearest Neighbour algorithms were used for each time series.

2) The embedding dimensions and separations suggested were used.

3) Training patterns were generated.

4) The Local approximation algorithm was used to perform predictions.

5) A separate validation period was selected to test the predictive power.

6) Using iterated predictions the performance on predicting the validation set was calculated by comparison to the actual values over the validation period.

7) The R.M.S. error was calculated and subtracted from 1.0, to render a measure of fit.

8) For each helper time series the separation and dimension were perturbed by +/- 1 and +/- 2. Then (2),(3),(4), & (6) where used to evaluate the performance of the whole system.

The results are shown:

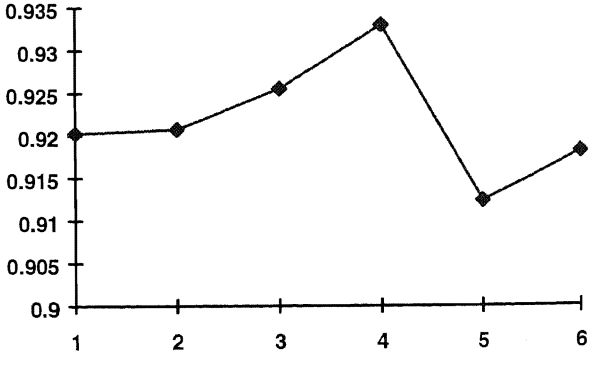


Figure 36, Effects of perturbing DM/$ embedding Dimension

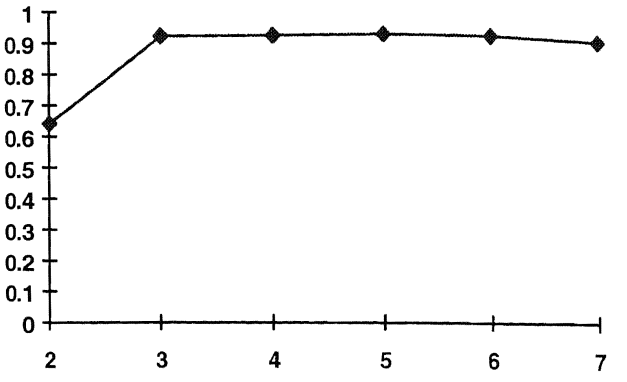


Figure 37, Effects of perturbing DM/$ Separation

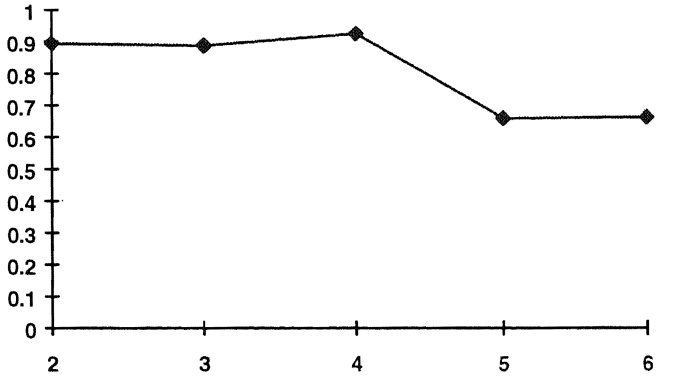


Figure 38, Effects of perturbing £/$ Dimension

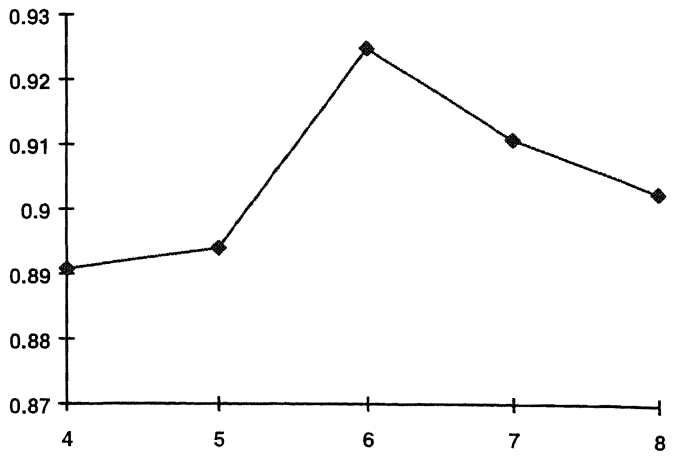
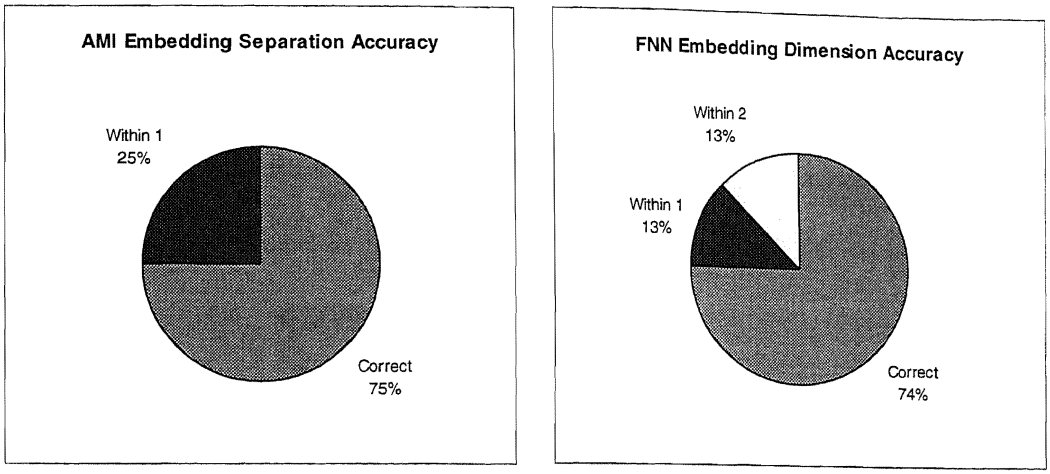


Figure 39, effects of perturbing £/$ D Separation

In order to further test the efficacy of these methods various other currency pairs were tried. In each case the rate against the American Dollar was used. The Currencies were Canadian Dollar and Swiss Frank, Hong Kong Dollar and Japanese Yen, Dutch Guilder and Deutschmark, Irish Punt and Japanese Yen, Swiss Frank and Japanese Yen, Cypriot pound and Greek Drachma, Belgian Frank and Deutschmark, Italian Lire and Deutschmark. The results for perturbation of the dimension gave the peak observed performance 75% of the trials, within 1, 12.5%, and within 2, 12.5%. For separation they gave the peak observed performance 75% of the trials, and were within 1 for the remaining 25%.

Figure 40, Analysis of performance of FNN & AMI on various series



5.4 Discussion of predictability of financial series

The reasons for attempting to predict financial time series are the rewards to be gained, and the interest that generates, and the challenge of attempting to predict the most difficult of series.

Unlike the simpler, though still difficult, processes of blind matter, as described in the next chapter, financial market prices are generated by the interaction of hundreds of thousands of intelligent and highly motivated people, each with many months or years of experience spent observing the markets trying his or her best to guess the next movement, the next trend, the next wave.

That there is any predictability in the markets at all is a marvel. The Efficient Market Hypothesis (Fama, 65), which held sway until the early 90s suggested that the time progression of the markets would appear as a random walk. The explanation for this that is often given is that a trading opportunity, like a ten pound note lying on a pavement, would be instantly snapped up. The argument runs that there would therefore be no sense of history in the behaviour of the markets. All the information that would be of

any use to a trader was present in the instantaneous state of the market and common knowledge. A further justification for this point of view is found in measuring the serial correlation of major financial time series. It has been shown by LeBaron, (LeBaron 96) that the correlation of The Dow Jones index returns from 1901- 1995 with a copy of itself lagged by one day is 0.04. Implying, if correlation can be used as a measure of a process without a normal distribution, that the markets are memory less. Interestingly LeBaron also shows that if you square the returns, thus emphasising the effect of larger changes, the correlation of the above shoots up to 0.22. Taking the simple magnitude of the returns produces 0.29. Similar evidence is found in the S&P.

The evidence of low dimensional chaos in many markets, and thus limited predictability (Medio,92), has undermined the efficient market hypothesis in some eyes, and a new theory, the Coherent Market Hypothesis (Vaga ,90), has been proposed that takes these discoveries of chaos and non-linearity in financial series into account. There is an ongoing battle between these rival theories and variations of them, and no clear winner has emerged.

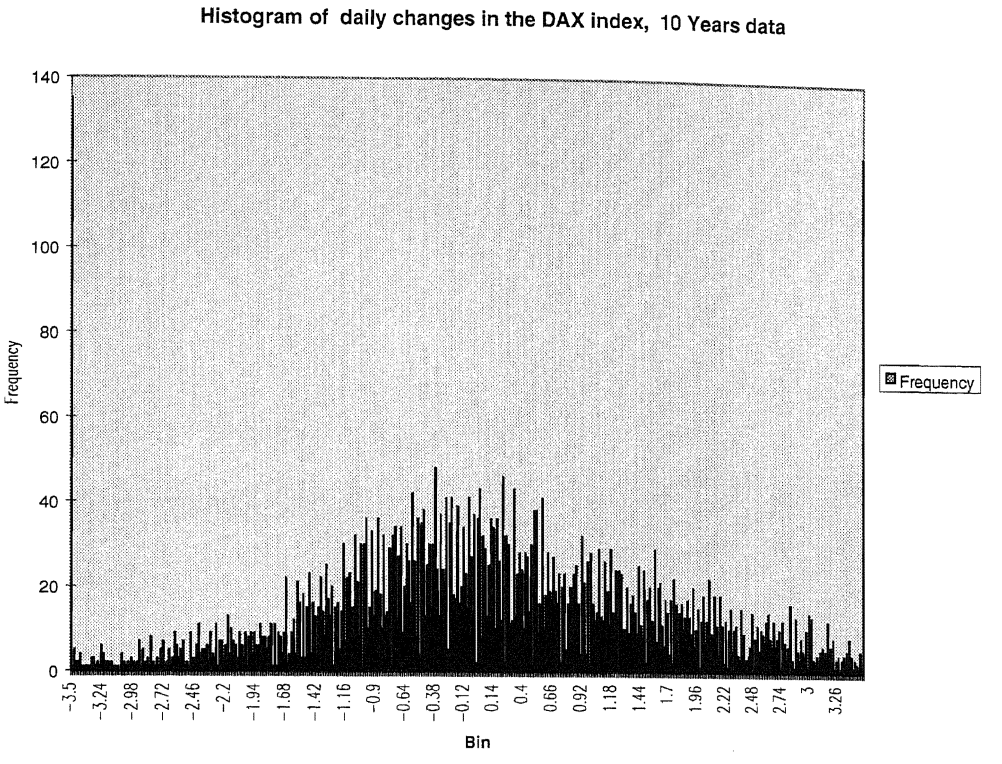
5.5 Discussion of the relevance of the results and implied models of the markets.

Various results, some with academic verification and many without, have been generated in recent years that suggest predictability of the markets (Peters,91), and sometimes offer metrics that support their case. Financial software vendors claim that they can predict some carefully chosen

financial instrument, and that the predictions are accurate to within say 90% over a period such as six months. This choice of metric is meaningless; except at times of financial crisis most international instruments such as exchange rates or market indices do not fluctuate outside of 10% bands for very long periods. One could easily do as well with a first order trivial predictor. More honest workers in this field use "hit rate" as a measure. This is the percentage of correct predictions that the predictive system makes of the change in price over some period, using presumably out of sample data. Now one would expect the proportion of these changes to tend towards 50% each way.

For instance for the last two years of Dax data, sampled every half hour (5524 data points) the ratio of up to down changes is 0.528

Now if we look at the distribution of these daily changes for the Dax



we can see an approximately Gaussian distribution with pronounced leptokurtosis.

If a user of the output of a predictive algorithm uses the information to trade in the chosen market, by buying the instrument when a positive change is predicted and "shorting" or borrowing the instrument when a negative change is predicted he or she will incur trading charges. These are dependant on the instrument, volume traded, and the status of the trader, in some markets these charges are symmetrical, in others "shorting" is more expensive. In either case one can draw vertical lines on the distribution graph corresponding to these charges, and shade the area between them that represents unprofitable trades. Depending on all the variables the shaded area can easily exceed 10% of the total, and of course the smaller and less well connected the trader is the greater this

area is likely to be. Thus if the measure of utility of a predictive tool is that it makes money for its user, a large percentage of the predictions it gives must lie outside this shaded area. Typically hit rates of 65% are considered good, since we would expect a 2nd order trivial predictor to return 50%.

It is clear from the above that a predictive system could score 65% hit rate and still lose money, and conversely a 40% successful predictor could be very profitable.

In work separate from this thesis, described in outline in (Edmonds,95b), both these circumstances have been observed.

It is not the intention of this thesis to imply that the interesting characteristics of limited predictability discerned in the financial time series used by these methods can be used to make money.

6. Performance of the predictor in a biotechnology application

In early 1994 the opportunity arose to test these methodologies on an application outside of the world of financial forecasting. Hazel & Christopher Davey of the University of Wales at Aberystwyth had generated a time series in the process of running a biological reactor over many hundreds of hours that seemed to show evidence of chaotic behaviour. Their superior, Professor Douglas Kell, asked me to use the techniques detailed in this thesis to analyse the series.

6.1 Background to the analysis.

The following is my understanding of the processes involved in the generation of the time series drawn from the paper that documented the work (Davey.96).

6.1.1 Measurement of Biomass

Prof. Kell had designed several years previously a device called a biomass monitor, that uses radio-frequency radiation to measure the capacitance of living cells in a suspension. It has been shown by (Harris,87),(Kell,87),and (Davey,93) that there is a characteristic capacitance curve as shown in figure 6 for a given concentration of living cells.

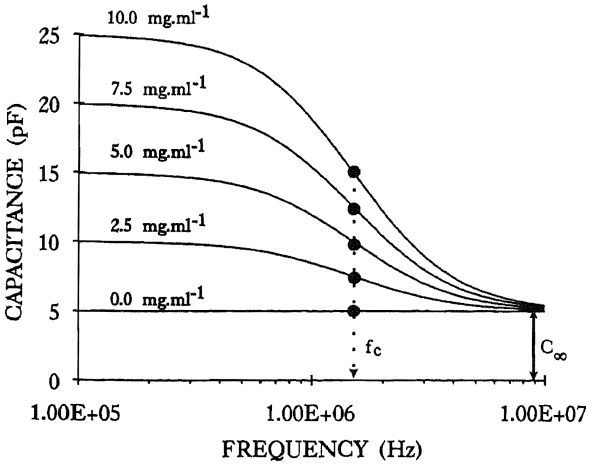


Figure 41 capacitance of various biomass concentrations against frequency

A half way point as shown can be calculated by scanning over various frequencies and this is linearly proportional to the concentration of living cells in the sample. The source of the capacitance in the cell suspension is the non-conducting outer membrane of the cell wall.

Dead cells do not have complete membranes, and therefore do not have measurable capacitance. The biomass monitor is an accurate measuring tool for determining the amount of living material in a vessel, and is unaffected by pollutants.

6.1.2 Culturing Yeast in a fermentor

Baker's Yeast was grown in a 1 Litre vessel that had a biomass monitor probe inserted. The contents of the vessel were aerated with pumped air and stirred at 450 r.p.m.. The biomass was continually monitored and a PC, data collection hardware and some specially written software were used to control a pump that added nutrient to the vessel. The software was configured to regulate the size of the biomass by running the pump at a variable rate and therefore adding more nutrient whenever the size appeared to drop, or less whenever it threatened to exceed the required size. Using this set up the biomass in the vessel could be held constant over several months.

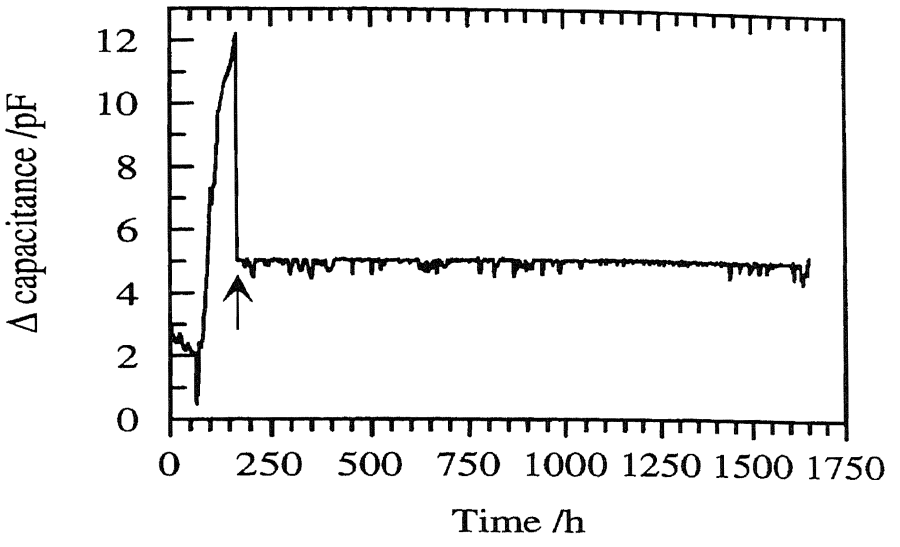


Figure 42 Measured capacitance during the experiment

Figure 42 Measured capacitance during the experimentFigure 42 shows the capacitance, and hence the biomass, measured by the bug meter over 1,400 hours.

The arrow marks the point at which the control system was switched on.

6.2 The data supplied

Clearly there is not very much of great interest in the data in Figure 42. However the real surprise was the behaviour of the pump stimulus over the same period. For a set-point of 5pF, large and seemingly random changes were detected in the pump output as shown in figures 8,9,10

which represent the data from hours 200 to 1400, split into equal thirds. These should be compared to Figure 46 which shows the pump demand for a lower 4pF set point at which this non-linear behaviour did not occur. The very large fluctuations in the data detected were a novel phenomena. Hitherto it had been assumed that the yeast cells would be equally distributed through the various stages of a cells life at any one time, and thus demand should be constant or gently rising as in Figure 46. Figure 43 to Figure 45 imply that dramatic changes occurred to the population over time when the concentration was increased over some critical threshold.

My task was to analyse this data and to try to extract whatever information my techniques could find, and to attempt to predict the data so as to hopefully show a simple deterministic cause for the strange behaviour seen.

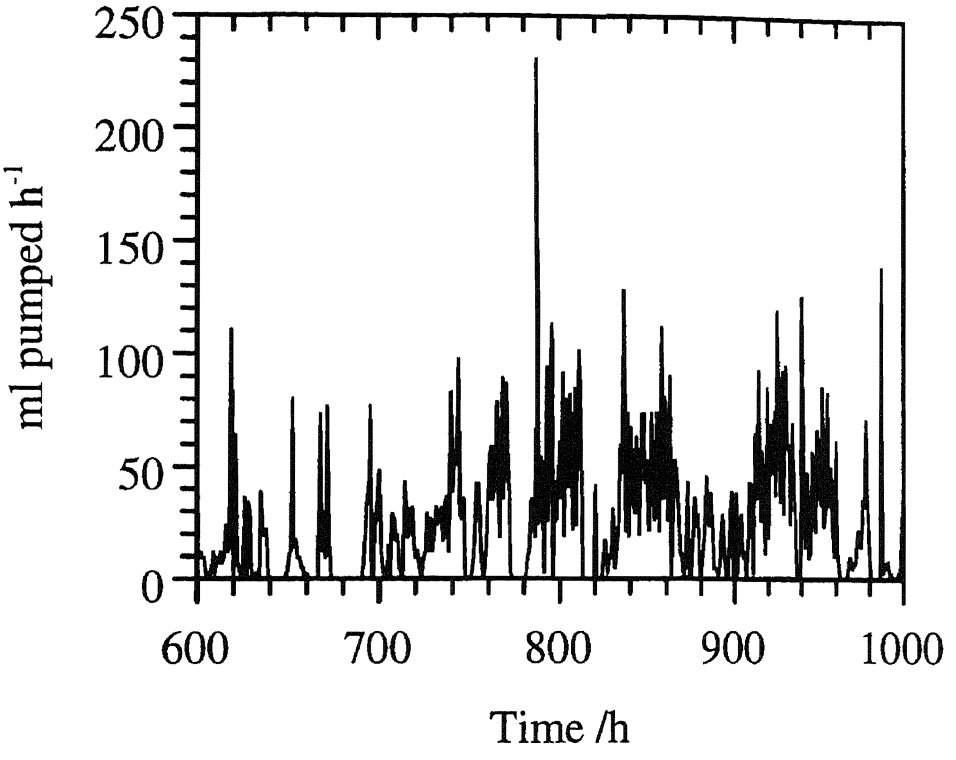


Figure 43 Nutrient feed for hours 600 - 1,000.

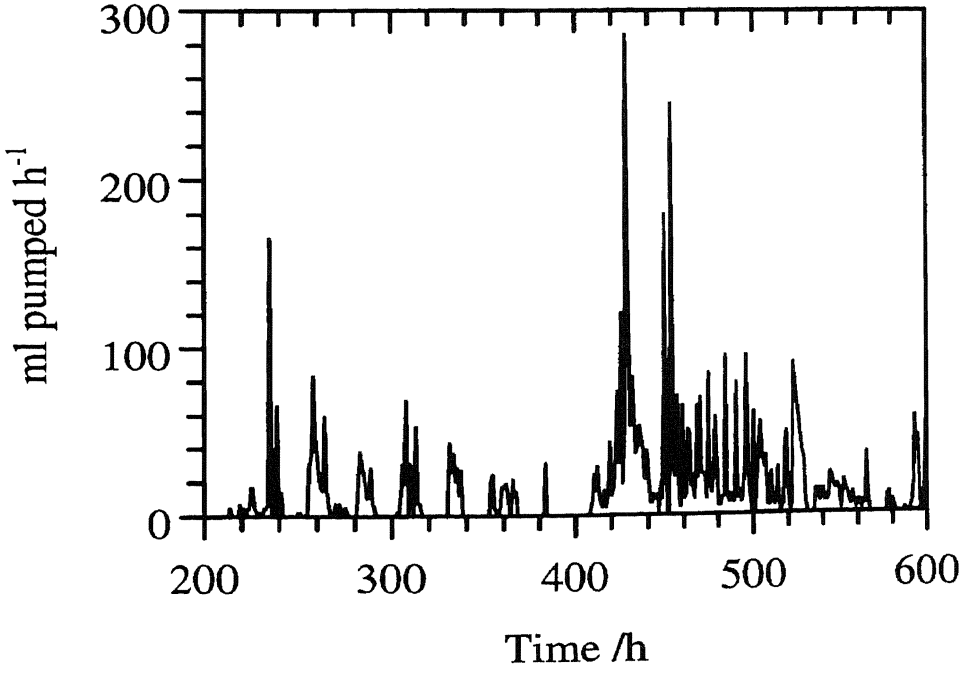


Figure 44 Nutrient feed for hours 200-600.

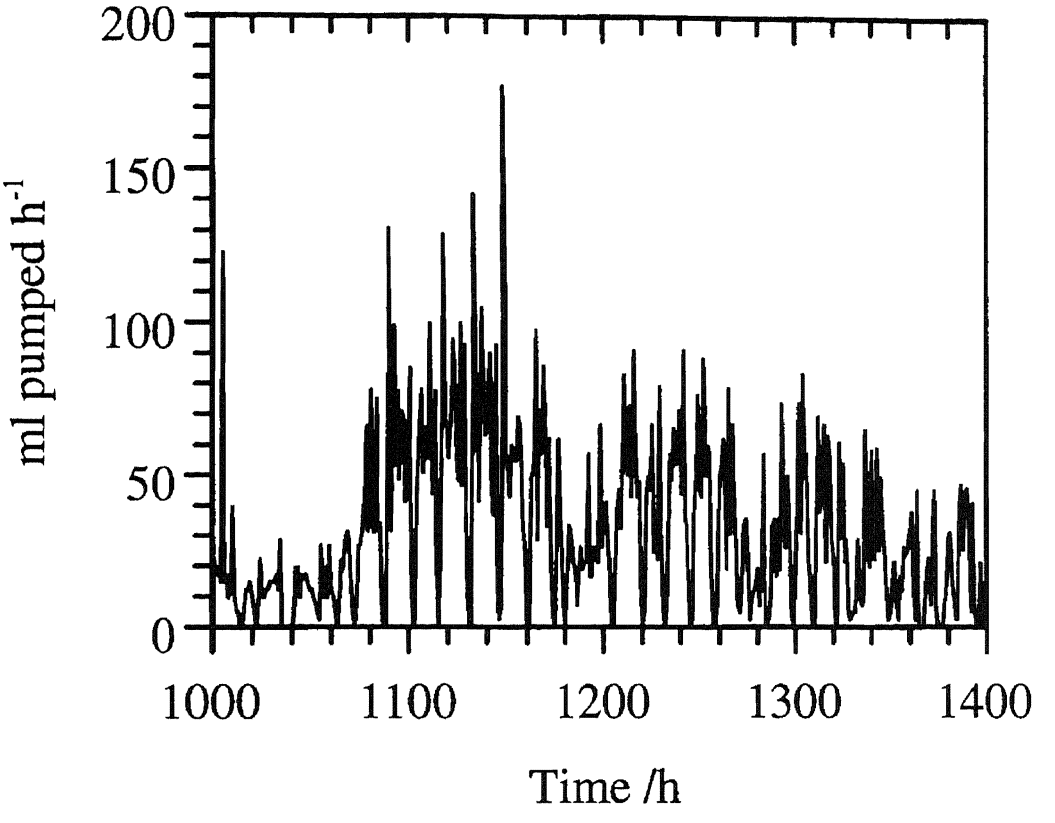


Figure 45, Nutrient feed for hours 1,000-1,400.

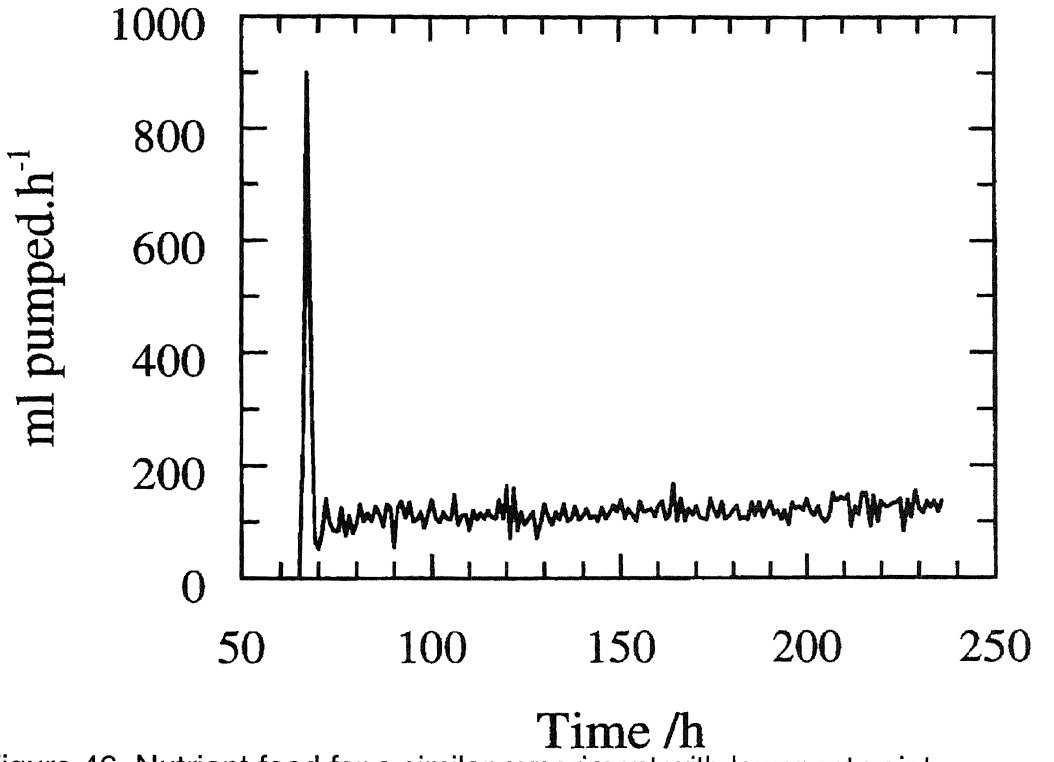


Figure 46, Nutrient feed for a similar experiment with lower set point

6.3 Analysis of the data

Since the work had to be compared and documented alongside other analyses, the same subdivisions of data were used as shown in Figure 43, Figure 44 and Figure 45. The forms of analysis available to me were: Hurst Exponent, Lyapunov exponent, Embedding dimension and separation, with their implied estimate of attractor dimension, and local approximation based prediction. The following are the results of the analyses.

Table 1 Non - linear analyses of yeast growth rate

Time Series Data (Hours)	200-1400	200-600	600-1000	1000-1400	Units
Lyapunov exponent	0.510	0.316	0.147	0.510	Bits/hour
Calculated Embedding Dimension	5	10	8	5	
Calculated minimum embedding separation	2	2	1	2	Hours
Hurst Exponent	0.755	0.680	0.747	0.773	
Correlation Coefficient for Predicted vs. Actual, Local Approximation	0.974	0.959	0.827	0.679	
Correlation Coefficient for Predicted vs. Actual, 1st Order Trivial Predictor	0.436	0.800	-0.050	0.214	
Correlation Coefficient for Predicted vs. Actual, 2nd order trivial predictor	0.865	0.894	0.901	0.872	
Correlation Coefficient for Predicted vs. Actual, 6th order autoregressive model	0.219				
RMS error of predictions vs.	6.749	8.380	17.65	13.44	ml/hour

Actual, Local Approximation					
RMS error of predictions vs. Actual, 1st and 2nd order trivial predictors	26.11	17.63	45.67	22.74	ml/hour
RMS error of predictions vs. Actual, 6th order autoregressive model	26.53				ml/hour
Average pump rate	31.14	21.43	47.77	24.22	ml/hour

Table 1 shows various different analyses of the time series data. Predictions were generated using the Local approximation algorithm, trivial predictors and auto-regressive moving averages. In each case the 10% of the data between the 80% and 90% points were used as the validation or out of sample test set, and the correlation and RMS error figures are for the performance of the method as a predictor of those data sets.

In sample performance is not shown.

The first data column in Table 1 shows results for the entire test data; the rest for successive thirds. The auto-regressive predictions were calculated by the Economics department at the University of Wales. The software used was called T.S.P., and the order of 6 was chosen by experiment to give the best predictions.

6.4 Predictions generated from the data

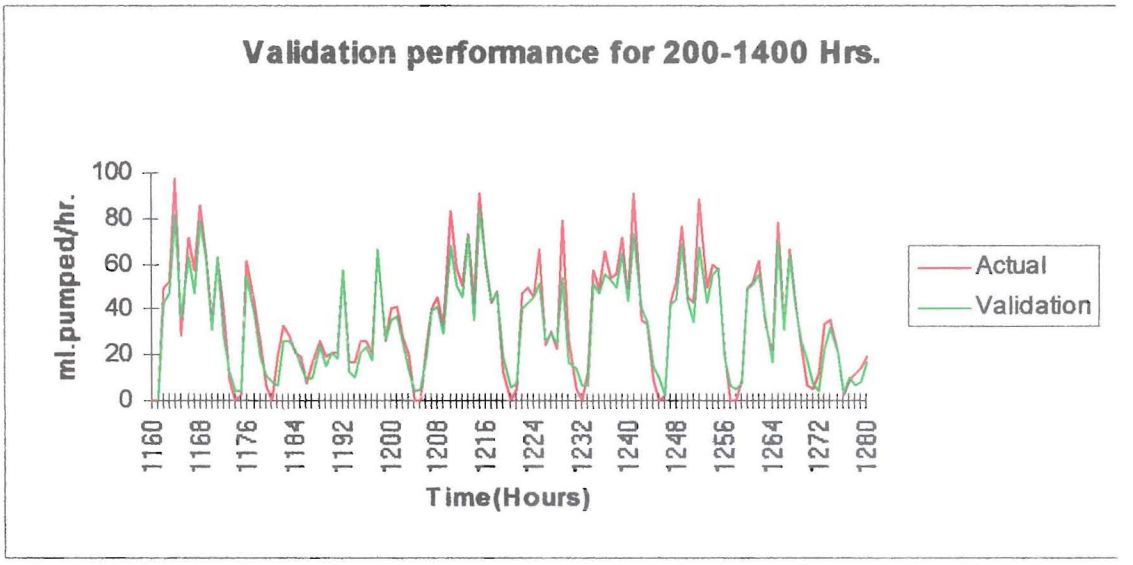


Figure 47, Local approximation predictions, Hours 200 -1400

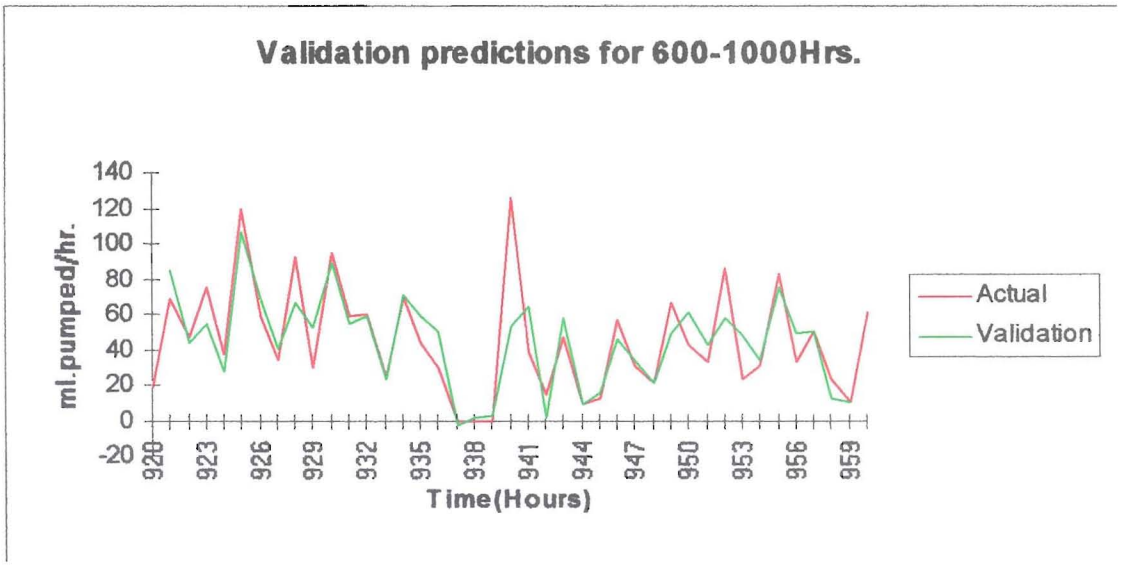


Figure 48, Local approximation predictions, 600-1000

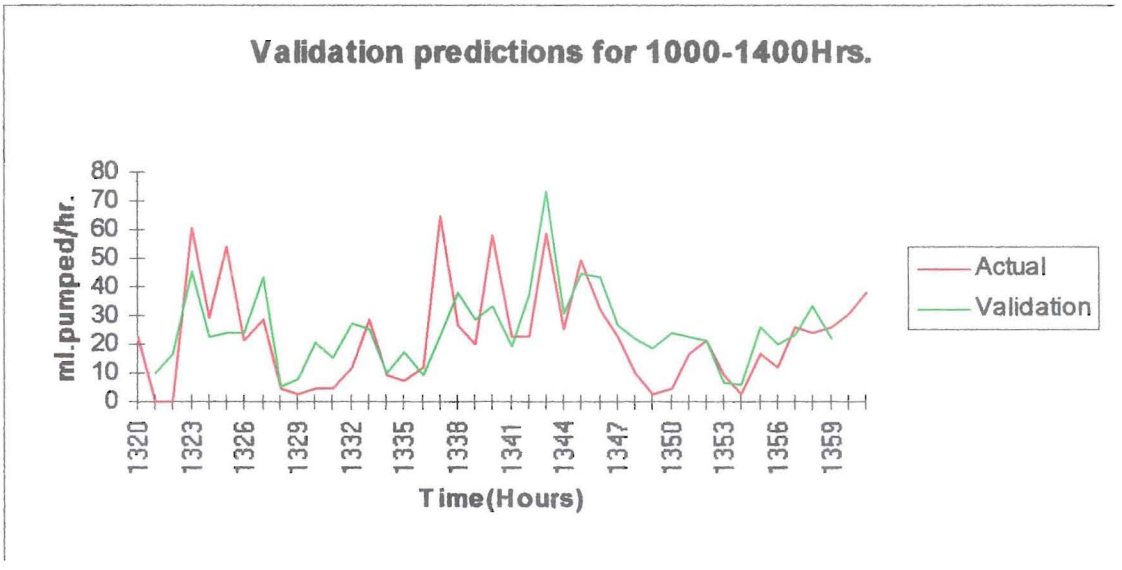


Figure 49, Local approximation based predictions, 1000-14000 hours

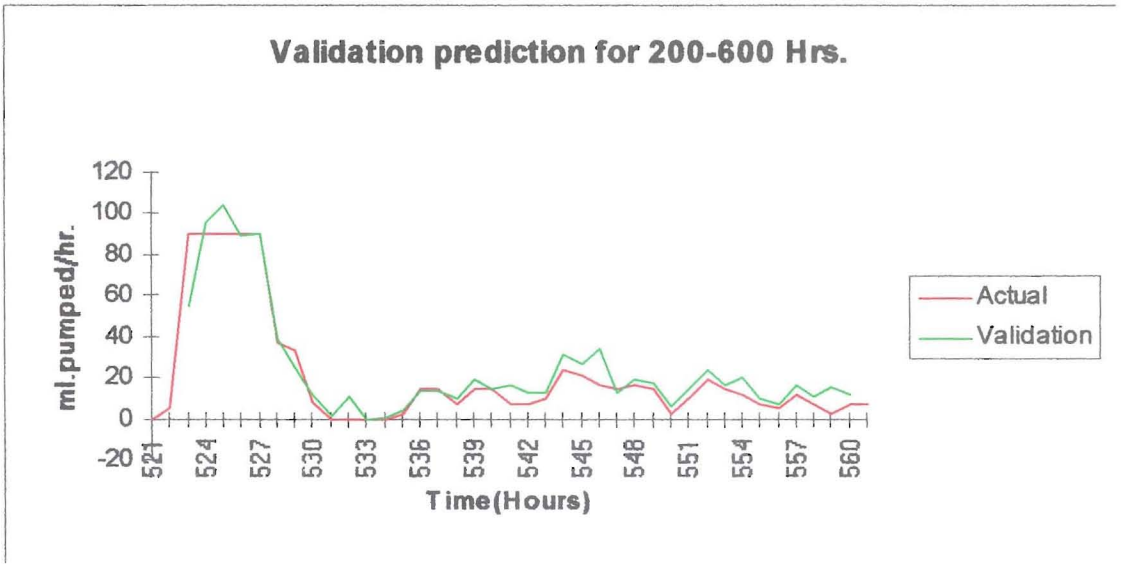


Figure 50, Local approximation based predictions, Hours 200-600

6.5 Conclusions drawn from the results

Figure 47 to Figure 50 show single step predictions of the time series using the version of the local approximation algorithm. It can be seen from Table 1 that the correlation coefficient of the overall prediction in

Figure 47 is very much better than any of the other predictions of sub sections. This implies that the prediction methodology has insufficient data points, and thus information about the attractor for the sub sections. Interestingly the data was originally split into three sections because it was thought that there were three distinctly different modes of behaviour judging from visual clues alone. The fact that the presence of all three data sets improves prediction performance implies that the same process is active in all the data sets. The difference in visual appearance is due, it appears, to the chaotic nature of the process.

The first row in Table 1 adds credence to this assumption because the Lyapunov exponents measured are all positive. It should be pointed out that the algorithm by Wolf (Wolf,85) used to generate the Lyapunov exponents was developed specifically for mathematically derived series where, unlike experimental data, large numbers of points are generally available. It is liable to be more reliable in its estimation of the whole series. The values for the sub sets should be considered merely indicative.

The Hurst exponent adds weight to the diagnosis of chaos, and more importantly the performance of the predictor adds indirect support. We can see by the poor showing of the autoregressive model, which used the ARMA (Box, 76) technique, that linear models do not predict this series well. We can therefore with some safety infer non-linear processes, a necessary pre-condition for chaos.

The calculated embedding dimension derived using false nearest neighbours is anomalous in that two of the sections show much higher embedding dimensions than that for the whole series. I believe the result is due to the greater importance given to noise in the data in the sub sections, and that these results are artefacts of the method itself. I refer the reader to section 2.3.2.1 & 4.3.2 for a discussion of the technique in detail. Since the false nearest neighbours technique settles on an embedding dimension when the number of false neighbours fall below some threshold, and the definition of a false neighbours itself depends on another threshold, a noisy section of data may well push up the calculated embedding dimension by artificially generating false neighbours. Sources of noise are to be found in the measurement process itself, and also in the physical organisation of the system. The system can add nutrient but not take it away. It is clear that during some periods of high growth the pump demand drops to zero, and if the system could have extracted nutrient no doubt the control system would have tried it.

Taking Takens' (Takens,81) embedding theorem in reverse we can estimate a fractal dimension of 2-3 from the embedding dimension, which implies 2-3 variable, which is said to accord well with the fundamental chemistry of the system,

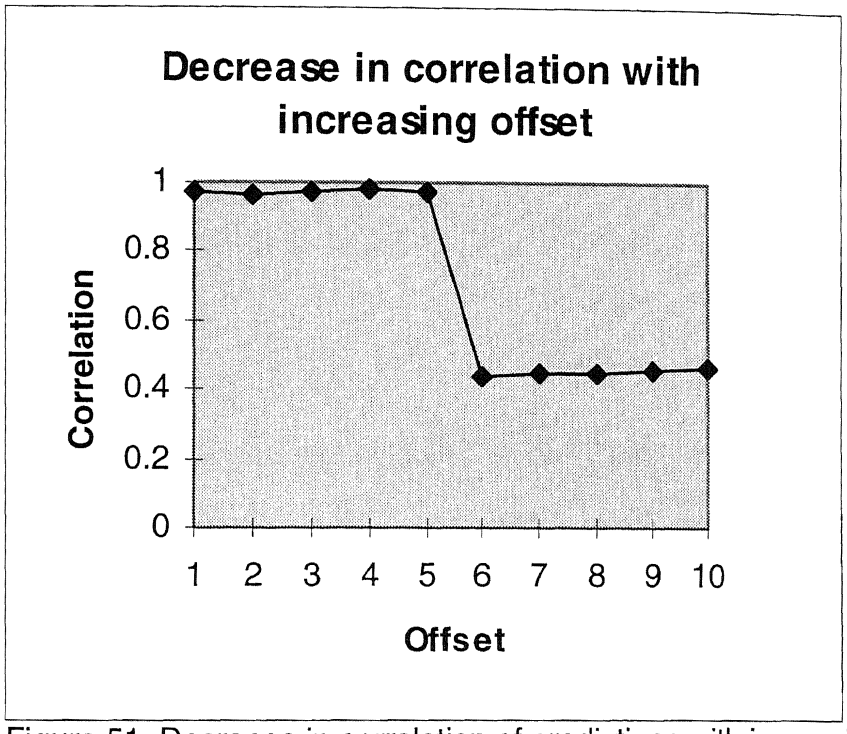


Figure 51, Decrease in correlation of predictions with increasing offset

Figure 51 shows the performance of the predictor when required to make predictions further ahead than one single time step. It can be seen that the predictions remain reasonably good up until an offset of 6 hours. At this stage the correlation coefficient drops to around 0.4. Sugihara and May (Sugihara,90) made much of the exponential drop of correlation for their experimental data. This is not in evidence here, indeed we have effectively a step function. The probability is that the truncation of the data discussed above, by having only positive pump demand, has added spurious correlations to the data above an offset of 5. It can be cited in evidence for this that the 1st order trivial predictor correlation, effectively the 1st order auto-correlation, is nearly identical to this level. There may or may not have been an exponential drop in correlation given a perfect control signal, impurities in the data do not permit us to find out.

David J. Wales (Wales,91) attempted to use a similar system to predict the Tent Map chaotic time series and arrived at a graph very similar to Figure 51 without any measurement noise, since the series was generated mathematically. I therefore consider the results to be in keeping with other published work.

Finally it is clear that the predictions are very effective, and that the underlying dynamics of the system have been well modelled. It can be seen from Figure 45 that given a smaller required biomass these chaotic regimes do not occur. If we scale this process up to that used in the industrial generation of high value chemicals such as drugs, a larger biomass will result in a higher yield for the same industrial plant.

Discovering chaos is part of the process of defence against it. Ott & Yorke (Ott,90) have shown that chaotic processes can be stabilised if they can be modelled, and we have therefore the prospect of generating an advanced control system making use of a predictive model that permits much higher stable concentrations of organisms than hitherto, and thus higher yields.

7. Conclusions and further work

At the beginning of the work the need was identified to predict and analyse real world time series exhibiting chaos and other non-linear behaviour. While tools existed to perform this task, they suffered from two drawbacks: the need to empirically choose a set of parameters that were vital to the prediction performance, and the large amount of computing resources they required. Where parameters are chosen empirically without resort to a theory or heuristic there must be doubt as to the quality of the results. For instance if the selection process took advantage of some transient characteristics of the training data, the model is not likely to work well with out of sample data.

This document has described work undertaken in reducing and finally eliminating the free parameters, and in reducing dramatically the time required to achieve predictions.

Comparison with other published work showed that no predictive performance has been lost in this process, indeed the predictions seem to be more accurate than a range of other methods on a particular bench mark series.

The techniques detailed in this document give a justification for a choice of embedding parameters. This justification has been shown to be valid for a range of financial time series data by the use of sensitivity analysis.

Finally the predictor was used to analyse and predict new experimental data drawn from a biotechnological experiment. The evidence of chaos

and short term predictability detected by the techniques and methodology detailed in this work has shown the presence of a new form of behaviour previously unsuspected in simple and common place organisms under certain environmental conditions.

There are many opportunities for future work that spring from this document.

In this work no real attention has been given to the problem of non-stationarity. The elements of the artificial time series were all generated using the same function and parameters. The biological time series seems to have had the same generating function throughout judging by the better prediction performance on the whole data set rather than the sub divisions. Financial series however are liable to change their properties over time. This can occur gradually, as a result of parameter drift, or dramatically.

One potentially fruitful solution might be the addition of forgetting to the local approximation algorithm used. Any embedded points older than a certain date could be de-rated or completely ignored in generating predictions. If it is true that the iterated system underlying the series was different before that cut off date, then better performance can be expected as a result of this modification.

Of course this adds back a free parameter, and the assumption has been made of a linear change in the underlying system that may not hold.

Discovering a methodology to determine this drift without resort to optimisation would be very valuable.

Another modification to the methodology could be investigated. Since the local approximation algorithm used stores the previous embedded points rather than a model drawn from them, as soon as a point has been used in a prediction, and once the next true value is known it can be added to the database of points. Using this method in and out of sample data cannot be confused, but the local approximation algorithm is always using the most recent data. If as discussed above, the underlying generating function is constantly changing this must have the effect of improving performance.

The local approximation algorithm employed is interpolative in nature, rather than extrapolative. That is to say that if a new predictee point is presented that is outside of the existing set of points in the embedding set, then the algorithm will not work well.
It is likely that neural nets will perform better in these circumstances. It would be interesting to know how much of a problem this is.

It is possible to define a reliability heuristic for predictions generated by this methodology. This might be composed of elements such as the density of predictor points around a predictee, the variation in the predictor outcomes and the degree of agreement in sign of the outcomes. It would

be interesting to know if this would track the measured reliability of the system.

8. References

(Abarbanel,92) Abarbanel, H, Brown R., Sidorowich J., Tsimring L. The analysis of observed Chaotic data in physical systems. Unpublished.

(Baba,89) Baba N., A new approach for finding the Global Minimum of Error Functions of Neural Networks, Neural Networks, 2.5, 367-374

(Benettin,80) G Bennetin, L Galgani & J.M. Strelcyn, Lyapunov Characteristic Exponents for smoth dynamical systems and for Hamiltonian systems; a method for computing all of them, Mechanica, 15, pp 9

(Box, 76) G.E.P. Box, Jenkins G.M., Time Series Analysis: Forecasting and Control, Holden-Day 1976

(Carmana,88) Carmana R.A. & Schaffer.D. (1988) Representation & Hidden Bias: Gray versus Binary coding for genetic Algorithms. in J.Laird (Ed.) Proceedings of the fifth international conference on machine learning. San Mateo Calif. Morgan Kaufmann.

(Casdagli,89) Casdagli M. Non-linear Prediction of Chaotic time series. Physica D 35: 335-356

(Cheeseman 90) Cheeseman, Peter. On finding the most probable model. In Computational models of Scientific Discovery and Theory Formation, Schrager, J. and Langley P. Eds. 1990, Morgan Kaufmann

(Darwin, 1859) Darwin Charles, On the Origin of Species by means of natural selection, or the preservation of favoured races in the struggle for

life. Cassel & Company 1909

(Davey,93) Davey, C.L., The theory of the ß dielectric dispersion and its use in the estimation of cellular biomass. Aber Instruments

(Davey et al. 96) Davey H.M., Davey C.L, Woodward A.M., Edmonds A.N., Lee A.W, Kell D.B., Oscillatory, Stochastic and Deterministically Chaotic Growth Rate Fluctuations in Permittistatically-Controlled Yeast Cultures, Biosystems, 39 pp 43-61 1996.

(Davis,91) Davis L., Handbook of Genetic Algorithms, Van Nostrand Reinhold 1991.

(Eckmann,86) J.P.Eckmann and S.Oliffson Kamphorst,D.Ruelle and S.Ciliberto, Lyapunov exponents from time series, Physical Review A, 1986

(Edmonds,94a) A. Edmonds, D. Burkhardt & O. Adjei, Simultaneous Prediction of Multiple Time series using Supervised Learning & Chaos Theory, Proceedings of IEEE International Conference on Neural Networks, Vol4 pp3156-3164 1994

(Edmonds,94b) A. Edmonds, D. Burkhardt & O. Adjei, Multivariate prediction of financial time series using recent developments in chaos theory. Proceedings of the 1st International Workshop on Neural Networks in the Capital Markets. London Business School, Nov. 1993.

(Edmonds,95a) A. Edmonds, D. Burkhardt & O. Adjei, Genetic Programming of Fuzzy Logic Production Rules, Proceedings of the 1995 IEEE international Conference on Evolutionary Computing,

(Edmonds,95b) A. Edmonds, D. Burkhardt & O. Adjei, Genetic Programming of Fuzzy Logic Production Rules with Application to Financial Trading. Neural networks in Financial Engineering, pp. 179-188, World Scientific, Nov. 1995

(Ellner,91) S. Ellner, A.R. Gallant, D. McCaffrey, D. Nychka, Convergence Rates and data requirements for Jacobian based estimates of Lyapunov exponents from data., Physics Letters A Vol. 153, pp. 357-363

(Fahlman, 88) Fahlman S. An empirical study of learning speed in back propagation networks, CMU-CS 88-162 Carnegie-Mellon Technical Report

(Fahlman, 90) Fahlman, S. Lebiere, C. The Cascade-Correlation Learning Architecture. Neural Information Processing Systems 2, pp 524-532, Touretzky D. Ed. Morgan Kaufmann.

(Fama,65) Fama, E.F., "The Behaviour of Stock Market Prices," Journal of Business, p34-105, (January 1965)

(Farmer 87) J. Doyne Farmer and John J. Sidorowich, Predicting Chaotic Time Series, Physical Review 1987

(Fraser,86) Fraser A., Swinney H.L., Phys,. Rev. A. 33 1134 1986

(Frean,90) Frean M., The Upstart Algorithm: A method for constructing and training feed-forward Neural Networks. Neural Computation 2, 198-209

(Grefenstette,86) Grefenstette J.J., 1986 Optimization of Control Parameters for Genetic Algorithms IEEE transactions on systems, Man

and Cybernetics. SMC-16(1) pp. 122-128

(Harris,87) Harris.C.M, Todd, R.W., Bungard, S.J., Lovitt, R.W., Morris J.G. Kell, D.B., The dielectric Permittivity of Microbial Suspensions at Radio-Frequencies: A novel method for the Estimation of Microbial Biomass., Enzyme Microbial Technology, 9, pp. 181-186

(Hassibi,93) Hassibi, B., Stork, D., Second order derivatives for network pruning: Optimal Brain Surgeon. Neural Information Processing Systems 5, 164-171, 1993 Hanson, Cowan, & Giles eds. Morgan Kaufmann.

(Hecht-Nielsen,90) Hecht Nielsen, R., Neurocomputing, Addison Wesley 1990.

(Hinton, 86) Hinton, G.E.,Deterministic Boltzman Learning Performs Steepest descent in Weight Space. Neural Computation 1,143-150 1986

(Holland,75) John Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press,1975

(Hornik,89) Hornik, K, Stinchcombe, M., White, H., Multilayer feedforward networks are universal approximators, Neural Networks 2, 359-366

(Hurst, 65) Hurst, H.E., Black, R.P. and Simaika, Y.M., Long-Term Storage: An Experimental Study, Constable - London

(Kell,87) Kell, D.B. Samworth, C.M., Todd, R.W. Bungard, S.J., Morris J.G.,, Real time estimation of Microbial Biomass during Fermentations using a dielectric probe., Studia Biophysica, 119 pp. 153-156

(Kennel 92) Kennel, M. Brown, R. Abarbanel H.D.I., Determining Embedding dimension for phase space reconstruction using a geometrical

construction. Physical Review A, Vol 45 Number 6 3403-3411

(Koza, 92) Koza, J.R., Genetic programming: on the programming of computers by natural selection, MIT Press 1992

(Lafrance 90) Lafrance, P., Fundamental Concepts in Communication. Prentice Hall International 1990

(Lorenz, 63) Lorenz E., Deterministic non-periodic flow, Journal of Atmospheric Science, 20, pp. 130-146

(Mackey,77) M.C. Mackey & L. Glass, Oscillation and chaos in Physiological Control Systems, Science, 197, 287

(McClelland,86) James L. McClelland, D.E. Rumelhart, Parallel Distributed Processing, 1986 MIT Press.

(Medio, 92) Medio, Alfredo. Chaotic Dynamics: Theory and application to Economics. Chapter 141992 Cambridge University press, 0-521-39488-0

(Moody,89) Moody, John, Fast Learning in Multi-Resolution Heirarchies, Advances in Neural Information Processing Systems 1, Morgan Kaufmann 1989 p. 29-39

(Nie,94) Junhong Nie, A fuzzy-Neural approach to time series prediction. Proceedings of ICNN 1994 IEEE press 1994 Vol. 5, P3164-3169

(Ott,90) Edward Ott, Celso Grebogi, James A. Yorke, Controlling Chaos, Physics Review Letters, 64, pp. 1196-12000 1990

(Press,88) Press w., Flannery B., Teukolsky S., Vetterling W., Numerical Recipes in C, Cambridge.

(Refenes,93) Refenes, A.N., Azemi-Barac, M. Treleaven, P.C., Financial

Modelling using Neural Networks. In Commercial applications of parallel computing, Liddell H., ed. UNICOM 1993

(Ruelle,81) Ruelle D., Small random perturbations of dynamical systems and the definition of attractors., Commun Math. Phys. 137, pp. 82

(Schraudolph ,91) Schraudolph N. N. & Belew R.K. Dynamic Parameter encoding for genetic Algorithms. Unpublished 1991

(Shimada,79) I Shimada & T. Nagashima, A Numerical approach to Ergodic problem of dissipative Dynamical Systems, Progress of Theoretical Physics, pp. 61 1605-1616 1979

(Solis,81) F.J. Solis, J.B. Wets, Minimisation by Random Search Techniques, Mathematics of Operations Research, 6, 19-30 1981

(Sugihara,90) Sugihara, G., May, R. M., Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. Nature Vol 344, 734-741 1990

(Takens,81) Takens F.,. (1980 Detecting Strange attractors in turbulence. Lecture Notes in Mathematics, Vol 898 (Warwick 1980) eds D.A. Rand, l.s. Young pp. 368-381 Berlin: Springer Verlag.

(Tong, 90) Tong H., Non-Linear Time Series - A Dynamical System Approach, Oxford University Press, 1990

(Turchin, 93) Turchin p.,Chaos & Stability in rodent population dynamics: evidence from non-linear time-series analysis. Oikos 24178-6168 1993

(Vaga,90) Vaga, Tonis, The Coherent Market Hypothesis, Financial Analysts Journal, Nov/Dec 1990

(Wales,91) Wales, D., Calculating the rate of loss of information by

chaotic time series by forecasting. Nature, Vol 350 485-488 1991

(Wolf,85) Wolf, A., Swift, J.B., Swinney, H.L., Vastano, J.A., Determining Lyapunov exponents from a time series. Physica 16D 285-317 1985

(Kennel 93) Kennel, M., The Multiple Dimensions Mutual Information Program. unpublished.

(Lebaron, 96) LeBaron, B., "Question of returns"., email to the Comp.Finance discussion group on the internet. 29/8/96

(Shannon 49) Shannon C.E. & Weaver W. The Mathematical theory of communication. Urbana Il. University of Illinois press 1949.

(Watkins,94) Watkins, Chau, Tawel, Lambrigstem & Plutowski, Advances in Neural information processing systems 6, Morgan Kaufmann 1994 p. 850-857

Appendix 1 Pseudocode of Lyapunov implementation

```
Lyapunov()
{
Determine the range of the data, fSize.
Check for variation in the data, quit if none.
Load the data embedded using dimension and separation determined by AMI & FNN (See
2 & 2)
Normalise the data
Load the data into a binary Tree   (See 4 )
Variable Lyapunov = 0
Variable Found = 0
For each embedded row
        {
        Search in the binary tree for the 10 nearest neighbours
        If this is the first time round the loop
                {
                Check if the nearest neighbour is the last sample.
                If so use the second nearest
                Store the index of this neighbour
                Store the distance to the current pattern
                Increment the found count, Found.
                }
        else
                {
                For each of the nearest neighbours
                        {
                        If This neighbour is more than 10 samples from the current
                        pattern and is not the last
                            {
                            Find the neighbour with the smallest angle relative to the last
                            chosen neighbour in the previous loop
                            }
                        }
                }
        if a suitable neighbour was found
                {
                Fetch the distance between the current point and the chosen neighbour,
                fDist.
                calculate the distance between the next point after it and the next point
                after the current vector, fIterateDist.
                Lyapunov = log2(fIterateDist/fDist) + Lyapunov.
                Found = Found + 1
                Calculate the vector between the chosen point iterates to help select the
                next vector
                }
        }
        Lyapunov = Lyapunov / Found
}
```

Appendix 2 Binary tree pseudocode

Data structures:
pData, an array of floats containing the data.
wVectorCount;, the number of rows in the data
wVectorWidth, the number of columns in the data
pTreeHead, The root node of the tree representation
pBucketArray, an array of integers of size wVectorCount.

The implementation is in C++ and uses a class CBinaryTree to hold the tree.
Construction of the tree is automatic on the construction of a CBinaryTree object passed the data and the row and column counts.

```
CBinaryTree initialisation
{
For each position in pBucketArray i
   pBucketArray[I] = i.

pTreeHead = CreateNode(0, wVectorCount)
}

CBinaryTree CreateNode(Input Index, Number of points) // this recursive procedure
generates the tree.
{
if the Number of points is smaller than a constant, (15)
      {
      This is a leaf node. Create a  CBinaryTreeNode object with the index of  this objects
      section of the data,  A pointer to pBucketArray  and the number of points.
      }
Else
      {
      For each column
            {
            Calculate the mean of all the data values in that column
            Calculate the standard deviation of the data values in that column
            if the standard deviation is the greatest so far store it, the column index and the
            mean. (The selected column)
            }
      Check to make sure that there was some variation in the  sample
      if there was no variation
            {
            Call this function recursively passing the Input Index and half the number of
            points.
            Call this function recursively passing input index  + half the number of points and
            half the number of points.
            }
      else
            {
            For each of the number of points
            look in the data for the selected column at the row offset given by Input Index +
            the point index.
            If that data value is greater than the mean move it to the upper half  of the
            sample.
            If its less than the mean move it to the lower half.
            Call this function recursively passing the Input Index as the first value >= to the
            mean, and the number of points greater than the mean.
            Call this function recursively passing the existing input index and the number of
            points less than the mean
            }
      }
}
```

pBucketArray contains indices for each row of data. Initially ordered.
The function above recursively divides the array into smaller and smaller sections by choosing pivot points, and at each division the indices are sorted either side of the pivot point. Eventually the divisions are smaller than a pre-selected size, and are denoted as

leaf nodes. A tree is created in this process, and each node of the tree contains the pivot index, number of points below it, the pivot column and the pivot value.

When used on financial data, especially differenced data, large numbers of points had the same value and thus sooner or later a tree node would be generated controlling points with zero standard deviation. The above algorithm handles this case by splitting the indexes at an arbitrary mid point.

There are two kinds of search that can be performed on the newly created tree. The tree can return all the points within a certain radius, or the n points closest to a given point. The latter is used exclusively in this work, and the search proceeds as follows:

```
Search ( search vector, count of neighbours, array of indices, array of distances)
{
        Initialise Arrays
        Call Search node passing a reference to the head node of the tree
}

SearchNode (Current node)
{
        if current node is a leaf node
                {
                Calculate Euclidian distance for each point controlled by this node to the
                search vector, and place in the appropriate part of the distance and index
                array, shuffling down, and eventually out of the array any points that are
                further away
                }
        else
                {
                find the discriminant column for this node, extract the corresponding data
                value from the search vector, and calculate the difference between this
                and the pivot value, fDiff; A distance, fDist is also calculated as fDiff^2
                if fDiff is negative
                        {
                        Call this function recursively with the left branch of the tree.
                        If fDist is smaller than the  best found so far call this function
                        recursively with the right branch
                        }
                else
                        {
                        Call this function recursively with the right branch of the tree.
                        If fDist is smaller than the  best found so far call this function
                        recursively with the left branch
                        }
                }
}
```

Appendix 3 Pseudocode of Mutual information implementation

This proceeds as follows:
Input arrays are Vector1, and Vector2 both of length wLength

```
Mutual Information( Vector1, Vector2)
{
        Calculate the mean and standard deviation of vectors 1 & 2 and re-scale them both
        to have zero mean and unity standard deviation.
        Check for data with no variation. Quit if found.
        Form a 2 dimensional matrix, matrix3, by concatenating Vector1 and Vector2.
        Form a binary tree from each of Vector1, Vector2 and Matrix3.
```

Generate a vector H of length wLength to hold adaptive bandwidths and initialise each location to a pre-defined constant 0.01.
Perform a density estimation on Matrix3 and use the probabilities to adjust the bandwidth array H.
Perform Density estimation on Vector1, Vector2 and Matrix3 using the new bandwidths and calculate the mutual information using **Error! Reference source not found.**.7

}

Appendix 4 Pseudocode of False Nearest Neighbours implementation

```
False nearest neighbours
    {
    Find the range of the data, fSize.
    For dimension = 2 to dimension = 8
        {
        Embed the data using the AMI derived separation and the current dimension
        Create a binary tree representation of the data
        for each row
            {
            Find the nearest neighbour and the distance between them, fDist
            Take data from the time series to extend the embedding of the point in
            question and the nearest neighbour.
            Calculate the new distance between the two points fNew.
            If fNew/fDist > 5 or fNew > 0.2 * fSize
                Increase the nearest neighbour count by one
            }
        if the change in nearest neighbour count / number of rows  > -0.1  select the
        preceding embedding as the result and quit
        }
    If  the above has not resolved an embedding dimension
        Choose the first neighbour count / number of rows below 0.1 as the result
    }
```

Appendix 5 Pseudocode of Hurst implementation

```
Hurst()
{
Find the number of samples in the time series
Normalise the data
Determine the number of measures, = Sample Count / 10
Allocate a 2 dimensional array to receive the RSI measurements
for each of the measures, p
        {
        RSAve = 0
        Size of block = Sample Count / p
        For  each  block in the data
                {
                Calculate the average
                Calculate the standard deviation using the average
                Find the range of the block
                RSAve = RSAve + The range / the standard deviation
                }
        RSAve = RSAve / p
        Put the log of the RSAve into the x side of the RSI array in the pth position.
        Put the log of (Size of Block * sample time in seconds) into the y side of the RSI
        array in the pth position.
        }
Perform linear regression on the array of tuples.
The Hurst exponent is the gradient of the best fit line.
}
```

Appendix 6 Pseudocode of local approximation algorithm interpolation

```
Local Approximation()
{
Result = 0.0;
b = 0.0;
av = 0.0;
for each of the vertices
   av = av plus that vertex's distance;
if the sum of the distances is not zero (this can happen with financial data)
   av = 4.0 / av;
else av = 4.0;
for each vertex calculate the average of the outcome and the average distance
for each vertex
  {
  b = b+  exp(- the vertex's distance from the predictee * av);
  Result = Result + exp(-the vertex's distance from the predictee * av) * the vertex's
outcome;
  }
if(b != 0.0) // check for potential divide by zero
    b = 1.0/b;
Result = Result * b;
}
```