winGamma^{TM}: a non-linear data analysis and modelling tool with applications to flood prediction

Peter James Durrant
Department of Computer Science, Cardiff University,
P.O. Box 916, Cardiff, CF24 3XF, Wales, UK

May 26, 2001

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated.
Other sources are acknowledged by citations and footnotes giving explicit references.
A bibliography is appended.

Signed ..... (candidate)

Date .....

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

Acknowledgements

I would like to thank Professor Antonia J Jones for her support and continuous encouragement throughout this endeavour, and for the hospitatility that Antonia and Tina have shown during my visits to the *research farm* in the Brecon Beacons.

I would also like to thank Brian Greenfield at the UK Environment Agency for supplying the data for the flood prediction study, and McCann-Erickson for sponsoring this work. I would also like to thank my family, friends and colleagues in the Department of Computer Science for their support over the last three years. In particular the assistance given by Steve Margetts during the development of *winGamma* enabled us to exceed our expectations for the software.

Abstract

This work is based on developments in non-linear modelling which allow the possibility of quickly examining input-output data and quantifying the extent to which this data can be modelled by a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ with bounded derivatives. This algorithm, the Gamma test, which quantifies the noise variance associated with the unknown smooth mapping, was first described in [Aðalbjörn Stefánsson et al., 1997].

After a brief introduction to non-parametric, non-linear modelling, with special reference to feed-forward neural networks, we describe the Gamma test and demonstrate a number of example analyses. These experiments are designed to illustrate the underlying rationale of the Gamma test and also to demonstrate feature selection, which is a natural extension. Where possible we also give comparisons between the Gamma test analyses and more conventional feature extraction algorithms.

A new extension of the Gamma test is then discussed that allows the possibility of not merely quantifying the noise variance but, under certain circumstances, actually re-constructing the noise *distribution* directly from the data.

We then go on to describe at some length the construction and function of the non-linear analysis workbench *winGamma*, a user-friendly Microsoft Windows application developed around the Gamma test.

Finally, these techniques are applied to the problem of modelling level and flow in the River Thames to produce accurate short term predictions for downstream values of level and flow, which can be used for the purposes of river management and flood prediction. This application represents a completely novel adaptive modelling approach to river level and flow prediction.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

The construction of non-linear models from sampled data is very much a subjective process. This in part stems from the enormous diversity of possible modelling techniques and the difficulty of assessing the quality of the data. For example, for data describing discrete input attributes with continuous or discrete outputs one might consider a rule based system of modelling such as a decision-tree approach [Quinlan, 1986]. At the other extreme, input and output variables are continuous and, if the unknown process being described by the data is suspected to be non-linear, one might consider a modelling technique based on neural networks (see [Bishop, 1996] for an excellent up-to-date account). The validation of the chosen modelling technique is frequently purely empirical – the best possible non-linear model is built using the selected technique. If these attempts are successful then the original choice is deemed to be vindicated, otherwise an alternative technique is tried or the failure simply ascribed to 'bad data'.

A dispassionate observer might be forgiven for concluding that the above state of affairs is somewhat unsatisfactory, perhaps lacking in good scientific methodology.

No single thesis can address all of the above problems – the extraction of good models from data of diverse types and diverse quality is a very broad problem. However, some aspects of these issues can be addressed in a more systematic fashion. In this thesis the focus is on *smooth* models of *continuous* variables. We do not consider the case of discrete input or output variables, although some of the techniques developed here might be applicable. We begin with a brief study of non-linear modelling techniques in Chapter 2. We consider the current state of non-linear modelling

and introduce the modelling techniques that are used throughout this thesis. One consequence of this decision is that we sidestep the question of 'what type of model should be constructed'.

It transpires that if we are prepared to assume that the underlying unknown process f is smooth, of the form

$$y = f(x_1, \ldots, x_d) + r \tag{1.1}$$

where the noise r may be due to real noise or a lack of functional determination and $x_1, \ldots, x_d$ are input variables and y is an output, then many of the methodological problems of model building can be rectified. These include being able to answer such questions as:

- To what extent do the inputs determine the output by a smooth model?
- Given an input vector $\mathbf{x}$ how accurately can the output y be predicted?
- How many data points are required to make a prediction with the best possible accuracy?
- Which inputs are relevant in making the prediction and which are irrelevant?

This thesis addresses some important aspects of non-linear data analysis and modelling. In particular it is possible to estimate the variance of the noise var(r), extract relevant input variables, and determine how much data is required to build a model to a pre-specified accuracy. Moreover, this information can often be directly computed from the raw data using efficient, scalable algorithms. These ideas originated with the Gamma test [Aðalbjörn Stefánsson et al., 1997] and the work of Končar [Končar, 1997] and are discussed in Chapter 3. An empirical justification for the Gamma test is provided using several examples, including a chaotic system. A detailed theoretical discussion is given in [Evans, 2001].

Chapter 4 describes the extensions to the Gamma test for data analysis. Techniques for feature selection, estimating the model complexity, embedding dimension search, and irregular embedding dimension search are all discussed. The experiments used within the chapter illustrate how each techniques can be applied to real-world problems.

Until recently the Gamma test had been used to measure the variance of the noise, i.e. the second moment of the noise distribution. It became clear that a simple extension of the Gamma test could be made to calculate the higher moments of the noise distribution and, to some extent, these measurements allow the noise distribution to be reconstructed. This novel process is discussed in Chapter 5.

In one sense this thesis is about the construction of *winGamma*[1], a commercial package which acts as a 'smooth data modelling' test bench. We have integrated the techniques developed around the Gamma test into a user-friendly, reliable, and comprehensive toolkit for the analysis of smooth non-linear systems and then added the best available non-linear model construction algorithms. The design, implementation, and use of *winGamma* is discussed in Chapter 6.

However, the construction of *winGamma* is really just the first step. Once the tool became available it became possible to accomplish as routine many tasks, of interest in their own right, which hitherto would have required considerable time and effort. Examples include the analysis and modelling (and hence the control) of modestly high dimensional chaotic systems, feature extraction from genome strings for the classification of species [Chuzhanova et al., 1998], the analysis of solar array data, and more effective commercial property price prediction [James and Connellan, 2000].

Finally we apply the tools developed to the interesting question of flood prediction. Once precipitation has occurred the process of rainfall leading to flowing rivers is arguably a smooth, albeit complex, dynamical system which should be amenable to analysis using the Gamma test and smooth non-linear modelling techniques.

We can state the simplified problem as follows: Can we determine the river flow from historic measurements of the river catchment area such as flow rates and levels and current environmental factors such as rainfall? In Chapter 7 we initially approach this question in a purely theoretical way by building a reasonably accurate 'river simulator' data generator and then by analysing the data files produced. We then use this preliminary study of the simulator to analyse actual river and environmental data from the UK Environment Agency to build a reliable flood prediction system.

[1] *winGamma* is available under licence for commercial and research use from the University of Wales.

CHAPTER 2

An Introduction to Non-linear Modelling

The classical statistical approach to building models involves linear regression: a process of fitting a straight line through some sampled data. This approach has been sufficient for many years to describe those systems that behave linearly or approximately linearly, or can be appropriately transformed to create a linear relationship.

However, there is a greater set of non-linear problems that cannot be approached non-parametrically using these linear techniques. The realisation that linear regression could not accommodate the analysis of these more complex problems led to the study of non-linear systems and the development of modelling techniques to describe those systems.

2.1 Non-linear modelling techniques

The modern statistical approach to non-linear model building has led to techniques such as local-linear regression, polynomial regression, kernel discriminant analysis, k-means cluster analysis and principal component analysis, to mention but a few.

Recent inspiration for non-linear modelling has also come from the study of biological and evolutionary systems, most notably producing the artificial neural network. This introduced a set of techniques for non-parametric non-linear regression which include feedforward artificial neural networks, radial basis function networks, and general regression neural networks. Training algo-

rithms for neural networks have developed to include the standard backpropagation and algorithms based on Levenberg-Marquardt and conjugate gradient optimisation techniques. These algorithms have provided the ability to train a neural network (using example data) to create successful non-linear models.

We introduce two of these techniques for non-linear model building which will be used exclusively in this thesis: *artificial neural networks* and *local-linear regression*.

2.2 Artificial neural networks

McCulloch and Pitts developed the first computational representation of a neuron, a device that took a weighted sum as the input and generated an output of 0 or 1 depending whether the sum was above a given threshold [McCulloch and Pitts, 1943]. They showed that such circuits could implement any given logic function, but did not provide a training mechanism to produce such a circuit – in fact this remains a very difficult unsolved problem.

Rosenblatt developed *perceptrons* [Rosenblatt, 1962], single layer feedforward networks of McCulloch-Pitts neurons, and focussed on the problem of how to find appropriate weights for a particular computational task. At about the same time *adalines* (which were similar to perceptrons) were developed by Widrow and Hoff [Widrow and Hoff, 1960]. The training algorithms developed were only applicable to single layer networks, i.e. one layer of inputs connected to an output layer.

The weaknesses of the single layer perceptrons were highlighted by Minsky and Papert. They demonstrated that perceptrons could only solve linearly separable problems, that many classes of interesting problems were *not* linearly separable, and conjectured that a suitable training algorithm for multi-layer perceptrons would be difficult to develop [Minsky and Papert, 1969]. The conclusions of their inquiry largely destroyed the scientific interest in artificial neural networks for 15 years.

The situation changed in the mid 1980s with the advent of two quite separate developments: *back-propagation*, actually developed earlier but whose significance was not immediately appreciated, and Hopfield networks, which are largely outside the domain of this thesis.

Backpropagation, originally developed in the context of adaptive control theory, was certainly the most influential development in the field of artificial neural networks applied to non-linear mod-

elling. It regenerated the interest in neural networks which was lost after Minsky and Papert published their work on perceptrons. Backpropagation was first discovered by Werbos [Werbos, 1974], and re-discovered by Rummelhart, Hinton, and Williams [Rumelhart et al., 1986], [Le Cun, 1986], and Parker [Parker, 1985]. Backpropagation was a technique of training multi-layer perceptrons by adjusting the connecting weights in successive layers.

The theoretical basis for feedforward neural network approximation stems from the fact that standard feedforward neural networks, with as few as one hidden layer, using (fixed) arbitrary sigmoidal functions, can approximate to any desired degree of accuracy any continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ over a compact subset of $\mathbb{R}^n$, provided sufficiently many hidden units are available [Hornik et al., 1989] and [Cybenko, 1989]. This is, of course, an existence theorem and gives no guarantee that any particular training method will converge to the required approximation, nor any indication of the number of hidden units required. However, it is an important result analogous to the approximation of continuous functions by polynomials (Weierstrass's theorem).

In practice a second hidden layer can often be used to reduce the number of hidden units in a single hidden layer network, so leading to a more efficient representation.

Multi-layer perceptrons trained using backpropagation must have differentiable output functions, so the threshold unit proposed by McCulloch and Pitts had to be replaced. The typical choice of output function is a sigmoid (see Figure 2.2).

2.2.1 How a feedforward neural network works

An input is passed into the network in a feedforward process to produce an output. The input to each node is calculated for the first layer using (2.1), the processing at each node is performed, and the output is fed forward to the next layer. This process continues through the network until the outputs are produced from the network.

A representation of a neural network is shown in Figure 2.1. This particular network has a 2-3-3-3 architecture[1]. The nodes in the input layer and the bias nodes do not perform any processing (the role of bias nodes will be discussed again in connection with the network activation function). The remaining nodes in the network perform processing, and are indicated with a sigmoid in Figure 2.1.

[1]The bias nodes, shaded in grey in Figure 2.1 do not perform any processing and are in effect hidden from the user. It is the connection weights from the bias nodes to the rest of the network that are significant.

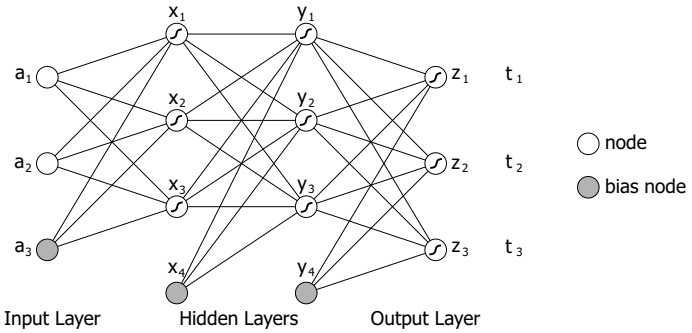


Figure 2.1: An artificial neural network with a 2-3-3-3 architecture.

Input layer nodes

The input layer nodes do not perform any processing[2], so the output of the input layer nodes is the input to the network.

Activation function

The input to the i^{th} node is processed as a sum of the outputs of the nodes in the previous layer multiplied by their connection weights to the i^{th} node, defined as

$$net_i(y_1, \ldots, y_m) = \sum_{j=1}^{m} w_{ij} y_j - \theta_i \tag{2.1}$$

where m is the number of nodes in the previous layer, w_{ij} is the weight from the j^{th} node to the i^{th} node, θ_i is the threshold for the i^{th} node, and y_j is the output of the j^{th} node (in the previous layer). In practice the thresholds are implemented by adding a 'bias node' to each layer, which is always on (indicated as shaded node in Figure 2.1), and is connected only to the nodes in the next layer. These connection weights can then be adjusted during learning without special treatment in the implementation to create the threshold.

We shall call (2.1) the *activation function* (the literature is rather confused with regard to this terminology).

Output function

Given the activation net_i the output function determines the response of a node. The chosen function is typically differentiable, non-linear and monotonic to provide a smooth mapping between

[2] In practice we often pre-process the data using standard scaling and other routines.

continuous variables as shown in Figure 2.2.

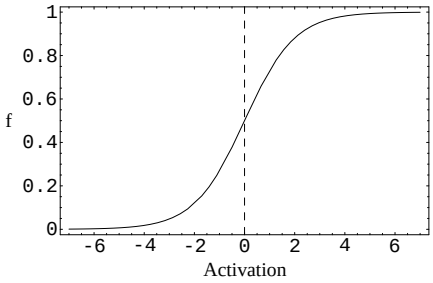


Figure 2.2: Sigmoidal output function. This example is the logistic sigmoid function (2.2) using the activation function (2.1) with $\theta_i = 0$.

The conventional output functions are either the logistic sigmoidal function

$$f(net_i) = \frac{1}{1 + \exp(-net_i)} \qquad 0 < f < 1 \tag{2.2}$$

or the $tanh$ function

$$f(net_i) = \frac{e^{net_i} - e^{-net_i}}{e^{net_i} + e^{-net_i}} \qquad -1 < f < 1 \tag{2.3}$$

Bishop suggests that the $tanh$ function may offer a slight practical advantage over the logistic sigmoid for the hidden layer nodes, although his evidence is purely empirical [Bishop, 1996]. In fact the precise details of the sigmoidal output function are largely irrelevant to the overall scheme of things, although some choices may offer implementational advantages.

2.2.2 An introduction to training algorithms

Neural networks for non-linear regression are trained by example using input-output data. The purpose of training is to minimise some measure of error on the training data by adjusting the model parameters.

If we define the error function to be a differentiable function of the outputs, e.g. sum-of-squares, then the error becomes a differentiable function of the weights. In this way the minimisation of the error becomes an optimisation in weight space.

Consider the standard sum-of-squares error function

$$E(\mathbf{w}) = E(\mathbf{z}, \mathbf{t}) = \frac{1}{2} \sum_{j=1}^{n} (z_j - t_j)^2 \tag{2.4}$$

where $\mathbf{z}$ is the network output, $\mathbf{t}$ is the target output, and n is the number of nodes in the output

layer. If we differentiate we obtain the error at each output node

$$\frac{\partial E}{\partial z_j} = z_j - t_j \tag{2.5}$$

for $1 \leq j \leq n$.

A simple algorithm for network training is described in Algorithm 1. The training data is periodically shuffled to avoid repetitive cycles that may result in the algorithm getting stuck at local minima. Each vector is then fed into the network and the error calculated between the expected output and the actual output and the weights in the network are adjusted accordingly. The algorithm tests to see if the stopping criteria has been reached after each iteration.

```
{initialisation}
establish stopping criteria
determine the network architecture
initialise the weights

{training loop}
while the stopping criteria has not been reached do
  shuffle the data
  for i = 1 to M do
    feedforward x(i) through the network to calculate the error
    adjust the network weights to reduce error
  end for
end while
```

Algorithm 1: A generalised algorithm for neural network training.

In Algorithm 1, the method of adjusting the weights to reduce the error is not discussed. Several techniques for error minimisation are examined in the next section.

2.2.3 Error minimisation techniques for training algorithms

Training is a technique used to minimise the error of a network. The network weights are adjusted until the error is at a minimum or a predefined limit has been reached.

In the neighbourhood of a minimum the error surface $E(\mathbf{w})$ is approximately quadratic and some training algorithms are optimised for this case. Consider the second order Taylor expansion of $E(\mathbf{w})$ around the minimum point $\mathbf{w}^*$

$$E(\mathbf{w}) = E(\mathbf{w}^*) + (\mathbf{w} - \mathbf{w}^*)^T \nabla E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H} (\mathbf{w} - \mathbf{w}^*) \tag{2.6}$$

where $\mathbf{H}$ is the Hessian matrix. At the minimum $\mathbf{w}^*$ the linear term is eliminated since $\nabla E(\mathbf{w}^*) \equiv 0$ and the quadratic error function can be expressed as

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*). \tag{2.7}$$

We shall consider quadratic error surface minimisation for *steepest descent*, *conjugate gradient descent*, and *quasi-Newton methods* to compare the utility of each technique.

It is important to note that both conjugate gradient descent and quasi-Newton methods compute the Hessian matrix to perform the minimisation, whereas gradient descent does not. It will become apparent that gradient descent cannot match the optimisation performance of these other techniques.

Steepest descent

Steepest descent is an iterative minimisation process that descends in the direction of the steepest local gradient.

The rate of convergence to the minimum is dependent on the step size[3] and the addition of other factors, for example momentum. Consider the step size: too small and the algorithm takes a long time to converge, too large and the algorithm fails to converge.

The concept of gradient descent is illustrated in Figure 2.3. The error function (shown as contours in the figure) is minimised by repeatedly taking steps down the steepest gradient. In this example the step size is decreased as the minimum is approached.

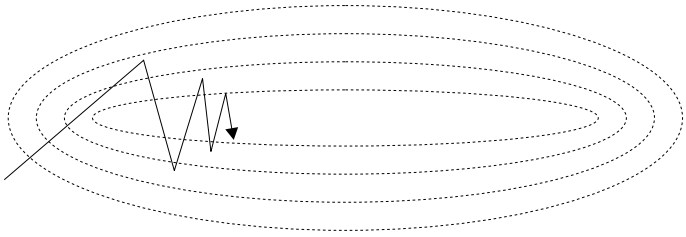


Figure 2.3: Gradient descent.

In addition to having an adaptive step size, momentum can be applied to the algorithm to ensure that advantageous directions are maintained, as shown in Figure 2.4.

[3]The step size corresponds to the learning rate in neural network terminology.

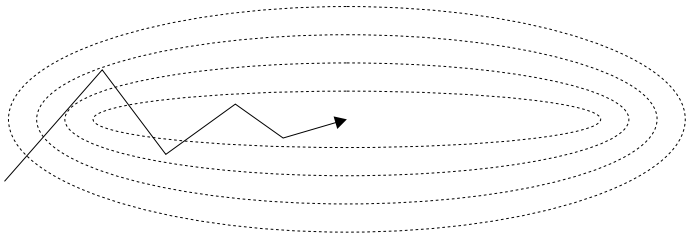


Figure 2.4: Gradient descent with momentum.

Conjugate gradient descent

Conjugate gradient descent uses past gradient measures to improve the minimisation process. After a minimisation has been performed in one direction, that direction is not considered for minimisation again.

The minimisation shown in Figure 2.5 for 2-dimensions requires only two steps to minimise the function, whereas the steepest descent minimisation for the same function (shown in Figure 2.3) requires many more.

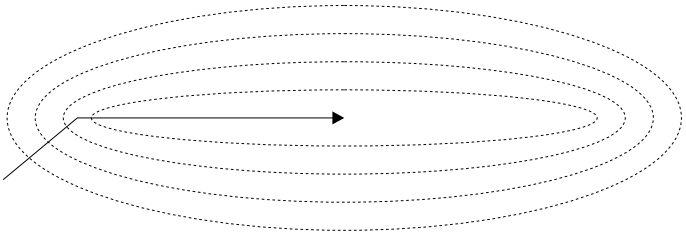


Figure 2.5: Conjugate gradient descent.

Quasi-Newton methods

Quasi-Newton methods attempt to obtain the location of the minimum of the quadratic surface from the Newton direction, obtained using the inverse Hessian.

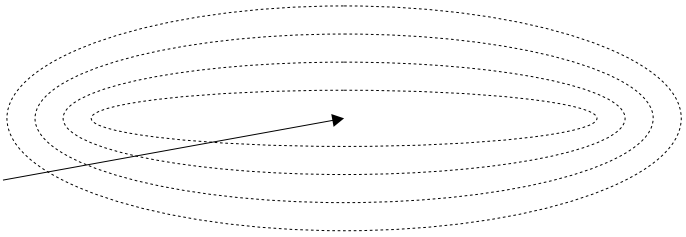


Figure 2.6: The Newton direction used in Quasi-Newton methods.

In reality it is computationally expensive[4] to compute the Hessian directly, so an approximation is achieved iteratively. Initially the algorithm performs gradient descent, but as the approximation of the Hessian improves, the convergence to the minimum can be rapid.

2.3 Backpropagation

The backpropagation learning algorithm [Rumelhart et al., 1986] is the original multi-layer local gradient descent minimisation technique.

2.3.1 Weight adjustment to the output layer

The change in connection weights between the output layer and the preceding layer is defined as

$$\Delta w_{jz} = -\eta \frac{\partial E}{\partial w_{jz}} \tag{2.8}$$

where j is the output layer node index ($1 \leq j \leq n$), z is the preceding layer node index ($1 \leq z \leq t$), and $\eta > 0$ is the learning rate.

Expressing (2.8) for known terms we obtain

$$\Delta w_{jz} = -\eta \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{jz}} = \eta \delta_j \frac{\partial net_j}{\partial w_{jz}} \tag{2.9}$$

where

$$\delta_j = -\frac{\partial E}{\partial net_j} = -\frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial net_j} = -f'(net_j) \frac{\partial E}{\partial z_j} \tag{2.10}$$

If we now use the logistic sigmoid output function (2.2), sum-of-squares error function (2.4) and (2.5), we obtain

$$\Delta w_{jz} = \eta \delta_j y_z \tag{2.11}$$

where y is the output from the previous layer and

$$\delta_j = -f(net_j)(1 - f(net_j))(z_j - t_j) \tag{2.12}$$

and now (2.11) expresses the weight adjustment in terms of known quantities.

[4] Evaluation of the Hessian has time complexity $O(NW^2)$ and computation of the inverse is $O(W^3)$ (where N is the number of data samples, and W is the number of weights).

2.3.2 Weight adjustment to the hidden layers

We adjust the parameters in the hidden layers to minimise the error using

$$\Delta w_{iz} = -\eta \frac{\partial E}{\partial net_i} \frac{\partial net_i}{\partial w_{iz}} = \eta \delta_i \frac{\partial net_i}{\partial w_{iz}} \tag{2.13}$$

where i is the hidden layer node index, z is the preceding layer node index, and write, using the chain rule,

$$\delta_i = -\frac{\partial E}{\partial net_i} = -\frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial net_i} = -f'(net_i) \frac{\partial E}{\partial y_i} = -f'(net_i) \sum_{j=1}^{n} \delta_j \frac{\partial net_j}{\partial y_i} \tag{2.14}$$

where n is the number of nodes in the succeeding layer (i.e. the output layer if we are considering the final hidden layer), and y_i is the output of node i.

Using the logistic sigmoid output function (2.2), the sum-of-squares error function (2.4), and (2.5), then

$$\Delta w_{iz} = \eta \delta_i x_z \tag{2.15}$$

where x is the output from the previous layer and

$$\delta_i = -f(net_i)(1 - f(net_i)) \sum_{j=1}^{m} \delta_j w_{ji} \tag{2.16}$$

which again expresses the weight adjustment in terms of quantities known at this stage.

The algorithm then proceeds by recursing these steps backwards through the layers until the last set of weights are adjusted. Threshold adjustment is effected without special provision using the bias nodes.

Thus the implementation of backpropagation involves a forward pass through the layers to estimate the error, and then a backward pass modifying the weights to decrease the error. Practical implementations are not difficult, but without modification it is still rather slow, especially for systems with many layers. Still, it is at present the most popular learning algorithm for multi-layer networks.

Backpropagation, being based on local gradient descent, can in principle fail and become stuck in a local minimum in weight space. In this case it is customary to re-initialise the weights, perhaps adding more hidden nodes, and re-start training. However, it is interesting to note that, provided sufficient hidden nodes are present, such failure rarely occurs in practice. This appears to be because ‘good solutions’ are quite prevalent in weight space. Were it not for this rather fortuitous fact, the backpropagation algorithm would not be nearly so useful and Minsky and Papert’s doubts

about the existence or practicality of locally computed multi-layer learning algorithms for feedforward networks might have proved correct.

2.3.3 Learning rate

To achieve convergence with a local minimum we must set η sufficiently small so that the error E will decrease at each successive step to satisfy the condition

$$\frac{\delta E}{\delta w_{ij}} = \mathbf{0} \quad \forall w_{ij} \tag{2.17}$$

Then $\mathbf{w}$ in (2.17) will hopefully correspond to the global minimum (or at least a local minimum sufficiently close to the global minimum). We would expect a steady reduction in error for a sufficiently small η because the average direction in weight space should be approximate to the negative of the local gradient. We need to be aware that the optimum value of η will typically change during the minimisation process.

2.4 Conjugate gradient descent

A full account of conjugate gradient descent can be found in [Bishop, 1996]. What follows is a summary of the technique.

2.4.1 Conjugate directions

The concept of conjugate directions is illustrated in Figure 2.7. Consider a minimisation along $\mathbf{d}_j$ which is achieved when point $\mathbf{w}_{j+1}$ is reached. At this stage a new direction $\mathbf{d}_{j+1}$ is chosen such that the new direction is conjugate, i.e. the gradient parallel to the direction $\mathbf{d}_j$ remains zero (indicated by the dashed line). The curved dotted lines represent the contours of the function.

To achieve successive conjugate search directions, the gradient $\mathbf{g} \equiv \nabla E(\mathbf{w})$ of the error surface at the next point must be a minimum in the current search direction $\mathbf{d}_j$. This is satisfied when

$$\mathbf{d}_{j+1}\mathbf{H}\mathbf{d}_j = 0 \tag{2.18}$$

where $\mathbf{H}$ is the Hessian matrix evaluated at the point $\mathbf{w}_{j+1}$. Search directions which satisfy (2.18) are said to be *conjugate*.

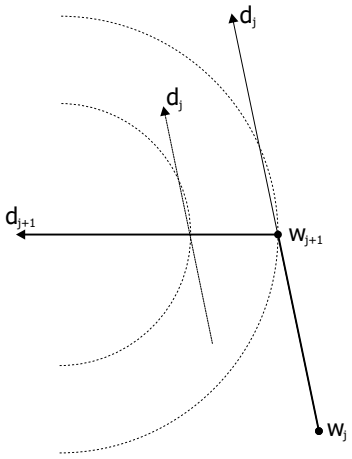


Figure 2.7: Conjugate directions.

When choosing successive search directions $\mathbf{d}_{j+1}$ it is possible to express (2.18) for a set of conjugate search directions (up to the dimensionality W of the weight space) where each direction is conjugate to all others

$$\mathbf{d}_j^T \mathbf{H} \mathbf{d}_i = 0 \quad j \neq i \tag{2.19}$$

Once a search direction has been established we can use a line search technique to find the minimum along the search direction.

2.4.2 Line search

Once we have established along which direction $\mathbf{d}_j$ from $\mathbf{w}_j$ we need to minimise, we can apply a line search algorithm to perform a 1-dimensional minimisation of the error surface to give the new weight $\mathbf{w}_{j+1}$. The new weight is generated by

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_j \mathbf{d}_j \tag{2.20}$$

where the parameter α_j is calculated using a line search technique such that

$$\mathbf{g}_{j+1} \equiv \nabla E(\mathbf{w}_j + \alpha_j \mathbf{d}_j) = \mathbf{0} \tag{2.21}$$

is a minimum.

A full description of *Brent's Method,* a robust line search algorithm that uses inverse parabolic interpolation to perform the line minimisation, is provided in [Press et al., 1992].

2.4.3 Search direction

We have introduced the method to generate conjugate directions to reach the minimum of a quadratic in W steps using the Hessian (2.19).

Now we need to consider the practical implications of constructing the set of mutually conjugate directions. This can be achieved by selecting the first direction to be the negative gradient $\mathbf{d}_1 = -\mathbf{g}_1$ and then choosing each successive direction to be a linear combination of the current gradient and the previous search direction

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j \tag{2.22}$$

where the co-efficients β_j can be found by using the conjugacy condition to give

$$\beta_j = \frac{\mathbf{g}_{j+1}^T(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{g}_j^T \mathbf{g}_j} \tag{2.23}$$

when expressed in the *Polak-Ribiere* form (considered the superior conjugate gradient algorithm [Press et al., 1992]). Note that we have reformulated the method to avoid calculating the Hessian.

2.4.4 Algorithm

What should be apparent is that we have managed to derive a procedure for finding the next search direction and the required step size, all without explicitly using the Hessian.

The conjugate gradient descent procedure is expressed formally in Algorithm 2.

```
{initialisation}
j = 1
choose an initial weight vector w(j)
compute the gradient vector g(j) at w(j)
set initial search direction d(j) = -g(j)
perform line minimisation along search direction d(j)
compute w(j + 1)

{main loop}
while the stopping criteria has not been reached do
  j = j + 1
  compute the new gradient vector g(j) at w(j)
  compute the new search direction d(j)
  perform line minimisation along search direction d(j)
  compute w(j + 1)
end while
```

Algorithm 2: Conjugate gradient descent.

2.5 BFGS (Quasi-Newton) method

The BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm is a *variable metric* or *quasi-Newton* method. Consider again the quadratic error function evaluated at $\mathbf{w}$ near to the minimum $\mathbf{w}^*$

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*) \tag{2.24}$$

The location of the minimum can be determined directly by differentiating (2.24)

$$\mathbf{g} \equiv \nabla E(\mathbf{w}) = \mathbf{H}(\mathbf{w} - \mathbf{w}^*) = \mathbf{0} \tag{2.25}$$

to give an expression for the minimum $\mathbf{w}^*$

$$\mathbf{w}^* = \mathbf{w} - \mathbf{H}^{-1}\mathbf{g} \tag{2.26}$$

The vector $-\mathbf{H}^{-1}\mathbf{g}$ is the *Newton direction* and when evaluated at any $\mathbf{w}$ on a quadratic error surface will point to the minimum of the error function $\mathbf{w}^*$.

2.5.1 Updating the weights

The algorithm is iterated to minimise the error surface since, in reality, it will only be approximately quadratic near to a minimum

$$\mathbf{w}_{j+1} = \mathbf{w}_j - \mathbf{H}^{-1}\mathbf{g}_j \tag{2.27}$$

If we consider the relationship between the weight and gradient vectors generated at two successive steps then we derive the *quasi-Newton condition*

$$\mathbf{w}_{j+1} - \mathbf{w}_j = -\mathbf{H}^{-1}(\mathbf{g}_{j+1} - \mathbf{g}_j) \tag{2.28}$$

The computational cost of generating the inverse Hessian is prohibitive so quasi-Newton algorithms operate by iteratively generating more accurate approximations to the inverse Hessian matrix $\mathbf{G}$. The approximation must be constructed to satisfy the quasi-Newton condition (2.28).

2.5.2 Approximating the inverse Hessian matrix

The BFGS expression provides a way to iteratively estimate the inverse Hessian matrix

$$\mathbf{G}_{j+1} = \mathbf{G}_j + \frac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{v}} - \frac{(\mathbf{G}_j\mathbf{v})\mathbf{v}^T\mathbf{G}_j}{\mathbf{v}^T\mathbf{G}_j\mathbf{v}} + (\mathbf{v}^T\mathbf{G}_j\mathbf{v})\mathbf{u}\mathbf{u}^T \tag{2.29}$$

where

$$\mathbf{p} = \mathbf{w}_{j+1} - \mathbf{w}_j \tag{2.30}$$

$$\mathbf{v} = \mathbf{g}_{j+1} - \mathbf{g}_j \tag{2.31}$$

$$\mathbf{u} = \frac{\mathbf{p}}{\mathbf{p}^T\mathbf{v}} - \frac{\mathbf{G}_j\mathbf{v}}{\mathbf{v}^T\mathbf{G}_j\mathbf{v}} \tag{2.32}$$

We can verify that the BFGS method does satisfy the quasi-Newton condition through direct substitution of $\mathbf{p}$, $\mathbf{v}$ and $\mathbf{u}$ into (2.29).

The algorithm is initialised by setting $\mathbf{G}$ equal to the identity matrix, taking the first step down the steepest gradient.

2.5.3 Line search

Line search is used to ensure that the Newton step (2.26) does not take the algorithm outside of the quadratic approximation

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_j\mathbf{G}_j\mathbf{g}_j \tag{2.33}$$

where α_j is found by line search. The line search ensures that successive iterations of the algorithm reduce the error. One technique that can be used to perform the line minimisation is *Brent's Method* [Press et al., 1992].

2.5.4 Algorithm

The BFGS algorithm is described in Algorithm 3.

2.6 Local-linear regression

Local-linear regression performs linear regression through the p_{max} nearest points to a query point to produce a linear model in the locality of that query point. This process is repeated across the

```
{initialisation}
j = 1
set the inverse Hessian to the identity matrix G(j) = I
choose an initial weight vector w(j)
compute the gradient vector g(j)

{main loop}
while the stopping criteria has not been reached do
  compute the search direction d(j) = G(j)g(j)
  perform line minimisation in search direction d(j) to compute w(j+1)
  compute the gradient vector g(j+1)
  update the inverse Hessian G(j+1) according to the BFGS method
  j = j + 1
end while
```

Algorithm 3: The BFGS algorithm.

training data to produce a piece-wise linear model. One of the many methods available to perform a nearest neighbour search is the k-d tree described in Appendix A.

Given a neighbourhood of p_{max} points we must solve the linear matrix equation

$$\mathbf{Xm} = \mathbf{y} \tag{2.34}$$

where $\mathbf{X}$ is a $p_{max} \times d$ matrix of the p_{max} input points in d-dimensions, $\mathbf{x}_i$ $(1 \leq i \leq p_{max})$ are the nearest neighbour points, $\mathbf{y}$ is a column vector of length p_{max} of the corresponding outputs, and $\mathbf{m}$ is a column vector of parameters that must be determined to provide the optimal mapping from $\mathbf{X}$ to $\mathbf{y}$, such that

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1d} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{p_{max}1} & x_{p_{max}2} & x_{p_{max}3} & \cdots & x_{p_{max}d} \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_d \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{p_{max}} \end{pmatrix} \tag{2.35}$$

The *rank* r of the matrix $\mathbf{X}$ is the number of linearly independent rows, which will affect the existence or uniqueness of solutions for $\mathbf{m}$.

If the matrix $\mathbf{X}$ is square and non-singular then the unique solution to (2.34) is $\mathbf{m} = \mathbf{X}^{-1}\mathbf{y}$. If $\mathbf{X}$ is not square or singular then we modify (2.34) and attempt to find a vector $\mathbf{m}$ which minimises

$$|\mathbf{Xm} - \mathbf{y}|^2 \tag{2.36}$$

As was proved by Penrose the unique solution to this problem is provided by $\mathbf{m} = \mathbf{X}^{\#}\mathbf{y}$ where $\mathbf{X}^{\#}$ is the pseudo-inverse matrix [Penrose, 1955], [Penrose, 1956].

For any given matrix $\mathbf{X} \in \mathbb{R}^{p_{max} \times d}$, the matrix $\mathbf{X}^{\#} \in \mathbb{R}^{d \times p_{max}}$ is said to be a *pseudo-inverse* of $\mathbf{X}$ if the following conditions are satisfied

$$\begin{aligned} \mathbf{X}\mathbf{X}^{\#}\mathbf{X} &= \mathbf{X} \\ \mathbf{X}^{\#}\mathbf{X}\mathbf{X}^{\#} &= \mathbf{X} \\ (\mathbf{X}\mathbf{X}^{\#})^T &= \mathbf{X}\mathbf{X}^{\#} \\ (\mathbf{X}^{\#}\mathbf{X})^T &= \mathbf{X}^{\#}\mathbf{X} \end{aligned} \tag{2.37}$$

where T denotes the transpose of the matrix. The terms *generalised inverse* or *Moorse-Penrose inverse* are also commonly used for such an $\mathbf{X}^{\#}$.

If $\mathbf{X}$ is square and non-singular then $\mathbf{X}^{\#}$ is just the inverse matrix $\mathbf{X}^{-1}$. In practice, the computation of $\mathbf{X}^{\#}$ is modestly demanding for large matrices. There are many algorithms for approximating pseudo-inverses [Kerr, 1985] and [Penrose, 1955].

A generalised technique to solve (2.34) for $\mathbf{m}$ is *singular value decomposition* (SVD), which is a computationally expensive but a widely accepted technique for its accuracy. Both [Press et al., 1992] and [Cherkassky and Mulier, 1998] provide good introductions to linear algebra and SVD, especially in the wider context of learning from data. For the purpose of this discussion of modelling techniques, we shall focus on introducing SVD for local-linear regression and side-step the more general subject of linear algebra.

2.6.1 Singular value decomposition

[Tsui, 1999] has taken the separate theories given in [Press et al., 1992][5] to provide a unified account of SVD, where the context of the discussion is similar to this thesis. The detail contained within that thesis will not be replicated here.

SVD is based on a generalisation in linear algebra that any symmetric matrix can be diagonalised via an orthogonal transformation. This leads to a technique to obtain the inverse of a non-singular square matrix, in this case $\mathbf{X}^{-1}$. SVD also solves the linear least squares approximation (2.36) without requiring $\mathbf{X}$ to be non-singular or even square.

From (2.34), $\mathbf{X}$ is an $p_{max} \times d$ matrix that can be written, using a standard theorem of linear algebra, as

$$\mathbf{X} = \mathbf{U}\mathbf{W}\mathbf{V}^T \tag{2.38}$$

[5] [Press et al., 1992] focus on providing an algorithm rather than defining the theory behind SVD in a contained manner.

where $\mathbf{U}$ is a $p_{max} \times d$ orthogonal[6] matrix, $\mathbf{V}$ is a $d \times d$ orthogonal matrix, and $\mathbf{W}$ is a $d \times d$ matrix with positive or zero elements (singular values) w_j, such that $w_1 \geq w_2 \geq \ldots \geq w_r > 0$, where r is the rank of $\mathbf{X}$. Then rewriting $\mathbf{W}$

$$\mathbf{W} = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} \tag{2.39}$$

where

$$\mathbf{A} = diag(w_1, \ldots, w_r) \tag{2.40}$$

provides a definition for the pseudo-inverse $\mathbf{X}^{\#}$

$$\mathbf{X}^{\#} = \mathbf{V} \begin{pmatrix} A^{-1} & 0 \\ 0 & 0 \end{pmatrix} \mathbf{U}^T \tag{2.41}$$

Then to find a solution for $\mathbf{m}$ that minimises (2.36), let

$$\begin{aligned} \mathbf{m} &= \mathbf{X}^{\#}\mathbf{y} \\ &= \mathbf{V} \begin{pmatrix} A^{-1} & 0 \\ 0 & 0 \end{pmatrix} \mathbf{U}^T\mathbf{y} \end{aligned} \tag{2.42}$$

where

$$\mathbf{A}^{-1} = diag(1/w_1, \ldots, 1/w_r) \tag{2.43}$$

The inverse of the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ are their own transpose (i.e. $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ and $\mathbf{V}\mathbf{V}^T = \mathbf{I}$). Hence the process of finding $\mathbf{m}$ using (2.42) is trivial once $\mathbf{X}$ is decomposed using (2.38). It is this decomposition of $\mathbf{X}$ to generate $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ that now needs explanation.

The columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{X}\mathbf{X}^T$, the columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{X}^T\mathbf{X}$, and the singular values on the diagonal of $\mathbf{W}$ are the square roots of the eigenvalues of $\mathbf{X}\mathbf{X}^T$ or $\mathbf{X}^T\mathbf{X}$ (they have the same eigenvalues). The process of extracting the eigenvectors and eigenvalues is more difficult to explain and beyond the scope of this general introduction to modelling. The full computational process to perform the decomposition of $\mathbf{X}$ given in (2.38) is provided in [Tsui, 1999] and [Press et al., 1992] provide an algorithm written in C.

2.6.2 Dynamic local-linear regression

A local-linear regression model constructed from training data is fixed (or static) once constructed. However, a simple modification can be made to take account of newly available data to give the

[6] A matrix is orthogonal if its inverse equals its transpose.

model a dynamic behaviour. This dynamic behaviour is relatively straightforward to implement: after the model is tested with a new input-output data point, the new test point is added to the training data. When the next query is made to the model, the new point is available to use in the local-linear regression.

The dynamic nature of the model makes it ideal where initially limited training data is available, or where rapid learning is required (a neural network would have to be re-trained to take account of new data).

2.7 Comparison of modelling techniques

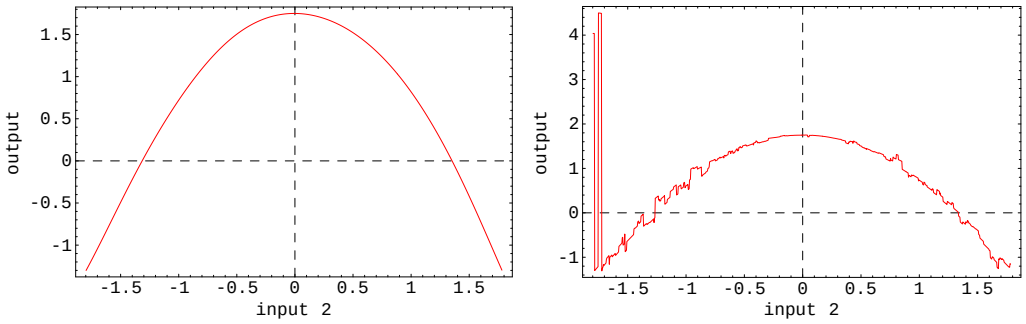
We have discussed two techniques for modelling: *feedforward neural networks* and *local-linear regression*. This section provides some practical information regarding the main differences.

Local-linear regression models do not require training in the same way that neural network models do, although they do require a nearest neighbour algorithm to operate efficiently. If a k-d tree (see Appendix A) is chosen as the nearest neighbour algorithm then the time complexity of model construction is $O(M \log M)$ for M training points (this is typically many orders of magnitude faster than training a neural network). In addition to the nearest neighbour algorithm, the local-linear regression models need to consist of the training data (since that defines the model's experience) and some modelling parameters. During testing, the p_{max} nearest neighbours are found for each query point, so for a training set consisting of M vectors and a test set of N vectors a full test has a time complexity of $O(N \log M)$ with additional complexity due to the dimensionality d of the input-space. The resultant local-linear regression model is a piece-wise linear model producing an approximation of the underlying smooth model.

Neural network models require training and this can be a time consuming process for large or complicated data sets. However, in comparison to the local-linear regression models, testing is much faster due to the simpler calculations involved. Contrary to local-linear regression, the network produces a smooth non-linear model.

Figure 2.8 shows a cross-section through a neural network and local-linear regression model trained using the same data.

The prediction performance of a local-linear regression model is limited by the number of example data points in the vicinity of the query point. If the nearest neighbours are distant then the



(a) A neural network provides a smooth non-linear model.

(b) A local-linear regression model produces a piece-wise linear model that approximates a smooth function.

Figure 2.8: A comparison between neural networks and local-linear regression.

prediction quality is very low. This is evident in Figure 2.8(b) where the spikes on the LHS of the plot are produced because the query point has no reliable near neighbours for reference.

2.8 Conclusions

There are many neural network training algorithms ranging from the *original* backpropagation, which relies on a gradient descent minimisation of the error surface with respect to the network weights, to the more mathematically advanced optimisation routines using the Hessian matrix and conjugate gradient descent. For most neural network applications it is now unrealistic to think that backpropagation provides the best training algorithm since the performance is much lower than the alternatives.

Regardless of the technique used to train a neural network it will always produce a smooth non-linear model. The model will often behave reasonably well outside of the region for which it was trained. Local-linear regression models on the other hand perform a piece-wise linear regression that at best provides a close approximation to a non-linear surface, but will never provide a smooth or continuous function. Local-linear regression models also cannot extrapolate beyond the region enclosing the training data. This is a particular shortcoming of any data-derived model, but for a nearest neighbour modelling routine it is especially difficult to overcome.

Local-linear regression models provide some benefit for rapid modelling. They do not require training, unlike a neural network, and although testing a model can be lengthy it is rarely prohibitive. It is also very easy to add new data points to the nearest neighbour structure (at least for

a k-d tree) allowing the model to adapt when new data becomes available. In comparison a neural network would require a period of re-training to achieve a similar dynamic effect.

It can often be difficult to train a neural network to an appropriatley low error when the data justifies it. Sometimes this can be achieved for local-linear regression models when there is a high density of data, which is not an unreasonable assumption when working with artificially produced data or data produced from instrumentation.

CHAPTER 3

The Gamma Test

In this chapter we present the *Gamma test* as a non-linear data analysis routine to be used prior to modelling. A more general background to the problem of predictive learning and function approximation can be found in [Friedman, 1994].

To construct a successful model we must capture the systematic behaviour of the observed system. For a data-derived model this can be achieved by sampling the system at characteristic points or intervals to create a representative data set that can later be used to build the model. However, there may be several reasons why the systematic behaviour cannot be fully captured within the data: there may be measurement errors, the system may contain noise, for time series the sampling rate may be too low, or perhaps the relevant variables were not measured[1].

The *Gamma test* is a smooth non-linear data analysis technique that can measure the extent to which the systematic behaviour of a smooth system can be captured. In the most basic sense it can be used to determine the variance of the noise in a data set. However, the application of the Gamma test can be applied to embedding dimension search and feature extraction. More generally it appears that, given sufficient data, an elegant extension of the Gamma test can be used to estimate *all* the higher *even* moments of the noise distribution, or at least as many as is justified by the amount of data. These techniques are discussed in more detail in Chapters 4 and 5 respectively.

The premise for the Gamma test arose from the definitions of *continuity* and *smoothness*, and a

[1] We shall use the term *noise* to collectively refer to all of these aspects (unless otherwise stated).

paper by [Pi and Peterson, 1994]. The idea being that if f is a continuous function then, in the absence of noise, if inputs $\mathbf{x}(i)$ and $\mathbf{x}(j)$ are close we should expect $y(i)$ and $y(j)$ to be close, where $y(i) = f(\mathbf{x}(i))$ and $y(j) = f(\mathbf{x}(j))$. If $y(i)$ and $y(j)$ are *not* close it can only be because of the presence of noise. The Gamma test attempts to quantify this observation using the average distance between near neighbours $\mathbf{x}(i)$ and $\mathbf{x}(j)$ and the corresponding average distance between $y(i)$ and $y(j)$ to derive an estimate of the amount of noise present.

3.1 Techniques to improve model quality

It is widely recognised that the best models capture only the systematic aspects of the data. Other aspects of the data due to noise are reduced or eliminated. This ability to include only the systematic behaviour is called *generalisation*.

The quality of a model is primarily determined by its ability to generalise for unseen data. There are several secondary considerations that will affect the generalisation capability of the model, such as '*is there enough training data?*', '*are the chosen variables relevant?*', and '*what is the expected model performance?*'.

In this chapter we offer techniques designed to answer these questions and study their practical benefits using several examples.

3.1.1 Generalisation

Early stopping in training is a technique that can be used to facilitate good generalisation. Conventionally this requires two or three data sets[2]: a training set to build the model, a validation set to test the model during training (not necessarily required), and a test set to test the model once training has been completed.

The model is trained using the training set and periodically tested with the validation set. If the training error and the validation error reduce then the training process is continued, otherwise if the validation error rises then training is stopped. This *ad-hoc* technique ensures that the model does not overfit the training data and so should provide a model with good generalisation capabilities[3].

Other techniques designed to enhance or maintain generalisation focus on (i) *structural stabili-*

[2] All data sets are assumed to arise from the same system.

[3] Using a validation set cannot guarantee to improve the generalisation performance.

sation (where the number of adaptive parameters in the network are adjusted), (ii) *regularisation* (where a penalty is added to the error function for less smooth mappings), or (iii) *training with noise* (noise is added to the input vectors to create additional samples).

All of the techniques discussed so far work without any prior knowledge of the amount of noise in the data. An additional approach to improving generalisation has recently arisen because we can now determine the noise present in a data set prior to model building using the Gamma test [Aðalbjörn Stefánsson et al., 1997]. This noise estimate provides a lower bound on the best model performance. This approach has a number of advantages:

1. The noise can be determined directly from the training data.
2. The best model performance is known prior to model building.
3. The noise estimate provides a stopping criteria for training.
4. The significance of variables can be tested (*feature selection*).
5. The quantity of available data can be tested to see if there is sufficient data to build a successful model.
6. The model complexity can be estimated.
7. The need for a separate validation set can be reduced or eliminated.

To a large extent these advantages of the Gamma test remove the necessity for a validation set.

In addition, existing techniques for model generalisation are enhanced by using the Gamma test to compute in advance what the best model performance can be without overtraining.

3.2 An introduction to the Gamma test

The Gamma test is a near-neighbour data analysis routine that estimates the variance of the noise in continuous data. The inspiration for the Gamma test came from the *Delta test* [Pi and Peterson, 1994] and the definition of continuity: if a function is continuous then near points in input-space should be close in output-space, i.e.

$$\lim_{\underline{\epsilon} \to 0} f(\underline{\epsilon} + \mathbf{x}) = f(\mathbf{x}) \tag{3.1}$$

where f is a continuous function, $\mathbf{x}$ is a point, and $\underline{\epsilon} + \mathbf{x}$ is a point close to $\mathbf{x}$.

3.2.1 A Discussion of non-linear regression

Although many physical phenomena are non-linear and continuous, our measurements of these systems are discrete. We therefore require mathematical techniques to reconstruct the continuum in order to study, predict and control these systems.

We are going to discuss some of the issues relating to the successful fitting of smooth surfaces through discrete data measured from continuous systems. In particular we are going to concentrate our immediate discussion on the continuity and smoothness of the underlying function that generated the data.

3.2.2 Assumptions

The principal assumptions associated with the Gamma test are:

1. The training set inputs are non-sparse in input-space (i.e. as the number of data points increases the first nearest neighbour distances reduce).
2. Each output is determined from the inputs by a deterministic process which is the same for both the training and test sets.
3. Each output is subjected to statistical noise whose distribution may be different for different outputs but which is the same in both training and test sets for corresponding outputs.

3.3 Supporting heuristic arguments

The theoretical proof of the Gamma test has only recently been completed and is the subject of another dissertation [Evans, 2001]. However, we provide an heuristic explanation of the basis of the method and subsequently illustrate the range of applicability using a chaotic time series as a further example.

3.3.1 Introduction

For simplicity we consider the case where we are given data samples $\{(\mathbf{x}(i), y(i)) : 1 \leq i \leq M\}$, where the *inputs* $\mathbf{x}$ (of dimension d) are confined to a closed bounded set C and the scalar *outputs*[4] y are generated by an unknown smooth function $f : C \subseteq \mathbb{R}^d \to \mathbb{R}$

$$y = f(x_1, x_2, \ldots, x_d) + r \tag{3.2}$$

The indeterminable part r may be due to real noise or a lack of functional determination. We shall consider explicitly the case where r is due to real noise (a lack of functional determination can only be remedied by measuring new variables).

We use the Gamma test to determine the variance of the noise $\text{var}(r)$. This provides a lower bound on the mean squared error of the output y (i.e. the variance of $y - f(\mathbf{x})$), beyond which any attempt to improve training will result in over-training. Note that we assume that the noise has zero mean, since a non-zero mean (bias) can be incorporated into the model.

Consider two data samples $(\mathbf{x}, y)$ and $(\mathbf{x}', y')$, where $\mathbf{x}'$ is the first nearest neighbour of $\mathbf{x}$, i.e. $|\mathbf{x}' - \mathbf{x}| > 0$ is minimal[5]. In the absence of noise it is reasonable to consider that y and y' must be close. Here $|\cdot|$ denotes Euclidean distance.

The Gamma test is based on the statistic

$$\gamma = \frac{1}{2M} \sum_{i=1}^{M} (y'(i) - y(i))^2 \tag{3.3}$$

Let $\delta = \max|\mathbf{x}' - \mathbf{x}|$ where the maximum is taken over all data samples ($1 \leq i \leq M$) then it follows from our assumption of non-sparseness that $\delta \to 0$ in probability as $M \to \infty$. Moreover under reasonable conditions, for example if f is continuous, one can show that

$$\lim_{\delta \to 0} \gamma = \text{var}(r) \tag{3.4}$$

where the convergence is *convergence in probability*. This is a highly intuitive result but not quite so easy to establish rigorously.

For finite data sets we cannot have arbitrarily small nearest neighbour distances, however in practice even the crude estimate provided by (3.3) often proves useful.

[4] We shall see later that vector outputs $\mathbf{y}$ are readily accommodated in the Gamma test at very little extra computational cost.

[5] Note that y' is not necessarily the first nearest neighbour of y.

If we assume that f is smooth with bounded first partial derivatives then we can generate a more precise estimate by fitting a regression line using the p_{max} near neighbours.

Given data samples $(\mathbf{x}(i), y(i))$, where $\mathbf{x}(i) = (x_1(i), \ldots, x_d(i))$, $1 \leq i \leq M$, let $N[i,p]$ be the list of p^{th} (equidistant) nearest neighbours[6] to $\mathbf{x}(i)$ then, using a convenient abuse of notation, we can write

$$\begin{aligned} \delta(p) &= \frac{1}{M}\sum_{i=1}^{M}\frac{1}{L(N[i,p])}\sum_{j\in N[i,p]}|\mathbf{x}(i)-\mathbf{x}(j)|^2 \\ &= \frac{1}{M}\sum_{i=1}^{M}|\mathbf{x}(i)-\mathbf{x}(N[i,p])|^2 \end{aligned} \tag{3.5}$$

where $L(N[i,p])$ is the length of the list $N[i,p]$. Thus $\delta(p)$ is the mean square distance to the p^{th} nearest neighbour. We also write

$$\gamma(p) = \frac{1}{2M}\sum_{i=1}^{M}\frac{1}{L(N[i,p])}\sum_{j\in N[i,p]}(y(j)-y(i))^2 \tag{3.6}$$

where the y observations are subject to statistical noise assumed independent of $\mathbf{x}$ and having bounded variance.

If now we assume that the unknown function f is smooth with bounded partial derivatives it can be proved that for any $\kappa > 0$

$$\gamma = \text{var}(r) + A\delta + o(\delta) + O\left(\frac{1}{M^{\frac{1}{2}-\kappa}}\right) \quad \text{as} \quad M \to \infty \tag{3.7}$$

where A is a constant depending on the expectation of $|\nabla f|^2$ with respect to the sampling distribution and the convergence is in probability. This is a much stronger result than (3.4) and very much harder to prove [Evans, 2001]. This result generalised to p^{th} nearest neighbours, for p bounded with respect to M, forms the basis of the Gamma test.

The Gamma test computes the mean-squared p^{th} nearest neighbour distances $\delta(p)$ and the corresponding $\gamma(p)$, where $1 \leq p \leq p_{max}$ and typically $p_{max} \approx 10$. Next the $(\delta(p), \gamma(p))$ regression line is computed and the vertical intercept is returned as the *Gamma statistic*. By virtue of the approximate linear relation (3.7) effectively this is the limit $\lim \gamma$ as $\delta \to 0$, which in theory is $\text{var}(r)$.

[6] In general the p^{th} nearest neighbour of $\mathbf{x}(i)$ may not be unique. If we define $N[i,p]$ as a *list* of indexes corresponding to the set of equidistant p^{th} nearest neighbours of $\mathbf{x}(i)$, this gives an implementation of the algorithm an opportunity to report on unusual data files.

3.3.2 Heuristic explanation

Using the Gamma statistic (3.3), consider the pair $(\mathbf{x}, y)$, $(\mathbf{x}', y')$, where $\mathbf{x}'$ is the first nearest neighbour of $\mathbf{x}$, then for an individual term $(y' - y)^2$ in (3.3) we can write

$$\begin{aligned}(y'-y)^2 &= \big(f(\mathbf{x}') + r' - f(\mathbf{x}) - r\big)^2 \\ &= (r'-r)^2 + 2\big(f(\mathbf{x}') - f(\mathbf{x})\big)(r'-r) + \big(f(\mathbf{x}') - f(\mathbf{x})\big)^2 \end{aligned} \tag{3.8}$$

Using the smoothness assumption on f we can expand the term $f(\mathbf{x}')$ by Taylor's theorem to obtain

$$(y'-y)^2 = (r'-r)^2 + \big((\mathbf{x}'-\mathbf{x})\cdot\nabla f\big)^2 + (r'-r)(\mathbf{x}'-\mathbf{x})\cdot\nabla f + o\left(|\mathbf{x}'-\mathbf{x}|^2\right) \tag{3.9}$$

where r' is the noise associated with the data pair $(\mathbf{x}', y')$.

The first step towards (3.7) is to average (3.9) over the M data points.

Now $(r'-r)^2 = r'^2 - 2r'r + r^2$. Since the introduction of a new data point can be expected to affect only a *small* number of the near neighbour relationships in the input data we might expect that the identically distributed variables $R_i = (r_i' - r_i)^2$ $(1 \le i \le M)$ are 'essentially independent'[7]. We call a sequence of identically distributed random variables $X_1, \ldots, X_M$ *L-dependent* (as $M \to \infty$) if there exists an integer $L \ge 1$ (independent of M) such that any one of the X_i is dependent on at most L of the others. One part of a rigorous proof consists of showing that $R_1, \ldots, R_M$ are indeed L-dependent with $L = 2K(d)$, where $K(d)$ $(= O(2^d))$ is the kissing number in d-dimensional space. It can be shown that one can treat sums of identically distributed L-dependent variables in very much the same way as one can treat sums of identically distributed independent variables, i.e. a form of central limit theorem applies. One important step of a rigorous proof consists in justifying this assertion.

However, if this is true then, when averaged over the M data points, we expect, since r' and r are independent and hence uncorrelated, that in probability the term $\frac{1}{M}\sum R_i$ approaches $2\mathrm{var}(r)$ as $M \to \infty$, i.e. the constant term in the approximate linear relationship (3.7). The second term on the RHS of (3.9) corresponds to the second term on the RHS of (3.7).

The proof now reduces to two cases. In (3.7) if $\delta << M^{-\frac{1}{2}+k}$ as $M \to \infty$ then the linear regression performed by the Gamma test algorithm is unnecessary, the algorithm is simply returning the asymptotic value of γ. Otherwise the terms in δ dominate, the rate of convergence is determined

[7] r_i' and r_i are independent and identically distributed so the distribution of R_i only depends on the distribution of r_i.

by the size of δ in terms of M, but the linear regression in δ will produce a vertical intercept that converges in probability to var(r).

Thus we can see that in many cases the probabilistic rate of convergence of the Gamma test will be determined by the expected size of δ. If C is a chaotic attractor we might expect δ to be small, but suppose the sampling distribution in input space has positive density then we might expect $\delta \approx c/M^{2/d}$, where $c > 0$, as $M \to \infty$.

There is a discussion of such a result for a uniform sampling distribution on a torus in [Cerf et al., 1997]. The difficulty of the analysis is caused by boundary effects, and the advantage of a torus is that it has no boundary. However, in our case C is a closed bounded region of $\mathbb{R}^d$ and the boundary effects are present. Nevertheless it emerges that for a uniform distribution over a closed bounded subset C of $\mathbb{R}^d$ the result remains true. This follows from a deep unpublished theorem of W. Schmidt, which asserts that in probability

$$\frac{1}{M}\sum_{i=1}^{M}|\mathbf{x}'_i - \mathbf{x}_i|^2 = \frac{c(d)}{M^{\frac{2}{d}}} + o\left(\frac{1}{M^{\frac{2}{d}}}\right) \tag{3.10}$$

as $M \to \infty$, and that this holds, subject to some reasonable side conditions on C, under a very wide range of circumstances (see [Evans, 2001]).

This deals with the first near neighbours. If we were to apply the test in this form we should have to progressively increase M until the estimate stabilised.

In fact, computing pairs (δ, γ) for increasing M is not the best or only way to exploit the local linearity of the (δ, γ) curve as $\delta \to 0$. Since for a particular value of M a point $\mathbf{x}'$, which is a first near neighbour of $\mathbf{x}$, is liable to be a second or third near neighbour of $\mathbf{x}$ for larger M, it is reasonable to conjecture that, provided p is small with respect to M, the behaviour of the p^{th} nearest neighbour distance as M increases will not substantially differ from the behaviour of the first nearest neighbour distance.

As long as p remains bounded, rather than recomputing the first nearest neighbour pairs (δ, γ) for increasing M, it is a one shot computation to determine not only the first nearest neighbour for each data vector $\mathbf{x}_i$ but to compute the lists $N[i, p]$ for $1 \leq i \leq M$ consisting of the p^{th} nearest neighbour of $\mathbf{x}(i)$ for $1 \leq p \leq p_{max}$ (where $p_{max} = 10$ say). This leads to the Gamma test in the form described.

3.3.3 Implementation

The Gamma test algorithm is given in Algorithm 4.

```
{initialisation}
generate near neighbour structure (e.g. k-d tree)
for p = 1 to p_max do
  δ(p) = 0
  γ(p) = 0
end for

{main algorithm}
for i = 1 to M do
  generate N[i,p] {find the p_max near neighbours of x(i)}
  for p = 1 to p_max do
    δ(p) = δ(p) + [x(i) − x(N[i,p])]^2
    z(p) = 0
    for j = 1 to L(N[i,p]) do
      z(p) = z(p) + [y(i) − y(N[i,p][j])]^2
    end for
    γ(p) = γ(p) + [z(p)/L(N[i,p])]
  end for
end for

for p = 1 to p_max do
  δ(p) = δ(p)/M
  γ(p) = γ(p)/2M
end for

{Gamma statistic}
Perform least squares fit on (δ(p), γ(p)) where (1 ≤ p ≤ p_max)
  to compute y = Ax + Γ
return (Γ,A)
```

Algorithm 4: The Gamma test algorithm.

The time complexity of the algorithm is determined by the data structure and search algorithm used to find the nearest neighbours. Using a k-d tree [Friedman et al., 1979] the time complexity is $O(M \log M)$, where M is the number of data points and the implied constant also depends on the dimensionality of the input data vectors. The k-d tree structure is discussed in Appendix A.

3.3.4 The Gamma scatter plot

It is possible to visualise the Gamma test. If we define

$$\delta = |\mathbf{x}(i) - \mathbf{x}(j)|^2 \tag{3.11}$$

$$\gamma = \frac{1}{2}(y(i) - y(j))^2 \tag{3.12}$$

for $1 \leq i \neq j \leq M$, then we can plot δ and γ to provide a *cloud* of points that can indicate the noise level visually. Superimposed on this scatter plot, we plot the averaged near neighbour distances $(\delta(p), \gamma(p))$ from (3.5) and (3.6), and perform a linear regression (3.7) through these points. The intercept with the axis at $\delta = 0$ gives the estimate for the variance of the noise, Γ.

We shall illustrate the Gamma scatter plot using a smooth function

$$f(x) = \sin(4\pi x) + \cos(2\pi x) \tag{3.13}$$

Uniformly distributed noise with variance 0.03 was added to the function and sampled at 1000 points in the interval [0,1]. Figure 3.1 shows the underlying smooth function and the sampled 'noisy' points.

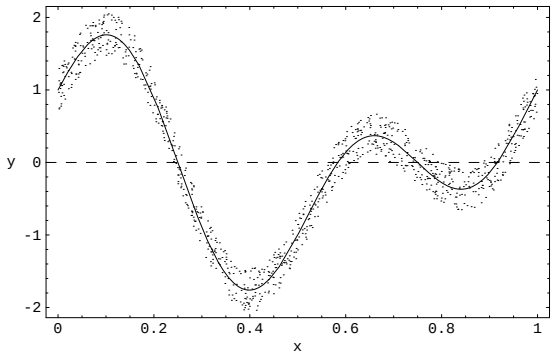


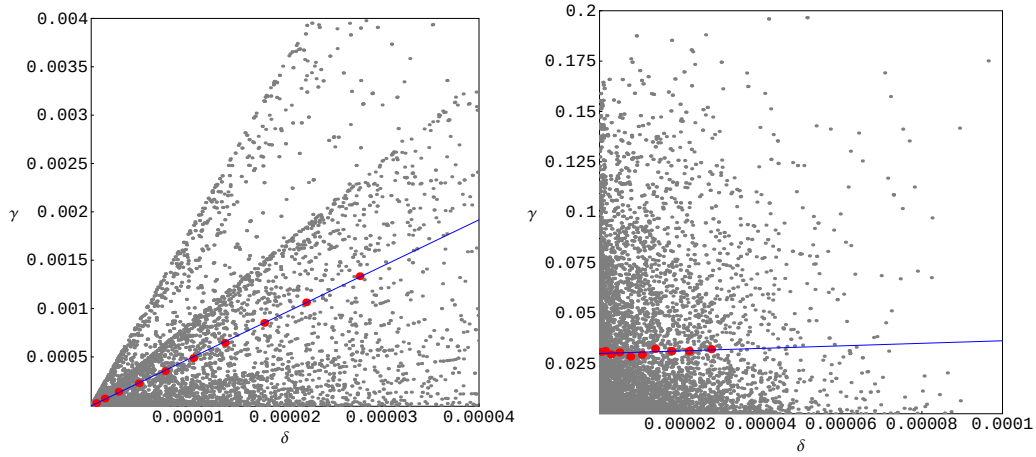
Figure 3.1: The smooth function (3.13) with added uniformly distributed noise (variance $\text{var}(r) = 0.03$, and $M = 1000$ sampled points).

Figure 3.2(a) shows a Gamma scatter plot for the smooth function (3.13) with no added noise. As expected for a noise-free function $\gamma \to 0$ as $\delta \to 0$ and the estimate for the variance of the noise, $\Gamma = 7.53 \times 10^{-7}$.

The form of the Gamma scatter plot changes when the Gamma test is run on the 'noisy' data set. The effect of the noise is apparent in Figure 3.2(b) because as $\delta \to 0$ then $\gamma \not\to 0$. The Gamma statistic for the noise in this example is $\Gamma = 0.0299$ (the actual noise variance $\text{var}(r) = 0.03$).

3.3.5 How reliable is the Gamma statistic: the M-test

The theory assures us that as $M \to \infty$ the result Γ of Algorithm 4 will converge to the true noise variance with probability one. However, this does not in itself tell us how large an M is required to give an accurate estimate of the true noise variance. We really need to know *how quickly* the estimate returned by the algorithm will stabilise to a close approximation of the true noise variance.



(a) Gamma scatter plot for the smooth function (3.13) with no added noise ($M = 1000$ sampled points).

(b) Gamma scatter plot for the smooth function (3.13) with uniformly distributed noise having variance $\mathrm{var}(r) = 0.03$ ($M = 1000$ sampled points).

Figure 3.2: Gamma scatter plots of the smooth function (3.13).

One simple way to accomplish this is to compute the Gamma statistic, Γ, for increasing M and examine the resulting graph to determine whether the graph appears to be approaching a stable asymptote[8]: we call this procedure the M-test.

Fortunately, a single Gamma test is normally a relatively fast procedure so that running an M-test with a suitably selected step size is not a prohibitively time intensive procedure.

Using the illustrative example (3.13), the M-test can determine whether the Gamma statistic stabilises for increasingly large sample sizes. This particular function satisfies the condition that the sampling in input space remains bounded to a closed set C (here we have constrained the sampling to take place in the interval $[0, 1]$). If this was not the case, and $\delta \not\to 0$ as $M \to \infty$, then the Gamma statistic may never stabilise.

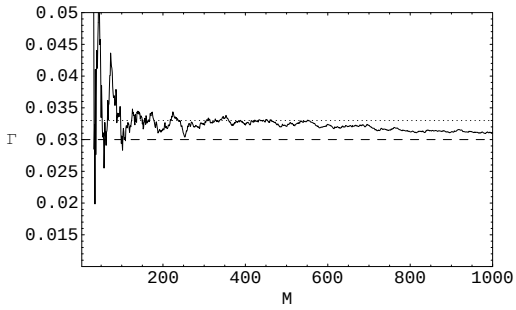
Two M-test experiments were run on sampled points $3 \leq M \leq 1000$ in steps of 1 point and for $1000 \leq M \leq 35000$ in steps of 200 points (where each point in input space was randomly sampled in the interval $[0, 1]$ using a uniform distribution). Figure 3.3 shows the resulting estimates for the Gamma statistic. The M-test for $3 \leq M \leq 1000$ is shown in Figure 3.3(a). The dashed line indicates the theoretical noise variance and the dotted line shows a value at 10% above the

[8] It is easier to do this by eye than to define a reliable and efficient algorithm.

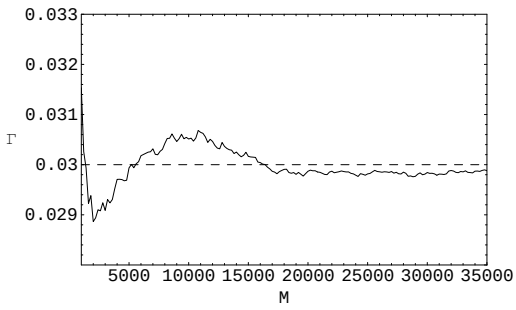
theoretical value. The Gamma statistic for $M > 200$ remains bounded between these two lines, providing a reasonably accurate estimate for low M.

Figure 3.3(b) shows that at $M \approx 18000$ points, the Gamma statistic stabilises to a constant value $\Gamma \approx 0.299$, which corresponds to an error no worse than $\approx 0.3\%$).

This experiment shows that if a *very* accurate estimate for var(r) is required then a significant amount of data is required. However, for most practical applications, an estimate that is $5 - 10\%$ accurate can be achieved with only modest amounts of data (in this case $M \approx 200$).



(a) The M-test shows the Gamma statistic for $M = 1000$ sampled points. The dashed line shows the theoretical noise variance and the dotted line shows a noise variance 10% higher. Γ is bounded by these two lines indicating that a reliable noise estimate can be achieved with less than 1000 data points.



(b) The M-test shows a stabilisation of the Gamma statistic at $M \approx 18000$ sampled points.

Figure 3.3: The M-test for the smooth function (3.13) with added uniformly distributed noise (var$(r) = 0.03$) and a variable M sample size.

Performing an M-test prior to model building can establish whether there is sufficient data to get a reliable Γ estimate. The fact that the graph has stabilised indicates that we have enough information (i.e. data) to accurately estimate the noise and so to construct a feasible surface with the performance corresponding to the measured noise level. The Gamma test itself provides the criterion for ceasing training of a non-parametric model such as a neural network. This is based on the idea that one criterion of a good model is that when tested on unseen data it can be expected to produce a *MSE* which is the same (or close to) the true or estimated noise variance of r associated with the data. We shall return to this discussion of what constitutes a good model in Chapter 5.

3.4 A further example using a chaotic time series

The utility of the Gamma test is illustrated using the Hénon map (a chaotic time series) which provides an interesting function to study without requiring a detailed knowledge of chaos. This section also illustrates that the Gamma test is remarkably robust with respect to the precise nature of the sampling distribution of input space.

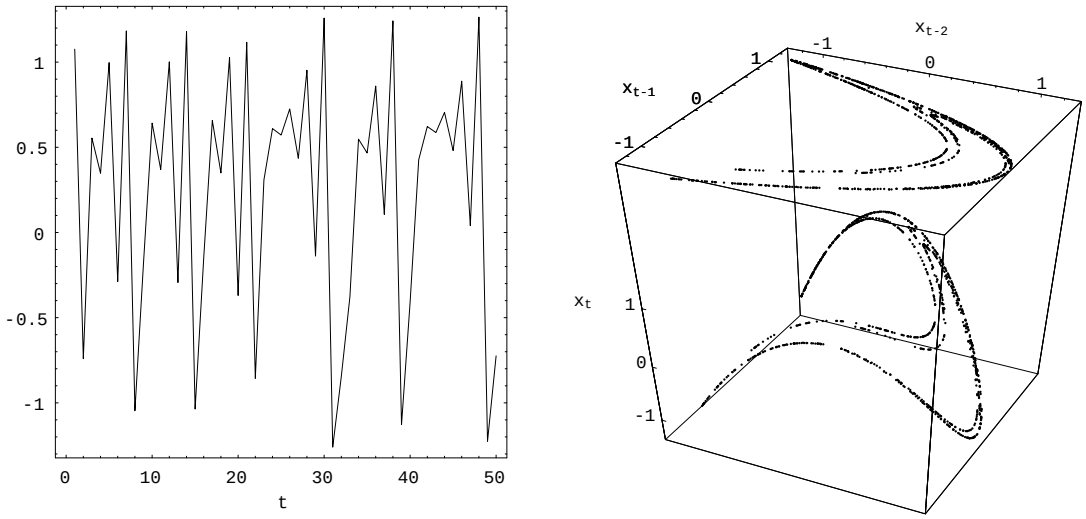
Hénon map

The Hénon map is generated iteratively using the equation

$$x_t = 1 - ax_{t-1}^2 + bx_{t-2} \tag{3.14}$$

where $x_0 = 0$, $x_1 = 0$, $a = 1.4$ and $b = 0.3$.

We can treat the Hénon map as a time series, as shown in Figure 3.4(a).



(a) The first 50 points of the Hénon time series x_t against t.

(b) The Hénon attractor with no added noise. The input space sampling is shown above the attractor.

Figure 3.4: Hénon map.

The points of the map ergodically sample a set of zero measure but positive Hausdorff dimension, called the *attractor* of the map. This can be extracted from the time series data and visualised by simply plotting the inputs to the function against the output as shown in Figure 3.4(b). At the bottom of the diagram (in the 3-dimensional representation), the relationship between the output

x_t and the input variables x_{t-1} and x_{t-2} is shown. This would be the hypothetical surface that we might seek to construct in a modelling exercise where x_t is treated as a function of x_{t-1} and x_{t-2}. At the top of the diagram a projection shows the input variables x_{t-1} and x_{t-2}.

3.4.1 Noise estimation

After generating the noise free data, noise with a known distribution and variance is added to the output only. Figure 3.5 shows the change. Note that the inputs are unaffected by the noise because the noise was added after the data set was constructed[9]. This was done to ensure the Gamma test measured the known noise on the output, and the result was not affected by noise on the inputs.

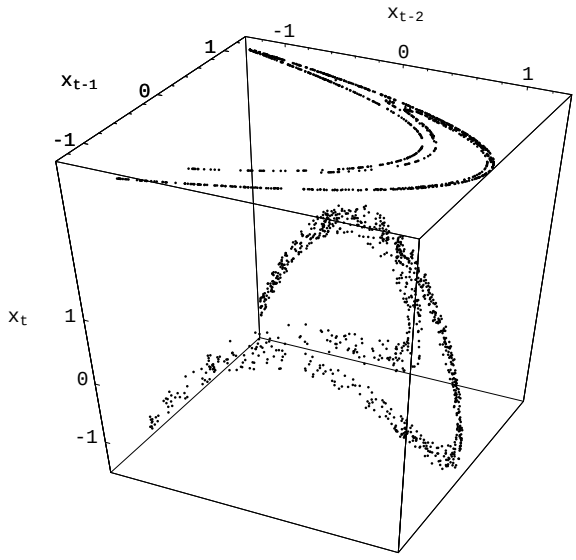


Figure 3.5: The Hénon attractor with uniformly distributed noise added to the output ($\text{var}(r) = 0.01$). The noise-free input space sampling is shown above the attractor.

Experiment description

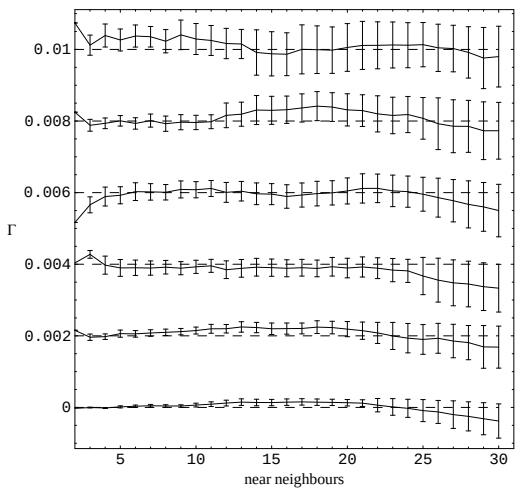
The Hénon map (3.14) was used to generate $M = 1000$ data points using two past values as the input (x_{t-1}, x_{t-2}) and x_t as the output.

A series of data sets was created by adding noise to the output with a uniform distribution with mean zero and variance ranging from 0 to 0.01 in steps of 0.002. This corresponds approximately to a noise amplitude between 0% and 4% of the original signal.

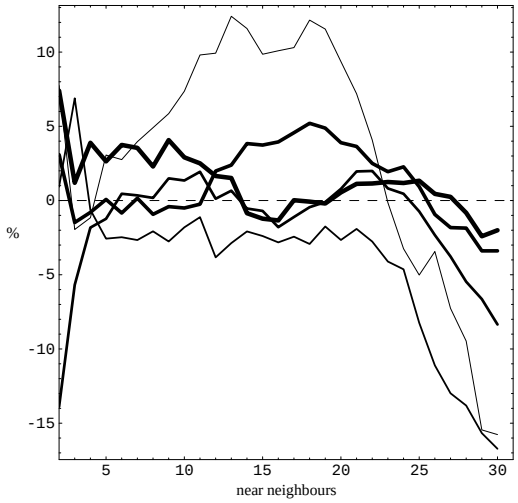
[9]Noise on the inputs produces *effective* noise on the output. The detailed analysis of this interplay would take us beyond the example, although it is an interesting issue.

Results

The Gamma test was run on each data set with 2 to 30 near neighbours. The results of this experiment are shown in Figure 3.6. The dashed lines indicate the noise level of each data set, the solid lines indicate the Gamma statistic for each near neighbour for each data set, and the error bars provide a rudimentary estimate of the error of the estimated noise variance using the standard error (SE) of the regression line fit (see Algorithm 4). We observe that the SE of the regression line fit is not itself a precise estimate of the accuracy of the Gamma statistic, although it seems reasonable to suppose that the two are related. In fact the SE would appear to over-estimate the error in Γ.



(a) Noise variance estimates for data generated from the Hénon map. There are 6 experiments shown for different added noise variance. The dashed lines show the actual noise variance, $\mathrm{var}(r)$, used for each experiment, the solid lines show the Gamma statistic, Γ, and the error bars show the standard error of the regression line fit.



(b) The percentage error of the noise variance estimates for the Hénon map. The thickness of each line corresponds to the actual noise variance added to the signal $(0 = thin, \ldots, 0.01 = thick)$.

Figure 3.6: Noise variance estimates of data generated from the Hénon map.

Figure 3.6 demonstrates the effectiveness of the Gamma test in estimating the variance of the noise r (despite the fact that the underlying function f is unknown) directly from data sampled from a non-linear process. The results show that the percentage error of the noise estimate is quite robust with respect to how many near neighbours are chosen when computing Γ. For a wide range of near neighbours the estimate of the noise variance for each data set was sufficiently close to the

known noise variance to be of practical use.

For fixed M, beyond a certain point the noise estimates get progressively worse as the number of near neighbours increase because the Gamma test starts examining points where the approximately linear relationship (3.7) ceases to hold.

When the experiment is repeated again for 50000 data points with a noise variance of 0.01 on the output, we can improve the stability of the Gamma statistic over a wider range of near neighbours. Even on the increased scale shown in Figure 3.7 the Gamma statistic is stable across the whole range.

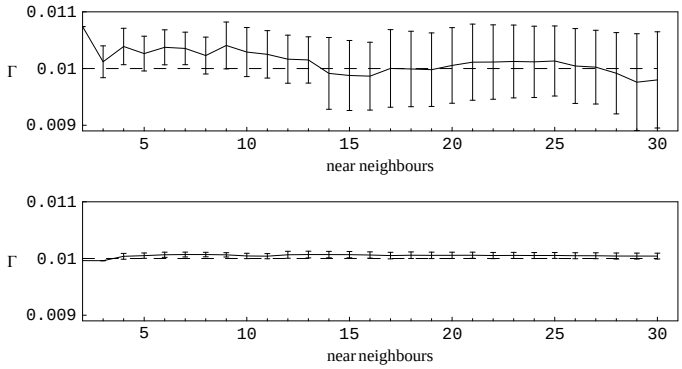
We stated at the beginning of Section 3.3.1 that data samples should be confined to a closed bounded set C. We know that the data samples will remain bounded because a chaotic attractor was used to generate the data, therefore the additional points used in this experiment increased the density of the points on the attractor to satisfy the condition of non-sparseness that $\delta \to 0$ as $M \to \infty$. There are various ways to illustrate this, such as box counting, but we will use the Gamma scatter plots in Figure 3.8 to show the change in average near neighbour distances. Figure 3.8(a) shows how widely spaced the average near neighbours are for $M = 1000$ compared to those shown in Figure 3.8(b) where $M = 50000$. It is interesting to note that the gradient, A, used as a measure of surface complexity is approximately constant irrespective of the number of data points M.

3.4.2 Longer range predictions of chaotic time series

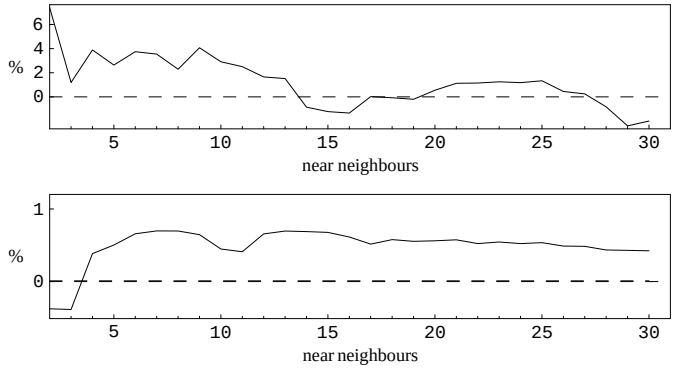
The results of the previous section show that for the Hénon map the functional surface $x_t = f(x_{t-1}, x_{t-2})$ is quite simple and can, in fact, be modelled quite accurately using only a few hundred points.

Whilst accurate *short term* predictions for chaotic time series are quite feasible the nature of chaos is such that as the prediction interval increases, all things being equal, the accuracy of the prediction rapidly decreases. This is graphically illustrated if we consider the complexity of successive surfaces $x_t = f_1(x_{t-1}, x_{t-2})$, $x_{t+1} = f_2(x_{t-1}, x_{t-2})$, ..., $x_{t+k} = f_k(x_{t-1}, x_{t-2})$ as k increases. These surfaces are shown for $1 \leq k \leq 6$ in Figure 3.9. We see that the complexity rises rapidly.

Now suppose that we compute the Gamma statistic for these surfaces. Of course, with no added noise and arbitrarily large amounts of data we should expect the Gamma statistic to approach zero. But suppose we *fix* M and then compute the Gamma statistics. A sparsely sampled complex

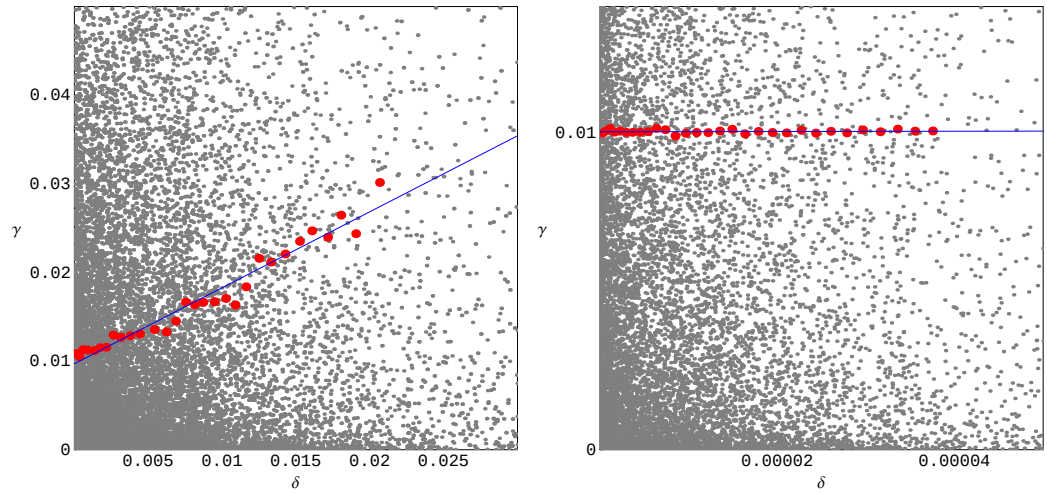


(a) Noise variance estimates of data generated from the Hénon map. The dashed line shows the actual noise variance, the solid line shows the Gamma statistic noise variance, and the error bars show the standard error of the regression line fit. The top chart shows the Gamma statistic for $M = 1000$ sampled points and the lower chart has $M = 50000$ sampled points.



(b) The percentage error of the noise variance estimates for the Hénon map. The top chart shows percentage error for $M = 1000$ sampled points and the lower chart has $M = 50000$ sampled points.

Figure 3.7: Confidence of the noise estimates for the Hénon map.



(a) $M = 1000$ attractor points, $A = 1.0988$. (b) $M = 50000$ attractor points, $A = 1.0841$.

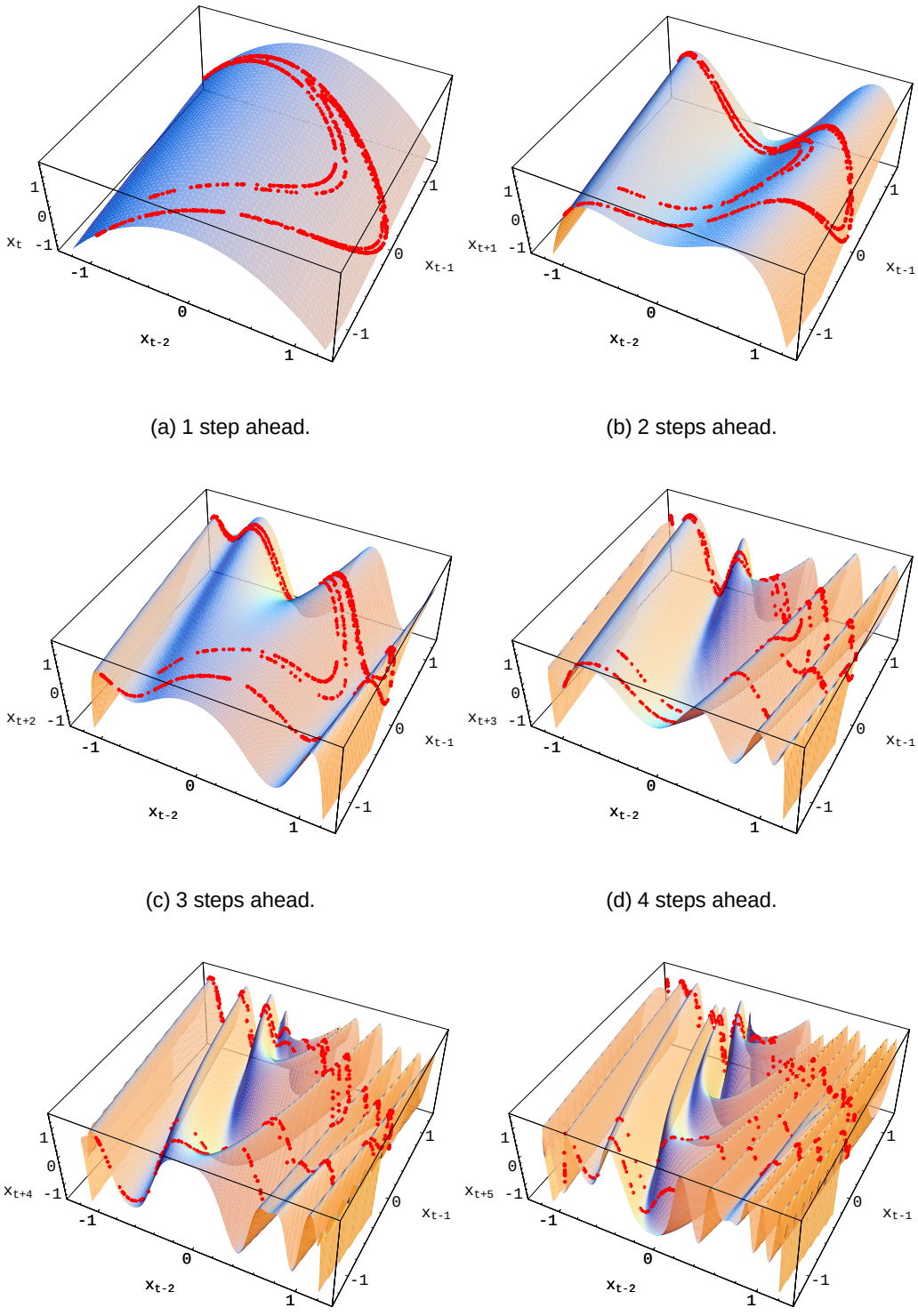
Figure 3.8: Gamma scatter plots for the Hénon map with added noise ($\text{var}(r) = 0.01$).

surface might be expected to give results comparable to a simpler surface sampled with noise.

The results of this experiment are shown in Figure 3.10. The Gamma statistic, Γ, is plotted in Figure 3.10(a) showing the number of sampled data points M against the number of steps ahead (k) for the Hénon map. We see that, for fixed M, as k increases the Gamma statistic rises rapidly. The graph also shows that the rate of increase in Gamma is dependent on the value of M (the Gamma statistic converges to zero for sufficiently large M). Moreover, the slope estimate A in Figure 3.10(b) (see Algorithm 4) also increases rapidly. Taken together, these observations are a useful indicator of chaos[10].

The rudimentary error estimate for the Gamma statistic using the SE of the regression line fit shown in Figure 3.10(c) (discussed previously in conjunction with Figure 3.7) demonstrates that it is much harder to accurately estimate the noise variance for complex functions, in this case for a Hénon map with large k.

[10] A more conventional indicator of chaos is to compute the Lyapounov exponents. However, this process is computationally expensive and (what is worse) existing algorithms provide no associated estimate of the errors. Thus assertions of chaos based on estimates of the Lyapounov exponents must be treated with caution.



(a) 1 step ahead.

(b) 2 steps ahead.

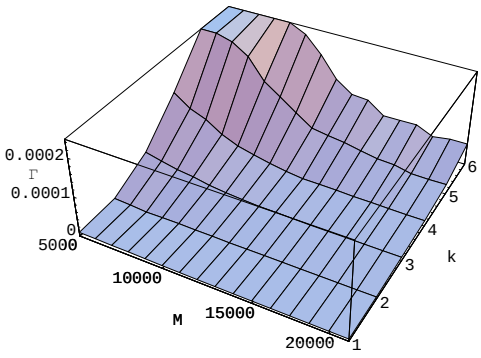
(c) 3 steps ahead.

(d) 4 steps ahead.

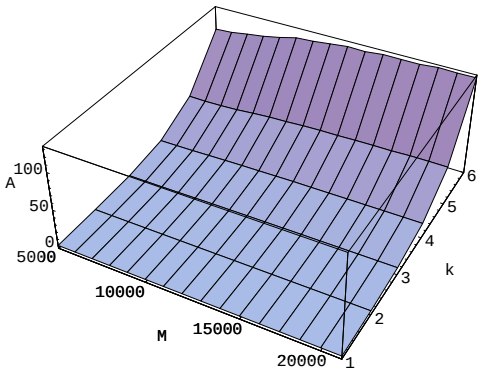
(e) 5 steps ahead.

(f) 6 steps ahead.

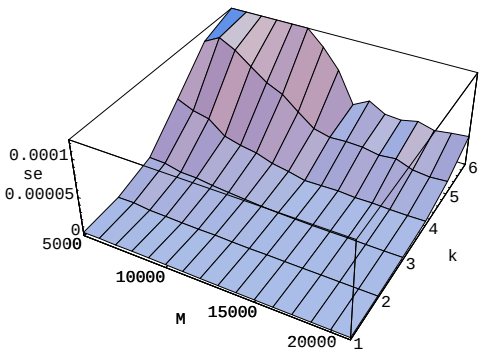
Figure 3.9: Longer range predictions of the Hénon map.



(a) The Gamma statistic Γ. For large sample sizes M the effective noise variance reduces indicating that long range predictions (high k) are possible.



(b) The Gradient A. Short range predictions (low k) are situated on less complicated surfaces than long range predictions (high k). The increasing sample size M provides little benefit for estimating the surface complexity A in this example.



(c) The SE of the regression line fit. A larger sample size M provides a more accurate estimate of the Gamma statistic.

Figure 3.10: Analysis of long range predictions of the Hénon map for increasing M and k ($p_{max} = 10$ near neighbours).

3.4.3 Conclusions

In this chapter we have described the Gamma test for estimating noise in smooth non-linear systems and briefly sketched some of the issues associated with the theoretical justification of this algorithm.

We described some of the associated visualisations developed for the Gamma test, such as the scatter plot, which (quite independently of the Gamma test) provide useful diagnostic tools for the examination of non-linear data.

We have provided several examples which illustrate the relative robustness of the accuracy of the test with respect to the number of near neighbours used and the nature of the sampling distribution in input space. We also illustrated how the Gamma test can provide a relatively efficient indicator test for the presence of chaos in a time dependent non-linear process.

In the next chapter we shall further examine the utility of the Gamma test in the process of feature selection and model building.

CHAPTER 4

Data Analysis using the Gamma Test

This chapter demonstrates how the Gamma test can assist with non-linear data analysis as a precursor to modelling. The techniques shown here are used to illustrate that non-linear data sets can be assessed for quality, used to select important features, and determine whether there is sufficient data to construct a smooth non-linear model.

Although the Gamma test is, in the first instance, an algorithm designed to estimate noise it can be used very effectively to select relevant features for a non-linear model in both noisy and low or zero noise situations. Our first examples are designed to illustrated why the Gamma test can be used to select relevant features in a zero noise case. We then add noise to determine how the ability to effectively select features degrades under moderate or high noise levels. In these examples the sampling distribution over the input space is uniform.

Finally we illustrate how the Gamma test can be used for feature selection in a zero noise chaotic time series. In this case feature selection corresponds to the selection of appropriate lags in an embedding model, and the sampling distribution over the input space corresponds to ergodic sampling of a fractional dimension attractor. We also compare the initial estimate of the embedding dimension from the Gamma test using an 'increasing embedding' with the more conventional 'false nearest neighbour' algorithm.

4.1 Feature selection

The technique of feature selection is used to extract useful information (or features) from a data set. Redundant or irrelevant variables in the data should be excluded. With the Gamma test we can define useful information as being those inputs which contribute to lowering the noise estimate of our input-output data set. In theory, the combination of inputs with the lowest noise estimate will provide the best model. We shall see using a series of examples that this is the case. In a mathematical context the features correspond to the *independent variables* and the output corresponds to the *dependent variable*.

Feature selection algorithms have two main components: a *criterion function* and a *search strategy* [Scherf and Brauer, 1997]. The criterion function determines how good a particular feature set is and the search strategy decides which set to try next.

The search through feature space has to be performed to ensure that all (or as many) combinations of inputs are tested within reasonable computational time. For a small number of inputs, for example up to 10-20, all possible combinations can be tested. In general, for d inputs, there are $2^d - 1$ combinations of those inputs[1].

For larger data sets, or for rapid feature selection, an heuristic search technique must be applied. The primary technique that we propose uses a genetic algorithm [Holland, 1975]. Other techniques involve hill-climbing and similar heuristics. We must recognise that these heuristic methods are not guaranteed to find the best possible feature set.

Whatever search strategy we choose, we clearly need an efficient criterion function. The two main types are *filters* and *wrappers* [Pfleger et al., 1994]. A wrapper uses a model to evaluate the feature set: the performance of a model constructed using the chosen features determines the significance of the feature set. One aspect of this thesis attempts to show that the Gamma test has made this method redundant in a number of cases.

The filter method does not rely on model building for the evaluation of a set of features. Instead, it uses the data directly to evaluate a given feature set. Our intention is to show that the Gamma test performs this task and has other benefits, such as determining the variance of the noise. For reference, some other examples of filter methods are described in [Cherkauer and Shavlik, 1995]).

[1] To perform a full search of 10 inputs requires $1,023$ Gamma test experiments, whereas $1,048,575$ experiments are required for 20 inputs.

4.1.1 Masks: describing a combination of inputs

We describe feature sets in an efficient way using a mask. For any given data set, the inputs $(x_1, x_2, x_3, \ldots, x_d)$ can be masked off to describe a subset of inputs. Using a 4 input data set as an example, we can describe the selection of input x_4 using the mask 0001, and the selection of all the inputs using the mask 1111. We use this representation within the context of feature selection to describe which inputs are used (1) and which are not (0).

4.2 Complete feature space search

A complete feature space search requires all possible combinations of inputs to be analysed.

4.2.1 2-dimensional input space

In these examples we try to provide an intuitive explanation of why, even in the zero noise case, the Gamma test can be used as an effective tool for the selection of relevant input variables.

We consider sections through two 3-dimensional objects: a cylinder, and a cone. 500 data points were sampled uniformly in input space across each surface to produce a 3-dimensional data structure of inputs (x, y) and an output z.

The cylinder

Figure 4.1 shows the section of the cylinder. The height of the cylinder, z, is dependent only upon x and *independent* of y. In this case we might expect to find that the best input is x, the combination of inputs (x, y) is only marginally less effective, but that the choice of the single input y leads to a very poor result.

The results of a search of feature space are shown in Table 4.1. We see that the Gamma statistic $\Gamma = -0.000249$ for the input x is very close to zero. The corresponding result for the combination (x, y) is somewhat larger at $\Gamma = 0.0089$, whereas the result for the single input y is $\Gamma = 5.78$.

The result for input y was dramatically higher because there is *no* relationship between y and z. If we treat z as a function of y alone the variation of z due to changes in x appears as a form of noise, reflecting our ignorance of x. In fact the variance of this 'effective noise' corresponds to the

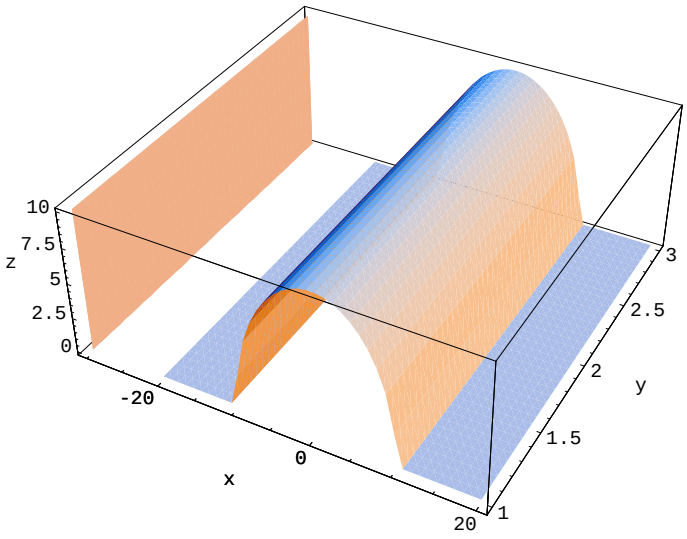


Figure 4.1: A cylindrical function. The effective noise from sampling in the y-dimension only is shown projected onto the y-z plane.

variance of z (var$(z) \approx 5.555$) in the interval over which y was sampled.

Γ	A	xy
-0.00024903	1.5633	10
0.0089737	0.48245	11
5.7815	136.11	01

Table 4.1: Feature space search results for the 3-dimensional cylinder section ($M = 500$).

The cone

Suppose now there is some small dependence of z on y. We replace the cylinder by the cone of Figure 4.2.

Part of the cone is shown in Figure 4.2. The height of the cone z is dependent on the (x, y) co-ordinates. We should discover that using x or y alone will not determine z, but using x and y together will.

The darkly shaded projection onto the x-z plane in Figure 4.2 corresponds to the component part of the signal that is expected to act like noise when data is sampled from across the cone but where only the x input is used to model z. This effective noise is not uniform across the x-input space and the variation of noise variance as a function of x is shown in Figure 4.3. If we average this

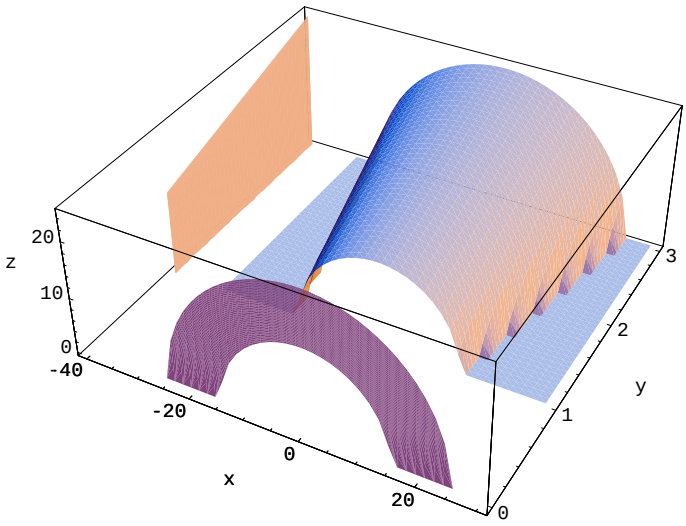


Figure 4.2: A conical function. The darkly shaded projection on the x-z plane shows the effective noise from sampling in the x-dimension only. The lighter shaded projection on the y-z plane shows the effective noise from sampling in the y-dimension only.

noise variance across the x input space we obtain the value 14.0126. Thus we might expect the associated Gamma statistic to be approximately this value. Similarly, if we project the cone onto the y-z plane (shown as the lighter shaded region) we see an even larger effective noise variance when sampling across the cone but using only input y to model z. These projections allow us to see geometrically that z is far more sensitive to variation in x than in y.

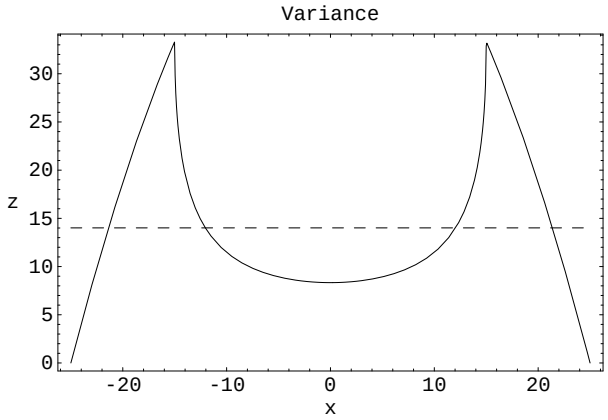


Figure 4.3: The effective noise variance of output z determined by input x. The dashed line indicates the average noise variance 14.0126 in the sampling interval $[-25, 25]$.

Table 4.2 lists the feature space search results. As expected, the effective noise variance was lowest when inputs x and y were used together. For the results where either x or y were exclusively used, the noise variance corresponds to the variance of z sampled in the interval over which x or y were

correspondingly sampled.

When input x is used to determine z, the estimated noise variance, $\Gamma = 14.76$, corresponds to the average noise variance shown in Figure 4.3.

Γ	A	xy
0.44217	11.257	11
14.76	6.6419	10
52.569	4896	01

Table 4.2: Feature space search results for the 3-dimensional cone section ($M = 500$).

4.2.2 16-dimensional input space (zero noise)

The previous example was intended to give an intuitive understanding of why, even using noise-free data, the Gamma test results for different selections of input variables can be used to discriminate significant inputs for a non-linear model.

In the following experiments we illustrate that this procedure remains effective where functional dependences are more subtle and many more input variables are present.

We consider 16 inputs and 1 output. The first 10 inputs, $x_1, x_2, \ldots, x_{10}$, are all *random numbers* in the range $(0, \pi)$. The final 6 inputs are:

$$\begin{aligned} x_{11} &= \sin(2x_1) \\ x_{12} &= \cos(4x_2) \\ x_{13} &= \sin(x_3^2) + \cos(x_4^2) \\ x_{14} &= \exp(x_5) \\ x_{15} &= -x_6^2 \\ x_{16} &= x_7^3 \end{aligned} \tag{4.1}$$

The target output is

$$\begin{aligned} y &= \sin(2x_1) - \cos(4x_2) \\ y &= x_{11} - \cos(4x_2) \\ y &= \sin(2x_1) - x_{12} \\ y &= x_{11} - x_{12} \end{aligned} \tag{4.2}$$

These functions are plotted in Appendix B.1. $M = 5000$ points were generated and in the initial experiment no noise is added to the output.

The output y is a relatively complicated function of inputs x_1 and x_2. There is a much simpler relationship between the output and x_{11} and x_{12}. There are also the intermediate relationships involving x_1 and x_{12}, and x_2 and x_{11}. In order to demonstrate the effectiveness of the Gamma test, the feature space search should discover these relationships. It should also highlight the simplest relationship as being the best.

The best results from the complete feature space search are shown in Table 4.3. Γ is the estimate for the variance of the noise and A is the gradient of the regression line fit – see equation (3.7).

$\lvert\Gamma\rvert$	A	**Mask** $(x_1 \ldots x_{16})$ $x_1x_2 \ldots x_{11}x_{12} \ldots$
3.01×10^{-7}	0.142124	0000110100110011
6.35×10^{-7}	0.12549	0001101010111100
2.00×10^{-6}	0.0881483	0101111110111001
2.49×10^{-6}	0.33071	0000010000110000
4.08×10^{-6}	0.293724	0100000001110000
4.15×10^{-6}	0.0955764	0100111101111001
4.79×10^{-6}	0.506928	0000000000110000
5.71×10^{-6}	0.149792	0001000010111001
5.80×10^{-6}	0.17813	0000000101110010
6.31×10^{-6}	0.0997976	0110101010111010
6.36×10^{-6}	0.224083	0000000000111010
6.86×10^{-6}	0.143837	0010110100110010
8.70×10^{-6}	0.0910738	0111011100111100
9.79×10^{-6}	0.107996	0001110011110001

Table 4.3: Best results from a complete feature space search ($|\Gamma| < 1 \times 10^{-5}$), $M = 5000$.

Inputs x_1, x_2, x_{11} and x_{12} are underlined in the mask to highlight their expected significance given by (4.2). These results do show the importance of inputs x_{11} and x_{12} in determining the output y; the inputs were used[2] in *all* of the best results for $|\Gamma| < 1 \times 10^{-5}$. The histogram of features shown in Figure 4.4 confirms this.

[2] A 1 in the mask indicates the inclusion of the input in the calculation, a 0 indicates exclusion.

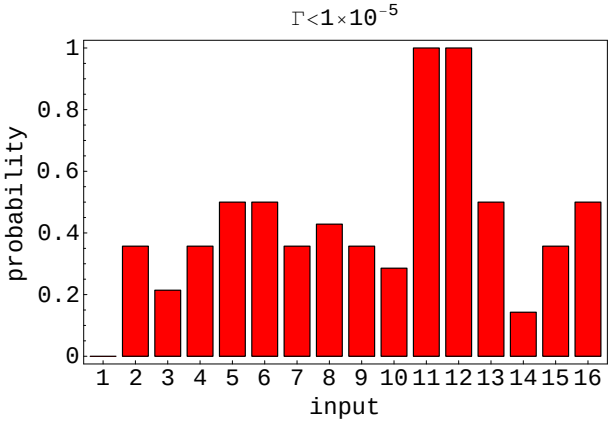


Figure 4.4: Feature set of best results ($|\Gamma| < 1 \times 10^{-5}$).

The results shown in Table 4.3 for $|\Gamma| < 1 \times 10^{-5}$ are significant because they show that the Gamma test can select the best combination of inputs. If we were to take any of the individual results we would include more inputs than are required to model the function. However, the necessary information would be incorporated since x_{11} and x_{12} appear in all of the results. The power of this technique comes from analysis of a set of results. By looking at the frequency of occurrence of the inputs for these results we have been able to establish that only a small subset of inputs are actually relevant. In the following section we discuss this further with a more detailed analysis of all of the results for this example.

Gamma histogram

A *Gamma histogram* can be used to show the distribution of the noise variance estimates for different feature sets. Using the previous example, we obtain the Gamma histogram shown in Figure 4.5 for the complete feature space search.

There are 6 significant parts to the distribution:

1. The first peak, $\Gamma < 0.03$.
2. The second peak, $0.03 \leq \Gamma < 0.1$.
3. The space between the second and third peak, $0.1 \leq \Gamma < 0.4$.
4. The third peak, $0.4 \leq \Gamma < 0.6$.
5. The space between the third and fourth peak, $0.6 \leq \Gamma < 0.95$.

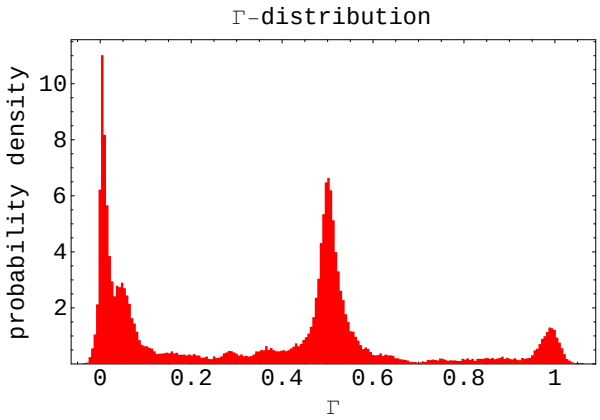


Figure 4.5: Gamma histogram for a complete feature space search over 16 inputs. The output of the function contained no noise and so the histogram starts at $\Gamma \approx 0$.

6. The fourth peak, $\Gamma \geq 0.95$.

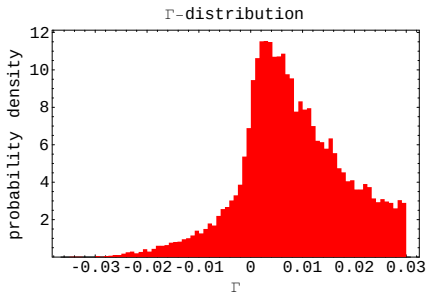
The first peak in the Gamma histogram is shown in Figure 4.6(a). This peak contains the feature sets that produced results with $\Gamma < 0.03$. A histogram of these features is shown in Figure 4.6(b). According to this histogram, inputs x_{11} and x_{12} are the most significant features, appearing in very nearly all of the results. The remaining inputs appeared with approximately equal frequency. Thus the Gamma test feature selection analysis supports the fact that x_{11} and x_{12} should provide the simplest function which smoothly models the output y.

The second peak in the Gamma histogram is shown in Figure 4.6(c) and indicates that the feature combination x_1 and x_{12} is also significant, as shown in Figure 4.6(d).

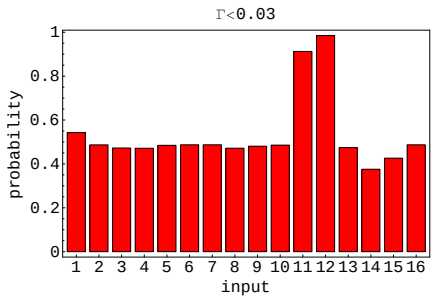
The space between the second and third peak is shown in Figure 4.6(e). This region in the histogram covers a large range of results ($0.1 \leq \Gamma < 0.4$) where the noise variance estimate approaches a level that is too high to be of practical significance. However, the feature set indicates the significance of x_1 and x_2 which is the most complicated form of the relationship given in (4.2).

Figures 4.7(a), 4.7(c) and 4.7(e) cover a region of the histogram where we would expect the relevant inputs (x_1, x_2, x_{11} and x_{12}) to have reduced influence. Indeed in the corresponding feature sets shown in Figures 4.7(b), 4.7(d) and 4.7(f) these inputs appear with less frequency or are completely eliminated. The exception is the high frequency of occurrence of input x_2 in Figure 4.7(d). The reason for its significance is unclear, but it may in part be due to the fact that y in (4.2) is more sensitive to variations in x_2 from the contribution of $\cos(4x_2)$ than to variations in x_1 from the contribution of $\sin(2x_1)$.

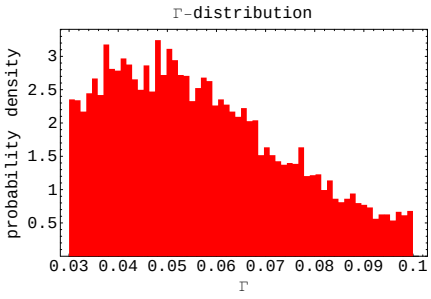
The highest Γ results for $\Gamma \geq 0.95$ illustrates the importance of this technique. In Figure 4.7(f) the significant inputs x_1, x_2, x_{11} and x_{12} are virtually eliminated. Used in conjunction with the results for the lowest Γ, we see that the Gamma test analysis can be used to justify the selection of inputs x_{11} and x_{12} (shown in Figure 4.6(b)) for use in a model of the output y.



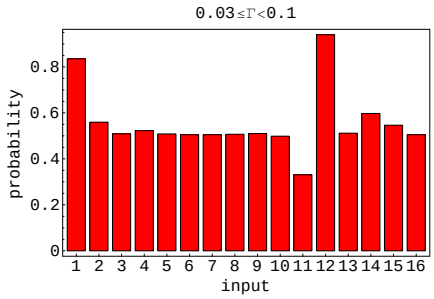
(a) Gamma histogram ($\Gamma < 0.03$).



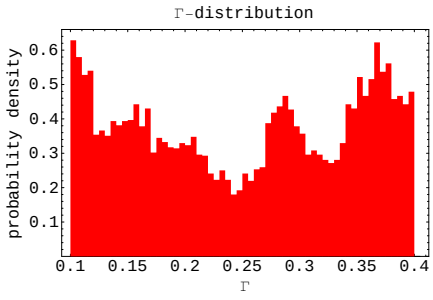
(b) Feature set ($\Gamma < 0.03$).



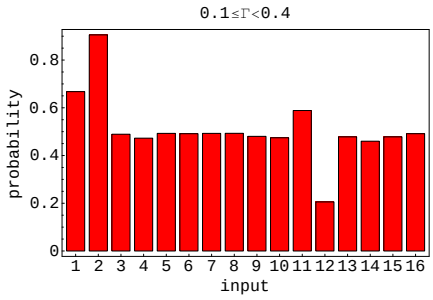
(c) Gamma histogram ($0.03 \leq \Gamma < 0.1$).



(d) Feature set ($0.03 \leq \Gamma < 0.1$).

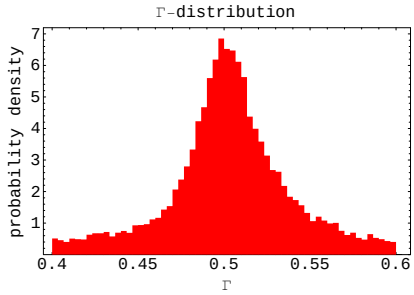


(e) Gamma histogram ($0.1 \leq \Gamma < 0.4$).

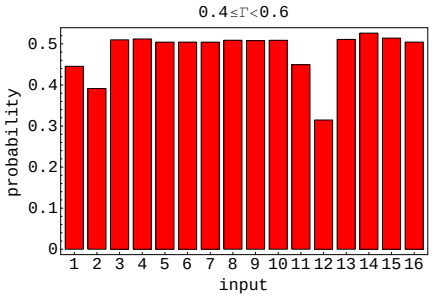


(f) Feature set ($0.1 \leq \Gamma < 0.4$).

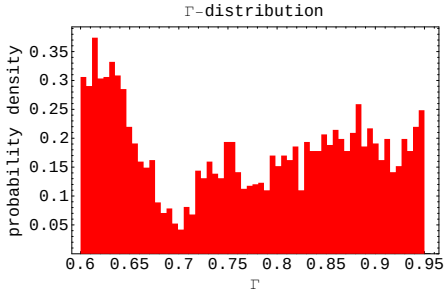
Figure 4.6: Gamma histogram components ($\Gamma < 0.4$).



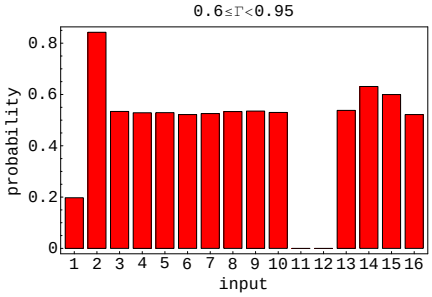
(a) Gamma histogram ($0.4 \leq \Gamma < 0.6$).



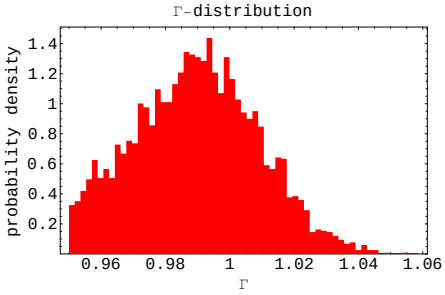
(b) Feature set ($0.4 \leq \Gamma < 0.6$).



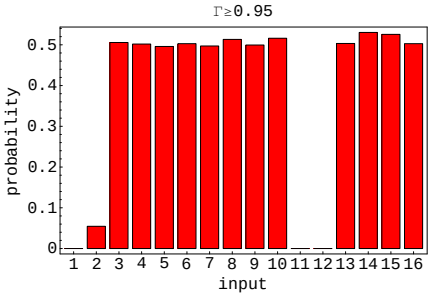
(c) Gamma histogram ($0.6 \leq \Gamma < 0.95$).



(d) Feature set ($0.6 \leq \Gamma < 0.95$).



(e) Gamma histogram ($\Gamma \geq 0.95$).



(f) Feature set ($\Gamma \geq 0.95$).

Figure 4.7: Gamma histogram components ($\Gamma \geq 0.4$).

4.2.3 16-dimensional input space (noisy output)

We can adapt the example in Section 4.2.2 by adding noise to the output y to examine how the feature selection algorithm performs. In this example, a noise variance, $\text{var}(r) = 0.25$, was added to the output.

The Gamma histogram in Figure 4.8 illustrates how the addition of noise affects the histogram. The Gamma histogram is in the interval $[0.195, 1.338]$. There are three major peaks in the Gamma histogram, which exist in roughly the same relative positions in the previous Gamma histogram in Figure 4.5. The two largest peaks exist in the centre of the histogram and at the low Γ end. The third peak is smaller and exists at the high Γ end of the histogram.

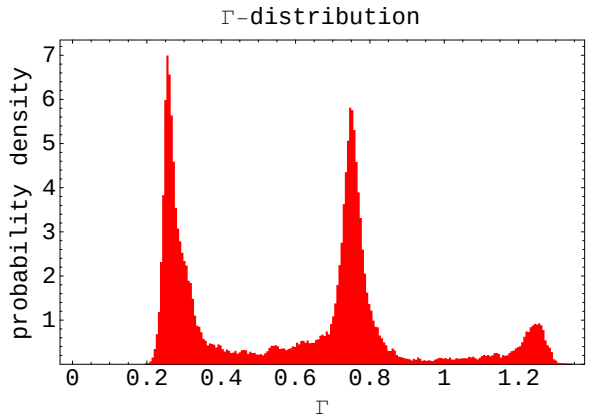


Figure 4.8: Gamma histogram for a complete feature space search of 16 inputs. Noise with a variance $\text{var}(r) = 0.25$ was added to the output – consequently the Gamma histogram starts at $\Gamma \approx 0.25$.

There are 5 significant parts to the distribution:

1. The first peak, $\Gamma < 0.3$.
2. The space between the first and second peak, $0.3 \leq \Gamma < 0.7$.
3. The second peak, $0.7 \leq \Gamma < 0.8$.
4. The space between the second and third peak, $0.8 \leq \Gamma < 1.2$.
5. The third peak, $\Gamma \geq 1.2$.

The first peak in the Gamma histogram is shown in Figure 4.9(a). This peak contains the feature sets that produced results with $\Gamma < 0.3$. A histogram of these features is shown in Figure 4.9(b).

This histogram shows that inputs x_{11} and x_{12} are the most significant features since they appear in most of the results. The remaining inputs appeared with approximately equal frequency (with the exception of input x_1 which appeared slightly more frequently). On this evidence, the Gamma test feature analysis supports the fact that x_{11} and x_{12} provide the information necessary to model the output y.

The space between the first and second peaks in the Gamma histogram, shown in Figure 4.9(c), indicates that the feature combination of x_1 and x_2 is also significant, as shown in Figure 4.9(d).

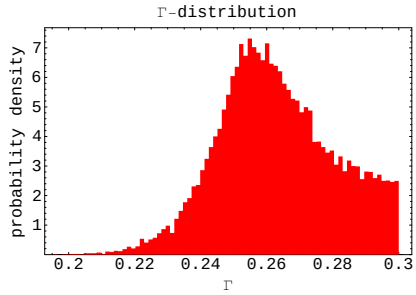
The remaining Figures 4.9(e)-(f) and Figures 4.10(a)-(d) provide additional evidence that inputs x_1, x_2, x_{11} and x_{12} are significant. This is demonstrated by the absence of these significant inputs in the worst results.

4.2.4 Feature selection hypothesis

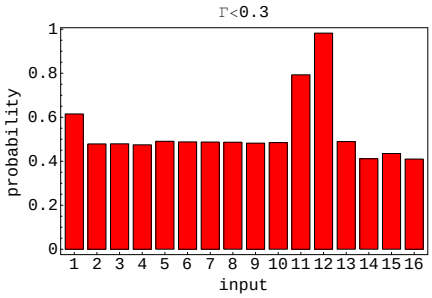
It is our hypothesis that the peaks visible in the Gamma histograms contain information that could be used to determine the number of significant input variables. A peak at the lower end of the Gamma histogram should contain results that use all of the available relevant input variables. A peak at the higher end of the Gamma histogram should show results generated from input variables that have little or no relevance in determining the output.

A re-examination of the experiments in Sections 4.2.2 and 4.2.3, the 16-dimension input feature space searches with no added noise and 0.25 variance noise added to the output respectively, provides support for our hypothesis. There are 4 inputs that can be used to define the output (x_1, x_2, x_{11} and x_{12}), although this essentially reduces to two inputs since x_{11} is a function of x_1, and x_{12} is a function of x_2. Any combination of these inputs (x_1 and/or x_{11} and x_2 and/or x_{12}) provides all of the information to determine the output.

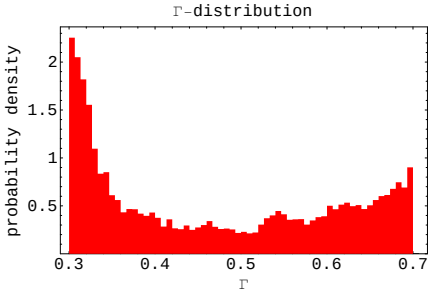
The peak at the lower end of the Gamma histogram should contain all of the results that use x_1 and/or x_{11} and x_2 and/or x_{12} in combination. The central peak would then contain most of the results that use only one of the significant variables, either x_1, x_2, x_{11} or x_{12}. The final peak then contains results which use none of the significant variables.



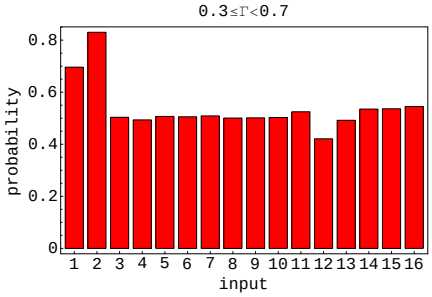
(a) Gamma histogram ($\Gamma < 0.3$).



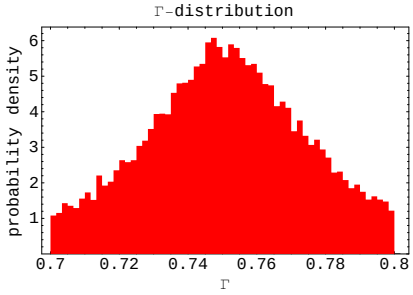
(b) Feature set ($\Gamma < 0.3$).



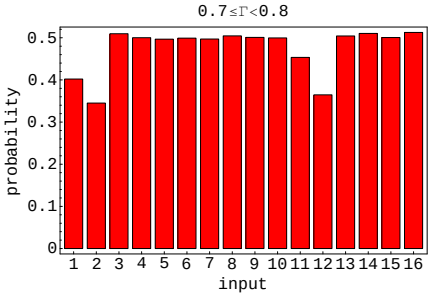
(c) Gamma histogram ($0.3 \leq \Gamma < 0.7$).



(d) Feature set ($0.3 \leq \Gamma < 0.7$).

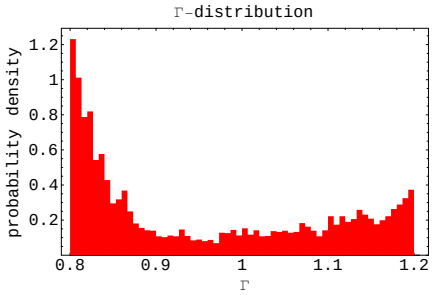


(e) Gamma histogram ($0.7 \leq \Gamma < 0.8$).

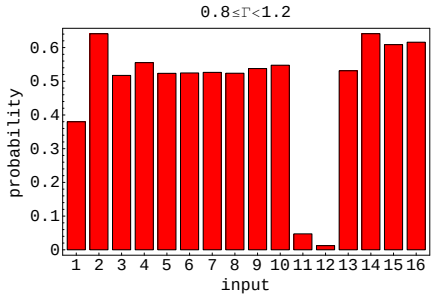


(f) Feature set ($0.7 \leq \Gamma < 0.8$).

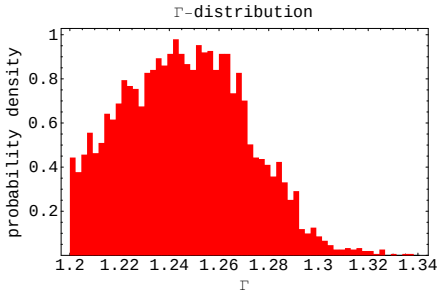
Figure 4.9: Gamma histogram components ($\Gamma < 0.8$).



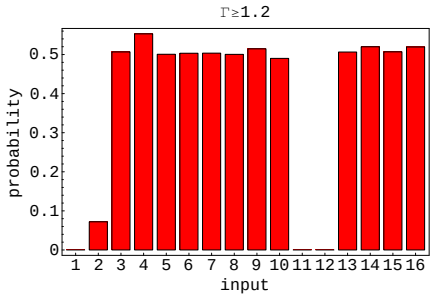
(a) Gamma histogram ($0.8 \leq \Gamma < 1.2$).



(b) Feature set ($0.8 \leq \Gamma < 1.2$).



(c) Gamma histogram ($\Gamma \geq 1.2$).



(d) Feature set ($\Gamma \geq 1.2$).

Figure 4.10: Gamma histogram components ($\Gamma < 0.8$).

11-dimensional function

The hypothesis that the peaks in the Gamma histogram contain useful information is re-inforced with an additional example. 11 inputs were generated, $x_1, \ldots, x_8$ are random numbers uniformly distributed in the interval $[0, \pi]$. The other inputs are defined as

$$\begin{aligned} x_9 &= \sin(2x_1) \\ x_{10} &= \cos(4x_2) \\ x_{11} &= -\cos(2x_3) \end{aligned} \tag{4.3}$$

The target output is

$$\begin{aligned} y &= \sin(2x_1) + \cos(4x_2) - \cos(2x_3) \\ y &= x_9 + x_{10} + x_{11} \end{aligned} \tag{4.4}$$

The output is directly dependent on three input variables: x_9, x_{10} and x_{11}. However, a combination of inputs consisting of x_1 or x_9, and x_2 or x_{10}, and x_3 or x_{11} will provide sufficient information to define the output.

A full feature space search produces the Gamma histogram shown in Figure 4.11.

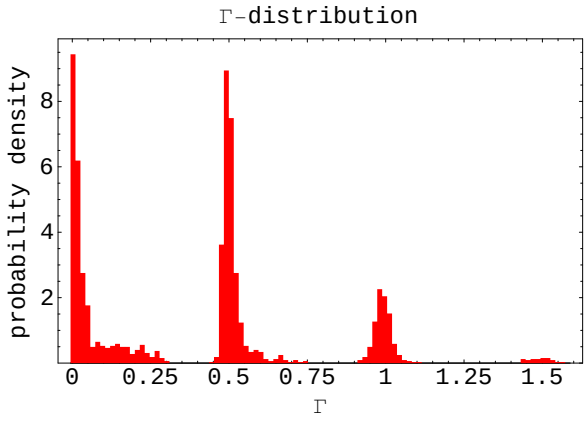


Figure 4.11: Gamma histogram for a complete feature space search of 11 inputs.

The Gamma histograms shown in Figures 4.5, 4.8 and 4.11 demonstrate a striking similarity in that the number of peaks in the respective Gamma histograms is a function of the number of significant inputs variables. Figures 4.5 and 4.8 had two significant inputs and three peaks in the histogram, whereas Figure 4.11 has three significant inputs and four peaks. The number of significant inputs is then given directly from the number peaks in the histogram. These examples have shown that there is always one more peak than the number of significant inputs. The extra peak lies at the

high noise end of each histogram and signifies results that contain no useful information, whereas the other peaks in the histogram signify results that contain at least one significant input.

A more detailed examination of the results of all of these experiments shows that the first peak contains results using all n significant inputs, the next peak contains $n-1$ significant inputs, and so on.

Conclusions

The observations made of the full feature space search have shown that the Gamma histogram provides a method to estimate the number of significant input variables. The results from the full feature space can then be used identify those inputs.

When there is noise on the output, the clarity of the analysis is reduced, but, providing that the noise level is not excessive, the full feature space search can pinpoint the relevant inputs.

4.3 Heuristic feature space search

The complete feature space search described in Section 4.2.2 is only practical for a relatively small number of inputs, limited to no more than 20 variables. As an illustration, the 11-dimensional example took approximately 3 hours to compute, whereas the 16-dimensional examples took approximately 4-5 days to compute[3].

In light of the computation required for even a modest number of inputs we have developed a number of heuristic search techniques to find good solutions in reasonable computational time. We shall discuss our main heuristic search technique: the genetic algorithm. A general description of evolutionary algorithms can be found in [Michalewicz and Fogel, 2000] where the discussion revolves around the general application of heuristics rather than providing solutions to specific problems.

Feature Space Search using a Genetic Algorithm

The introduction of genetic algorithms by [Holland, 1975] provides the primary inspiration for the design of our genetic algorithm heuristic search technique. For simplicity we quote in Algorithm

[3]Machine specification: AMD Athlon 800MHz, 320MB RAM, Windows 2000.

5 the general form of an evolutionary algorithm, as described in [Michalewicz and Fogel, 2000].

```
t = 0
initialise P(t)
evaluate P(t)
while not termination condition do
  t = t + 1
  select P(t) from P(t − 1)
  alter P(t)
  evaluate P(t)
end while
```

Algorithm 5: Evolutionary algorithm.

The algorithm maintains a population P of potential individual solutions x_i, in this case x_i represents a particular mask. The population $P(t) = \{x_1^t, x_2^t, \ldots, x_n^t\}$ undergoes modification in an iterative process that mimics genetic evolution. The initial population $P(0)$ is created randomly.

The selection of individual solutions x_i^{t-1} from $P(t-1)$ is performed in a probabilistic manner. The better solutions, that is the masks that represent solutions with the lowest Gamma statistics, have a greater chance of being selected for the next generation. We call the probability with which a solution is likely to be selected its *fitness*. The alteration of $P(t)$ in our genetic algorithm uses mutation, the unary genetic operator to modify individual masks, and the crossover operator to generate a new mask from two parent masks (the worst solution is rejected to maintain a constant population size). The evaluation of $P(t)$ then performs a Gamma test on each x_i^t in the population, from which the fitness can be calculated.

The fitness of a particular mask or feature set can be determined by three principle factors of the Gamma test:

1. The *intercept* - the Gamma statistic.
2. The *gradient* - the model complexity estimate.
3. The *length* - the number of inputs required.

The relative significance of these factors can be adjusted to *tune* solutions towards a particular requirement. For example, we may be interested in solutions that use only a small subset of the available inputs, therefore the length fitness would be given a greater significance than either the intercept or gradient. Hence the calculation of the fitness

$$\begin{aligned} \texttt{fitness}(mask) \;=\; & 1 - [W_{intercept} \times \texttt{interceptFitness}(mask) + \\ & W_{gradient} \times \texttt{gradientFitness}(mask) + \\ & W_{length} \times \texttt{lengthFitness}(mask)] \end{aligned} \tag{4.5}$$

can be weighted by means of user defined parameters $W_{intercept}$, $W_{gradient}$ and W_{length} in accordance to the importance ascribed to the three fitness factors. The maximum fitness will occur when the fitness function (4.5) reaches a maximum.

There are limits applied to the fitness components

$$\begin{aligned} \texttt{fitness}(mask) &\leq 1 \\ \left.\begin{array}{r} \texttt{interceptFitness}(mask) \\ \texttt{gradientFitness}(mask) \\ \texttt{lengthFitness}(mask) \end{array}\right\} &\geq 0 \\ 0 \leq \left\{\begin{array}{c} W_{intercept} \\ W_{gradient} \\ W_{length} \end{array}\right\} &\leq 1 \\ W_{intercept} + W_{gradient} + W_{length} &> 0 \end{aligned} \tag{4.6}$$

The individual components of (4.5) are defined as

$$\texttt{interceptFitness}(mask) = \begin{cases} 1 - \frac{1}{1-10\times\texttt{vRatio}(mask)} & : \quad \texttt{vRatio}(mask) < 0 \\ 2 - 2\frac{1}{1+\texttt{vRatio}(mask)} & : \quad \texttt{vRatio}(mask) \geq 0 \end{cases} \tag{4.7}$$

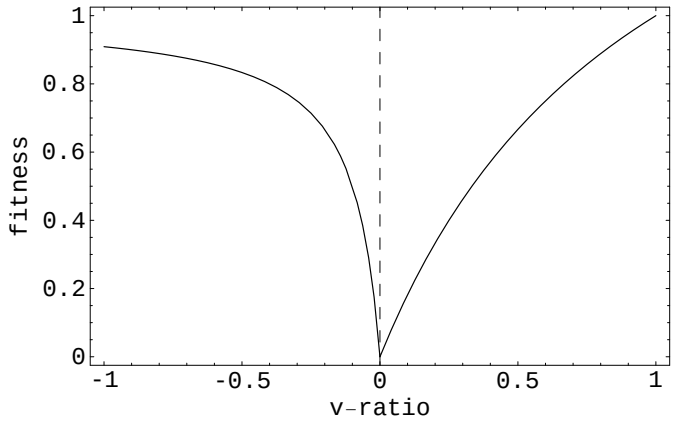


Figure 4.12: Genetic algorithm intercept fitness.

$$\texttt{gradientFitness}(mask) = 1 - \frac{1}{1 + |\frac{\texttt{gradient}(mask)}{\texttt{outputRange}}|} \tag{4.8}$$

$$\texttt{lengthFitness}(mask) = \frac{\texttt{ones}(mask)}{\texttt{length}(mask)} \tag{4.9}$$

where $\texttt{ones}(mask)$ counts the number of ones in the mask (i.e. the number of inputs selected) and $\texttt{length}(mask)$ returns the length of the mask.

4.3.1 16-dimensional input space (zero noise)

We repeat the experiment of Section 4.2.2 to demonstrate the usefulness of the genetic algorithm to search for good solutions in reasonable computational time. The same data set using 16 inputs and a single output with no added noise has been used to test the genetic algorithm.

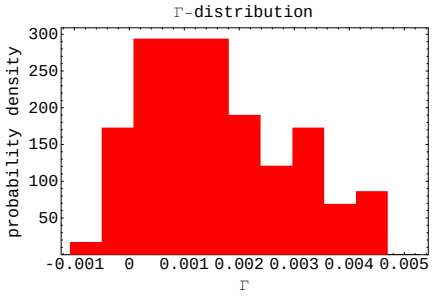
Two experiments have been performed to demonstrate the GA. The first experiment is tuned to produce solutions with a low intercept irrespective of the number of inputs used, whereas the second experiment is tuned to search for solutions with both a low intercept and a low mask length. This subtle difference between the two experiments aims to show that the second experiment can find good solutions using the minimum number of inputs.

The full search took 4-5 days of continuous computation to perform 65535 unique Gamma tests. These new experiments generate a random population of 100 individual masks, with only an additional 200 Gamma tests performed after the initialisation stage. The 300 Gamma tests required for each experiment took approximately 10 minutes to compute and the results are shown in Figures 4.13 and 4.14.

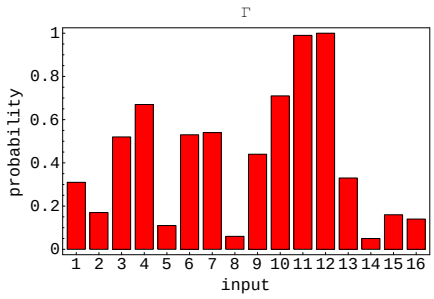
Figure 4.13(a) shows that all of the solutions were reasonably good and Figure 4.13(b) demonstrates that the GA identified inputs x_{11} and x_{12} as being the most significant. The evolution of the population is shown in Figure 4.13(c) where the average population fitness converged to the fitness of the best solution towards the end of the execution of the GA. This indicates that the GA had probably run for a sufficient time using the current settings (we do not intend to discuss too deeply the intricacies of interpreting the results generated using genetic algorithms).

Figure 4.14(a) shows that there were two clusters of solutions with the majority of solutions being accurately classified as being noise free. Figure 4.14(b) demonstrates that the GA identified inputs x_2 and x_{11} as being the most significant. The evolution of the population is shown in Figure 4.14(c).

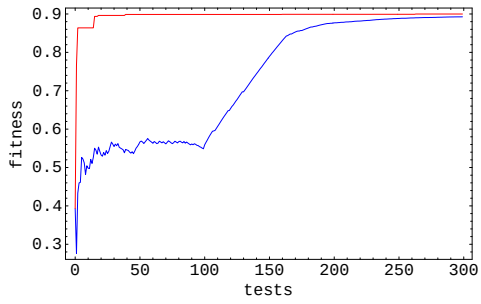
Both of these experiments show that the GA can be applied to complex problems to provide good solutions in reasonable time. Although a more detailed analysis could be achieved with a larger population size and a longer run, the aim here was to demonstrate the concept that mask searches can be achieved quickly using an heuristic technique such as a GA.



(a) Gamma histogram.

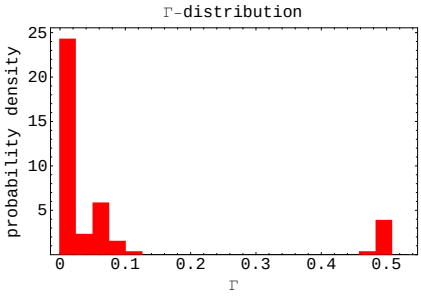


(b) Feature set.

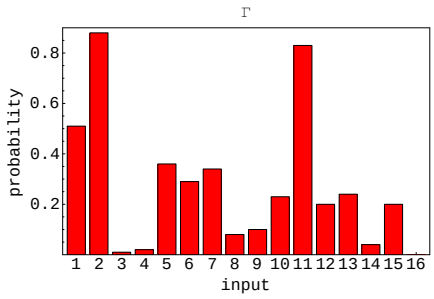


(c) The change in population fitness during the execution of the GA. The blue line shows the average fitness of the population. The red line shows the fitness of the best individual.

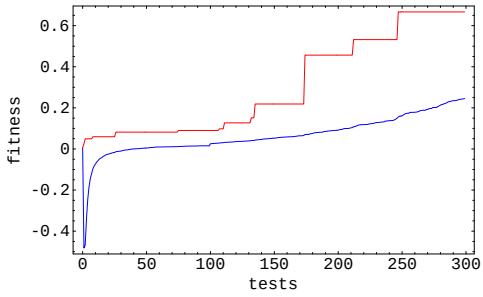
Figure 4.13: Gamma histogram for an heuristic search using a genetic algorithm. The two most significant inputs are x_{11} and x_{12} and these appear in the majority of solutions. The GA settings are population size $= 100$, $\texttt{interceptFitness}(mask) = 1$, $\texttt{gradientFitness}(mask) = 0.1$ and $\texttt{lengthFitness}(mask) = 0.1$. In approximately 10 minutes the algorithm performed 300 Gamma tests, of which 100 were used to initialise the population.



(a) Gamma histogram.



(b) Feature set.



(c) The change in population fitness during the execution of the GA. The blue line shows the average fitness of the population. The red line shows the fitness of the best individual.

Figure 4.14: Gamma histogram for an heuristic search using a genetic algorithm. The two most significant inputs are x_{11} and x_{12} and these appear in the majority of solutions. The GA settings are population size $= 100$, $\texttt{interceptFitness}(mask) = 1$, $\texttt{gradientFitness}(mask) = 0.1$ and $\texttt{lengthFitness}(mask) = 1$. In approximately 10 minutes the algorithm performed 300 Gamma tests, of which 100 were used to initialise the population.

4.4 Embeddings: the analysis of chaotic time series

It is possible to model a continuous dynamical system, which in the first instance may be defined by a system of differential equations, by a smooth non-linear input/output model which over time generates new states of the system based on a finite window of previous states. This observation is in fact a quite deep theorem due originally to [Takens, 1981] and later extended by [Sauer et al., 1991].

The false nearest neighbours algorithm [Kennel et al., 1992] was developed to find a suitable embedding dimension for a time series. We propose the *increasing embedding* as an extension of the Gamma test to perform the same task.

After introducing the two embedding dimension search techniques we shall analyse two chaotic time series, the Hénon map and the generalised Chua's circuit, to provide a comparison between the techniques.

4.4.1 False nearest neighbours

The *false nearest neighbour* (FNN) algorithm [Kennel et al., 1992] is a technique to determine the embedding dimension for phase-space reconstruction. A chaotic attractor is typically a compact object in phase-space, such that points of an orbit on the attractor acquire neighbours. It has been suggested that the evolution of phase-space neighbourhoods can determine how points on or near the attractor will evolve, and also provide a way to accurately compute the Lyapunov exponents. However, in this restricted discussion we are purely concerned with identifying the correct embedding dimension.

If the embedding dimension of an attractor is sufficient there will be a one-to-one mapping from the delay-space (the time series) to the original phase-space of the attractor such that the topological properties of the attractor will be maintained. The assumed smoothness of the function means that neighbourhoods of points in delay-space will map to neighbourhoods of points in phase-space.

An embedding dimension that is too small will not preserve the topological structure of the attractor, so that points that are neighbours in one embedding dimension, d, will not necessarily be neighbours in the next higher embedding dimension, $d + 1$, because the attractor has not been completely unfolded. It is these points that are classified as *false nearest neighbours* and the number present for a particular embedding dimension determine whether that embedding dimension,

d, sufficiently describes the attractor. The FNN algorithm identifies these points for a range of embedding dimensions and (in theory) the optimal embedding dimension has the minimum number of false nearest neighbours.

In order to describe the FNN algorithm we define the delay-space points as $x(i)$ and the corresponding phase-space points as $\mathbf{z}(i)$, where $\mathbf{z}(i) = (x_{i+T}, x_{i+2T}, \ldots, x_{i+(d-1)T})$, where T is the time-lag[4]. In phase-space, the p nearest neighbours to $\mathbf{z}(i)$ are $\mathbf{z}(N[i,p])$, using the same nearest neighbour notation introduced in Chapter 3. For notational simplicity, we define $j = N[i,p]$ for a given i and p.

The phase-space dimension is increased from d to $d+1$ to identify the false nearest neighbours. This identification compares the distance between near neighbours in d dimensional phase-space, the distance between $\mathbf{z}_d(i)$ and $\mathbf{z}_d(N[i,p])$, to the distance between near neighbours in $d+1$ dimensional phase-space, the distance between $\mathbf{z}_{d+1}(i)$ and $\mathbf{z}_{d+1}(N[i,p])$. If the difference between the two distances is high then the point is a false nearest neighbour.

We can conclude that the attractor has been suitably unfolded in phase-space when, for a given embedding dimension d, the number of false nearest neighbours is minimum compared to results for a wide range of embedding dimensions.

The squared-Euclidean distance between $\mathbf{z}_d(i)$ and $\mathbf{z}_d(N[i,p])$ in d-dimensions is

$$\begin{aligned} D_d(i,p)^2 &= [\mathbf{z}_d(i) - \mathbf{z}_d(N[i,p])]^2 \\ &= \sum_{q=0}^{d-1} [x(i+qT) - x(j+qT)]^2 \end{aligned} \tag{4.10}$$

If we increase the embedding dimension to $d+1$ we obtain the corresponding squared-Euclidean distance between nearest neighbours $\mathbf{z}_{d+1}(i)$ and $\mathbf{z}_{d+1}(N[i,p])$

$$\begin{aligned} D_{d+1}(i,p)^2 &= [\mathbf{z}_{d+1}(i) - \mathbf{z}_{d+1}(N[i,p])]^2 \\ &= D(i,p)_d^2 + [x(i+dT) - x(j+dT)]^2 \end{aligned} \tag{4.11}$$

Using (4.10) and (4.11) we can compute the change in distance, S, between $\mathbf{z}(i)$ and $\mathbf{z}(N[i,p])$ when changing dimension from d to $d+1$ to be

$$\begin{aligned} S &= \left[\frac{D_{d+1}(i,p)^2 - D_d(i,p)^2}{D_d(i,p)^2}\right]^{\frac{1}{2}} \\ &= \frac{|x(i+dT) - x(j+dT)|}{D_d(i,p)} \end{aligned} \tag{4.12}$$

[4] A full discussion of the significance of the time-lag T is beyond the scope of this thesis. In all of the experiments used within this thesis we assume that T is fixed.

The final expression of S in (4.12) describes the distance metric as a ratio of the distance between a point $x(i + dT)$ and $x(j + dT)$ in delay-space, and the corresponding point $\mathbf{z}(i)$ and its nearest neighbour $\mathbf{z}(N[i, p])$ in phase-space.

The false nearest neighbours can be defined to exist beyond a distance D_{tol} such that

$$S > D_{tol} \tag{4.13}$$

The distance metric S is the ratio of the nearest neighbour distance in delay-space to the corresponding nearest neighbour distance in phase-space. If the measure is large then the point is a false nearest neighbour.

A second criterion is applied to handle the issue of limited data. Let D_A be a measure of the size of the attractor

$$D_A^2 = \frac{1}{2} \sum_{i=1}^{N} [x(i) - \bar{x}]^2 \tag{4.14}$$

where

$$\bar{x} = \frac{1}{N} \sum_{i=d-1}^{N} x(i) \tag{4.15}$$

If a nearest neighbour in $d + 1$ dimensional phase-space is distant (compared to the size of the attractor) then we consider that point to also be a false nearest neighbour. This arises because if the nearest neighbour $\mathbf{z}_d(N[i, p])$ is not close to $\mathbf{z}_d(i)$, i.e. $D_d(i, p) \approx D_A$, then in $d + 1$ dimensional phase-space[5] $D_{d+1}(i, p) > cD_A$ where $1 < c < 2$.

In this second criterion, any point beyond A_{tol} is classified as a false nearest neighbour

$$\frac{D_{d+1}(i, p)}{D_A} > A_{tol} \tag{4.16}$$

This ensures that distant near neighbours, which are stretched to the extremities of the attractor as the embedding dimension increases, are classed as false nearest neighbours.

As explained in [Kennel et al., 1992], the utility of the second criterion is to distinguish between low dimensional chaos and high dimensional chaos or noise.

The algorithm computes the total proportion of false nearest neighbours in the data set as determined by the two criteria (4.13) and (4.16). It is sufficient to use the first nearest neighbours only ($p_{max} = 1$) for each of the M points in the data set.

[5] [Kennel et al., 1992] refer to the condition being $D_{d+1}(i, p) \approx 2D_A$ but this is not intuitive.

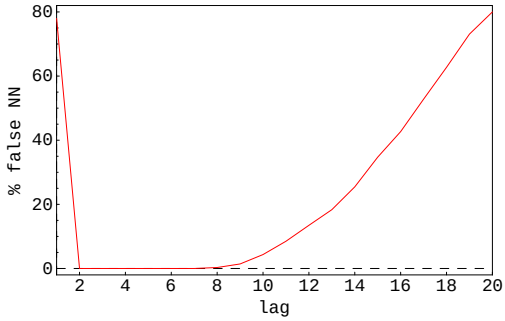
4.4.2 Increasing embedding

The increasing embedding examines the relationship between $\mathbf{z}(i)$, generated from the past d points in phase-space such that $\mathbf{z}(i) = \sum_{q=0}^{d-1}[x(i+qT)]$, and the next point in the time series $x(i+dT)$ using the Gamma test (T is the time-lag and d is the dimension). d is increased until the magnitude of the Gamma statistic reaches a minimum, at which stage d should provide the optimal embedding dimension.

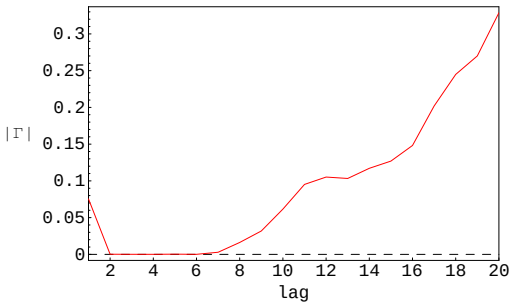
4.4.3 The Hénon map

The Hénon map was introduced in Section 3.4. From the defining equation (3.14) we can instantly see that the output of the function x_t is dependent on the two previous values x_{t-1} and x_{t-2}. Figure 3.4(b) confirms that the attractor is a smooth function using these 3 parameters (2 inputs and 1 output).

Figure 4.15 shows the results from the FNN algorithm and the increasing embedding on the time series data generated from the Hénon map. The optimal embedding dimension selected by both methods uses 2 lags in the embedding window.



(a) False nearest neighbour embedding for the Hénon map. The graph shows that an embedding dimension $d = 2$ would be suitable.



(b) Increasing embedding for the Hénon map. The graph shows that an embedding dimension $2 \leq d \leq 7$ may be suitable. Empirical evidence suggests that selecting the lowest embedding dimension from a range of possible solutions is best, in this case $d = 2$.

Figure 4.15: The comparison between the false nearest neighbour method of estimating an embedding dimension and the increasing embedding using the Gamma test for the Hénon map.

4.4.4 Generalised Chua's circuit

The generalised Chua's circuit is defined as

$$\begin{aligned} \dot{x_1} &= \alpha[x_2 - h(x_1)] \\ \dot{x_2} &= x_1 - x_2 + x_3 \\ \dot{x_3} &= -\beta x_2 \end{aligned} \tag{4.17}$$

where

$$h(x_1) = m_{2q-1}x_1 + \frac{1}{2}\sum_{i=1}^{2q-1}(m_{i-1} - m_i)(|x_1 + c_i| - |x_1 - c_i|) \tag{4.18}$$

and q denotes a natural number. We want to obtain the 5-scroll attractor used in the time series competition described in [Suykens and Vandewalle, 1998]. That 5-scroll attractor had parameters

$$\begin{aligned} q &= 3 \\ \mathbf{m} &= (0.9/7, -3/7, 3.5/7, -2.7/7, 4/7, -2.4/7) \\ \mathbf{c} &= (1, 2.15, 3.6, 6.2, 9) \end{aligned}$$

where $\mathbf{m} = (m_0, m_1, \ldots, m_{2q-1})$, $\mathbf{c} = (c_1, c_2, \ldots, c_{2q-1})$, and the initial state $\mathbf{x} = (0.1, -0.2, 0.3)$.

The 5-scroll attractor for these parameters is shown in Figure 4.16.

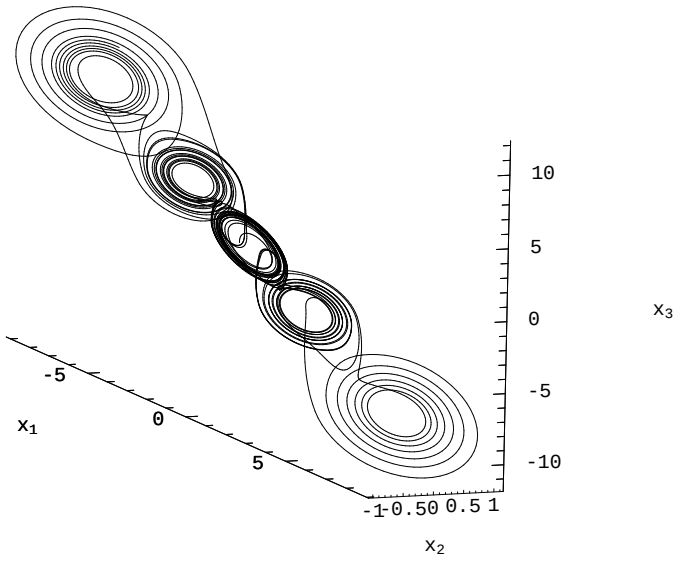


Figure 4.16: The 5-scroll attractor generated from Chua's generalised circuit.

The time series data $\mathbf{x}_t$ generated by (4.17) and (4.18) was passed through a multi-layer perceptron with 3-hidden nodes to produce a non-linear scalar y_t that hides the underlying structure of the attractor

$$y_t = \mathbf{W} \tanh(\mathbf{V}\mathbf{x}_t) \tag{4.19}$$

The multi-layer perceptron is defined as

$$\begin{aligned} \mathbf{W} &= (-0.0124, 0.3267, 1.2288) \\ \mathbf{V} &= \begin{bmatrix} -0.1004 & -0.1102 & -0.2784 \\ 0.0009 & 0.5792 & 0.6892 \\ 0.1063 & -0.0042 & 0.0943 \end{bmatrix} \end{aligned} \tag{4.20}$$

The time series competition involved constructing a model from 2000 points on the attractor, and using that model to predict the next 200 points. The data provided for the competition, and the data used to evaluate the competition entries is shown in Figure 4.17. The results of the competition were published in [Suykens and Vandewalle, 1998] and the winning solution was published in [McNames et al., 1999]. The quality of the contestants results were judged using the mean-squared error. Our rather hastily constructed model came sixth out of 17 entries.

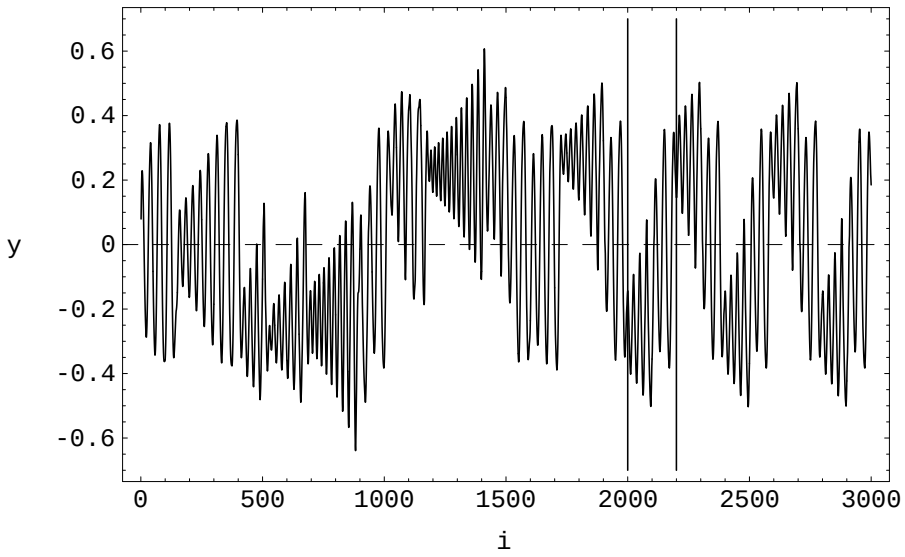
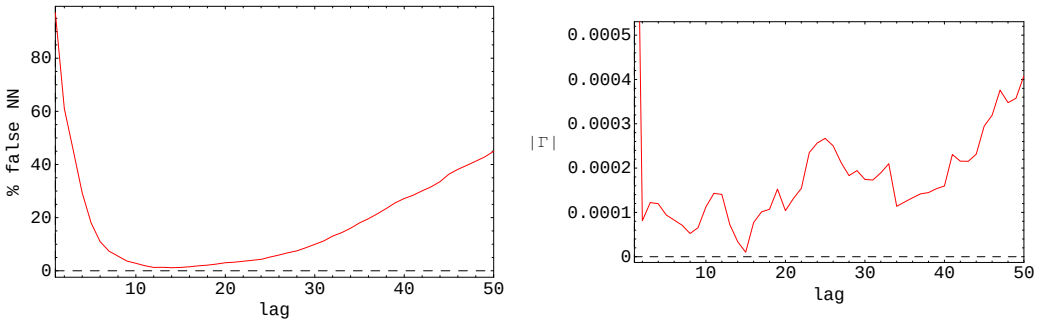


Figure 4.17: The data generated from the 5-scroll attractor for the time series competition. The first 2000 points (up to the first vertical bar) were provided as a training set for modelling. The next 200 points between the vertical bars were used to evaluate the submitted competition predictions. The final 800 points illustrate how the system developed.

We return to the analysis of the problem and show how the FNN algorithm and the increasing embedding can be used to find the embedding dimension for this problem. Figure 4.18 shows the

embedding dimension analysis for the two algorithms. The graphs show similar results: the FNN algorithm selects an embedding dimension $d = 14$ whereas the increasing embedding selects a dimension $d = 15$. The similarity in these results is striking since both of these analyses are dependent on a number of variables. D_{tol} and A_{tol} must be chosen for the FNN algorithm, whereas p_{max} affects the increasing embedding.



(a) False nearest neighbour embedding for the 5-scroll Chua attractor. The graph shows that an embedding dimension $d = 14$ would be suitable.

(b) Increasing embedding for the 5-scroll Chua attractor. The graph shows that an embedding dimension $d = 15$ would be suitable.

Figure 4.18: The comparison between the false nearest neighbour method of estimating an embedding dimension and the increasing embedding using the Gamma test for the 5-scroll Chua attractor.

4.4.5 Conclusion

The analysis of the Hénon map showed comparable results between the false nearest neighbours algorithm and the increasing embedding. The false nearest neighbour technique probably provided the more instructive result for the more complicated 5-scroll attractor indicating a range of suitable embedding dimensions. The Gamma test solution was not as conclusive as for the Hénon map, which can perhaps be attributed to the Gamma test requiring a lot of data to produce very accurate estimates for the Gamma statistic. It is worth noting that the data points generated from the 5-scroll attractor covered a very small region of the attractor [Suykens and Vandewalle, 1998] making measurement of the embedding dimension more difficult.

In terms of computation, both techniques are comparable since they use the same near neighbours routine.

The recommendation from this study for determining the optimal embedding dimension is to com-

pare the results from a number of techniques to see if a consensus can be reached.

4.4.6 Irregular embedding

An embedding window selected using the false nearest neighbour technique or increasing embedding often provides a starting point to refine the model. In many cases an optimal model may exist that requires only a subset of the inputs. If that is the case, we can describe the model using an *irregular embedding*.

We can perform an irregular embedding by first running an increasing embedding or false nearest neighbour test to identify the embedding window, then one of the feature selection techniques discussed in Section 4.1 is applied to find the best input combination.

We illustrate the technique using two examples: (1) the Hénon map and (2) the 5-scroll Chua attractor. In Section 4.4 we calculated the optimal embedding windows for both examples.

The Hénon map

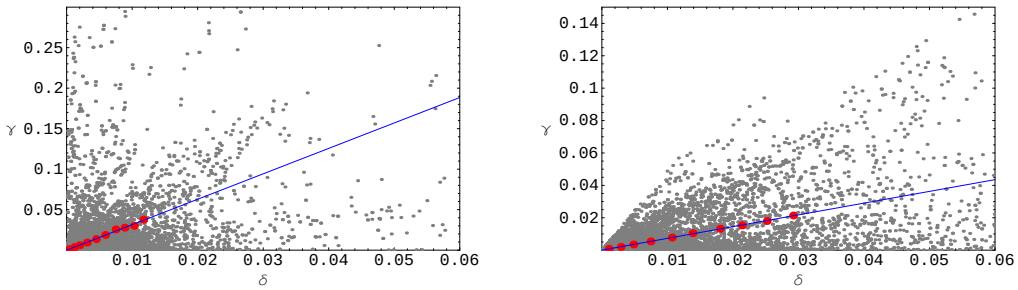
The analysis of the Hénon map in Section 4.4 indicated that an embedding dimension of between 2 and 6 would provide a good model.

We can demonstrate the effectiveness of the full feature space search on a 6-dimensional embedding. The top two results from the analysis are shown in Table 4.4. The two best results, measured purely in terms of their respective Gamma statistic values, demonstrate a noteworthy point. Remember that we know from (3.14) that the next value in the Hénon time series x_t is generated from a function of x_{t-1} and x_{t-2}. The result for the irregular embedding described by the mask `001010` uses inputs x_3 and x_5 which correspond to x_{t-4} and x_{t-2} respectively to predict x_t. It would seem that using this result would not provide the best model. However, if we use the second result from Table 4.4 then we can see that the output x_t does indeed directly depend on x_{t-1} and x_{t-2}. The key to interpreting these results is to use either the gradient A as an indicator of surface complexity, or the Gamma scatter plots to provide a visual measure of the noise (the Gamma scatter plots for these two results are shown in Figure 4.19).

The Gamma scatter plots shown in Figure 4.19 highlight the difference between the irregular embeddings `001010` and `001011`. The best irregular embedding is `001011` despite having a slightly higher Gamma statistic. The Gamma scatter plots show that this embedding has no scatter points

$\lvert\Gamma\rvert$	A	Mask $(x_1 \ldots x_6)$
3.054×10^{-5}	3.1404	001010
3.6029×10^{-5}	0.72488	001011

Table 4.4: The top two irregular embeddings from a complete feature space search of the time series generated from the Hénon map ($M = 994$, $d = 6$). The best result in this case cannot be measured on the Gamma statistic alone, but relies on the judgement that the gradient of the regression line fit, A, provides a simpler model for the second best result.



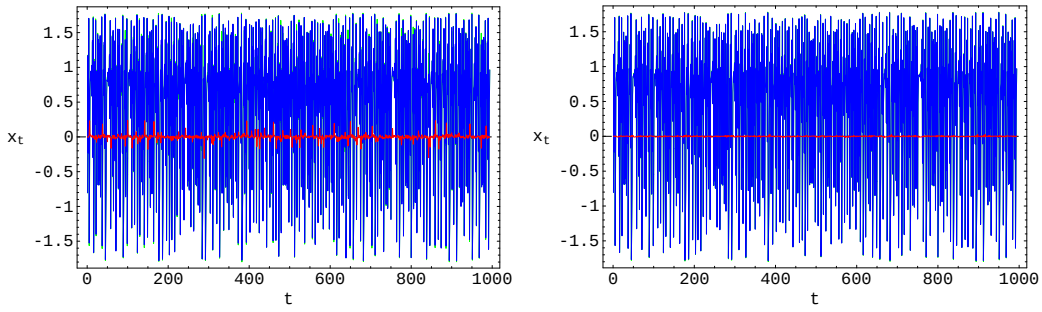
(a) The Gamma scatter plot for the irregular embedding 001010 shows noise (scatter points in the low δ high γ region) even though $\Gamma = 3.054 \times 10^{-5}$ was the lowest measured for the selected embedding dimension.

(b) The Gamma scatter plot for the irregular embedding 001011. This embedding did not have the lowest Gamma statistic, but in comparison had a lower gradient, A, and shows no noise on the scatter plot.

Figure 4.19: Two Gamma scatter plots, generated from irregular embeddings of the Hénon map, demonstrate that the Gamma statistic is not the only measure to consider when selecting inputs for a model.

in the region that indicates noise, whereas the irregular embedding 001010 does indicate that the data would be difficult to model.

Figure 4.20 shows the test results from two models built from the irregular embeddings 001010 and 001011 of the Hénon map. The models had the same architectural complexity (structure), but only the model based on the irregular embedding 001011 trained to its expected MSE, determined by Γ.



(a) The model built using the irregular embedding `001010`.

(b) The model built using the irregular embedding `001011`.

Figure 4.20: The models created from the irregular embeddings `001010` and `001011` of the Hénon map show that the irregular embedding `001011` provides a much better neural network model when using the same level of model complexity (in this case two hidden layers containing 5 nodes each). The actual output values are shown using the green line, which is obscured by the blue line that shows the model output. The error between the actual output and the model output is indicated by the red line.

The 5-scroll Chua attractor

Using the embedding dimension of 15 computed in Section 4.4.4 for the 5-scroll Chua attractor, a full feature space search was performed to identify the irregular embedding. The best results are shown in Table 4.5 with a corresponding chart showing the significance of each of the inputs in Figure 4.21.

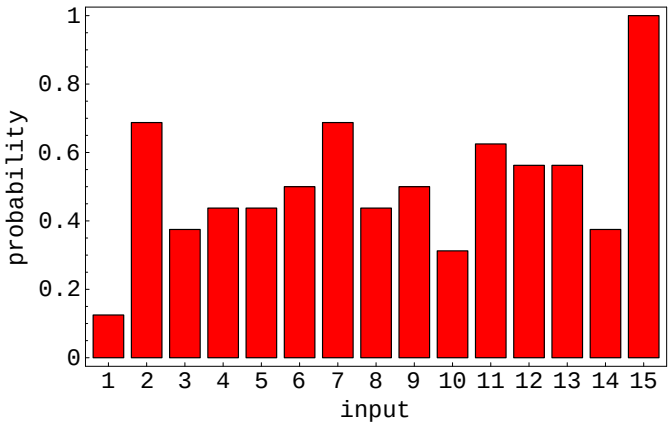


Figure 4.21: The best irregular embeddings from a complete feature space search of the time series generated from the Chua 5-scroll attractor ($|\Gamma| < 3.8 \times 10^{-8}$, $M = 1984$, $d = 15$) shows that inputs 2, 7, 11, 12, 13 and 15 should provide the best model.

$\|\Gamma\|$	A	Mask $(x_1 \ldots x_{15})$
5.98×10^{-9}	0.118676	010111100110011
9.81×10^{-9}	0.11795	011011001110011
1.06×10^{-8}	0.184786	000110100011001
1.10×10^{-8}	0.164729	001001001010101
1.14×10^{-8}	0.120221	100001100101111
1.41×10^{-8}	0.10644	011111101001111
1.63×10^{-8}	0.152616	010100111000101
2.29×10^{-8}	0.170846	010000110011001
2.62×10^{-8}	0.153349	010001100000111
2.66×10^{-8}	0.135786	010011111001101
3.02×10^{-8}	0.172887	011100110110111
3.05×10^{-8}	0.155446	001110101011001
3.06×10^{-8}	0.172044	001001010011001
3.72×10^{-8}	0.171036	110000001001101
3.73×10^{-8}	0.134671	010100110111101

Table 4.5: The best irregular embeddings from a complete feature space search of the time series generated from the Chua 5-scroll attractor ($|\Gamma| < 3.8 \times 10^{-8}, M = 1984, d = 15$).

Figure 4.21 shows six inputs from the 15 available that potentially contribute significant information. Here we are defining inputs as significant if they appear in at least 50% of the cases. This limit may be refined for other functions, for example the feature selection analysis in Section 4.1 used a much higher threshold to select relevant inputs.

4.5 The Gamma test analysis of a random walk: a salutary example

For time series analysis, particularly using a single time series, the *MSE* is *not* invariably a useful measure of the predictive value of a model. The following is a salutary example. A *random walk* time series is generated by taking the current value and, with probability 0.5, adding $+/-1$ to it to generate the next value. Figure 4.22 shows a random walk generated using 10000 points.

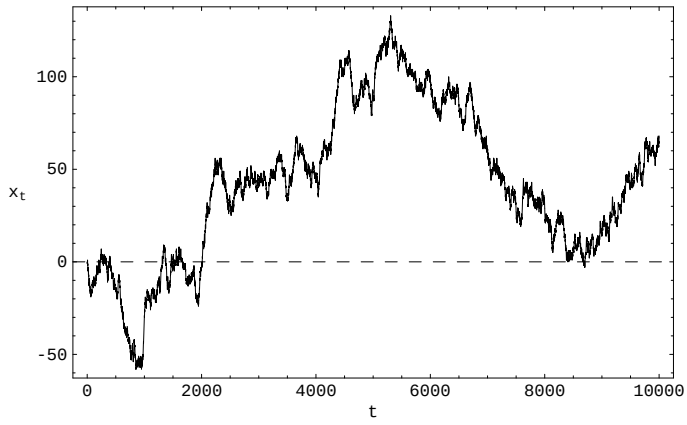


Figure 4.22: A random walk time series generated by taking the current value and, with probability 0.5, adding $+/-1$ to obtain the next value ($M = 10000$ points).

The expected absolute value of the series after M steps is around $\sqrt{M}$ and so after a large number of steps the change from one step to the next will be relatively small compared to the actual current value. This small local change will manifest itself as a kind of low noise measurement with the Gamma test returning a small value for the estimated *MSE*. Indeed we *can* construct one-step prediction models that predict with this MSE as we shall demonstrate.

Since the random walk is a time series, an embedding dimension analysis was performed to determine the optimal embedding dimension. Table 4.6 shows an estimated MSE $\Gamma \approx 1$ for embedding dimensions $1 \leq d \leq 4$. It is interesting to note that A appears to be inversely proportional to the embedding dimension.

Figure 4.23 shows the results of the increasing embedding dimensionality analysis for $1 \leq d \leq 20$. It would appear at first glance that the greater the dimensionality, the better the model. However this is a misleading assumption due to the *curse of dimensionality*, where $M = 10000$ points are insufficient in $d = 20$ dimensions to provide an accurate Γ statistic. M-tests performed in 2 and 20 dimensions shows that the Gamma statistic is stable in 2 dimensions but not in 20 dimensions indicating that more data is required for higher dimensional analysis.

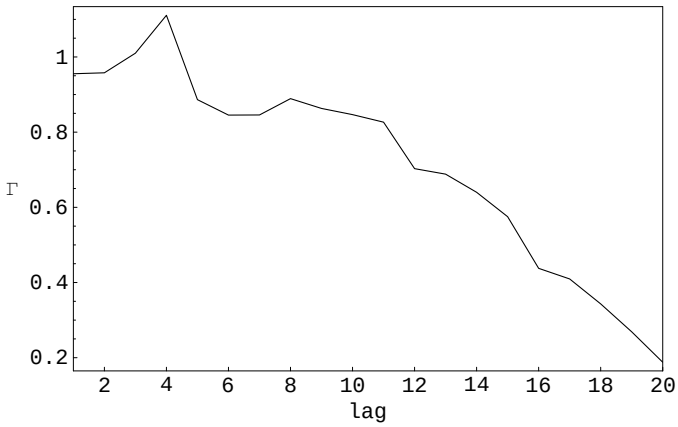
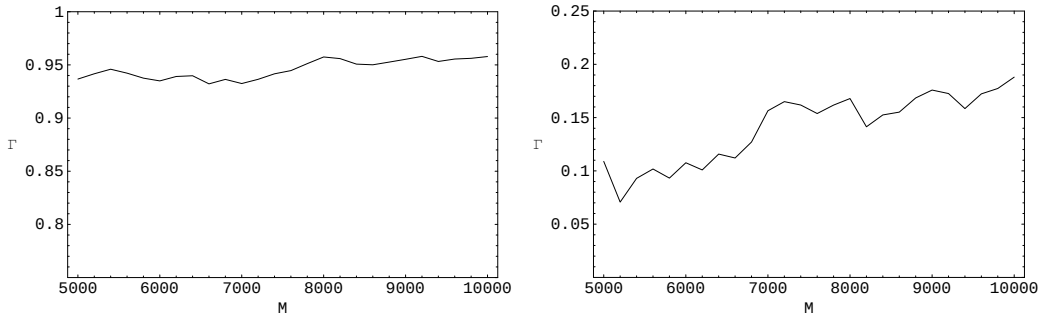


Figure 4.23: An increasing embedding performed on the random walk time series indicates that the analysis may be affected by the *curse of dimensionality*. Further analysis shows that more points are required to provide an accurate Gamma statistic.



(a) An M-test performed on the $d = 2$ dimensional embedding shows that $M = 10000$ points are sufficient to get an accurate estimate of the Gamma statistic.

(b) An M-test performed on the $d = 20$ dimensional embedding shows that the asymptotic convergence of $M = 10000$ points is insufficient to get an accurate estimate of the Gamma statistic.

Figure 4.24: The M-tests performed on the random walk time series for embedding dimensions $d = 2$ and $d = 20$.

Γ	A	V-ratio	embedding dimension
0.95535	0.49886	0.0005628	1
0.95786	0.24916	0.0005643	2
1.01004	0.17087	0.0005950	3
1.11083	0.12863	0.0006544	4

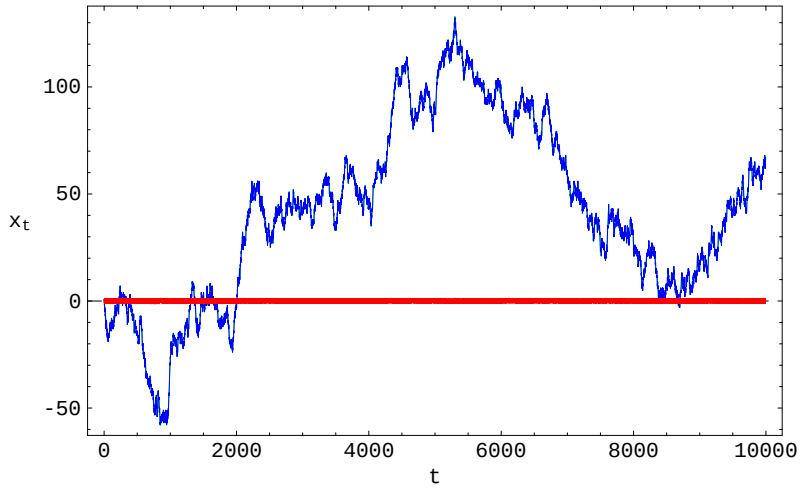
Table 4.6: An embedding dimension search for a random walk ($M = 10000$). The embedding dimension does not significantly change the value of the Γ statistic ($\Gamma \approx 1$). The V-ratio indicates that there would be approximately a 5.5% error on any prediction.

If we increase M to 65000 points then in 2 dimensions $\Gamma = 0.993568$ and in 20 dimensions $\Gamma = 0.49973$. In 2 dimension the Gamma statistic is virtually unchanged providing confidence that indeed the estimate for Γ was accurate. However, in 20 dimensions the corresponding Gamma statistic value has still not converged.

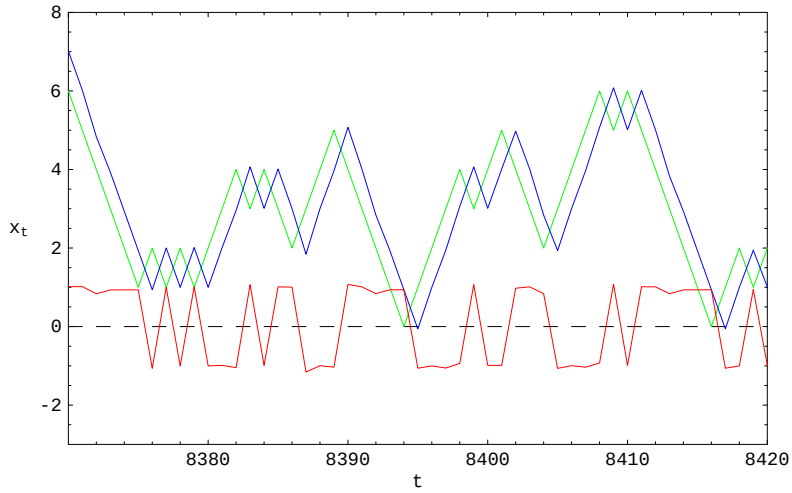
In an arbitrary test, a full feature space search over 5 dimensions shows that the previous value is the most significant at predicting the next value. This is illustrated in Table 4.7 where input $x_5 = x_{t-1}$ is selected for *all* of the best results.

Figure 4.25 shows a predictive model constructed from the random walk time series with embedding dimension $d = 5$ and an irregular embedding `01101` (the result with the lowest Γ value from Table 4.7). A model built using this input combination is shown in Figure 4.25. Figure 4.25(a) appears to show a satisfactory model, but if we look closer, as in Figure 4.25(b), then we can see that the model constantly predicts the next value as being approximately equal to the last value in the time series (hence there appears to be a lag of one time step between the model output and the actual output).

We can conclude that the predictive value of random walk *type* models is zero since the change of $+/-1$ from one step to the next is entirely random and the model has failed to predict any of the turning points. Indeed, it is apparent that the model uses approximately the previous value as the forecast for the next value. The absolute error between the actual output and the predicted value is then always approximately 1. The MSE or variance of this error is also 1 and this corresponds to the Gamma estimates in Table 4.6 for the lower dimensions $1 \leq d \leq 4$. Indeed, although the model was predicted to have a MSE performance of 0.858538 the model could only achieve a $MSE \approx 1$ on independent training and test sets. This difference between the predicted model performance and the actual attainable model performance can be explained by the M-test experiments shown in



(a) The response of the model appears to be identical to the actual signal. However, a closer inspection reveals that the model output lags behind the actual signal by one time step, as shown in Figure 4.25(b).



(b) The model output is one step behind the actual signal. This provides the model with a low MSE but provides no predictive capabilities.

Figure 4.25: A model built from the random walk time series with embedding dimension $d = 5$ and irregular embedding `01101`. The first 6000 points were used to train the model with an additional 4000 points being used to test the model. The error performance ($MSE \approx 1$) is constant for the training and test sets showing that the model has not been overtrained.

Γ	A	V-ratio	mask $x_1, \ldots, x_5$
0.858538	0.214628	0.000506	01101
0.886423	0.130978	0.000522	11111
0.92067	0.189405	0.000542	10101
0.950982	0.234933	0.00056	11001
0.955484	0.498851	0.000563	00001
0.958221	0.249163	0.000565	00011
0.972159	0.249172	0.000573	00101
0.976081	0.250479	0.000575	10001
0.98153	0.249072	0.000578	01001
1.000665	0.140286	0.00059	10011
1.010141	0.170882	0.000595	00111
1.013751	0.127712	0.000597	10111
1.019968	0.155698	0.000601	01011
1.054338	0.130582	0.000621	11011
1.110924	0.128635	0.000654	01111
1.244438	0.139867	0.000733	11101
1.828649	0.216258	0.001077	11010
1.901981	0.497633	0.001121	00010
1.925159	0.248459	0.001134	00110
1.955525	0.2482	0.001152	01010
1.959832	0.249239	0.001155	10010
1.994026	0.170071	0.001175	01110
1.994565	0.156041	0.001175	10110
2.090096	0.12856	0.001231	11110
2.873091	0.496461	0.001693	00100
2.906746	0.248162	0.001712	01100
2.953016	0.248013	0.00174	10100
2.982243	0.170445	0.001757	11100
3.871387	0.495112	0.002281	01000
3.915813	0.247974	0.002307	11000
4.904878	0.493297	0.00289	10000

Table 4.7: A feature selection search on a random walk shows that the most recent lag $x_5 = x_{t-1}$ is the most significant input, where all of the top results use input x_{t-1} and $0.85 < \Gamma < 1.25$. The worst results exclude x_{t-1} where $1.8 < \Gamma < 5$. This arises because the last value is (in probability) the closest to the next randomly generated value in the time series. The embedding dimension is $d = 5$ and $M = 10000$.

Figure 4.24, since that, in all likelihood, there was insufficient data at this dimension to accurately estimate the expected MSE.

The Gamma test analysis of this data set using 5 inputs and 1 output and an irregular embedding of 01101 gives a V-ratio (i.e. the normalised Gamma statistic $\Gamma/\text{var}(x_t)$) of around 0.000506 which

in the normal course of events would suggest that this time series is highly predictable (roughly a 5% error). In a sense it is highly predictable for we know that the next value will never differ from the current value by more than $+/-1$.

When analysing financial time series or stock market data we often (but not necessarily inevitably) find that the time series behaves very much like a random walk. Since many financial time series exhibit the same characteristics as a random walk, a prediction method will have to look for multiple time series that contain leading indicator information for the time series to be predicted.

4.6 Estimating model complexity

In Section 4.4.6 we described the gradient A as a statistic that describes model complexity. The comparison between two irregular embeddings for the Hénon map illustrated that Γ is not the only useful measure to determine the optimal model.

Although we do not intend to go into great detail about the gradient A of the regression line fit, it is important to mention that when selecting relevant or causal features it is also helpful to consider the complexity of the resulting functional surface. For two possible subsets of features with the same Gamma statistic it can easily happen that one will result in a more complex model surface than the other.

We know from the theoretical analysis that the slope estimate A returned by the Gamma test (Algorithm 4) depends to a large extent on the average of $|\nabla f|^2$ over the input space. To date, various proposals have been made to automatically determine optimal neural network architectures from the gradient A by [Končar, 1997] and [Tsui, 1999], however we believe this requires further investigation.

4.7 Conclusions

Feature selection using the Gamma test has provided a practical benefit for dimensionality reduction and model optimisation. One of the first practical demonstrations was to classify genetic sequences, [Chuzhanova et al., 1998]. Many unpublished examples of feature selection have been demonstrated by the sponsors of this work, Universal Solutions, who have produced many successful modelling projects in the field of advertising and marketing.

Feature selection prior to model building has revolutionised the way in which neural networks and other models are constructed. In the past, feature selection was made using a constructed model, where many models had to be built to provide comparison between different input selections. In effect, the Gamma test has decoupled feature selection from the model building process, making it independent of the model type.

A complete feature space search can usually be carried out in reasonable time for $d < 20$. However, we have demonstrated that an heuristic technique, such as using a genetic algorithm, can provide a reliable estimate of the optimal features in a short time.

In Section 4.2.1 we demonstrated that the Gamma statistic returns the average noise variance across the data set. This is an important result that shows that the noise distribution need not be constant to get a reliable estimate of the noise variance.

Time series analysis using the Gamma test provides comparable results to the False Nearest Neighbour (FNN) technique, described in Section 4.4.1, for finding the optimal embedding dimension. This is encouraging because the FNN technique is an established method for determining the optimal embedding dimension and the Gamma test produces very similar results.

The example of the random walk in Section 4.5 shows that the Gamma test can find the optimal model for the given data, but it does not always produce the anticipated result. Trying to predict the future of a time series using previous values from the same time series will often produce a model with no predictive power. The exceptions are models constructed from smooth dynamical systems where the attractors can often be found, as shown in the examples using the Hénon map in this and the previous chapter.

CHAPTER 5

Higher Moments Gamma Test

In Chapter 3 we introduced the Gamma test as a technique for estimating the variance of the noise var(r) contained within data generated by a smooth, continuous, non-linear system.

In this chapter we propose a natural extension to the Gamma test in the form of a system of equations which we conjecture provide a link between the higher moments of the noise distribution and certain easily computed regression line intercepts. The goal here is to derive further information regarding the noise distribution by estimating the moments. We first introduce a heuristic derivation of these equations and then provide some experimental evidence.

For non-symmetric unbounded noise distributions it emerges that even if the conjectured equations are true so much data is required to give accurate estimates of the regression line intercepts that the method would in most cases be impractical.

There is a further problem in that for non-symmetric distributions the equations alone do not provide sufficient information to solve for the higher moments. However, for *symmetric* noise distributions much less data is required and, because we may assume that all odd moments are zero, there are sufficient equations to solve for as many higher *even* moments as is justified by the amount of data.

Finally we show that if the noise distribution is *assumed symmetric* then the estimates for the even moments can be used to approximately reconstruct the original noise distribution.

5.1 Moments

Data is often represented in a relatively simple way in order to understand and characterise it. In the introduction to the Gamma test in Chapter 3, a noise distribution was characterised in terms of its variance var(r). The data can be further characterised using the higher moments of the noise distribution, and in addition these can often be used to reconstruct the noise distribution.

In our discussions so far we have used the mean and variance, which are the first and second moments of a distribution respectively, to describe the noise. The mean of a noise distribution is assumed to be zero since any bias can be incorporated into the model. However, the mean and variance tell us very little about the overall *shape* of the distribution.

Higher moments can provide greater evidence for the shape of a distribution. For example the third moment, *skewness*, is often used to describe the asymmetry of a distribution. The ratio of the fourth moment about the mean of a distribution to the square of the variance, is independent of the unit employed. This invariant of the distribution is called its *kurtosis*, and is frequently denoted by β_2 [Kendall and Stuart, 1963]. For a normal distribution $\beta_2 = 3$.

The idea of moments has its origin in mechanics where we can describe the moment of a force, M, as being the force, f_1, multiplied by the perpendicular distance, x_1, from the force to the fulcrum. Hence the moment of the force is $f_1 x_1$. If there are several forces acting then the total moment of all these forces is $M = f_1 x_1 + f_2 x_2 + f_3 x_3 + \ldots = \sum f_i x_i$.

Moments are defined either about the origin, the mean of a distribution, or around a reference point [Rosander, 1957]. Unfortunately we cannot use these existing techniques to calculate the moments of the noise distribution from our non-linear data because the noise distribution is entangled within the data arising from a smooth model. However, we already have the Gamma test to calculate the second moment, and it turns out to be relatively straightforward to adapt it to calculate the higher moments as we shall demonstrate in Section 5.2.

If we consider the moments about the mean, a, we can write a general formula for the moment calculation

$$M_l = \frac{1}{M} \sum_{i=1}^{j} f_i (r_i - a)^l \tag{5.1}$$

where l is the order of the moment ($l = 1$ corresponds to the mean, $l = 2$ the variance, $l = 3$ the skewness, and so on) and M is the number of samples in the population. f_i is the number of sample values of $(r_i - a)$ in the i^{th} group and j is the number of groups such that $\sum_{i=1}^{j} f_i = M$.

5.2 Higher moments: an extension of the Gamma test

The argument of Section 3.3.1, which provides a relationship $G_2 = 2M_2$ between the second moment $\Gamma = M_2$ of the noise distribution and the vertical intercept G_2 of the regression line between $\delta_M(k)$ and $\gamma_M(k)$, can be extended.

For even $l \geq 2$ let G_l be the vertical intercept of the regression line between $\delta_M(k)$ and $\gamma_M(k,l)$ where

$$\delta_M(k) = \frac{1}{M}\sum_{i=1}^{M}(x_{N[i,k]} - x_i)^2 \tag{5.2}$$

and

$$\gamma_M(k,l) = \frac{1}{M}\sum_{i=1}^{M}(y_{N[i,k]} - y_i)^l \tag{5.3}$$

Now consider the analogous expression to (3.9) obtained by raising (3.8) to the l^{th} power, i.e.

$$\begin{aligned}(y'-y)^l &= \left((r'-r) + (\mathbf{x}'-\mathbf{x})\cdot\nabla f(\mathbf{x}) + o(|\mathbf{x}'-\mathbf{x}|)\right)^l \\ &= (r'-r)^l + \binom{l}{1}(r'-r)^{l-1}(\mathbf{x}'-\mathbf{x})\cdot\nabla f(\mathbf{x}) + \\ &\quad \binom{l}{2}(r'-r)^{l-2}\left((\mathbf{x}'-\mathbf{x})\cdot\nabla f(\mathbf{x})\right)^2 + o(|\mathbf{x}'-\mathbf{x}|^2)\end{aligned} \tag{5.4}$$

where r' denotes the noise associated with a near neighbour $\mathbf{x}'$ of $\mathbf{x}$ etc. Since r' and r are independent we have

$$\mathcal{E}\left((r'-r)^l\right) = \sum_{j=0}^{l}(-1)^{l-j}\binom{l}{j}\mathcal{E}(r'^j)\mathcal{E}(r^{l-j}) \tag{5.5}$$

We recall that r' and r are distributed identically so that (5.5) becomes

$$\mathcal{E}\left((r'-r)^l\right) = \sum_{j=0}^{l}(-1)^{l-j}\binom{l}{j}M_jM_{l-j} \tag{5.6}$$

where $M_0 = 1$, $M_1 = 0$ and M_l (for $l \geq 2$) is the l^{th} moment about the mean for the distribution of r.

Summing both sides of (5.4) we obtain

$$\begin{aligned}\frac{1}{M}\sum_{i=1}^{M}(y_{N[i,k]} - y_i)^l &= \frac{1}{M}\sum_{i=1}^{M}(r_{N[i,k]} - r_i)^l \\ &\quad + \frac{1}{M}\sum_{i=1}^{M}\binom{l}{1}(r_{N[i,k]} - r_i)^{l-1}(\mathbf{x_{N[i,k]}} - \mathbf{x}_i)\cdot\nabla f(\mathbf{x}_i) \\ &\quad + \frac{1}{M}\sum_{i=1}^{M}\binom{l}{2}(r_{N[i,k]} - r_i)^{l-2}\left((\mathbf{x_{N[i,k]}} - \mathbf{x}_i)\cdot\nabla f(\mathbf{x}_i)\right)^2 \\ &\quad + o(\delta_M(k))\end{aligned} \tag{5.7}$$

as $M \to \infty$. Now, since l is even, $l-1$ is odd so that $\mathcal{E}\left((r_{N[i,k]} - r_i)^{l-1}\right) = 0$. Hence asymptotically the second term on the RHS is zero and

$$\begin{aligned}\frac{1}{M}\sum_{i=1}^{M}(y_{N[i,k]} - y_i)^l \quad &\approx \quad \frac{1}{M}\sum_{i=1}^{M}(r_{N[i,k]} - r_i)^l \\ &\quad + \frac{1}{M}\sum_{i=1}^{M}\binom{l}{2}(r_{N[i,k]} - r_i)^{l-2}\left((\mathbf{x}_{\mathbf{N[i,k]}} - \mathbf{x}_i)\cdot\nabla f(\mathbf{x}_i)\right)^2 \\ &\quad + o(\delta_M(k)) \end{aligned} \tag{5.8}$$

Notice that, proceeding as before, the slope of the regression line now really does depend on l in addition to possibly depending on k, and from (5.6) also depends on the noise moments $M_0, \ldots, M_{l-2}$. Following the line of proof in the Gamma test we can show that the regression line intercept G_l converges in probability to $\mathcal{E}\left((r'-r)^l\right)$ as $M \to \infty$. Then from (5.6) and (5.8) using $M_1 = 0$ we obtain for $l = 2, 4, 6, 8, 10, \ldots$ the following limiting equations as $M \to \infty$ (with convergence in probability)

$$\begin{aligned} G_2 &\approx 2M_2 \\ G_4 &\approx 2M_4 + 6M_2^2 \\ G_6 &\approx 2M_6 + 30M_2M_4 - 20M_3^2 \\ G_8 &\approx 2M_8 + 70M_4^2 - 112M_3M_5 + 56M_2M_6 \\ G_{10} &\approx 2M_{10} - 240M_7M_3 + 420M_4M_6 - 252M_5^2 + 90M_2M_8 \\ &\ldots \end{aligned} \tag{5.9}$$

Computing the G_l and solving equations (5.9) successively will immediately give M_2 and M_4 from the equations for G_2 and G_4. We cannot proceed beyond this, at least without further equations or assumptions, since there are too many unknown quantities. Computing G_6 for example leaves two unknown terms M_6 and M_3. Unfortunately, computing G_l for odd l is easily seen to provide no useful information[1] so that the odd moments cannot be determined in this way. Hence, if we are interested in determining the higher moments of the noise distribution we must make further assumptions.

5.3 Non-symmetric noise distributions

The conjectured equations (5.9) link G_l for even l with the theoretical moments of the noise distribution. Since these equations also involve odd moments, plainly for $l > 4$ we have more unknowns

[1] $G_l \to 0$ as $M \to \infty$ for odd l using (5.6).

than equations. Hence measuring G_l for $l > 4$ on data suspected of having a *non-symmetric* noise distribution is not helpful because at present we have no method for estimating the odd moments.

We can however perform some experiments with known non-symmetric noise distributions in order to check whether equations (5.9) seem likely to be true. This is achieved using the known moments of a variety of non-symmetric noise distributions.

The function defined in (3.13)

$$f(x) = \sin(4\pi x) + \cos(2\pi x) \tag{5.10}$$

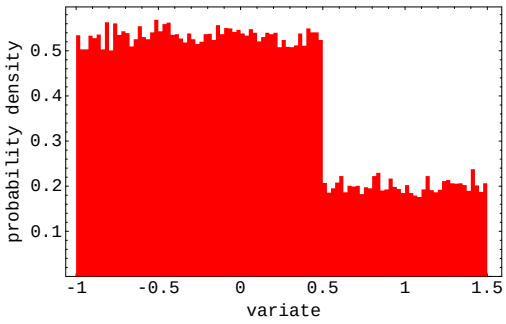
will be used to produce the underlying smooth function for the experiments in this section. The experiments performed use two different non-symmetric noise distributions: (1) a distribution composed by combining a pair of uniform distributions (we refer to this as a *uniform distribution-pair*), and (2) a lognormal distribution. These distributions are described in Appendices C.1 and C.2 respectively.

The experiments to calculate G_l use a uniform distribution-pair with `widthLeft` $= 1.5$, `widthRight` $= 1$, `meanRight` $= 1$, `numPoints` $= 50000$, and `proportionRight` $= 0.2$ as required for Algorithm 14. There are two additional experiments using the lognormal distribution; the first uses a distribution whose shape is less asymmetric with $\mu = 2$ and $\sigma = \sqrt{0.4}$, closer to the symmetric noise distribution case, and the second uses a more pronounced asymmetric distribution with $\mu = 0.5$ and $\sigma = \sqrt{0.4}$. These two lognormal distributions may indicate how the asymmetric nature of the distribution affects the calculation of G_l with respect to the expected values given by (5.9). For brevity, these distributions will be identified as *uniform distribution-pair*, *lognormal distribution* $\mu = 2$, and *lognormal distribution* $\mu = 0.5$.

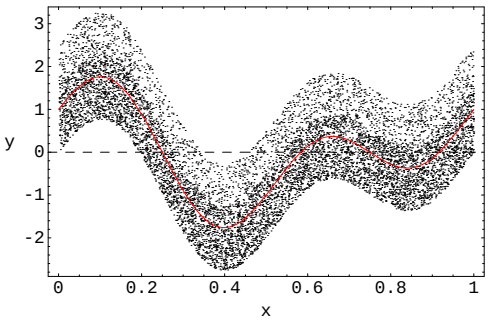
The noise distributions used for these experiments, and some example sampled points around the smooth function (5.10), are shown in Figure 5.1.

5.3.1 Experimental verification of (5.9)

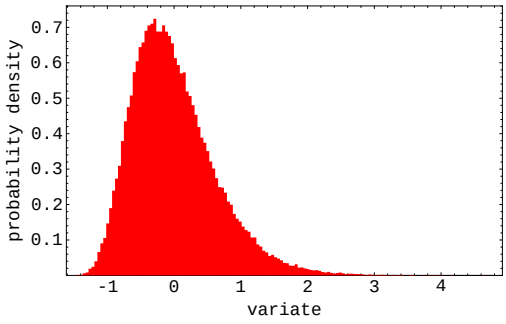
We attempt to verify (5.9) by inserting into these equations the known moments of our artificial noise. In this way we can arrive at a prediction for the values of G_l for even l. The higher moments Gamma test is used to compute the appropriate regression lines, for increasing M, to arrive at numerical estimates for G_l. Comparison of these values enable us to test the validity of (5.9).



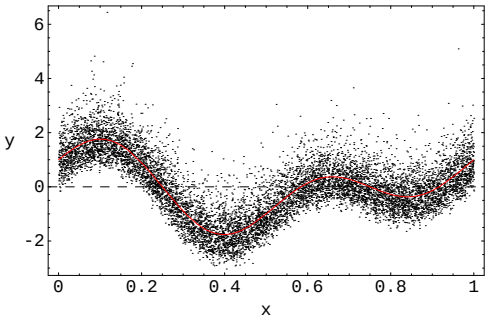
(a) A uniform distribution-pair with `widthLeft` $= 1.5$, `widthRight` $= 1$, `meanRight` $= 1$, `numPoints` $= 50000$, and `proportionRight` $= 0.2$ (Algorithm 14).



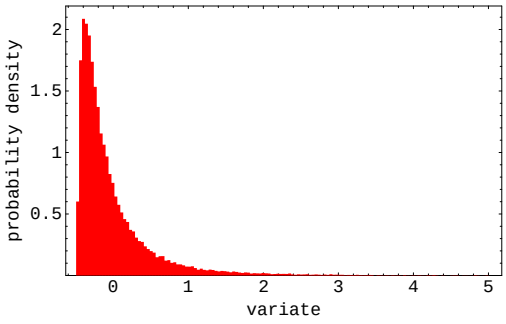
(b) The data points generated from the uniform distribution-pair in Figure 5.1(a) shown with the underlying smooth function.



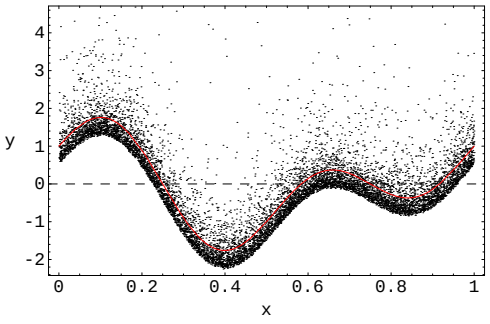
(c) The lognormal distribution with $\mu = 2$ and $\sigma = \sqrt{0.4}$.



(d) The data points generated from the lognormal distribution in Figure 5.1(c) shown with the underlying smooth function.



(e) The lognormal distribution with $\mu = 0.5$ and $\sigma = \sqrt{0.4}$.



(f) The data points generated from the lognormal distribution in Figure 5.1(e) shown with the underlying smooth function.

Figure 5.1: The non-symmetric noise distributions used to estimate the asymptotic nature of the moments.

The known theoretical moments for the three non-symmetric noise distributions are shown in Table 5.1 where each distribution is shown to have approximately the same variance ($M_2 \approx 0.4$).

Moment	Uniform distribution-pair	Lognormal distribution	
		$\mu = 2$	$\mu = 0.5$
1	0	0	0
2	0.41666	0.4	0.4
3	0.12500	0.248	1.472
4	0.41250	0.76097	15.7007
5	0.29166	1.47	397.684
6	0.56473	4.36534	2.49511×10^4
7	0.57421	13.8841	3.95841×10^6
8	0.91362	52.6426	1.60958×10^9
9	1.10000	226.69	1.69006×10^{12}
10	1.62118	1111.27	4.59940×10^{15}

Table 5.1: The theoretical moments of the non-symmetric noise distributions.

One problem associated with calculating the higher moments directly from the data is that a great deal of data is required for lognormal distributions to get accurate results. Table 5.2 illustrates the problem clearly for a lognormal distribution with $\mu = 2$ and $\sigma = \sqrt{0.4}$. Using $M = 50000$ data points and calculating directly from the generated noise data, the higher moments are 'inaccurate' compared to the theoretical values of the moments, and even for $M = 10^6$ or $M = 10^7$ convergence to the theoretical values is not really achieved.

The theoretical values of G_l for the three distributions were calculated using (5.9) with the theoretical moments given in Table 5.1. These theoretical values of G_l are shown in Table 5.3. The data derived estimates of G_l calculated using the `CentralMoment` routine in *Mathematica* are shown in Table 5.4.

Experimental validation of (5.9) should show that the higher moments Gamma test estimate for G_l converges to either the theoretical G_l given in Table 5.3, or to the data derived estimates for G_l given in Table 5.4, depending on the noise distribution.

Moment	Theoretical	$M = 50000$	$M = 10^6$	$M = 10^7$
1	0	0	0	0
2	0.4	0.39700	0.39928	0.39978
3	0.248	0.23966	0.24661	0.24790
4	0.76097	0.72525	0.75564	0.76080
5	1.47	1.32874	1.44478	1.46947
6	4.36534	3.71976	4.22978	4.35184
7	13.8841	10.6917	13.0859	13.7170
8	52.6426	35.3785	47.5822	50.9471
9	226.69	125.909	192.103	210.681
10	1111.27	480.292	857.108	963.253

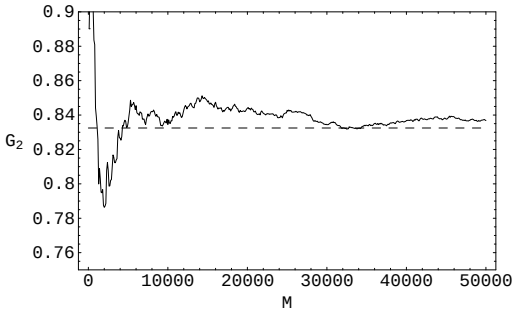
Table 5.2: The theoretical and experimental moments of a lognormal noise distribution with $\mu = 2$, $\sigma = \sqrt{0.4}$. The experimental moments were calculated using the `CentralMoment` routine in *Mathematica*.

	Uniform	Lognormal distribution	
G_l	distribution-pair	$\mu = 2$	$\mu = 0.5$
2	0.83333	0.8	0.8
4	1.86667	2.48195	32.3614
6	5.97321	16.6323	5.00474×10^4
8	22.8319	202.774	3.21968×10^9
10	96.6792	4141.95	9.19886×10^{15}

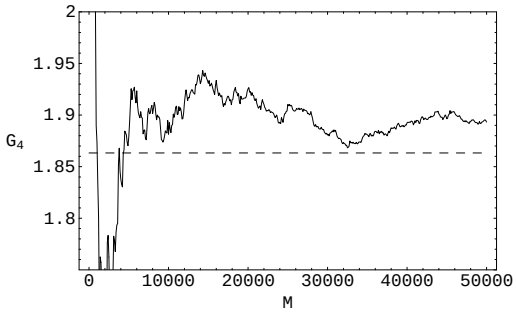
Table 5.3: The theoretical G_l derived from (5.9) and the theoretical moments of the non-symmetric noise distributions.

Experimental results

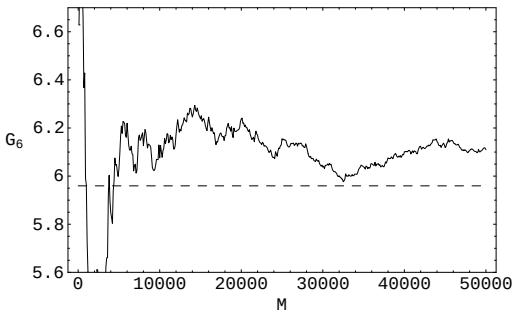
Figures 5.2, 5.3, and 5.4 show the higher moments Gamma test estimate for G_l with (where possible) two predicted lines: (1) the theoretical estimate of G_l, and (2) the noise-data derived estimate of G_l.



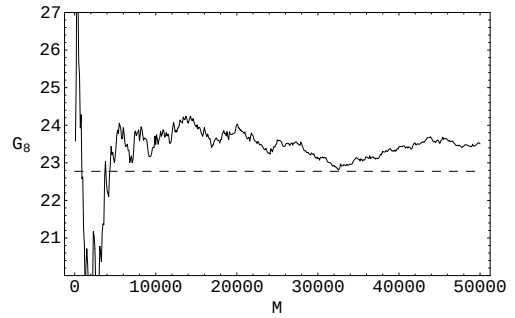
(a) The estimate for G_2 converges to the theoretical value.



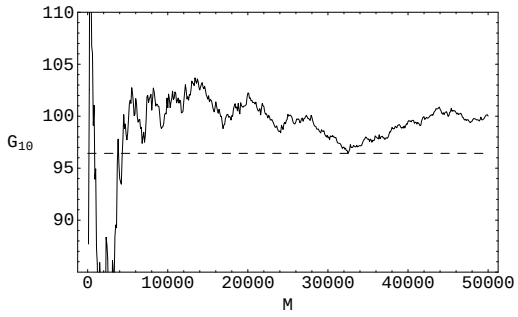
(b) The estimate for G_4 asymptotes to a value close to the theoretical value.



(c) The estimate for G_6 asymptotes to a value that over-estimates the theoretical value.

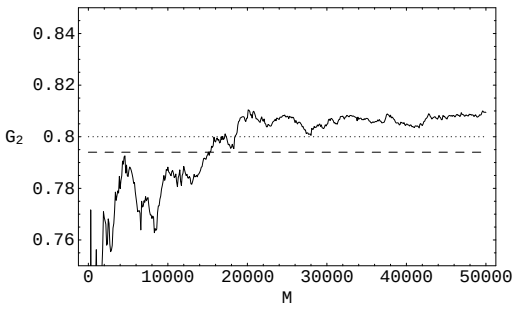


(d) The estimate for G_8 asymptotes to a value that over-estimates the theoretical value.

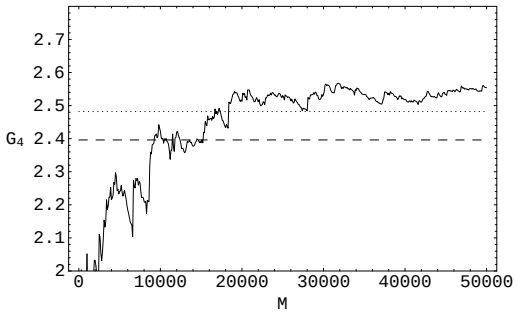


(e) The estimate for G_{10} asymptotes to a value that over-estimates the theoretical value.

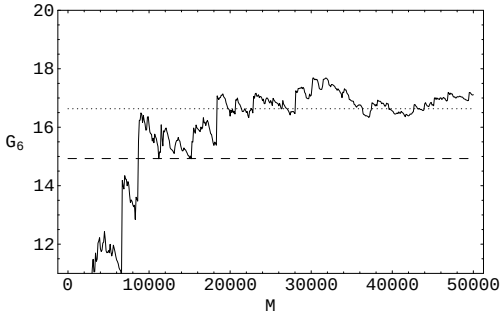
Figure 5.2: Asymptotic nature of G_l for the uniform distribution-pair described in Figure 5.1(a) with $M = 50000$ sampled points. The higher moments Gamma test estimate for G_l is shown relative to the values predicted by (5.9) using the data derived moments (shown as the dashed line). The value of G_l calculated from the theoretical moments are not shown since they approximately equal the noise-data derived moments.



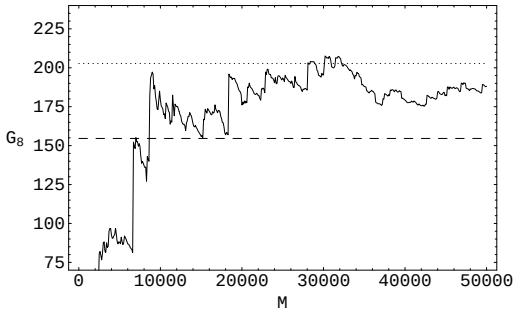
(a) The estimate for G_2 asymptotes to a value that over-estimates the theoretical value.



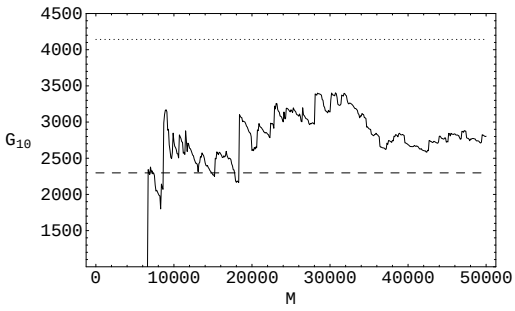
(b) The estimate for G_4 asymptotes to a value that over-estimates the theoretical value.



(c) The estimate for G_6 asymptotes to the theoretical estimate.

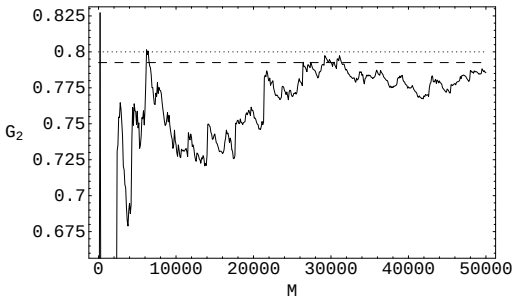


(d) The estimate for G_8 is bounded by the theoretical and data derived estimates.

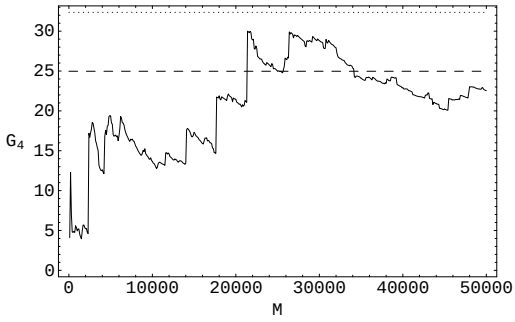


(e) The estimate for G_{10} is bounded by the theoretical and data derived estimates.

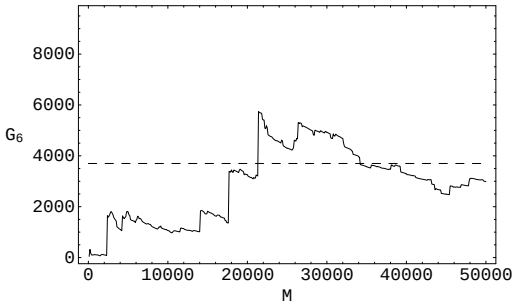
Figure 5.3: Asymptotic nature of G_l for a lognormal distribution with $\mu = 2$, $\sigma = \sqrt{0.4}$, and $M = 50000$ sampled points. The higher moments Gamma test estimate for G_l is shown relative to the values predicted by (5.9) using the theoretical moments (shown as the dotted line) and the noise-data derived moments calculated for $M = 50000$ (shown as the dashed line).



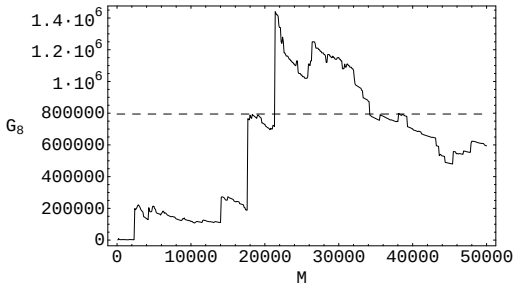
(a) The estimate for G_2 converges to the data derived estimate.



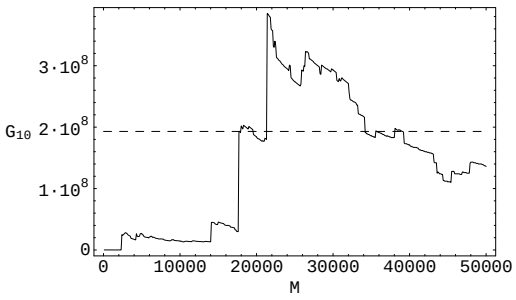
(b) The estimate for G_4 converges to the data derived estimate.



(c) The estimate for G_6 converges to the data derived estimate. The theoretical value $\approx 5 \times 10^4$.



(d) The estimate for G_8 converges to the data derived estimate. The theoretical value $\approx 3 \times 10^9$.



(e) The estimate for G_{10} converges to the data derived estimate. The theoretical value $\approx 9 \times 10^{15}$.

Figure 5.4: Asymptotic nature of G_l for a lognormal distribution with $\mu = 0.5$, $\sigma = \sqrt{0.4}$, and $M = 50000$ sampled points. The higher moments Gamma test estimate for G_l is shown relative to the values predicted by (5.9) using the theoretical moments (shown as the dotted line) and the noise-data derived moments calculated for $M = 50000$ (shown as the dashed line).

G_l	Uniform distribution-pair	Lognormal distribution	
		$\mu = 2$	$\mu = 0.5$
2	0.83247	0.79400	0.79256
4	1.86327	2.39618	24.9706
6	5.95950	14.9286	3699.17
8	22.7740	154.609	7.94720×10^5
10	96.4257	2297.83	1.93001×10^8

Table 5.4: The G_l derived from (5.9) using the experimental moments of the non-symmetric noise distributions calculated using the `CentralMoment` routine in *Mathematica* ($M = 50000$).

Analysis of results

Figure 5.2 verifies (5.9) for the uniform distribution-pair. Although the higher moments Gamma test slightly over-estimated G_l against the noise-data derived estimates and the theoretical estimates, the results were nevertheless very good indeed considering the algorithm had to deal with the smooth underlying function as well.

From the conclusion drawn from Table 5.2, the Gamma test estimates for G_l for the lognormal distributions given in Figures 5.3 and 5.4 were closer to the noise-data derived estimates for G_l than the theoretical values. This is because a substantial amount of data is required to closely approximate the theoretical values. However, since the higher moments Gamma test estimates for G_l were close to the noise-data derived estimates for the known noise distribution, we can be satisfied that the Gamma test is 'no worse' than existing techniques of measuring the moments, and is in fact the only known technique available to deal with data that contains an entangled unknown smooth function and noise distribution.

The confirmation of (5.9) has been much easier to demonstrate for a bounded distribution using the uniform distribution-pair than for the unbounded lognormal distributions. The less conclusive results for the lognormal distributions show a discrepancy between the higher moments Gamma test and the theoretical moments, which is likely to be directly attributable to the sample size. This hypothesis that (5.9) are correct has been partly substantiated using a separate analytical technique to calculate the moments from the noise distribution for large M (Table 5.2).

If indeed (5.9) are correct, then measuring G_l accurately for large l requires massively large data sets particularly if the distribution is unbounded. The sample size M also appears to be dependent on the skewness of the distribution, for example for the less skewed lognormal noise distribution

($\mu = 2$) the experimental and theoretical moments were similar and (for large l) bounded the higher moments Gamma test estimates for G_l. Both the higher moments Gamma test and other analytical techniques severely underestimate the theoretical values for G_l for reasonable M, $l > 2$, and high skewness (large M_3).

Table 5.5 shows the comparison between the theoretical moments and the higher moments Gamma test estimates for the three non-symmetric distributions. The G_l estimates for the uniform distribution were reasonably accurate at low $M = 1000$. The lognormal distribution with $\mu = 2$ provides reasonably accurate for $G_2 \ldots G_8$ for $M = 10000$. The highly skewed lognormal distribution with $\mu = 0.5$ could only approximate G_2 at $M = 50000$. This table confirms that highly skewed and unbounded distributions require many data points to calculate G_l.

One technical aspect to note for these experiments is the precision of the arithmetic used, since the algorithm depends on taking *differences* raised to the l^{th} power. These results were computed in both single precision and double precision in C++ to see whether there was a significant affect on accuracy. Although there were slight differences in the results, they were largely insignificant in their affect on the overall results. For large quantities of data where $\delta \to 0$ as $M \to \infty$, the measurements of δ and γ must be performed in at least double precision to maintain accuracy.

There is little point in examining the relationship between G_l and M_l because the odd moments cannot be determined using the higher moments Gamma test.

Uniform distribution-pair				
		M		
G_l	Theoretical	1000	10000	50000
2	0.83333	0.839124	0.834511	0.836781
4	1.86667	1.87385	1.8814	1.89319
6	5.97321	5.964	6.07577	6.10984
8	22.8319	22.6236	23.4911	23.5029
10	96.6792	94.9311	100.8080	100.0060

Lognormal distribution $\mu = 2$				
		M		
G_l	Theoretical	1000	10000	50000
2	0.8	0.756204	0.788235	0.809417
4	2.48195	2.05152	2.42253	2.55415
6	16.6323	10.1903	16.0798	17.1127
8	202.774	69.7921	179.286	188.159
10	4141.95	567.746	2730.99	2804.13

Lognormal distribution $\mu = 0.5$				
		M		
G_l	Theoretical	1000	10000	50000
2	0.8	0.55948	0.73561	0.785579
4	32.3614	4.94418	13.6598	22.5250
6	5.005×10^4	103.6920	1065.38	2985.32
8	3.220×10^9	2794.85	118771	594083
10	9.199×10^{15}	82805.5	1.450×10^7	1.360×10^8

Table 5.5: A comparison of the theoretical and experimental G_l of the non-symmetric noise distributions.

5.4 Symmetric noise distributions

Let us now assume that the noise distribution is symmetric, so that all higher odd moments are zero. Then equations (5.9) become (with convergence in probability)

$$\begin{aligned} G_2 &\approx 2M_2 \\ G_4 &\approx 2M_4 + 6M_2^2 \\ G_6 &\approx 2M_6 + 30M_2M_4 \\ G_8 &\approx 2M_8 + 70M_4^2 + 56M_2M_6 \\ G_{10} &\approx 2M_{10} + 420M_4M_6 + 90M_2M_8 \\ &\dots \end{aligned} \tag{5.11}$$

as $M \to \infty$.

We shall use the higher even moments Gamma test algorithm described in Section 5.4.1 to calculate the G_l from (5.11) and solve for the higher even moments M_l.

5.4.1 The higher even moments Gamma test algorithm

The higher even moments Gamma test algorithm is given in Algorithm 6. It is a natural extension of the original Gamma test algorithm described in Algorithm 4.

5.4.2 Experimental verification of (5.11)

If our diverse heuristic arguments are valid then under the assumption of symmetric noise these equations should permit us to efficiently compute as many higher order even moments as the quantity of data justifies.

We first show using a series of experiments that the computed values of G_l for symmetric noise distributions do indeed asymptote to the values predicted by (5.11) using the true values for the M_l.

We then show several experiments which illustrate that this method is in fact a remarkably effective way to estimate the higher even moments M_l. Of course, once the higher order moments are known or estimated we can then proceed to reconstruct the noise distribution and we shall return to this issue in Section 5.5.

```
{initialisation}
generate near neighbour structure (e.g. k-d tree)
for p = 1 to p_max do
  δ(p) = 0
  for l = 2 to k step 2 do
    γ(p,l) = 0
  end for
end for

{main algorithm}
for i = 1 to M do
  generate N[i,p] {find the p_max near neighbours of x(i)}
  for p = 1 to p_max do
    δ(p) = δ(p) + [x(i) − x(N[i,p])]^2
    for l = 2 to k step 2 do
      z(p) = 0
      for j = 1 to L(N[i,p]) do
        z(p) = z(p) + [y(i) − y(N[i,p][j])]^l
      end for
      γ(p,l) = γ(p,l) + [z(p)/L(N[i,p])]
    end for
  end for
end for

for p = 1 to p_max do
  δ(p) = δ(p)/M
  for l = 2 to k step 2 do
    γ(p,l) = γ(p,l)/M
  end for
end for

{Gamma statistics}
for l = 2 to k step 2 do
  Perform least squares fit on (δ(p), γ(p,l)) where (1 ≤ p ≤ p_max)
    to compute G_l from y = A_l x + G_l
end for

{Moments}
for l = 2 to k step 2 do
  Solve M_l from known G_n where (2 ≤ n ≤ l)
end for

return (M_l, A_l) for (2 ≤ l ≤ k)
```

Algorithm 6: The higher even moments Gamma test algorithm.

The following examples use (5.10) as the underlying function. Symmetric noise distributions were added to the function to examine the effectiveness of calculating G_l and hence estimating the moments M_l.

The three noise distributions chosen for these experiments are shown in Figure 5.5. Each distribution has a variance $M_2 \approx 0.4$ which is comparable to that used for the experiments using the non-symmetric noise distributions. The distributions used are: (1) a uniform noise distribution, (2) a normal distribution, and (3) a bimodal distribution that consists of two separated, but identically shaped, normal distributions.

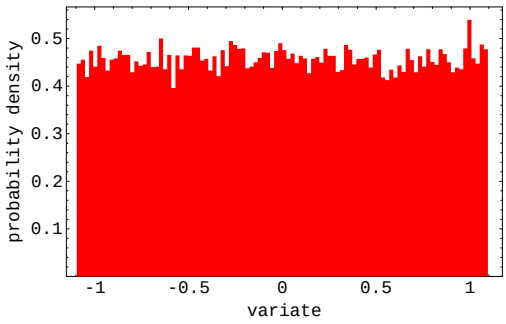
The theoretical moments for the three symmetric noise distributions are shown in Table 5.6 where each distribution is shown to have approximately the same variance ($M_2 \approx 0.4$). The uniform distribution is on the interval $[-1.095445, 1.095445]$ to give a variance of 0.4. The normal distribution has mean zero and variance 0.4, and the bimodal distribution is a combination of two normal distributions with mean $+/-\sqrt{0.4}/1.155$ and variance $\sqrt{0.4}/2$ to give a variance of 0.39984.

Moment	Uniform	Normal	Bimodal
1	0	0	0
2	0.4	0.4	0.399844
3	0	0	0
4	0.288	0.48	0.29981
5	0	0	0
6	0.24685	0.96	0.31174
7	0	0	0
8	0.2304	2.688	0.40880
9	0	0	0
10	0.22621	9.6768	0.64296

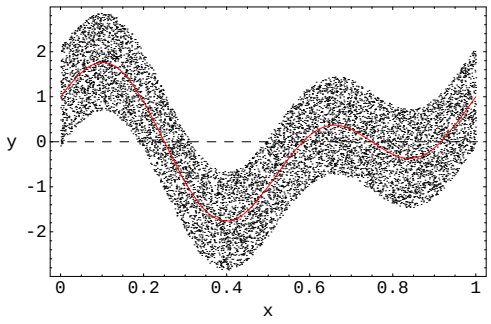
Table 5.6: The theoretical moments of the symmetric noise distributions

The theoretical moments given in Table 5.6 are inserted into (5.11) to predict a value for G_l for the distributions. Verification of the higher moments Gamma test estimate for G_l can be established if those values asymptote to the predicted theoretical values shown in Table 5.7.

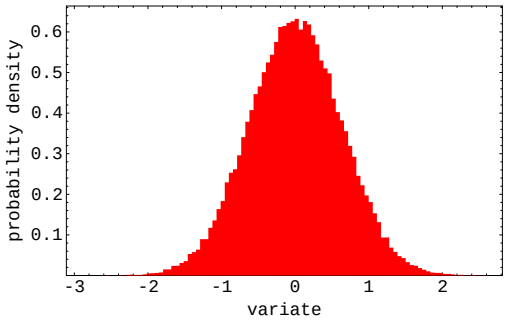
G_l has been independently calculated using data generated from the noise distributions. The moments were calculated using the `CentralMoment` routine in *Mathematica* using 50000 sampled points from each of the noise distributions. These values were then substituted into (5.11) to cal-



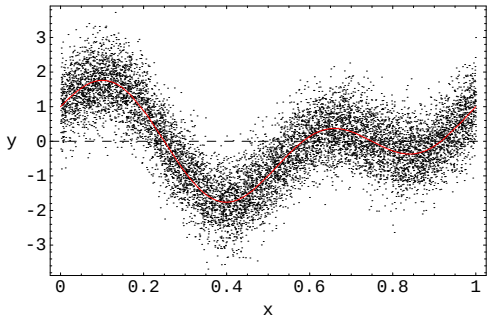
(a) A uniform distribution on the interval $[-1.095445, 1.095445]$ (variance 0.4).



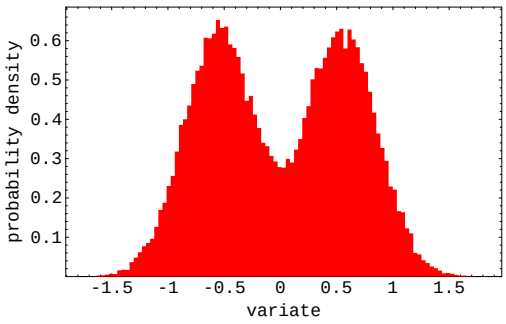
(b) The data points generated from the uniform distribution in Figure 5.5(a) shown with the underlying smooth function.



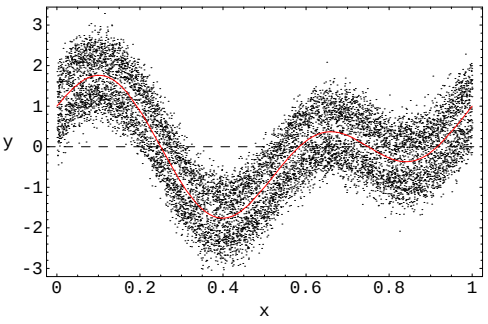
(c) The normal (Gaussian) distribution with mean zero and variance 0.4.



(d) The data points generated from the normal distribution in Figure 5.5(c) shown with the underlying smooth function.



(e) The bimodal distribution consisting of two normal distributions with mean +/- $\sqrt{0.4}/1.155$ and variance $\sqrt{0.4}/2$, which gives an overall variance of 0.39984 for the distribution.



(f) The data points generated from the bimodal distribution in Figure 5.5(e) shown with the underlying smooth function.

Figure 5.5: The symmetric noise distributions used to estimate the asymptotic nature of the moments.

G_l	Uniform	Normal	Bimodal
2	0.8	0.8	0.79968
4	1.536	1.92	1.55888
6	3.94971	7.68	4.21986
8	11.7965	43.008	14.0902
10	38.6066	309.658	55.253

Table 5.7: The theoretical G_l derived from (5.11) and the theoretical moments of the symmetric noise distributions.

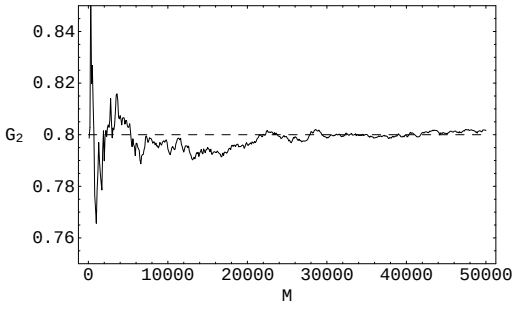
culate G_l for the distributions, as shown in Table 5.8. These values were intended to provide an experimental comparison to the higher moments Gamma test. However, the difference between the theoretical and experimental G_l for the symmetric noise distributions appears to be less important than for the non-symmetric case, so that only the theoretical values for G_l need to be used in these experiments.

G_l	Uniform	Normal	Bimodal
2	0.80155	0.80893	0.79774
4	1.54058	1.97111	1.5511
6	3.96158	8.0365	4.18669
8	11.8248	46.1027	13.9325
10	38.6613	342.198	54.4216

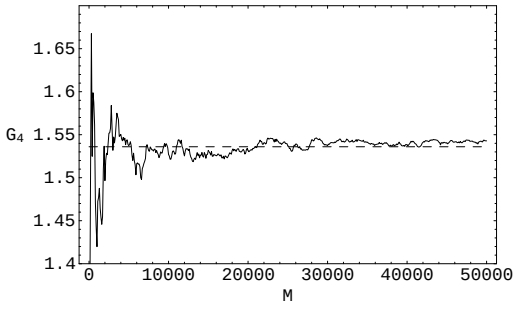
Table 5.8: The G_l derived from (5.11) using the experimental moments of the symmetric noise distributions calculated using the `CentralMoment` routine in *Mathematica* ($M = 50000$).

Experimental results

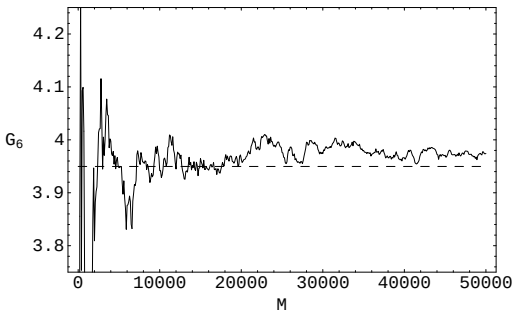
Figures 5.6, 5.7 and 5.8 show the higher moments Gamma test estimate for G_l for increasing sample sizes M up to $M = 50000$ sampled points. The theoretical values for G_l are shown by the dashed lines. The estimates for G_l (for even l) for the distributions will asymptote to the theoretical values as $M \to \infty$, providing the technique described in Section 5.4 is reliable.



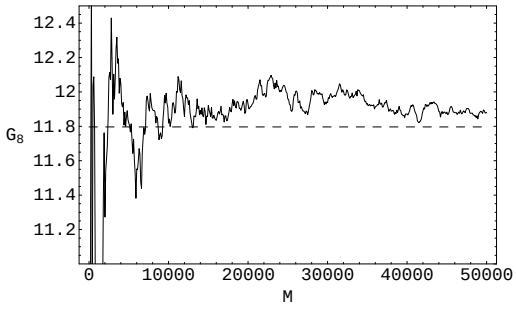
(a) The estimate for G_2 asymptotes to the theoretical value.



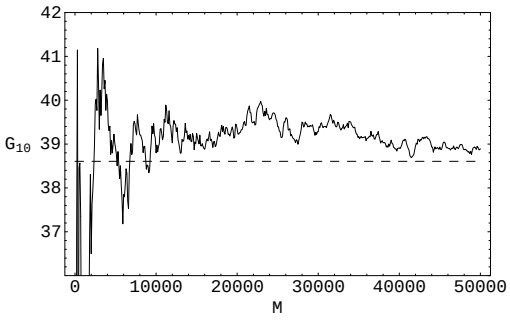
(b) The estimate for G_4 asymptotes to the theoretical value.



(c) The estimate for G_6 asymptotes to the theoretical value.

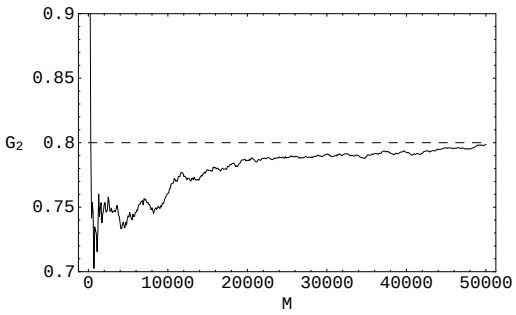


(d) The estimate for G_8 asymptotes to the theoretical value.

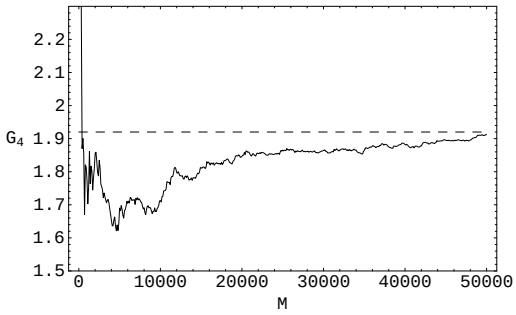


(e) The estimate for G_{10} asymptotes to the theoretical value.

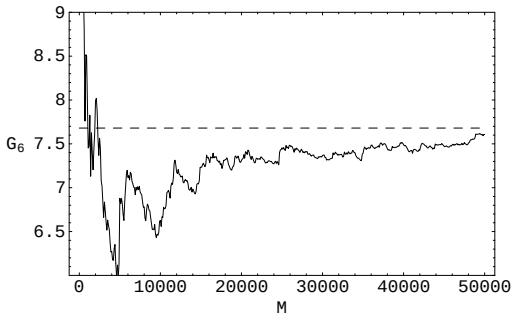
Figure 5.6: Asymptotic nature of G_l for the uniform distribution described in Figure 5.5(a) with $M = 50000$ sampled points. The higher moments Gamma test estimate for G_l is shown relative to the value predicted by (5.11) using the theoretical moments (shown as the dashed line).



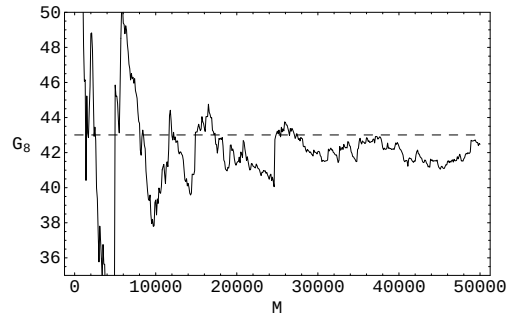
(a) The estimate for G_2 converges to the theoretical value.



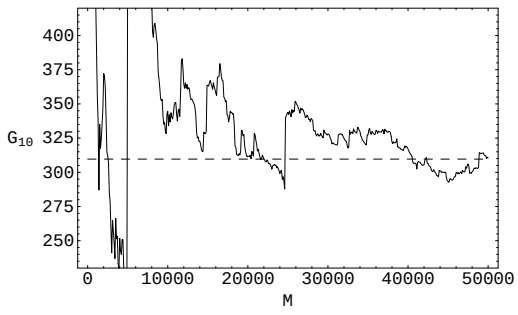
(b) The estimate for G_4 converges to the theoretical value.



(c) The estimate for G_6 converges to the theoretical value.

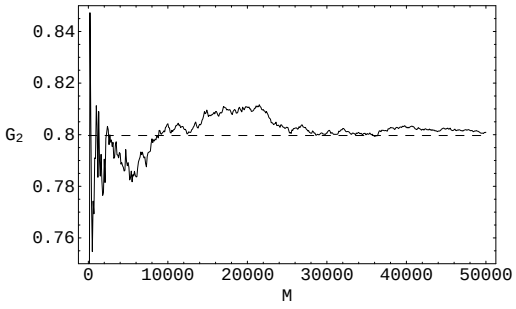


(d) The estimate for G_8 converges to the theoretical value.

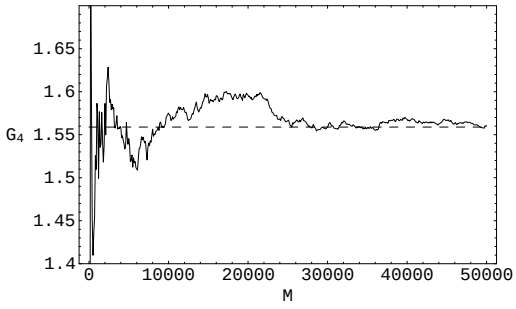


(e) The estimate for G_{10} converges to the theoretical value.

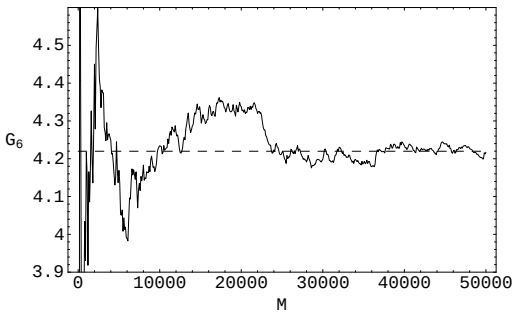
Figure 5.7: Asymptotic nature of G_l for the normal distribution described in Figure 5.5(c) with $M = 50000$ sampled points. The higher moments Gamma test estimate for G_l is shown relative to the value predicted by (5.11) using the theoretical moments (shown as the dashed line).



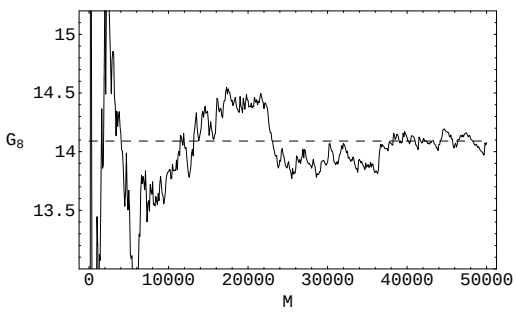
(a) The estimate for G_2 asymptotes to the theoretical value.



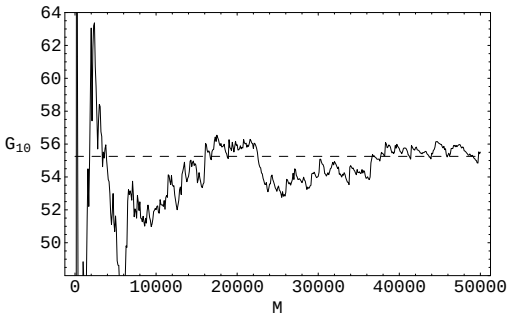
(b) The estimate for G_4 asymptotes to the theoretical value.



(c) The estimate for G_6 asymptotes to the theoretical value.



(d) The estimate for G_8 asymptotes to the theoretical value.



(e) The estimate for G_{10} asymptotes to the theoretical value.

Figure 5.8: Asymptotic nature of G_l for the bimodal distribution described in Figure 5.5(e) with $M = 50000$ sampled points. The higher moments Gamma test estimate for G_l is shown relative to the value predicted by (5.11) using the theoretical moments (shown as the dashed line).

Analysis of results

Figures 5.6, 5.7 and 5.8 empirically verify (5.11) for the three symmetric distributions. The convergence of the higher moments Gamma test estimate of G_l to the theoretical G_l was evident for all three distributions (Table 5.9 provides a numerical comparison). This verification is very encouraging, especially considering that the estimates were made from data containing an underlying smooth function.

The confirmation of (5.11) is more conclusive than for the non-symmetric distributions used to verify (5.9). It has become evident through these experiments that much less data is required to accurately estimate G_l for symmetric noise distributions than for (unbounded) non-symmetric distributions.

Now that we are confident that the results for G_l are accurate, the higher moments can be calculated using (5.11) using the higher moments Gamma test estimates for G_l.

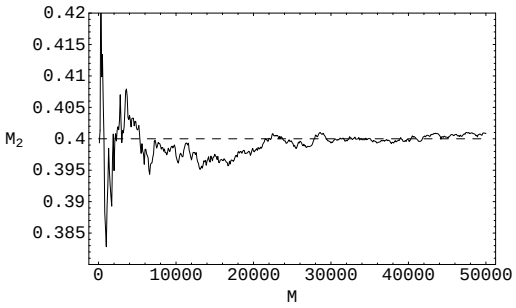
5.4.3 Using G_l to estimate the even moments

The values of G_l calculated for the previous experiments were sufficiently accurate to justify (5.11) and provide confidence that the moments can be calculated with reasonable accuracy. The calculation of the moments from G_l was performed and the results shown in the Figures 5.9, 5.10, and 5.11 for the three noise distributions.

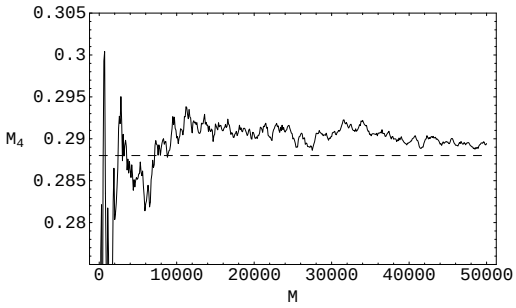
The moments M_l measured using the higher moments Gamma test for each of the distributions asymptoted to the values predicted by the theoretical moments. The asymptote to the theoretical values required approximately 50000 sampled points to provide sufficiently accurate results for large l (see Table 5.10). In one sense this is an expected result since a lot of data is required to accurately define the original probability density functions. This fact aside, in practice the moments determined by the higher moments Gamma test for much less data often provides an acceptable measure.

In these experiments the moments of the noise distributions determined experimentally using *Mathematica* were similar to the theoretical moments making comparison using `CentralMoment` unnecessary (contrary to the experiments using the lognormal distributions for the non-symmetric noise case).

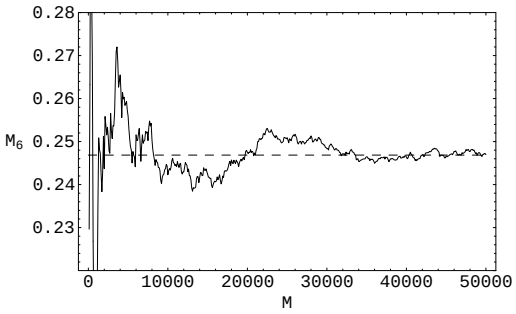
These results show that, for *symmetric* noise distributions, the moments estimated from the higher



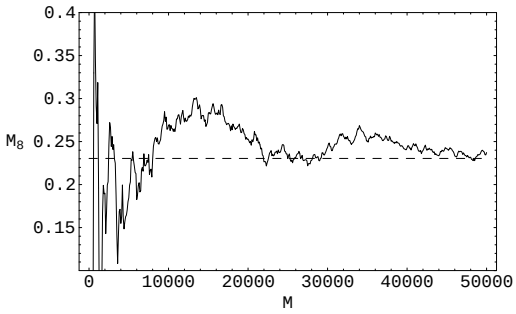
(a) The Gamma test estimated M_2 converges to the true moment value shown by the dashed line.



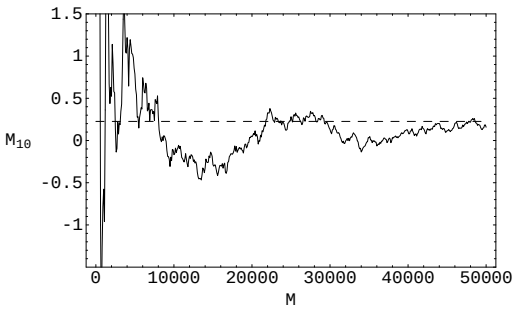
(b) The Gamma test estimated M_4 approximately converges to the true moment value shown by the dashed line.



(c) The Gamma test estimated M_6.

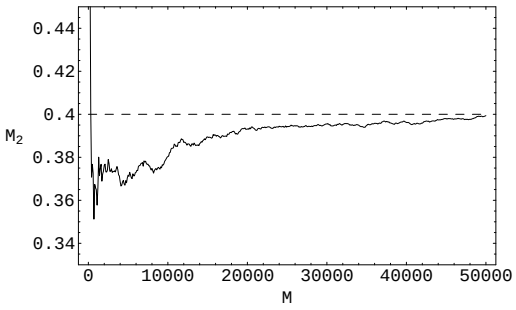


(d) The Gamma test estimated M_8.

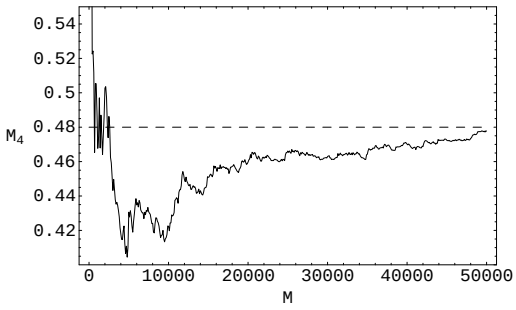


(e) The Gamma test estimated M_{10}.

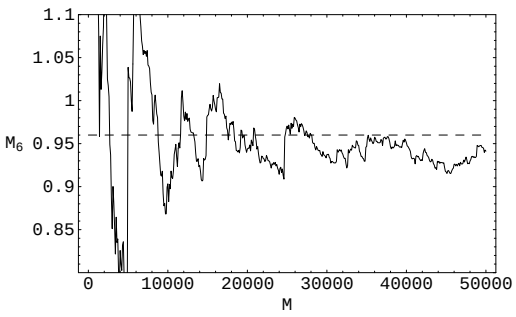
Figure 5.9: Asymptotic nature of moments for a uniform noise distribution, and $M = 50000$ sampled points.



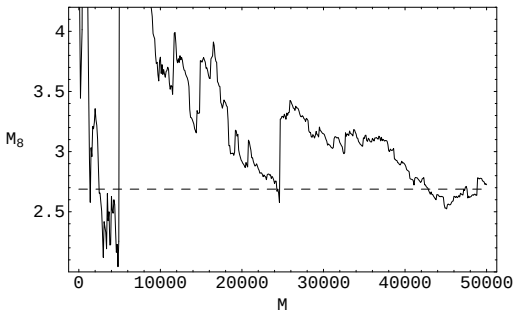
(a) The Gamma test estimated M_2 approximately converges to the true moment value shown by the dashed line.



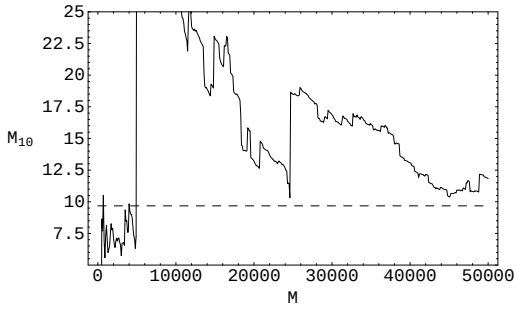
(b) The Gamma test estimated M_4 approximately converges to the true moment value shown by the dashed line.



(c) The Gamma test estimated M_6 converges to the true moment value shown by the dashed line (unexpected).

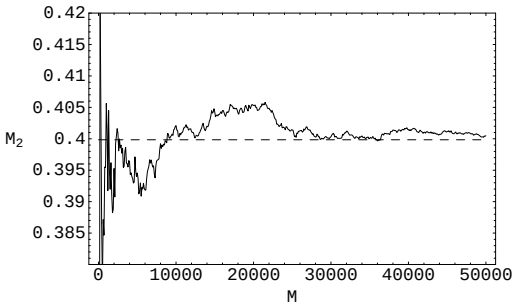


(d) The Gamma test estimated M_8.

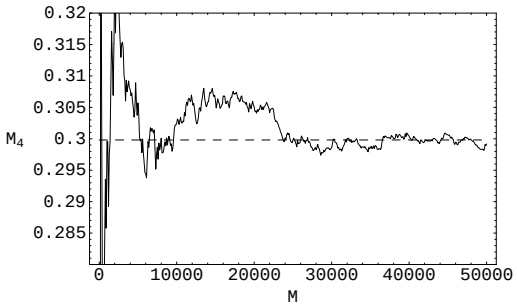


(e) The Gamma test estimated M_{10}.

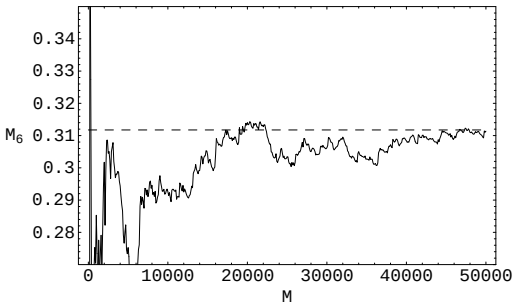
Figure 5.10: Asymptotic nature of moments for a normal noise distribution, and $M = 50000$ sampled points.



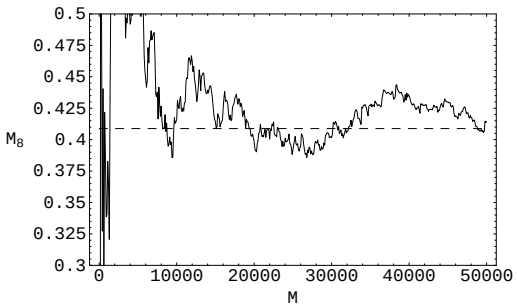
(a) The Gamma test estimated M_2 converges to the true moment value shown by the dashed line.



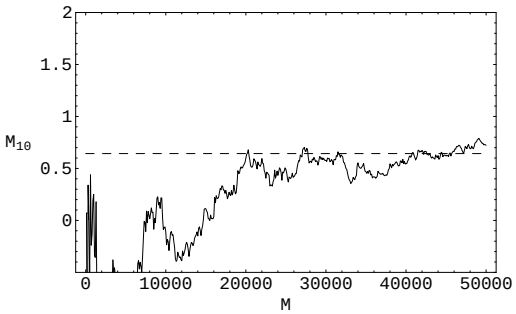
(b) The Gamma test estimated M_4 approximately converges to the true moment value shown by the dashed line.



(c) The Gamma test estimated M_6.



(d) The Gamma test estimated M_8.



(e) The Gamma test estimated M_{10}.

Figure 5.11: Asymptotic nature of moments for a bimodal noise distribution, and $M = 50000$ sampled points.

Uniform distribution				
		M		
G_l	Theoretical	1000	10000	50000
2	0.8	0.765647	0.794516	0.801646
4	1.536	1.41982	1.52707	1.54282
6	3.94971	3.51855	3.94181	3.97423
8	11.7965	10.1008	11.8186	11.8815
10	38.6066	31.7261	38.8007	38.8928

Normal distribution				
		M		
G_l	Theoretical	1000	10000	50000
2	0.8	0.72985	0.760654	0.798732
4	1.92	1.77652	1.71271	1.91237
6	7.68	7.96515	6.62619	7.60584
8	43.008	52.1634	39.2998	42.4734
10	309.658	426.373	344.51	310.442

Bimodal distribution				
		M		
G_l	Theoretical	1000	10000	50000
2	0.79968	0.811248	0.804177	0.800964
4	1.55888	1.58641	1.5751	1.56066
6	4.21986	4.2165	4.23444	4.21713
8	14.0902	13.4455	13.8269	14.0772
10	55.253	48.8346	52.0118	55.509

Table 5.9: A comparison of the theoretical and experimental G_l of the symmetric noise distributions.

moments Gamma test do indeed asymptote to their theoretical values, which now allows us to discuss the reconstruction of the noise distributions.

Uniform distribution				
		M		
M_l	Theoretical	1000	10000	50000
2	0.4	0.382823	0.397258	0.400823
4	0.288	0.27025	0.290093	0.289432
6	0.246857	0.207407	0.242279	0.246949
8	0.2304	0.270981	0.268991	0.23724
10	0.226211	−0.576024	−0.167783	0.157596

Normal distribution				
		M		
M_l	Theoretical	1000	10000	50000
2	0.4	0.364925	0.380327	0.399366
4	0.48	0.488747	0.422411	0.477704
6	0.96	1.30724	0.903284	0.941242
8	2.688	4.36392	3.78563	2.72449
10	9.6768	7.35309	27.338	11.8347

Bimodal distribution				
		M		
M_l	Theoretical	1000	10000	50000
2	0.399844	0.405624	0.402089	0.400482
4	0.299813	0.299612	0.302522	0.299174
6	0.311748	0.285302	0.292607	0.311358
8	0.408804	0.340567	0.415944	0.414532
10	0.642966	0.250113	−0.109427	0.722397

Table 5.10: A comparison of the theoretical and experimental M_l of the symmetric noise distributions.

5.5 Reconstructing a symmetric noise distribution

It is well known that the moments do not in general completely determine the distribution even when moments of all orders exist [Kendall and Stuart, 1963]. Only under certain conditions will a

set of moments determine a distribution uniquely, but fortunately these conditions are satisfied for all distributions likely to be encountered in practice. For all practical purposes, a knowledge of the moments, when they exist, is equivalent to a knowledge of the distribution function. In particular we expect that if two distributions have a certain number of moments in common they will bear some resemblance to one another. Further discussion of this topic can be found in, for example, [Kendall and Stuart, 1963] Chapter 3.

For our purposes we should like to perform an approximate reconstruction of the probability density function (pdf) of the noise distribution based on estimates for the first few even moments and the hypothesis that this distribution is symmetric, so that the odd moments are zero. In practice we are unlikely to be interested in M_l for $l > 10$.

In Algorithm 7 we represent the unknown pdf as a normal distribution $\alpha(x)$ multiplied by a polynomial $p(x) = \sum a_{2i}x^{2i}$ of degree n having only even powers. This representation is a compromise based on the assumption that in most practical situations the noise distribution is likely to be approximately normal.

The mean and standard deviation (σ) for the normal distribution are assumed respectively to be zero and the square root of the variance ($\sigma = \sqrt{M_2}$) as estimated by the Gamma test. The unknown coefficients of the polynomial are determined from the moments by symbolically integrating

$$\int_{-R\sigma}^{R\sigma} x^j \alpha(x) p(x) dx \quad (j = 2, 4, 6, \ldots, n) \tag{5.12}$$

where R is chosen to be 5 as a suitable compromise determined empirically to give reasonable results. The symbolic expressions determined by (5.12) are then equated to the estimated moments and the resulting linear equations can be solved to give the polynomial coefficients.

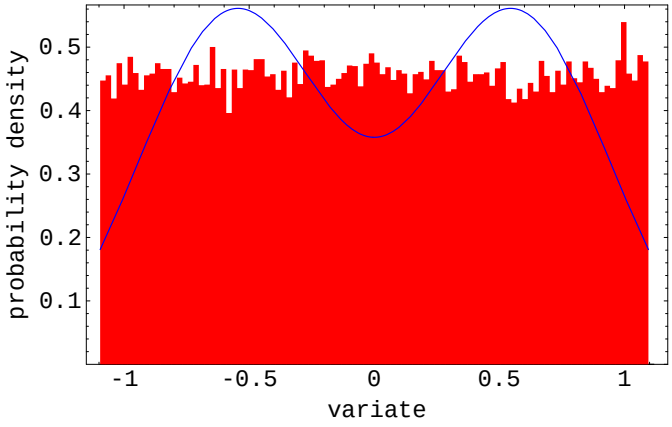
A *Mathematica* implementation of this noise reconstruction method is given in Algorithm 7, where `iLimit` corresponds to R in (5.12), and `xmin` and `xmax` define the interval within which the reconstructed noise distribution should be plotted. The plot of the noise distribution is then returned by the algorithm.

5.5.1 Experimental reconstruction

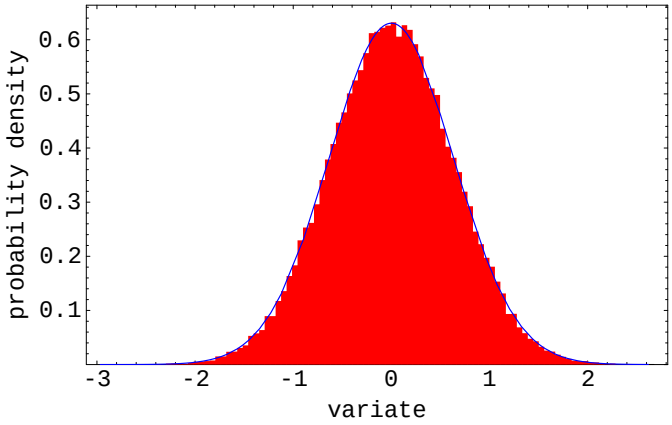
The moments computed for the three symmetric noise distributions ($M = 50000$) are shown in Figure 5.12. The noise distributions were reconstructed using Algorithm 7 and overlaid on to the original noise distributions shown in Figure 5.5 to provide a comparison between the reconstruction and the original distribution.

```
PlotEPNoise[allMoments_, numMoments_, iLimit_, {xmin_, xmax_}] :=
Module[
  { moments, n, stdDev, alpha, f, a, L0, L, j, eqns, coeffs },

  moments = Abs[Take[allMoments, numMoments]];
  n = Length[moments]/2;
  stdDev = Sqrt[moments[[2]]];

  alpha[x_] :=
    1 / (stdDev * Sqrt[2*Pi]) * Exp[-(x^2) / (2 * stdDev^2)];
  f[x_, n_] := alpha[x] * Sum[a[2*i] * x^(2*i), {i, 0, n}];

  L0 = Integrate[f[x, n], {x, -iLimit*stdDev, iLimit*stdDev}];
  L = Table[0, {i, 1, 2*n}];

  For[j = 2, j <= 2*n, j += 2,
    L[[j]] = Integrate[x^j * f[x, n], {x, -iLimit*stdDev,
    iLimit*stdDev}];
  ];

  eqns = Join[{L0 == 1}, Table[L[[2*i]] == moments[[2*i]], {i, 1,
  n}]];

  coeffs = Table[a[2*i], {i, 0, n}];
  Evaluate[coeffs] = coeffs /.  Solve[eqns, coeffs][[1]];

  Return[Plot[f[x, n], {x, xmin, xmax}]];
];
```

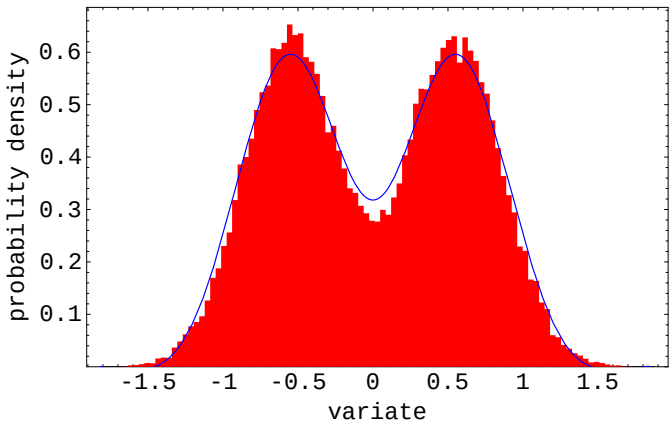
Algorithm 7: The EP (even polynomial) algorithm for reconstructing a distribution.



(a) The uniform noise distribution.



(b) The normal noise distribution.



(c) The bimodal noise distribution.

Figure 5.12: The reconstructed symmetric noise distributions using $M_2, \ldots, M_{10}$ (shown by the line) overlaid on to the original noise distribution.

The distributions were reconstructed using all of the computed even moments up to M_{10}. The bounded uniform distribution is only approximated by the EP algorithm, which considering that the distribution is being represented by an even polynomial times a normal distribution, is satisfactory. An improvement might be possible through careful consideration of the integration range of (5.12) but, due to the nature of the reconstruction method, this will only have a minor effect. The normal and bimodal distributions, as would be expected given the nature of the reconstruction, were well represented using this reconstruction method.

The EP algorithm has been shown to work well for the normal and bimodal noise distributions. However, if uniform noise is suspected then this may not be the most appropriate algorithm for reconstructing the noise distribution.

5.6 Perfect models

In building non-linear models we have previously accepted the idea that a model whose error variance is close to the noise variance estimated by the Gamma test is probably the best that can be accomplished. For example in training a neural network by error backpropagation we attempt to reduce the mean-squared error to the Gamma statistic.

However, the ability (albeit rather restricted) to estimate the higher order moments suggests a refinement of the idea of what constitutes a good model. We offer the following:

- An *ideal* model would have an error distribution *identical* to the noise distribution.

Of course, this is a useful criterion only if there is a method of accurately reconstructing the noise distribution. We have seen that approximate reconstruction of a noise distribution is possible provided it is symmetric. An interesting question now arises: if the goal of conventional backpropagation were modified so as to reduce some measure of the difference between the estimated higher moments and the actual error moments of the neural network, then would the resulting model have a superior performance?

5.7 Conclusions

We have shown that a plausible generalisation of the original Gamma test allows the higher even moments to be measured directly from the data (providing we assume that the noise distribution

is symmetric). This discovery has enabled a noise distribution to be reconstructed and this leads us to propose further work beyond the scope of this thesis such as: (1) what is the best method to reconstruct the noise? and (2) how can modelling techniques be improved using the reconstructed noise distribution?

Perhaps the most challenging development is to turn the Gamma test algorithm into one that can measure both the even and odd moments. It is not obvious how to proceed at present with the Gamma test in its current form, but perhaps this will become clearer as the research continues.

CHAPTER 6

winGamma

winGamma was specified to provide an easy-to-use Windows application for use by the consultants of Universal Solutions[1], as a research and teaching tool for academics and students, and as a commercial product available to data modellers requiring state-of-the-art analysis and modelling techniques.

This is the first commercial Gamma test software and took approximately three man-years to develop (including research, prototyping, design, implementation and testing). Some of the algorithms contained within *winGamma* had been previously implemented for use within the research group at Cardiff University using a rudimentary scripting interface. This project developed those algorithms further and added a suitable graphical interface to enable practitioners from any field to use the software tool with the minimum of effort.

A lot of the design, prototyping and programming was undertaken using some of the techniques now described as *extreme programming* (XP) and explained in [Beck, 2000] and [Hunt and Thomas, 2000]. These practices ensured that the software was delivered quickly, with appropriate quality and to specification. However, there are some limitations to the software that have come to light now that this first software tool has been developed, particularly involving scalability, maintainability and platform independence. These aspects are considered towards the end

[1] Universal Solutions are an advertising consultancy operating within Universal McCann to help media planners develop more effective marketing campaigns. Universal McCann is owned by the McCann-Erickson global advertising agency that sponsored this work. *winGamma* has been supplied to McCann-Erickson under a commercial agreement that prevents *winGamma* being released to their direct competitors.

of this chapter and new judgements are presented to show how to proceed with the design of the next version of the software. Most of the analysis and modelling techniques mentioned within this thesis have been included in *winGamma*, as have some techniques not previously discussed, all of which have implications for future development and research work. The techniques not implemented so far are the higher moments Gamma test and false nearest neighbours.

6.1 Requirements definition and specification

The *winGamma* specification focuses on code re-use[2] and the development of a quality user interface to provide an easy-to-use system. This broad starting point enabled the requirements to become more formally defined.

Requirements definition

> The software must provide a means of non-linear data analysis using the Gamma test and provide a method for non-linear modelling. The software must be of commercial quality and compatible with Microsoft Windows.

The requirements definition was developed into a flexible *requirements specification*, with the final scope of the project limited by the ability to develop the algorithms and provide a suitable *graphical user interface* (GUI) using the tools available and within the time-limits imposed by the research.

Requirements specification

1. Develop an environment for running Gamma test experiments and model building using neural networks and local-linear regression.
2. Provide a commercial quality interface compatible with Microsoft Windows 95/98/NT (and later ME/2000).
3. Re-use existing algorithms where possible.
4. Provide a means to access data files generated by other tools.
5. Maintain the functionality of the existing software.
6. Interface with *Mathematica* for additional analysis.

[2] A script based Gamma test and modelling system was developed for UNIX in 1997 by Steve Margetts.

6.2 Development environment

After the specification of *winGamma* had been agreed the development language and environment were chosen. The numerical content of the analysis and modelling routines required a language that offered favourable execution speed from compiled code. Speed was chosen as the criterion since there is little to choose between most high-level languages regarding accuracy. Some languages such as Java did not, at least at the time, provide suitable performance. In the end C++ was chosen as the development language for several reasons:

1. C++ offered the best combination of execution speed, object-oriented features and memory management.
2. The original Gamma test and modelling code was written in C++.
3. Suitable Microsoft Windows-based development tools exist for C++ including Microsoft Visual C++ and Borland C++ Builder.
4. Third party tools, for example for charting and reporting, are widely available with C++ and Microsoft Windows interfaces.

The two potential software development tools Microsoft Visual C++ and Borland C++ Builder were compared and Table 6.1 provides a summary of the main features[3]. Both development environments have a similar *integrated development environment* (IDE) and each support *Microsoft foundation classes* (MFC), essential for programming native Microsoft Windows applications. However, visual GUI building is absent from Microsoft Visual C++ and this is a major drawback for *rapid application development* (RAD). Conveniently Borland has also abstracted Microsoft's rather esoteric MFC to provide a simpler programming interface through their *visual component library* (VCL).

Borland C++ Builder was chosen as the development environment because it was more comprehensive, especially regarding the reasons highlighted over GUI development. The GUI building capability and the VCL abstraction of the MFC library combine to produce a well-designed RAD tool.

There were additional considerations that exerted a secondary influence on the choice of development environment that did not arise during the initial specification. For example, to create a distributed application, perhaps to take advantage of a 'pool of machines', an interface such as

[3] Source: Borland (only the relevant points are shown).

Feature	Borland C++ Builder 4	Microsoft Visual C++ 6
Real visual development	Yes	No, use Visual Basic
Class framework	MFC, VCL, OWL	MFC
CORBA ORB	VisiBroker	No
Integrated CORBA development	Yes	No
Distributed object interfaces	COM and CORBA	COM
High performance RDMS	Native and ODBC	ODBC
Visual database design	Yes	Yes
Internet protocol components/classes	Yes	Yes

Table 6.1: A comparison of Borland C++ Builder and Microsoft Visual C++ showing the key development features.

CORBA could be used. This has an advantage over the alternative COM proposed by Microsoft because it is platform independent and would allow computation to be spread over different machine architectures and operating systems (Borland support CORBA, Microsoft do not). This would allow the *winGamma* interface to run on a Microsoft Windows platform and communicate to a distributed network of (heterogenous) computers. Database connectivity using *open database connectivity* (ODBC) would enable integration with industry standard databases such as Oracle and Objectivity to extract and store data and results.

6.3 Design and prototyping

The design and prototyping describes the structure of the existing code and ideas that have developed from prototype versions of *winGamma* including various application and interface ideas. Finally we introduce the latest design of *winGamma*. This discussion has been purposely produced to sufficiently describe the software without entering into a comprehensive software engineering review. This is for two reasons: (1) the thesis is intended to describe non-linear data analysis and modelling and psuedo-code has been provided throughout, and (2) a comprehensive review is not necessary understand how the program was constructed using standard (and familiar) Microsoft Windows interface components.

Many of the design elements were influenced by the existing code which is discussed in Section 6.3.1. Section 6.3.2 discusses some of the issues that arose during the prototyping stage of the

project, which in turn helps to justify many of the final design features described in Section D.

6.3.1 Existing code: the Gamma test and modelling components

The existing objects provide data manipulation, a nearest neighbour algorithm, the Gamma test and neural network and local-linear regression models. The objects were written in C++ (with some dependence on UNIX libraries).

The pre-existing components are represented in Figure 6.1 using the *Object Modelling Technique* (OMT) described in [Rumbaugh et al., 1991]. Through code re-use the fundamental design of these components remains unchanged.

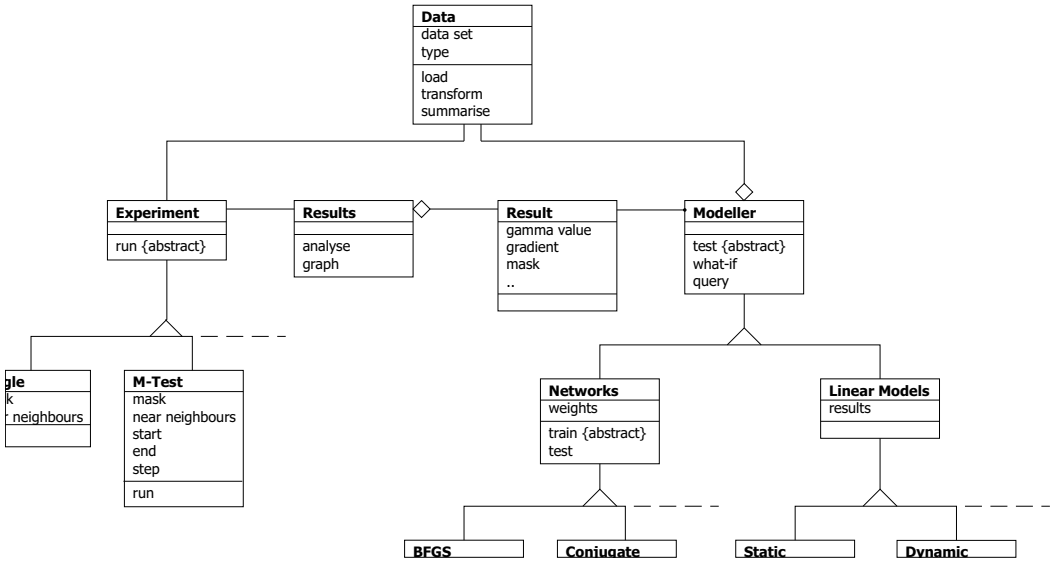


Figure 6.1: OMT representation of the *winGamma* components

These existing objects form a script-based UNIX program where the user is able to run a Gamma test and build or test a model. Results visualisation and additional analysis have to be done using external software.

6.3.2 Prototyping *winGamma*

Several prototype systems were developed to understand how an integrated environment could work. Although the prototypes no longer exist, they proved useful at the time to highlight several important requirements:

1. Users wanted to manipulate a data set within the application for repeated analysis, for example by changing the lags used in a time-series analysis.
2. It was desirable to perform standard results visualisation within the application.
3. An export facility should be provided to allow results to be analysed in standard applications (spreadsheets, databases and *Mathematica*).

Software modification of the existing objects (Section 6.3.1) was required to ensure that the interaction and feedback originally coded for a UNIX console was redirected to the GUI and the dependency on UNIX libraries was removed.

Project-based operation

Analysis of a data set can involve many experiments. The existing script-based UNIX software allowed single experiments to be run, where it was up to the user to manage the results as they were produced. The initial prototype versions of *winGamma* followed this implementation as a way of rapidly developing an example application. However, it soon became clear that it would be beneficial to manage repeated experimentation on a data set within the context of a *project environment*. This lead us to propose that experimentation should be recorded for a given data set and that the resultant project should be able to be saved to disk and loaded back into *winGamma* at a later time.

Data set management

One of the requirements that came from developing a prototype system included the facility to view multiple data sets. Data sets fall into three categories: (1) the *analysis data set* used for analysis with the Gamma test and to construct a model, (2) a *test data set* used to test constructed models, and (3) a *prediction data set* used to query a constructed model (where the outputs are unknown).

A number of components are available within Borland C++ Builder for retrieving and displaying data, ranging from reading the data into a text box (the simplest) to a full database option (the most complex). The database solution introduced additional complexity to the software, requiring either a database engine to be shipped with *winGamma* or by providing an ODBC interface to the user's existing databases.

For this first version of *winGamma* it was more important to provide the core functionality of data analysis and modelling so the database development was dropped. Ultimately it was felt that a user's data would most likely be available in spreadsheet, database or plain text format. Although this approach requires the user to export their data to an ASCII format file, it was felt that it provided the greatest flexibility for the least development effort leaving, as a priority, the main task of developing the analysis and modelling routines. A method to retrieve the text files and display the data in a spreadsheet-like grid was chosen. This solution introduced an additional problem: the VCL data grid available to perform this function can only handle modest size data sets before the component becomes unmanageable (during the prototyping stage we discovered that the component used a disproportionate amount of memory to store the data). To overcome this problem it was decided to split the data sets into *pages* that contained a manageable subset of the data.

Results visualisation

The standard numerical results from the Gamma test and modelling routines can easily be displayed in any of the standard Windows visual components: text boxes, text grids, etc. The main priority was to include the standard visualisation tools, including the Gamma scatter plot and model testing and what-if query charts. Additional real-time visualisation of the performance of the GA and model training algorithms was also desirable. The charting components available in Borland C++ Builder were compared during the protoyping stage. The *TeeChart* component was chosen because the component includes the ability to display scatter plots, histograms, 3-dimensional charts, line charts, real-time charts and custom drawing routines. In addition to these visualisation techniques, methods to zoom/pan, print/preview, export and customise the charts were also provided.

Exporting results

Although the analysis and modelling should be performed within *winGamma*, many users also use spreadsheet, database, presentation and mathematical software. All components that contain data, charts and results will be implemented with an export facility. It was decided to implement charts to be exported as either a chart image (in a number of standard image formats) or as raw data for re-generation in another application. These routines should provide enough flexibility to analyse the results and re-create the charts in most readily available software.

Threads

Perhaps the greatest architectural addition to arise from the prototyping stage involved *threading* the application. A GUI by default provides a single application thread that all computation is performed on. Using this basic application model the GUI cannot be interacted with while other operations are running, where the subtlest manifestation is that standard window operations, such as window repainting, have to wait until the computation on the main application thread is completed. To overcome this, the computationally intensive operations were placed on threads. This directly benefits the user in a number of ways: (1) interaction is maintained with the GUI, (2) threads can be easily paused/resumed to enable the user to regain the CPU for other tasks without aborting the current operation, and (3) threads can be terminated before the operation running on it is complete and (depending on the implementation) can return any results produced so far.

Threads offer additional computational complexity that is not present in single threaded applications. Suitable management of threads is required to avoid deadlock (where a thread continuously waits for another thread to finish) or to avoid overuse of the available computing resources.

It was apparent from the prototype systems that two threads were required for a single user, single processor machine. The application thread was left to automatically manage the GUI components, and another thread was created to perform the analysis or modelling computation as each request was issued. This required a thread management component to prevent multiple requests for analysis or modelling to maximise the efficiency of the computation.

Notification that a thread has terminated (either through the natural completion of a task or through user termination) is required by an application to ensure that the GUI reflects the current operational state of the program and to enable other operations to be executed that were previously denied.

6.4 Implementation

We shall focus this discussion on the issues surrounding the implementation of *winGamma* rather than describe in detail every aspect of the implementation. For reference, a general discussion of the operation of *winGamma* from a user interface perspective is provided in Appendix D.

6.4.1 Implementation lessons

The initial structure of *winGamma* manifested itself during the prototyping stage (Section 6.3.2) where many of the user requirements became apparent. This stage of development was also particularly important for *winGamma* because we were unsure what functionality could be developed in the time using the skills available to us. Although we had experience of developing algorithms and applications using rudimentary console based interfaces, this was the first project implemented to handle a fully interactive GUI environment and the first implemented using a professional quality IDE (integrated development environment). We quickly learnt that choosing the best IDE allowed us to easily produce applications with a complicated GUI.

From the prototyping stage we learnt that it was relatively straightforward to implement standard graphical interfaces for getting data into and out of a program. The real problem with GUI development stems from the interactiveness provided for the user and that requires careful management of user-triggered *events*. This is necessary, for example, to avoid simultaneous access to non-sharable resources or to provide considerate feedback to the user regarding the state of the application (for example by removing the ability to paste when the clipboard is empty).

Handling normal user interaction with the GUI had to be maintained even when *winGamma* was performing processor intensive tasks and this was achieved using threads. Although this produces a slight computational overhead in terms of thread management and multi-tasking, it is not onerous and is more desirable than removing all user interaction with the application whilst computation is being performed.

The implementation and adaption of the algorithms used some of the methods of extreme programming (XP). There were times during the development cycle, especially for writing particulary complicated code, when it was necassary to design, program and debug the code with another person present. This enabled the full impact of the design to be discussed before implementation and helped to avoid mistakes during coding. This was particularly useful when the existing Gamma test components had to be adapted to provide feedback to the user via the GUI. The adaptation of the code was performed in tandem with the person who wrote the original code to ensure that functionality and performance were not unduly affected.

Another aspect of XP that was put into practice helped us to *evolve* the software. This meant that instead of having a rigid design for *winGamma* at the start of the project, we allowed the software to be constructed in small stages. This allowed us to continuously review the application and add new features when it became apparent that they would benefit the user. For example, we had

no plans to write code to perform what-if scenarios on the models (see Appendix D.4.6). After implementing a query function (see Appendix D.4.5), we realised that it would be a small step to allow the user to analyse a scenario (the scenario algorithm just calls the query routine many times).

The main benefits of this approach to programming *winGamma* was that the project was flexible enough to adapt as we discovered new techniques and developed our skills. Now that *winGamma* has been produced and has been used for some time by a number of users, we can determine what features would be useful and how the application should be developed in the future.

6.5 Conclusions

winGamma is a fully functional application that meets the original design specification. In the evolution of the project it has become apparent that many other features could be useful and these are introduced in Section 6.6.

The application implemented all of the features available in the UNIX version of the code and added additional experiment and model types.

The *winGamma* interface evolved through various prototypes and in its current form has proven very usable.

There are some structural improvements that could be made to the code as complications have been introduced as *winGamma* has evolved. The original code has been repeatedly adapted to ensure that interface elements and the threading code worked correctly. This creep effect has been very hard to eliminate because many of the features now in *winGamma* were not envisioned during the preliminary design stage.

6.6 Future development

The future development of *winGamma* considers features that could be added to assist with data analysis and model building, or technological approaches that could be adopted to improve the performance of the software.

The intention of this section is to introduce ideas that can be used during a discussion of how to

develop future versions of *winGamma*.

6.6.1 Features

The proposed features describe concepts and ideas that could be implemented to improve the analysis and modelling capabilities of *winGamma*.

Stand-alone models

winGamma produces models that can be exported to *Mathematica*. Although useful for research, *Mathematica* is not widely used making this feature redundant for many potential users. There are a number of solutions to this problem that can be easily implemented to widen the market. The first is to produce models in more readily available formats such as Microsoft Excel. Another solution would be to produce a code segment that describes the model, which could then be included in the users own applications (a number of languages would have to be supported). Alternatively the models could also be exported in ASCII format and the users could implement their own method to use the model.

Higher order Gamma test

The code to perform the higher order Gamma test, described in Chapter 5, was not included in *winGamma*. Instead, it was developed independently with enhanced features for managing interface communication to provide machine independence since it does not rely on a particular GUI specification, such as VCL or MFC.

Heuristic data scaling

An heuristic scaling algorithm was developed to automatically determine the relative importance of inputs using the Gamma test and scale them accordingly. The technique and methodology are still being researched, but could potentially provide important insights into the nature of non-linear relationships.

Input selection and masks

The current implementation uses a mask to specify which inputs are used and which are not. The feature selection routines use these masks in a rather rudimentary fashion.

There is an obvious natural extension to the specification of a mask that would allow the user to apply more 'structure' to aid the selection of inputs. For example, this structure could take the form of a request to select n inputs from m (where $n \leq m$) or to exclude a number of inputs from an analysis.

Scripting language

A scripting language could be used to extend the functionality of *winGamma* without requiring developer intervention. The intention would be to provide users with the ability to implement new experiment types or to automate repetitive tasks.

Various scripting languages are available that could possibly be included in *winGamma*. Python and Perl are obvious choices, but during the development of such a system it may be that a custom language would be required to define full functionality.

Automatic analysis and modelling

winGamma requires a suitably qualified analyst to analyse and model data. This dependency on skilled users could be reduced by providing routines to automatically analyse and model data. These routines could then be extended to provide data mining facilities for the analysis of databases and data warehouses.

Visualisation

The graphical routines in *winGamma* were implemented using the *TeeChart* component. Although this component provides some very flexible features, it is virtually impossible to extend the usability of *TeeChart* beyond that envisioned by the authors. This is a limiting factor of the component when additional visualisation routines are required. The *TeeChart* component is also bonded to the Microsoft Windows environments, making the transition to a platform independent version of *winGamma* more problematic. At present some or all of these limitations would be overcome by

choosing new or additional charting components.

A technique that could enhance model analysis is volumetric data visualisation. For example, the current what-if routines vary a single input to measure the response of the output from the model. This could be extended to see how two input dimensions affect the output visualised using a 3-dimensional surface plot. Taken one stage further, a 4-dimensional volumetric visualisation could examine what effect three input variables had on an output. This form of visualisation, for example, would enable the analyst to look for multiplicative or synergetic effects between variables (i.e. where two or three inputs combined produce an effect different from the sum of their individual effects).

6.6.2 Technology

This section introduces some ideas arising from developments in computing technology.

Platform independence

The goal of platform independent software is primarily to extend the market audience. Importantly this goal provides a secondary benefit over multiple platform implementations in that only one variant of the source code is maintained and developed.

At the time when *winGamma* was being specified it was decided to use C++ because that produced the fastest code. Additionally Borland C++ Builder was chosen to provide the best way of coupling the algorithms to an interface. At the time Java was considered because it offered platform independence, but at the time was not a serious contender regarding execution speed, requiring a virtual machine to interpret the Java code. However, there have been massive performance improvements since the project was specified and Java would now provide a more credible development language.

However, using a single development language is not necessarily the only solution. The speed of C++ code still makes it superior to Java, and is available for most computer architectures. Providing development follows standard conventions, such as those specified by ANSI, then a single implementation of the code can be compiled for multiple architectures. In this way, the next version of *winGamma* may benefit from the speed of the C++ algorithms and a platform independent Java interface.

Distributed/parallel algorithm implementation

One of the limitations of the current implementation of *winGamma* is that it cannot handle very large data sets in reasonable time due to the required computation. For example, it is very difficult to perform a full feature selection search on data sets with over 20 inputs. If this sort of analysis is desirable, then the computation needs to be spread over several processors, whether they reside on local machine or distributed across a network. However the need for this sort of implementation reduces with the development of more advanced heuristic algorithms.

The are two obvious methods of parallelising the Gamma test. The first is to parallelise the algorithms, dividing the computation across processors. This form of algorithmic parallelisation is really only feasible on a dedicated parallel machine since a distributed environment would be hampered by the low communication rate. The second form of parallelisation arises when feature selection routines are decomposed. This form of analysis would allocate a series of Gamma test experiments to individual processors. The results (which are of a limited size) then pose no serious communication overhead.

Database support

Linking *winGamma* to a database could provide two important improvements. The first is that a database could be used to store all of the experimental results. The second is that a lot of commercial data is stored within a database and this could be accessed directly rather than requiring the user to export data from the database prior to analysis.

ODBC (open database connectivity) and JDBC (Java database connectivity) provide industry standard interfaces for database access. If Java or C++ is used as the development language then these tools could be employed accordingly.

The volume of data held on corporate databases can be very large, so if *winGamma* is turned into a data mining application accessing large corporate databases then the analysis algorithms would have to be speeded up using some of the ideas discussed in the previous section.

Internet access

Internet access to *winGamma* provides a number of attractive solutions. Firstly this approach provides an attractive way to handle software licensing because the user would be required to

connect to a license server that would validate the authenticity of the user/software. Secondly it is feasible to implement the interface to *winGamma* as a Java or ActiveX application that could be accessed using a web browser. This could provide a method of analysing data via an internet connection, perhaps providing access to a powerful remote machine that would otherwise not be available to the user.

However, there are still technological limitations to such an implementation where the bandwidth would limit how much communication could be made to the remote machine. In the case of running *winGamma* remotely, only modestly sized data files could be transferred in reasonable time. At present the software licensing application remains the attractive use of the internet and has been implemented for the *winGamma* software running in the Department of Computer Science. If the application is copied and taken away from the university then it becomes inactive because either *winGamma* cannot connect to the server to verify its authenticity or the server recognises that the software is off site.

CHAPTER 7

Flood Prediction System

This chapter investigates how the Gamma test and the non-linear modelling techniques described in the previous chapters can be applied to the problem of river modelling. Our main interest in developing these techniques is to produce a flood forecasting system. However, these same techniques could provide forecasts to aid the management of hydroelectric generation schemes where it is desirable to control the flow of water.

7.1 Introduction

The impact of flooding can be devastating causing loss of life and the destruction of crops, homes and industry. In the aftermath of a flood the health of the population is put at risk by water-borne diseases (cholera for example) that can quickly spread once the water supplies are contaminated and sewerage systems are destroyed [Smith, 1996].

In light of the devastating effects caused by flooding and the costly disaster recovery process, a flood prediction system would aid disaster impact reduction schemes by providing a reliable warning of any flood threat. It will enable action to be taken before a flood strikes which is more cost-effective both economically and socially [Anderson, 1991].

An accurate and timely flood warning should allow time for remedial action to be taken, ensuring that the affected population, emergency services, transportation routes and flood defence systems

are all managed effectively. A suitable response to a flood warning should ensure that the impact is reduced or eliminated, allowing the affected area to more rapidly recover.

This project highlights many of the problems that need to be overcome in order to build a reliable flood prediction system, and provides a basis for future research using the techniques described in this thesis.

It is worth noting that the same basic techniques might be applied to the management of hydro-electric reservoirs. An example might be the operation of a reservoir with an uncontrolled inflow but which has the means of regulating the outflow. If advance information regarding the inflow is available then the reservoir can be operated, perhaps by some rule based system, so as to optimise electricity production or minimise downstream flood damage. This problem is discussed in [Valença and Ludermir, 2000], which provides a comparison between higher-order neural networks (see [Kumoluyi et al., 1995] for an excellent introduction to higher-order neural networks) and *PARMA* (Periodic Auto-Regressive Moving Averages) models. The model was implemented to forecast weekly average inflow on a step-ahead basis and was tested on four hydroelectric plants located in different river basins in Brazil.

7.2 Statement of the problem

The aim of the proposed flood prediction system is to demonstrate that reliable predictions can be made of the water flowing in a river using observations of the river, its tributaries and the environment.

Perhaps the most crucial factor in the design of a prediction system is the forecasting time. The effectiveness of a prediction can be maximised if a suitably large forecasting time can be achieved with sufficient accuracy. It is our long-term aim to design a flood prediction system that maximises the forecast time and, if constructed, will therefore require rapid data collection and fast model operation to provide a rapid prediction. It is also essential that the system operates automatically and continuously to provide constant monitoring and warning.

This project is intended to provide a prototype for a real flood forecasting system, and as such ignores the physical problems of how to measure and record data. For example, the engineering requirements of a real-time system require remotely distributed monitoring equipment that can relay their measurements to a base station for analysis.

7.2.1 Modelling approaches

To construct an effective flood warning system requires the dynamics of the river to be modelled. There are several favoured approaches to modelling river systems and predicting flooding, many providing analytical solutions. For example the Poisson-Parseval solution to the wave equation provides a semi-analytical solution of the Saint-Venant equation in two dimensions [Stephens and Stapleton, 1983]. Additionally, the use of finite differences or finite element methods allows the flow to be modelled if the dimensions of the river system are known.

Continuous forecasting of water levels can be made by indirect or direct methods. The indirect method initially involves prediction of runoff either through a rainfall-runoff model or by routing the flow observed at an upstream gauge to the desired location downstream. The predicted runoff is later converted to a water level by use of a rating curve. The rainfall-runoff models, for example [Kitadinis and Bras, 1980a], [Kitadinis and Bras, 1980b], [Georgakakos, 1986a] and [Georgakakos, 1986b], require knowledge of the underlying hydrology and establishment of many rain gauges together with a good telemetry system. Routing techniques are more useful when the travel time is longer and the downstream flow is low or controlled. For direct prediction of water levels, statistical correlation techniques have been employed [Mutreja et al., 1987]. Unlike indirect methods these techniques are not dependent on the rating curves.

These techniques can provide accurate results in a reasonably wide range of circumstances. However, the empirical models require careful construction for each particular catchment area and so it would be hard to envisage a general purpose adaptive system which could proceed from such a basis.

[Thirumalaiah and Deo, 1988] use neural networks as a pattern recognition technique for river stage forecasting in the Godavari Basin (India) and their results show that adaptive modelling for level prediction is quite practical.

The geographical area covered by a single system is a specification parameter which should be considered carefully. Too large an area will require too many monitoring units and hence too many inputs for the model. We envisage that a model will be competent to deal with up to 50 input variables, although further experimentation on real data is required to confirm this expectation. A modular system could then be constructed from networks of individual models where single modules are combined, and predictions from one model act as inputs to downstream models. This would provide an extendable modular architecture covering a much larger area and under some circumstances allow greater prediction times.

Our approach uses a data-derived model created from a series of observations made throughout the river system. We shall determine which variables are relevant (for example flow rate, river level, rainfall) to provide a suitable long term prediction for the water level of the river.

One advantage of data-derived modelling techniques is that the underlying fluid flow equations do not need to be defined (they could never be defined exactly anyway). Instead, the relationships inherent within the data are used to determine the behaviour of the river flow.

The placement of sensors at discrete points in the river environment leaves the difficult problem of how to interpolate between sensor points. This is essential if flood prediction is to be made along the whole length of a river. The finite-difference and finite-element techniques used within the semi-analytical models allow flows to be modelled at points between any two consecutive sensors, potentially allowing flood prediction to be made at all points along the river. However, this is a computationally expensive process and we are not proposing to provide interpolation between sensors at this time. It is therefore essential for our models that sensors are positioned at the most critical points along the river.

7.2.2 Data sources

There are two main sources of data: (1) measurements made of the river and its tributaries, and (2) measurements taken of the environment. Measurements taken of the environment should increase the prediction time, even though, for example, it is difficult to accurately predict rainfall.

The following is a list of variables that could be measured to produce a flood prediction system:

1. Measurements taken directly from the river:
   (a) Flow rate.
   (b) Water level.
   (c) Quantity of water extracted.
2. Measurements made of the environment:
   (a) Rainfall - from rain-gauges or radar.
   (b) Cloud cover - from satellite.
   (c) Temperature.
   (d) Ground saturation.

(e) Tidal behaviour.

7.3 River simulator

A simulated river will be used to develop the analysis and modelling techniques for the flood prediction system. This artificial data provides a deterministic system for analysis. It is hoped that the techniques developed for the river simulator can be subsequently applied directly to a real river system.

In the following sections we describe the design of the simulator and methods for data analysis.

7.3.1 Simulator design

The simulator is an abstraction of a real river system. The main river is fed by its tributaries, where each tributary may be fed by other tributaries, and each tributary is affected by the local rainfall.

Figure 7.1 illustrates our simulated representation of a river. Each node represents an intersection in the river system where measurements of flow and rainfall are made. Each channel has an associated length and this determines how long the water measured at one node takes to flow to the successive node in the river. Although not implemented, flood conditions could be allowed to develop by specifying the maximum quantity of water that can flow down a channel. The river would then flood whenever the channel limit is exceeded.

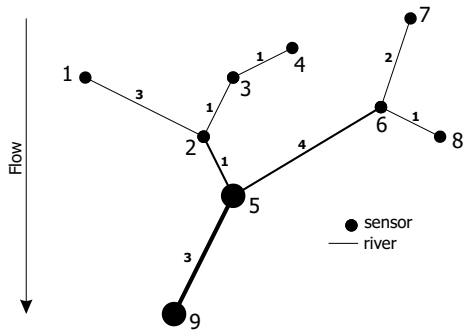


Figure 7.1: The river simulator used to prototype the analytical techniques used in the flood prediction system.

The behaviour of the simulated river is deterministic. Each node on the river is affected by the water flowing in from its tributaries and the water generated by the local rainfall. The source nodes, which are labelled 1, 4, 7 and 8 in Figure 7.1, provide inputs to the river system which are

created solely from rainfall.

Rainfall is simulated using a *bounded* random number generator to add a quantity of water to the river. The flow through the channel is then calculated by adding the local rainfall to the water flowing into the channel, as illustrated in Figure 7.2.

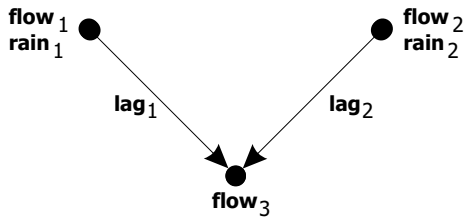


Figure 7.2: The flow dynamics of the simulator are modelled at each node. The water flowing at a node is a sum of the water and rain at the upstream node, lagged in time according to the distance between the nodes.

The flow through the river $flow_i$ can be defined at each node i using the local rainfall $rain_i$ and the flow $flow_j$ measured at the adjacent upstream nodes j

$$flow_i(t) = \sum_{j=1}^{n}(flow_j(t - lag_j) + rain_j(t - lag_j)) \tag{7.1}$$

where t indicates the time of measurement, lag is the time taken for the river to flow from the upstream node j to the downstream node i, and n is the number of adjacent upstream nodes.

For example, at nodes 1 and 2 in Figure 7.1, the flows are

$$\begin{aligned} flow_1(t) &= 0 \\ flow_2(t) &= 0 \\ flow_3(t) &= flow_1(t - lag_1) + rain_1(t - lag_1) + \\ &\quad flow_2(t - lag_2) + rain_2(t - lag_2) \end{aligned} \tag{7.2}$$

where, in this example, $flow_1 = 0$ and $flow_2 = 0$ because they are source nodes that have no water flowing into them from other rivers. In effect, the water flowing out of the channel will be determined by the rainfall only. Rain and flow are delayed using $(t - lag_j)$ according to how long it takes these two sources of water to flow to the receiving node.

We can see from (7.1) that in this simple model the flow is a linear combination of the flow and rainfall at an upstream node. A sufficiently large data set containing rainfall and river flow data should provide enough information to model the river.

7.4 Determining the relevant time lags

It is crucial to capture the simulated river dynamics in order to build a successful model. We assume that the rainfall and river levels are measured at each node in the simulator, which leaves (if we look at the data only) the lag time between the nodes as the unknown factor. Once the lag time has been discovered we can accordingly delay the measurements to produce an optimal model.

We chose to approach the task of estimating the lags in two ways: (1) using the Gamma test, and (2) using a lag correlation routine. The lag correlation routine *Delta correlation* is described in Section 7.4.1.

The results of the analysis of the simulator are described in Section 7.4.3 with a comparison between the two proposed techniques.

7.4.1 Delta correlation

One method to find the optimal embedding delay is to vary the delay on a set of input time series to see how they correlate to the target (output) time series. We have called this technique the Delta correlation.

From vector algebra we have

$$\cos\alpha = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} \tag{7.3}$$

Given an input time series $\mathbf{a}$ and an output time series $\mathbf{b}$, the correlation is determined by $\cos\alpha$, where α is the angle between the two vectors. Strongly correlated time series will have $\cos\alpha \to 1$ for a positive correlation and $\cos\alpha \to -1$ for a negative correlation.

In the context of river flows, the correlations will be positive since the expectation is that as an upstream river rises (or falls) then the downstream river will correspondingly rise (or fall) at a later point in time. The time taken for the flow to reach the downstream point is the delay time (or lag).

For optimal performance the time series are transformed to differences to accentuate the changes in river conditions (all of the experiments in this thesis that use the Delta correlation algorithm take differences). This gives better performance than using the direct river observations.

The Delta correlation algorithm is shown in Algorithm 8.

```
pre-conditions:
input_time_series contains one or more time series
output_time_series contains one time series

pre-processing (optional):
convert input and output time series to difference time series

{main algorithm}
vector lag_correlations
for lag = 1 to maximum_lag do
  lag_correlations[lag] = Correlation(input_time_series, output_time_series, lag)
end for
{end of main algorithm}


Correlation(input_time_series, output_time_series, lag)
vector correlation
M = number of elements in time series
number_items = M − 1 − lag
b = last number_items in output_series
for input = 1 to number of input_series do
  a = first number_items in input_series[input]
  correlation[input] = a · b/ab
end for
return (correlation)
```

Algorithm 8: The Delta correlation algorithm.

7.4.2 Gamma test lag correlation

The Gamma test lag correlation is an adaptation of the feature selection techniques described in Chapter 4. An input time series is generated from the upstream river sensor data and converted to a new data using a sensible embedding dimension. The output values of the data set are constructed from the time series measurements made at the downstream node.

The feature selection routine constructs a mask consisting of a single input that, as the feature space search continues, slides back in time through the embedded time series (compared to the static output). Running the Gamma test for each mask should show the minimum Gamma statistic occurring at the correct time lag between sensors.

Conversely the input time series could be constructed from the downstream node data with an output consisting of the upstream data, and moving the input mask forward in time.

7.4.3 Simulator analysis

This analysis is carried out on the river simulator shown in Figure 7.1 using the river flow only. The purpose of not recording the simulated rainfall is to introduce a level of uncertainty into the analysis that reflects the real problem of not being able to monitor all of the processes taking place in the real-world. This will help to test the robustness of the analysis techniques.

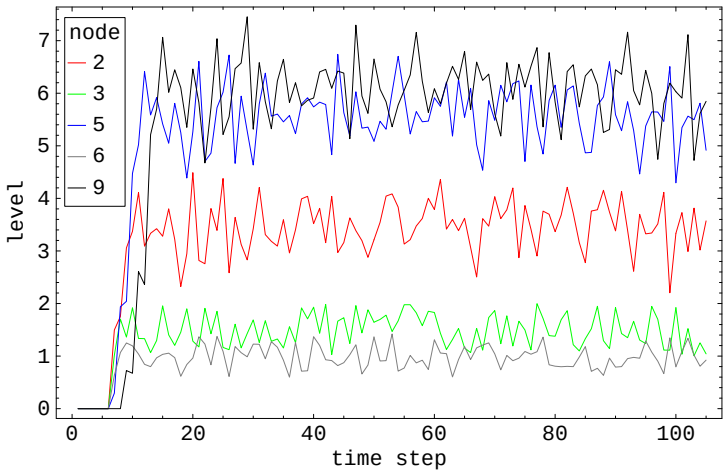


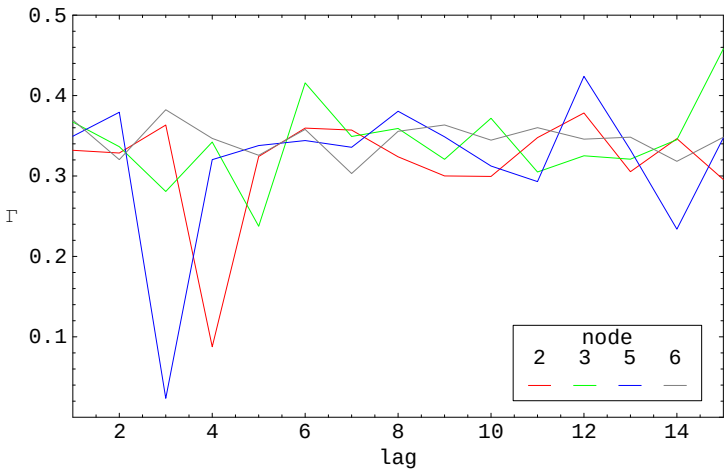
Figure 7.3: In these simulator flow graphs the correlations between node measurements are easily seen.

The flow at each stage was measured and correlated to the flow at the target node (node 9). Both the Gamma test lag correlation (Section 7.4.2) and the Delta correlation algorithm (Section 7.4.1) extracted the correct lag times for each sensor as illustrated in Figures 7.4 and 7.5 respectively. The results are summarised in Table 7.1.

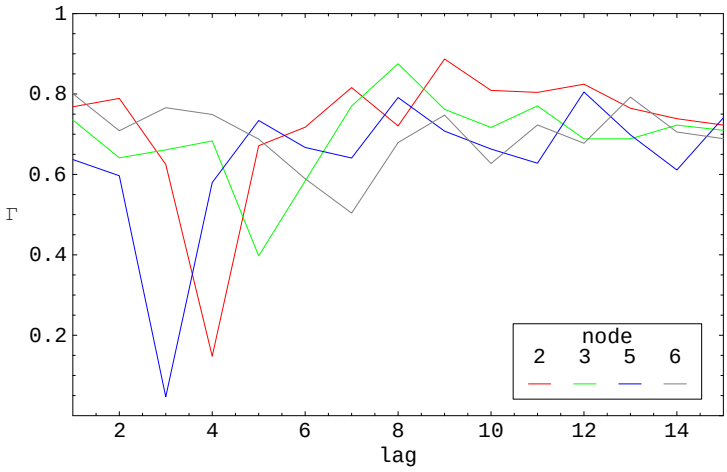
The Gamma test lag correlation of the river simulator data is shown in Figure 7.4. The optimal lags are indicated by a minimum in the reported Γ statistic. This analysis was performed twice, to demonstrate that taking differences of the time series can improve the quality of the results. Although we can see in Figure 7.4(a) that the analysis correctly identifies the lag times, the results of the analysis on the differences time series is much more exaggerated as shown in Figure 7.4(b).

In these figures the lag time from nodes 2, 3, 5 and 6 are reported to be 4, 5, 3 and 7 time steps respectively. We can see from Table 7.1 that the analysis has correctly estimated all of these lag times.

The Delta correlation lag analysis of the river simulator data is shown in Figure 7.5, where the optimal lag times are indicated by a maximum in the correlation. This analysis also correctly



(a) Gamma test lag correlation on the simulated river time series data.



(b) Gamma test lag correlation on the simulated river differenced time series data.

Figure 7.4: The Gamma test lag correlation shows that the lag to node 9 from node 2 is 4 time steps, from node 3 is 5 time steps, from node 5 is 3 time steps, and from node 6 is 7 time steps. Figure 7.4(a) shows the analysis performed on the time series river data and Figure 7.4(b) shows the analysis performed on the differenced time series river data.

identified the lag times from nodes 2, 3, 5 and 6 as being 4, 5, 3 and 7 respectively.

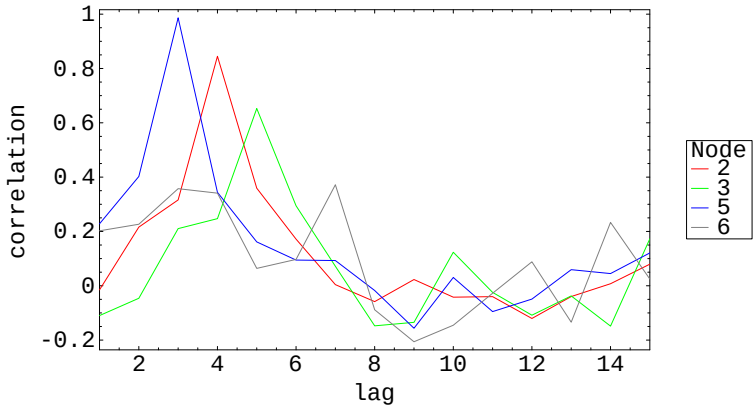


Figure 7.5: The correlation lag analysis shows that the lag to node 9 from node 2 is 4 time steps, from node 3 is 5 time steps, from node 5 is 3 time steps, and from node 6 is 7 time steps.

	Node							
measurement	1	2	3	4	5	6	7	8
actual lag	7	4	5	6	3	7	9	8
Gamma test lag	-	4	5	-	3	7	-	-
Gamma test lag (differences)	-	4	5	-	3	7	-	-
Delta correlation	-	4	5	-	3	7	-	-

Table 7.1: The lag times calculated for the simulated river system show that both the Gamma test lag correlation and the Delta correlation analysis can correctly identify the time delays between the upstream nodes (nodes 1-8) and the point of prediction at node 9. These lags correspond exactly to the distances between the nodes shown in Figure 7.1.

7.4.4 Simulator design limitations

The physical environment contains a level of complexity not represented in the river simulator. For example, the complicated dynamics of the geology and its effect on the level of ground saturation, or the effects of the weather on the amount of water lost to evaporation are not considered.

Rivers carry more water faster when full and this dynamical effect on the lag time is not represented in the simulator. When the volume of water increases the flow time between nodes should decrease accordingly.

These limitations do not in themselves prohibit the development of a flood prediction system.

However, a more realistic simulator would allow for more comprehensive testing of the analysis and modelling techniques, even when real data is in short supply.

7.5 The Thames area: a real river system

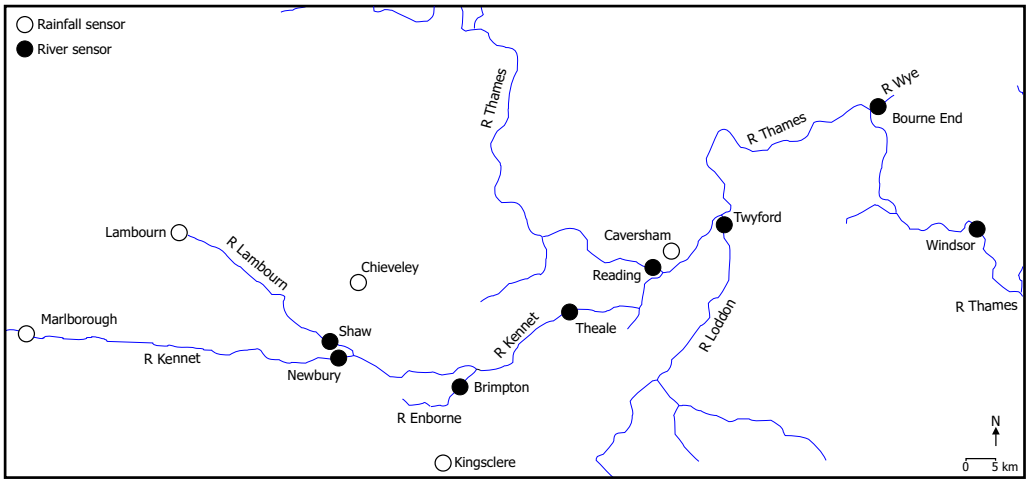
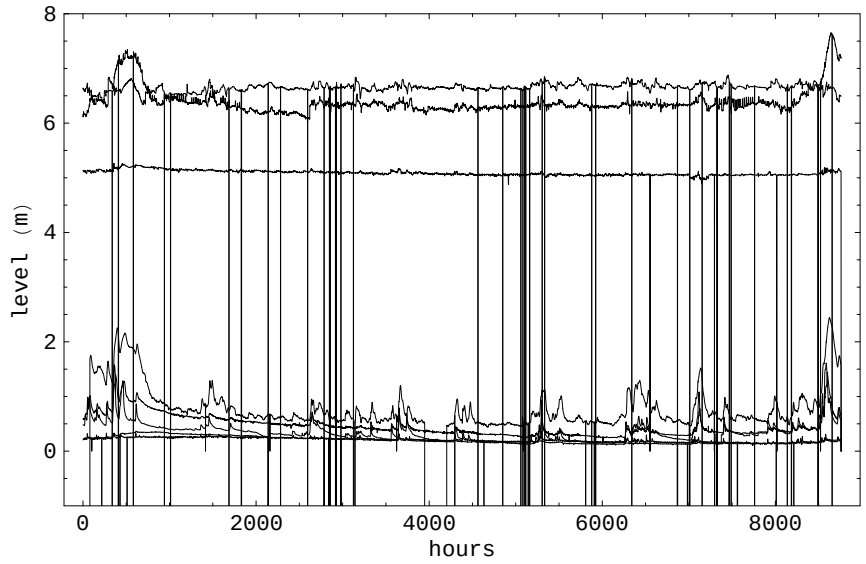


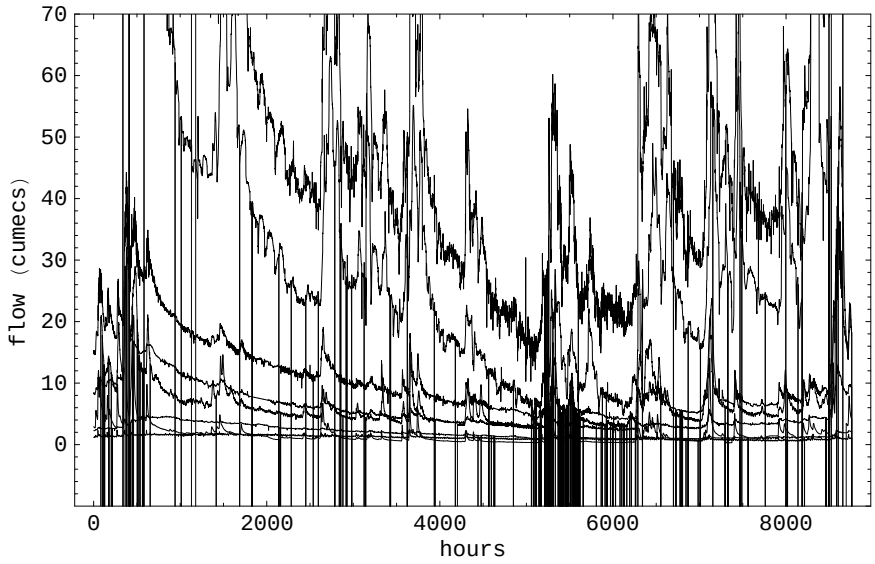
Figure 7.6: Thames region study area.

The river system used in this analysis was the Thames and Kennet river basin between Newbury and Windsor. The data, provided by the UK Environment Agency, consists of flow and level readings taken hourly over one calendar year (10am, 1 January 1999 to 9am, 1 January 2000). The flow rate and level were given in the standard measurements of cumecs (cubic metres per second) for flow, and metres for level. Hourly rainfall readings from five nearby sensor sites were also provided. The river basin is shown in Figure 7.6 with the river and rainfall measurement sites marked.

The river data is subject to the measurement errors often present in real-world applications and, unlike the clean simulated data, will introduce additional analytical complexity. Figure 7.7 immediately demonstrates how frequently the sensors have malfunctioned during the study period for the river level and flow measurements. In some cases these malfunctions can continue for significant periods. The cause of these malfunctions is unknown, but could be due to sensor failure or periods of routine maintenance.



(a) The raw river level data.



(b) The raw river flow rate data.

Figure 7.7: The raw river level and flow rate data. Faulty sensor readings are indicated on the graphs by the plunging vertical lines.

7.6 Data cleaning routines

It was apparent that, before any analysis or modelling could be attempted, the data would have to be cleaned to eliminate or reduce the effect of faulty sensor readings. One factor that influenced the type of data cleaning routine employed was the requirement that the final system should operate in real-time.

For the purposes of a purely theoretical reconstruction of a missing data point we could employ a more sophisticated interpolation routine. However, the techniques open to use are limited in a real-time system where the data is being used for forward prediction. Interpolation using future data values would severely limit the forward prediction time. An alternative may be to use a previously constructed model to estimate the missing values from other sensor readings, but since at this stage we do not have a model, it is not an available option.

The routine we employed compared each data point in the file with the previous one and, if it differed by more than a specified threshold, it was assigned the previous value. This routine was designed to work with time series values that gradually change over time. This same routine could not be applied, for example, to rainfall measurements where the values arrive in bursts.

Data cleaning algorithm

A *Mathematica* version of the data cleaning routine is given in Algorithm 9. The algorithm accepts a multiple time series `MSeries` and a threshold `theta`. The time series are scanned and values that are flagged as missing or appear faulty are replaced with the last known reliable value. Setting `showChanges` to true will display the errors to the user.

7.6.1 The river data

Table 7.2 shows the thresholds used in Algorithm 9 for each river time series. The thresholds are data-stream dependent and were determined by manual inspection of the data. A natural extension of this would be to automate the threshold assignment.

The results of the data cleaning process can be seen in Figures 7.8 and 7.9 where a considerable improvement in data quality is achieved. In most cases where there are single missing values this provides a simple and effective approximation. A disadvantage of this technique occurs when a string of missing values are assigned the last valid measured value. If the string is of considerable

```
RTClean[MSeries_, theta_, showChanges_] := Module[
  {numIn, lists, M, i, k, newMSeries},

  numIn = Length[MSeries[[1]]];
  lists = Transpose[MSeries];
  M = Length[lists[[1]]];

  For[i = 1, i <= numIn, i++,
    For[k = 2, k <= M, k++,
      DD = Abs[lists[[i, k]] - lists[[i, k - 1]]];

      If[(DD > theta || lists[[i, k]] == 0),
        If[showChanges == True,
          Print["Possible faulty sensor value in list ", i,
            " at time = ", k];
          Print["Actual value = ", lists[[i, k]],
            " Replaced with = ", lists[[i, k - 1]]];
        ];
        lists[[i, k]] = lists[[i, k - 1]]
      ];
    ];
    If[showChanges == True, Print["End list"]];
  ];

  newMSeries = Transpose[lists];
  Return[newMSeries]
];
```

Algorithm 9: *Mathematica* data cleaning algorithm.

River	Threshold	
	Level	Flow
Wye at Bourne End	0.2	10
Enbourne at Brimpton	2	10
Kennet at Newbury	5	50
Thames at Reading	5	120
Lambourne at Shaw	0.2	10
Kennet at Theale	2	50
Loddon at Twyford	2	50
Thames at Windsor	5	120

Table 7.2: The thresholds used in data-cleaning for flow and level: these are selected separately for each River.

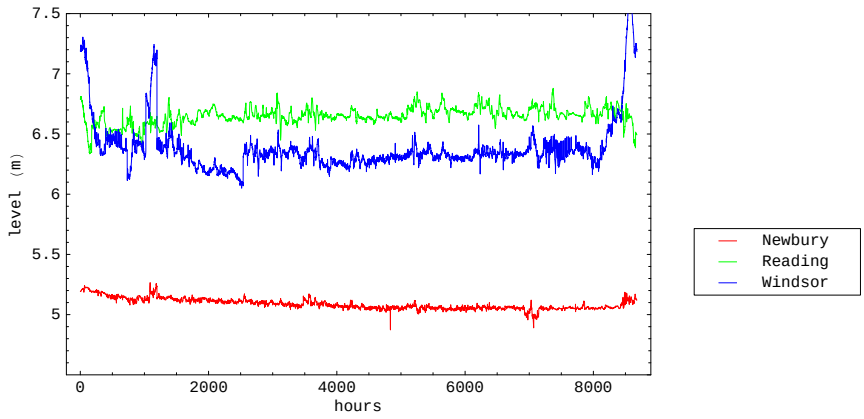
length then this would be liable to produce an increasingly inaccurate approximation.

7.6.2 The rainfall data

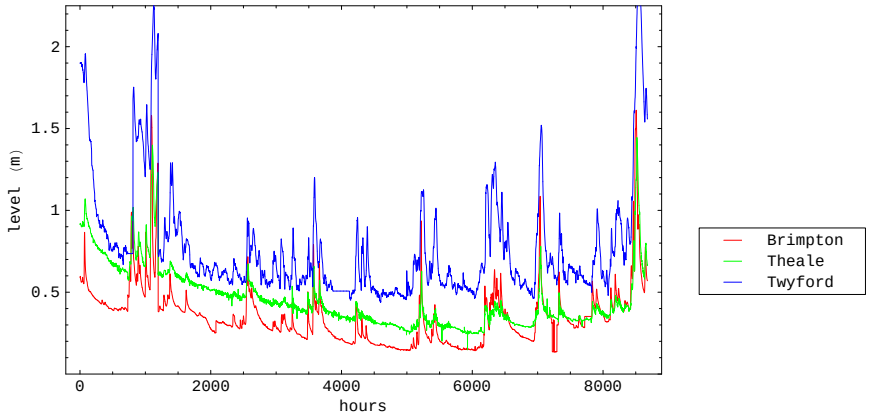
The raw data for the five hourly rainfall sites over the data period of one calendar year is shown in Figure 7.10. The graphs in Figure 7.10 show that the rainfall measurements were relatively stochastic and discontinuous and hence could not be subjected to the data cleaning routine described in Section 7.6. In fact we have assumed that the rainfall was correctly measured since it is much easier to measure rainfall reliably and much harder to estimate errors in the values.

After prolonged periods of heavy rainfall the surface soil of the catchment area becomes saturated and underground reservoirs become full. This results in a change in the runoff dynamics: more water arrives in the tributary and main watercourses, and it arrives more rapidly.

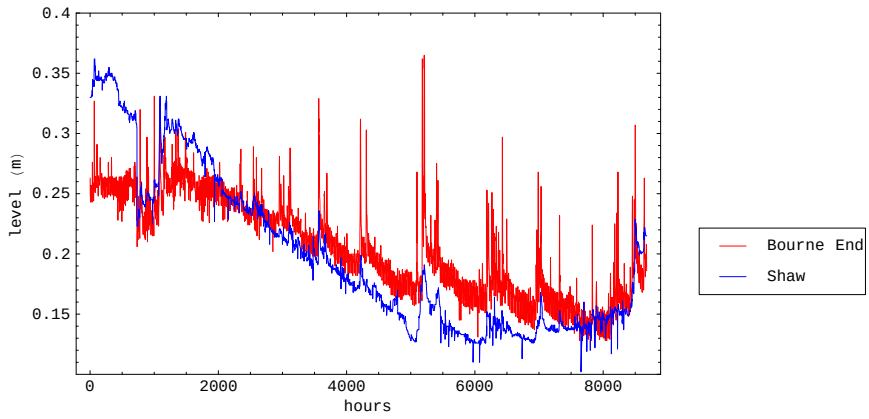
To incorporate these effects into the model and to investigate their relevance, the rainfall measurements were aggregated into moving averages windowed over different time intervals: 24-hours, 7-days and 28-days. These three values were easily calculated and can be included as inputs where the analysis proves them useful. These moving averages are shown in Figure 7.11.



(a) River levels at Newbury, Reading and Windsor.

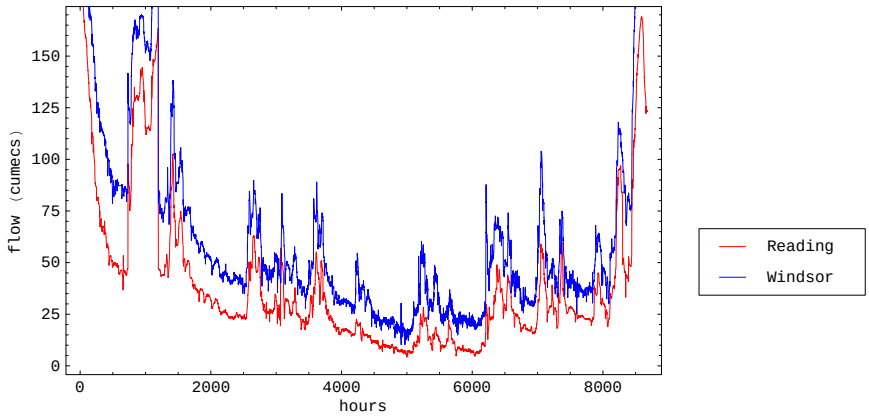


(b) River levels at Brimpton, Theale and Twyford.

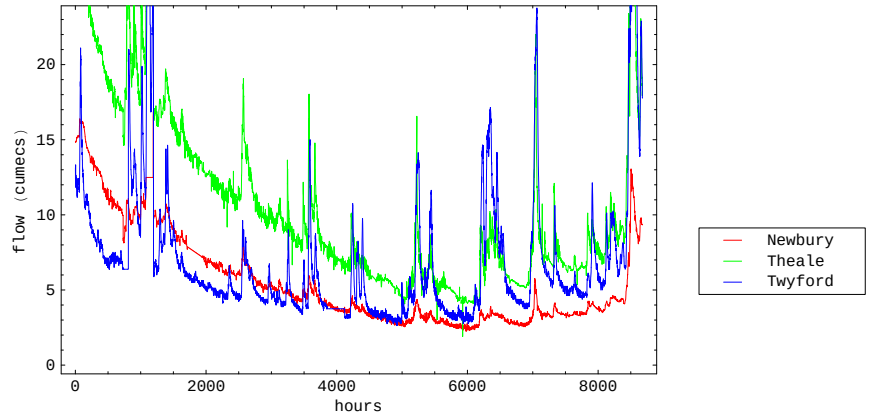


(c) River levels at Bourne End and Shaw.

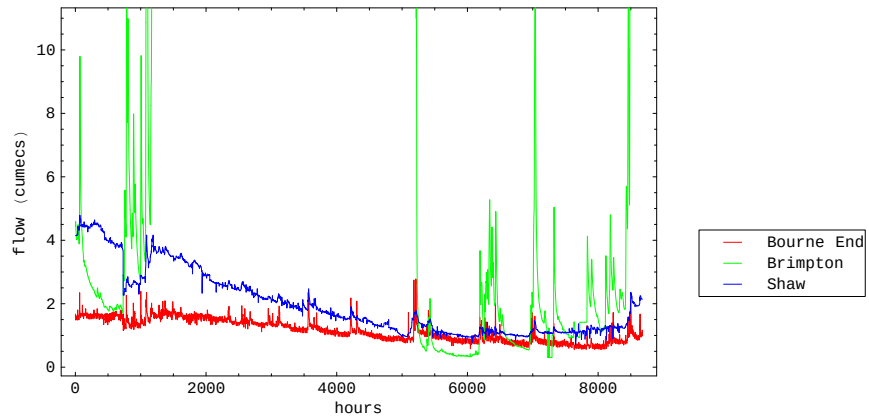
Figure 7.8: The cleaned river level data for the one year analysis period.



(a) River flows at Reading and Windsor.

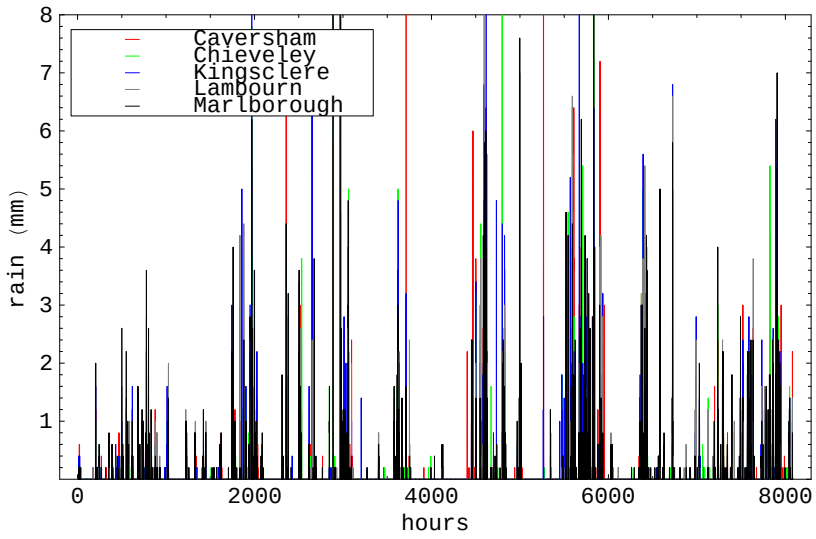


(b) River flows at Newbury, Theale and Twyford.

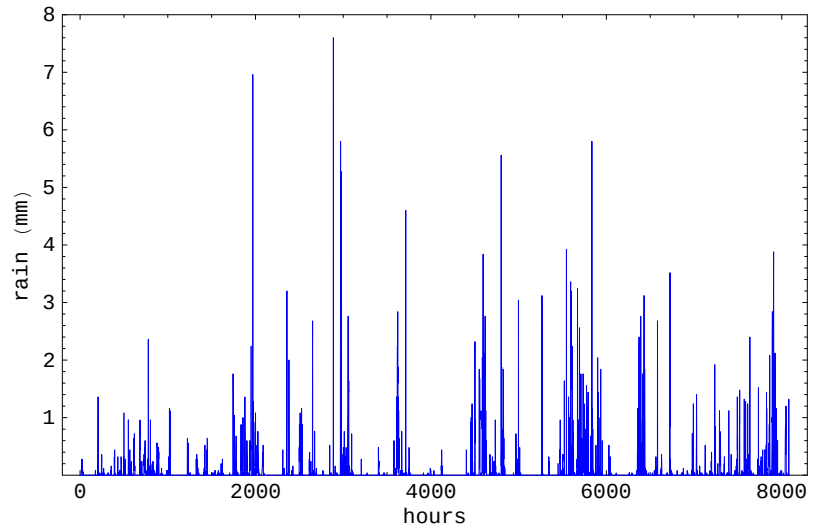


(c) River flows at Bourne End, Brimpton and Shaw.

Figure 7.9: The cleaned river flow data for the one year analysis period.

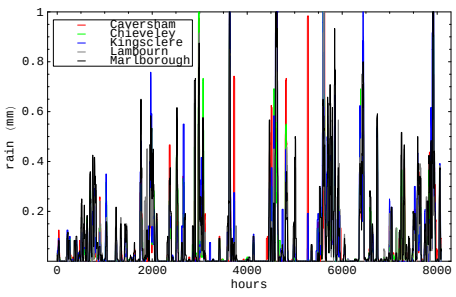


(a) The rainfall measured by each sensor.

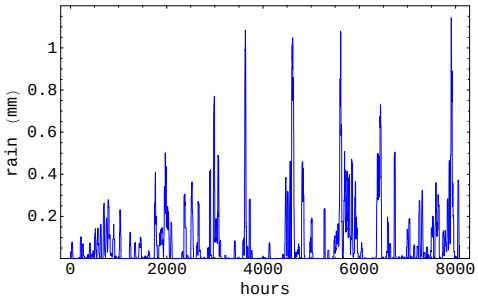


(b) The average rainfall across the region.

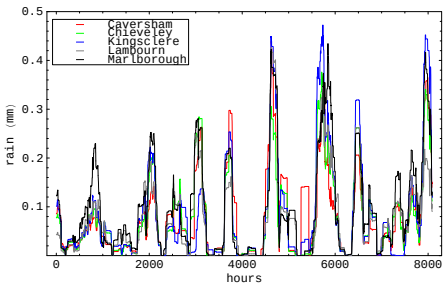
Figure 7.10: The rainfall monitored at the sites marked on the map shown in Figure 7.6.



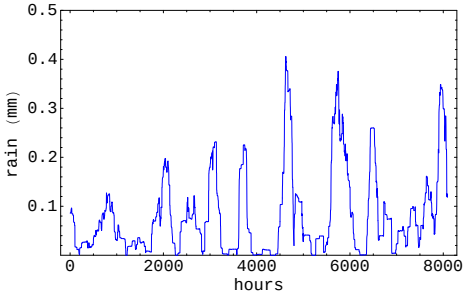
(a) The average rainfall during a 24-hour period at each sensor site.



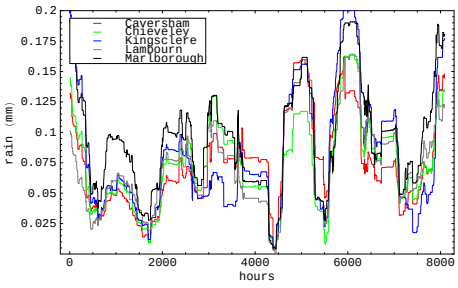
(b) The average rainfall during a 24-hour period across the region.



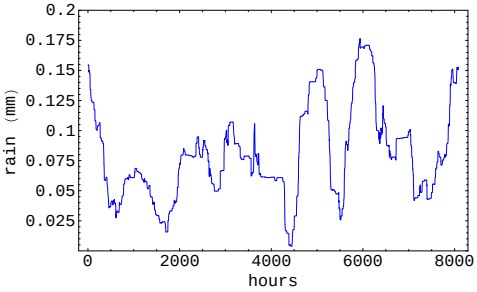
(c) The average rainfall during a 7-day period at each sensor site.



(d) The average rainfall during a 7-day period across the region.



(e) The average rainfall during a 28-day period across the region.



(f) The average rainfall during a 28-day period across the region.

Figure 7.11: The average rainfall calculated at (1) each sensor site and (2) accumulated across the region.

7.6.3 Sensor consistency

Apart from the issue of sensor malfunction there is also an issue regarding the accuracy and/or reliability of sensor readings which, although not obviously in error, can nevertheless produce some puzzling results when correlated with other sensor readings taken at the same site. For example, if we correlate flow and level readings taken at the same site at the same time we might expect to see a simple functional relationship in which increased flow produces a non-linear increase in level. Figure 7.12 shows flow-level correlation plots for the various sensor sites shown in Figure 7.6.

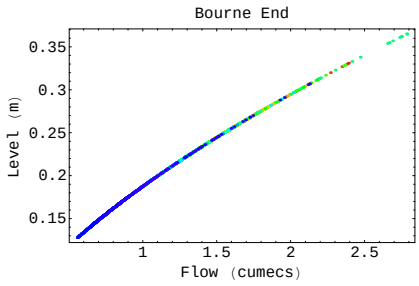
We can see that in some instances (River Wye at Bourne End, River Enbourne at Brimpton, River Lambourn at Shaw, and to some extent River Kennet at Theale) our naive expectations are confirmed. However, in other instances (River Kennet at Newbury, River Thames at Reading, River Loddon at Twyford, and River Thames at Windsor) the flow-level correlations show very unpredictable and widely differing scatter plots capable of varying interpretations. For example, in Figure 7.12(d) (River Thames at Reading) we might suspect progressive sensor *drift*. Indeed closer inspection of the data (the point colour changes progressively through the spectrum from red to blue over the calendar year) shows that the different 'lines' visible on the scatter plot do occur at different times. An alternative explanation might be that the river was dredged periodically and this fact was reflected in an alteration of the flow-level relationship. However, other examples, such as the River Kennet at Newbury or the River Thames at Windsor are less easy to interpret.

The inspection and cleaning of the raw data in conjunction with the interpretation of the flow-level correlation plots indicates that there is a serious issue of data accuracy and consistency to be addressed. Regardless of these data issues, it should be possible to produce moderately accurate flow/level predictions using this data.

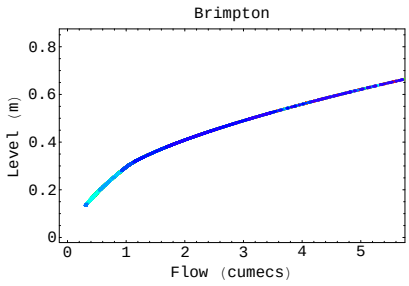
7.7 Model identification

Examination of the regional map in Figure 7.6 shows that two models can be sensibly constructed from the data measured at the marked sensor sites.

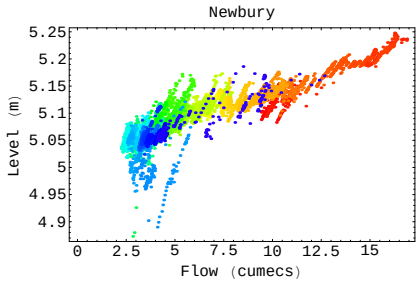
1. Theale area model: Rainfall, Newbury, Shaw and Brimpton to predict Theale.
2. Windsor area model: Rainfall, Theale, Reading, Twyford and Bourne End to predict Windsor.



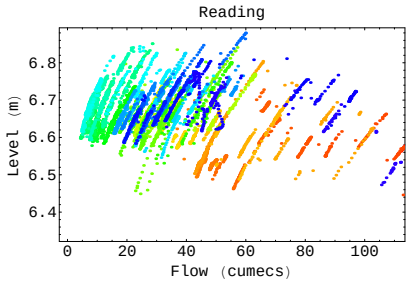
(a) River Wye at Bourne End.



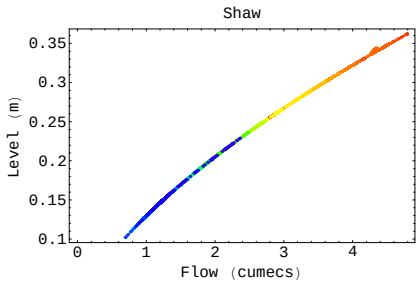
(b) River Enbourne at Brimpton.



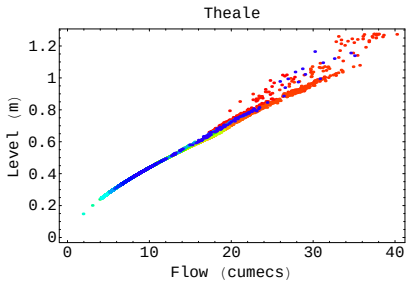
(c) River Kennet at Newbury.



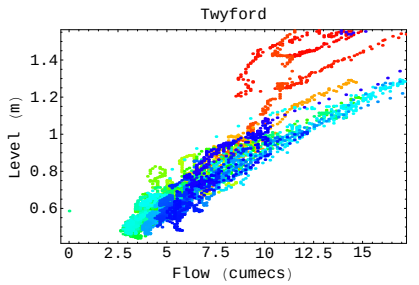
(d) River Thames at Reading.



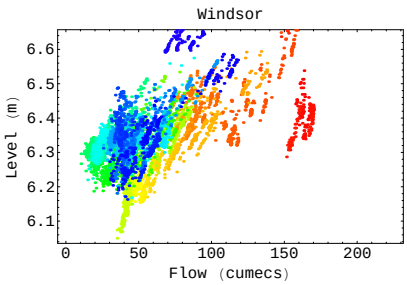
(e) River Lambourn at Shaw.



(f) River Kennet at Theale.



(g) River Loddon at Twyford.



(h) River Thames at Windsor.

Figure 7.12: The flow-level correlations of the river data measured at each sensor site. The hue of the points indicates the time of measurement (the colours change progressively through the spectrum: red points were measured at the start of the period and blue points at the end).

The first model covers the rivers flowing into Theale, primarily the River Kennet. The second model covers the rivers flowing into Windsor, primarily the River Thames, but also the flow from the River Kennet through Theale. This second model allows us to use either the real data measured at Theale, or the predicted river levels from the first model. This enables us to investigate the modular design of a predictive system.

The rainfall indicates some combination of lagged rainfall and aggregated rainfall measurements, and the site name indicates level and flow measurements from the relevant site. Having normalised these data series our first task is to determine lags where possible. For each model we compare the lags obtained from level and flow respectively using both Delta correlation and a Gamma test lag correlation.

7.7.1 Normalisation of data

In this analysis, we are using three different types of measurement (flow rates, levels and rainfall) each measured on a different scale. We decided to normalise all of the data being used before attempting to determine the relevant input variables. The standard normalisation routine used in *winGamma*, which maps the mean to zero and the standard deviation of a data stream to 0.5, was used to re-scale the data. This process of normalisation attempts to equalise the relative numerical significance between the input variables and aid the feature selection routines, especially in the absence of any prior knowledge regarding input variable relevance.

Normalising the data will produce a different set of nearest neighbour relationships compared to those for the unscaled data. However, any two metrics on a Euclidean space are equivalent to within a constant, so the Gamma test analysis on normalised data will not affect the asymptotic nature of the Gamma statistic. Normalisation can also affect the *rate* of convergence of the Gamma statistic and the *quantity* of data required to produce a model of given quality.

7.7.2 Determining the lags

We recall from the discussion of the random walk (see Section 4.5) that a small MSE is not necessarily a good indicator of model quality. Our early experiments in river level and flow prediction indicated that an embedding model constructed from time series data recorded at a *single* site could not be used to produce a model that effectively predicted turning points. Although we were able to obtain models with a relatively low MSE, these models invariably *lagged* the actual data by one

time step (in identical behaviour to the random walk) and so were ineffective in anticipating future changes of level or flow.

We concluded that the only way forward was to endeavour to use upstream data, taken prior to the time of prediction, to model flow or level at downstream sites. Only in this way could we be sure that we were genuinely capturing the flow dynamics.

The most obvious way to determine the correct transfer times between successive measurement points is by direct on-site measurement, preferably under a variety of flow rate conditions. This would be the recommended approach in a real system. It is relatively straightforward to accomplish and, once performed, leaves no room for doubt. Additionally such physical measurements act to validate algorithmic approaches to determining lags. In this account we investigate several algorithmic techniques to determine the transfer times directly from the data measurements.

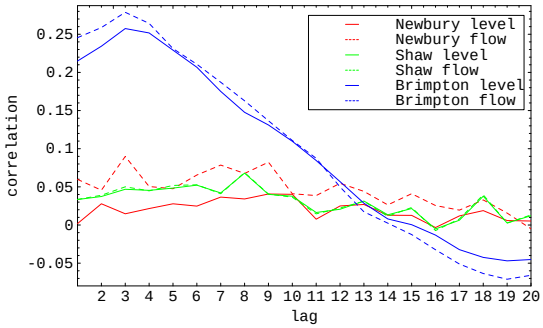
The Gamma test lag correlation or the Delta correlation used to analyse the simulator in Section 7.4.3 can be used directly here to determine the lag times. Another approach might be to use a variation of the False Nearest Neighbour algorithm if it can be modified to handle multiple time series. This concept has not been researched and as such is left for future work.

Determining the lags for the Theale area model

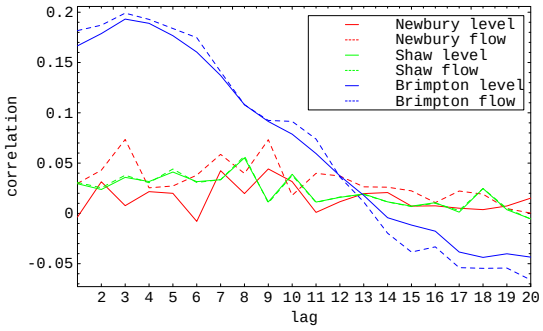
The Delta correlation produced the graphs in Figure 7.13. The level and flow measurements at Newbury, Shaw and Brimpton were correlated to the level and flow measurements taken at Theale. After determining the lags by selecting the maximum correlation we arrive at the Delta correlation results shown in Table 7.3.

Additional analysis was performed using the Gamma test lag correlation. These results are also shown in Table 7.3 to provide comparison to the Delta correlation analysis.

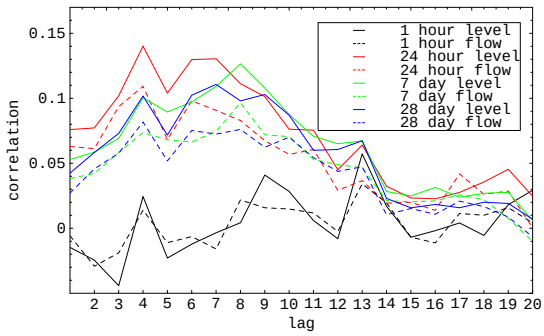
The Delta correlation analysis unambiguously identifies the lags from Shaw and Brimpton to Theale to be 8 hours and 3 hours respectively. The lag between Newbury and Theale is less clear cut. The analysis produces a 3 hour lag using the flow data and a 9 hour lag using the level data. The proximity of Newbury to Shaw would suggest that the lag to Theale should indeed be around 9 hours. A closer examination of the data used to produce Figure 7.13 shows that the 3 hour lag had a correlation of 0.0735 and a correlation of 0.0732 for 9 hours. We can conclude that the likely lag is indeed 9 hours given all of the available evidence.



(a) The Delta flow and level correlations from Newbury, Shaw and Brimpton measured against the river level at Theale.



(b) The Delta flow and level correlations from Newbury, Shaw and Brimpton measured against the river flow at Theale.



(c) The rainfall measured against the level and flow at Theale.

Figure 7.13: The Delta correlation plots for the Theale model.

measurement	Delta		Gamma		used
	level	flow	level	flow	lag
Newbury level	9	9	6	7	9
Newbury flow	3	3	1	1	9
Shaw level	8	8	1	3	8
Shaw flow	8	8	1	3	8
Brimpton level	3	3	6	6	3
Brimpton flow	3	3	1	1	3
Regional rainfall 1-hour	13	13	1	16	13
Regional rainfall 1-day	4	4	16	16	4
Regional rainfall 7-days	8	8	1	1	8
Regional rainfall 28-days	7	4	1	2	7
Marlborough rainfall	13	9	-	-	13
Lambourn rainfall	13	9	-	-	9
Chieveley rainfall	13	13	-	-	13
Kingsclere rainfall	20	20	-	-	8

Table 7.3: Estimated lags for the Theale area measurements. The lags chosen for the analysis were derived from the Delta correlation analysis. The lag for Kingsclere rainfall was manually selected as 8 hours.

For the regional rainfall aggregated over 28 days we obtain a Theale level correlation of 0.111 corresponding to a lag of 7 hours, whereas for the flow we obtain a correlation of 0.102 corresponding to a lag of 4 hours. In this case the meaning of a lag against a 28 day aggregated rainfall is less clear cut, but examining the graphs we decide that a 7 hour lag may be more appropriate here. The Delta correlation between individual rainfall sensor sites and Theale were also analysed as they could introduce additional local information that the aggregated regional rainfall cannot describe.

It is interesting to note that the results of the Delta correlation analysis are relatively consistent, regardless of whether the level or flow are used.

The results for the Gamma test lag correlation were less conclusive and did not produce such effective results as were generated using the river simulator. Although the results are shown in Table 7.3 they did not provide the expected reliability and were not used in this analysis.

Determining the lags for the Windsor area model

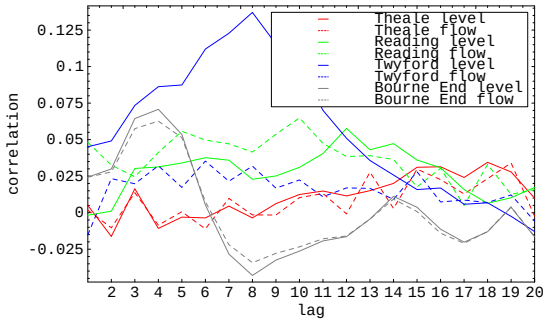
Running the Delta correlation for the Windsor area model produced the graphs shown in Figure 7.14. The level and flow measurements at Theale, Reading, Twyford and Bourne End were correlated to the level and flow measurements taken at Windsor. The lags were then determined from the Delta correlations shown in Table 7.4.

Following the approach used in the analysis of the Theale area model, the Gamma test lag correlation was performed with the results also shown in Table 7.4 to provide comparison to the Delta correlation analysis.

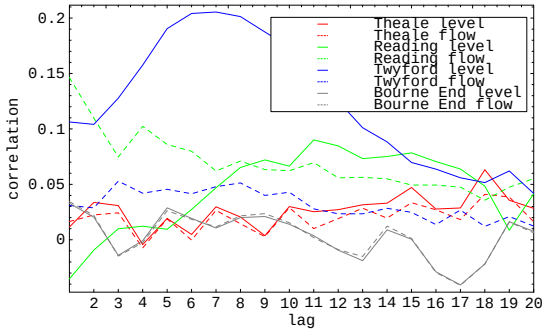
	Delta		Gamma		used
measurement	level	flow	level	flow	lag
Theale level	18	18	18	20	18
Theale flow	19	18	6	18	18
Reading level	12	11	5	5	11
Reading flow	10	1	8	19	11
Twyford level	8	7	9	17	8
Twyford flow	6	3	20	14	8
Bourne End level	4	1	11	3	4
Bourne End flow	4	1	16	15	4
Regional rainfall 1-hour	1	8	2	2	1
Regional rainfall 1-day	6	8	20	11	8
Regional rainfall 7-days	1	1	5	2	1
Regional rainfall 28-days	1	8	8	18	8
Caversham rainfall	11	8	1	14	8

Table 7.4: Estimated lags for the Windsor area measurements. The lags chosen from the analysis were derived primarily from the Delta correlation analysis.

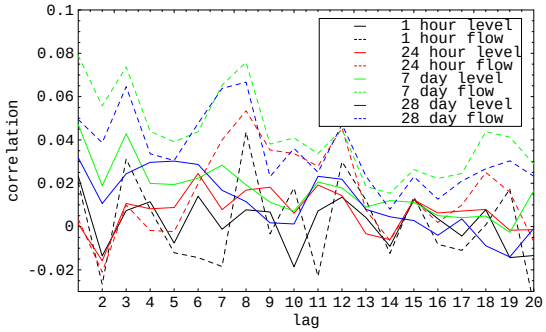
Both the Delta correlation and the Gamma test lag analysis identify that the lag between Theale and Windsor should be approximately 18 hours. The lag between Reading and Windsor is far more uncertain. 11 hours was chosen for the lag because, as we saw for the Theale model, the Delta correlation lags appear to be more reliable. Using Figure 7.14 we deduced that the remaining lags were likely to be 8 and 4 hours between Twyford and Windsor, and Bourne End and Windsor, respectively.



(a) The Delta flow and level correlations from Theale, Reading, Twyford and Bourne End measured against the river level at Windsor.



(b) The Delta flow and level correlations from Theale, Reading, Twyford and Bourne End measured against the river flow at Windsor.



(c) The rainfall measured against the level and flow at Windsor.

Figure 7.14: The Delta correlation plots for the Windsor model.

The regional rainfall was much harder to correlate. The evidence seems to suggest a lag of 1 hour for hourly and weekly aggregated rainfall. A longer lag of 8 hours is appropriate for the daily, 28 day, and Caversham rainfall.

The results of the Delta correlation analysis for level and flow were not as consistent as those calculated for the Theale area model. This could be due to the low correlation between level and flow measurements at the Reading, Twyford and Windsor sensors (Figure 7.12). The results for the Gamma test lag correlation could have been similarly affected.

7.8 Model building

Data files were constructed using the appropriate lags and imported into *winGamma*. These data files were used to build models for the Theale and Windsor areas discussed in the previous Section.

7.8.1 Theale area model

The lags calculated in Table 7.3 were used to construct a data set for the Theale area model. The choice of inputs were validated using the feature selection routines available in *winGamma* and previously discussed in Chapter 4.

The analysis determined that the rainfall at Lambourn and the flow at Newbury were irrelevant ($|\Gamma| = 0.00077$ with Lambourn rainfall and Newbury flow and $|\Gamma| = 1.9 \times 10^{-6}$ excluding Lambourn rainfall and Newbury flow). The results of the analysis are shown in Table 7.5.

This discovery that the rainfall measured at Lambourn and the flow at Newbury were not useful led us to try a number of manual revisions to the lag time, but at each stage the feature selection routines discarded the measurements. This observation may have arisen for a number of reasons. The Lambourn rainfall measurement site is relatively distant from Theale making it difficult to find a correlation. Considering that when rain occurs it is actually distributed across a region, it may well be the case that the Lambourn rainfall measurements, insofar as they contribute at all, are contributing at around the noise level. The Newbury flow-level correlation, shown in Figure 7.12(c), indicates that flow and level are not highly correlated. The analysis has subsequently selected the most useful of the Newbury measurements and discarded the less reliable flow information.

The consequence of the analysis is to use the inputs and lags in the Theale model that correspond to those shown in Table 7.3 without the rainfall measurements at Lambourn and the flow measure-

	Including Lambourn rainfall and Newbury flow	Excluding Lambourn rainfall and Newbury flow
$\|\Gamma\|$	0.00077	2.0638×10^{-6}
Gradient A	0.01865	0.022833
Standard error	0.00066	0.00037164
|V-ratio|	0.00306	8.255×10^{-6}
Near neighbours	10	10
M	8076	8076
Zero nearest neighbours	175	414
Lower 95% confidence	-0.00123	-0.0011044
Upper 95% confidence	0.001921	0.0017852
Mask	`11111111111111`	`11011111101111`

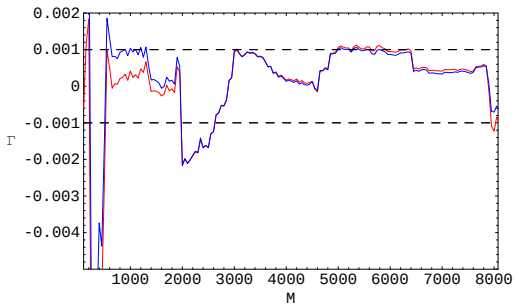
Table 7.5: The Gamma test analysis results on the Theale area data set. The two results compare the effect of including or excluding the Lambourn rainfall and the Newbury flow (indicated by a `1` or `0` in the mask respectively).

ments at Newbury.

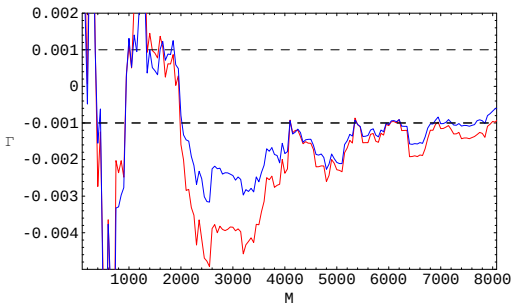
Once the optimal inputs were selected using the Gamma test, the quantity of data was analysed using the M-test to determine whether there was sufficient data to provide an asymptotic Gamma estimate and subsequently a reliable model. The results of this analysis are shown in Figure 7.15(a). To capture the seasonal dynamics of the data, an M-test was performed on randomised data and the results plotted (see Figure 7.15(a)). As the M-test proceeded, the Gamma test algorithm was exposed to points randomly sampled throughout the year. This produced an asymptotic convergence of the Gamma statistic, $\Gamma \approx 0.0007$, and indicated that there was sufficient data at around 6000 data points.

The M-tests were run for the level and flow at Theale. The form of the chart in Figure 7.15(a) indicates that there is very little difference between modelling the level or the flow (the measurements are reasonably well correlated at Theale, shown in Figure 7.12(f)). We know from Figure 7.12(f) that the level and flow measurements at Theale are similar so that the choice of modelling level or flow should be inconsequential. Since flood conditions are more directly related to level we decided to model the river level at Theale.

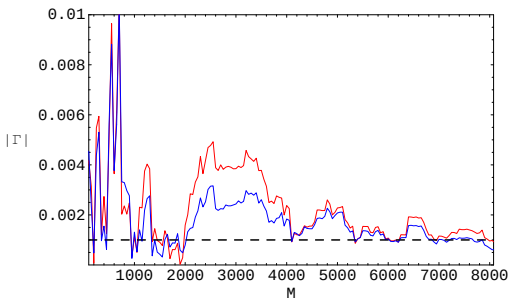
This M-test analysis shows that in order to capture the dynamics of the river system, the river



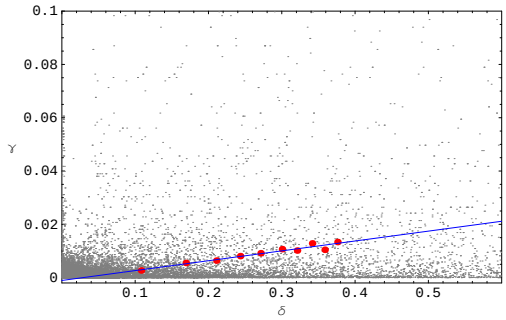
(a) The M-test on the chronologically ordered data oscillates between the Gamma statistic for $M = 6500$ where $|\Gamma| \approx 0.0007$.



(b) The M-test on the randomised data.



(c) The absolute Gamma statistic $|\Gamma|$ for the randomised data M-test.



(d) A Gamma scatter plot generated from the data for the Theale area model using level as the output. The scatter plot shows a high level of noise despite the choice of optimal lags and inputs.

Figure 7.15: M-test performed on the Theale area model data. The red lines correspond to the Gamma statistic calculated for the river level at Theale and the blue lines correspond to the flow.

environment must be sampled for most of one year. Since in this case we only have one year of available data we cannot build a successful model using the data in chronological order by selecting one continuous time period for model training and a second disjoint period for testing. Instead we chose to randomise the data and use a proportion of the data for training and a separate proportion for testing. Using the M-test in Figure 7.15(b) we know that at least 6000 randomised data points would be required to build a reliable model.

The Gamma scatter plots for this data set show a high level of noise, even though the lags have been optimised and the *best* combination of inputs selected. Figure 7.15(d) shows an example of a Gamma scatter plot where the output was chosen to be the Theale river level data.

The data order was randomised for model training. The target MSE for the models was 0.000841. This was calculated for the training set created from 6500 randomly selected data points and using the mask `11011111101111` from Table 7.5.

Two types of model were constructed and tested. The first was a LLR model (with 30 near neighbours) and the second was a 12-10-10-1 BFGS neural network. The LLR model is shown in Figure 7.16 and the BFGS model is shown in Figure 7.17. Since the minimum lag used is the 3 hour lag from Brimpton to Theale, these models give a three hour ahead prediction.

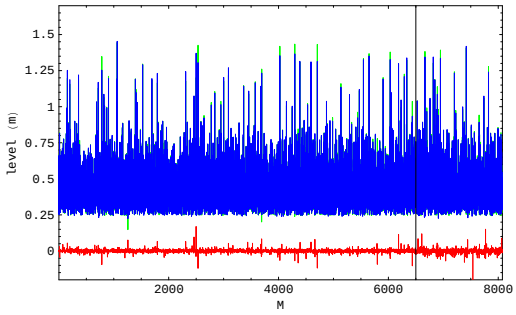
The neural network, trained using the BFGS method, reached a MSE of 0.00086 on the training set. On the unseen test set the MSE was a reasonable 0.0012. This would seem to indicate that the model generalises well and had not been overtrained. The LLR model produced a MSE of 0.000776 on the training data and 0.00202 on the unseen test set. These MSE figures for the scaled data are shown in Table 7.6 with the unscaled values.

	Local-linear regression		Neural network	
	Scaled	Unscaled	Scaled	Unscaled
Training data	0.000776	6.679×10^{-5}	0.000863	8.002×10^{-5}
Test data	0.00202	0.000187	0.00124	0.000115

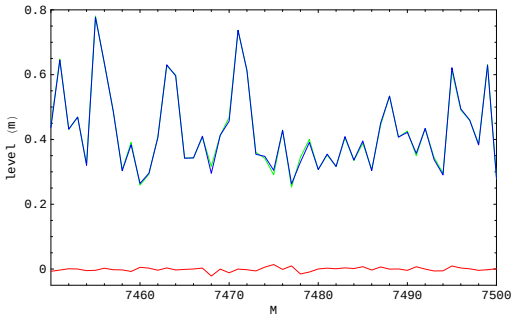
Table 7.6: A comparison of the MSE values of the two Theale area models showing the scaled and unscaled data performance.

Table 7.6 shows that the neural network performs better than the local-linear regression model on the unseen test data. This indicates that the neural network provides a better model. Comparing the two models in unscaled units, this means that the LLR model predicts the level with an average

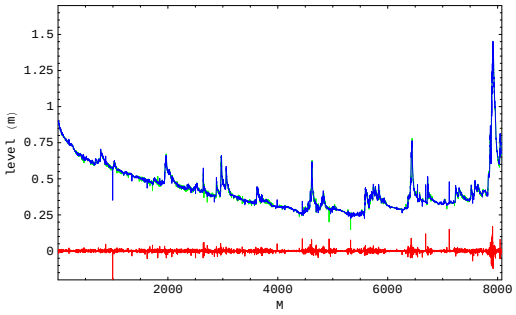
error[1] of 0.014m and the neural network predicts with an average error of 0.011m.



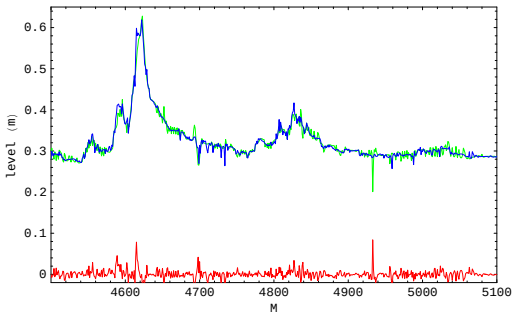
(a) The randomised river data is shown. The first 6500 points were used to construct the LLR model. The remaining points (after the vertical black line) were unseen and were used to test the model.



(b) A closer inspection of the LLR model performance on the unseen data shows an acceptable error level.



(c) The model response in chronological order (20% of points are unseen).



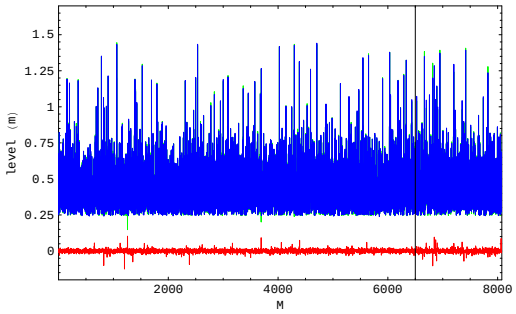
(d) A more detailed view of the chronologically ordered test (20% of points are unseen).

Figure 7.16: The performance of the 3 hour look-ahead LLR Theale area model. The green line shows the actual river level at Theale, the blue line shows the model prediction for the river level, and the red line shows the error between the actual and predicted level.

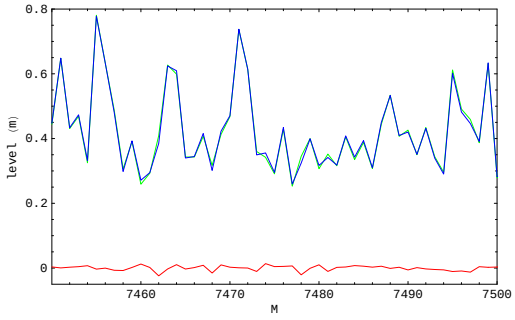
7.8.2 Windsor area model

A model for the Windsor area was constructed using the lags shown in Table 7.4. The estimated model performance was determined using the Gamma test, the results of which are shown in Table 7.5.

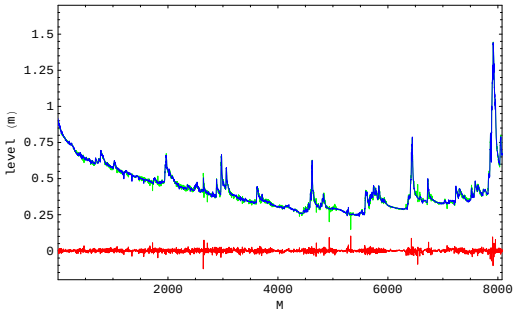
[1]The average error is the square-root of the MSE.



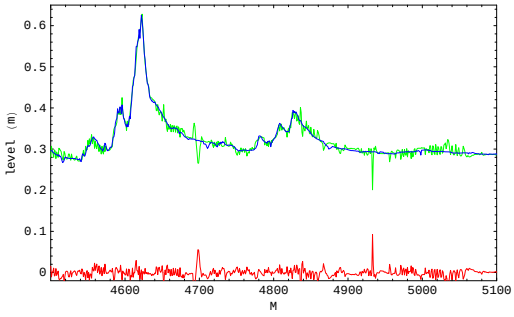
(a) The randomised river data is shown. The first 6500 points were used to train the BFGS model. The remaining points (after the vertical black line) were unseen and were used to test the model.



(b) A closer inspection of the BFGS model performance on the unseen data shows an acceptable error level.



(c) The model response in chronological order (20% of points are unseen).



(d) A more detailed view of the chronologically ordered test (20% of points are unseen).

Figure 7.17: The performance of the 3 hour look-ahead BFGS Theale area model. The green line shows the actual river level at Theale, the blue line shows the model prediction for the river level, and the red line shows the error between the actual and predicted level.

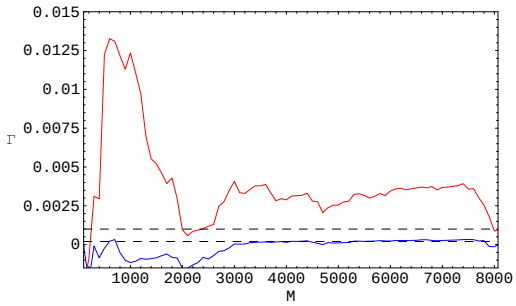
	All Windsor area inputs
$\lvert\Gamma\rvert$	0.000921
Gradient A	0.030042
Standard error	0.000196
$\lvert$V-ratio$\rvert$	0.003682
Near neighbours	10
M	8071
Zero nearest neighbours	2
Lower 95% confidence	-0.00136
Upper 95% confidence	0.003633
Mask	`111111111111`

Table 7.7: The Gamma test analysis result on the Windsor area data set.

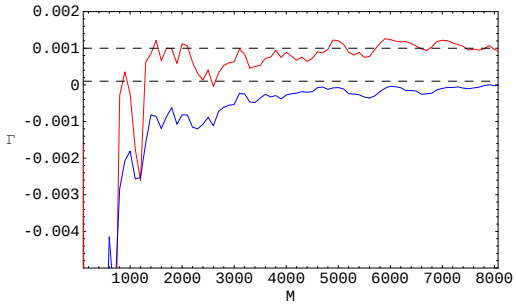
A series of M-tests were performed to determine whether the quantity of data was sufficient to build a reliable model. In the Theale area model the effect of the seasonal dynamics within the data were minimised using randomised data. The results of the M-tests in Figure 7.18 show that the seasonal dynamics can indeed be eliminated this way. In Figure 7.18(a) where the data is analysed in chronological order there is no obvious asymptote. The M-test performed on the randomised data (Figure 7.18(a)) shows that the Gamma statistic asymptotes, $\Gamma \approx 0.001$, at approximately 5000 points.

Each M-test was run on the level and flow data measured at Windsor. Figure 7.18(a) shows that the level and flow Gamma statistics do not converge to approximately the same value as the corresponding values for the Theale area model did, see Figure 7.15(a). This is most likely an artefact of the scaling routine where in the Theale model flow and level were highly correlated and the scaling of each would produce approximately the same value. In Windsor, where the correlation between level and flow was not as significant, the scaling routine performs differently. However, we can see a correlation between the shape of the level and flow M-test curves and this seems to indicate that the M-test is reasonably robust in the presence of noise.

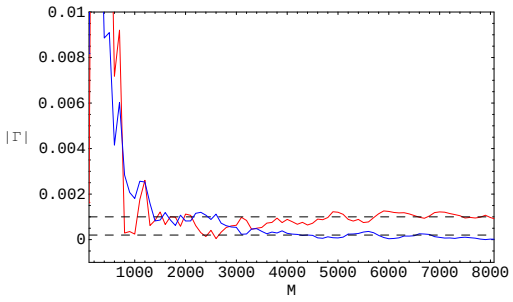
Figure 7.18(d) shows a Gamma scatter plot for the Windsor area model with the river level at Windsor as the output. Even though we are confident that the lags have been optimised and the *best* combination of inputs selected were selected the plot indicates the presence of noise. This is entirely reasonable since there are many unmeasured environmental factors that could affect



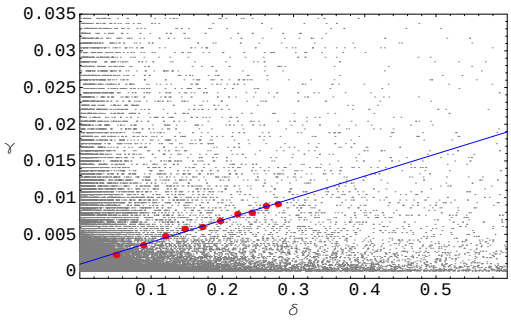
(a) The M-test on the chronologically ordered data does stabilise for $3000 \leq M \leq 7000$ but does not hold for $M > 7000$.



(b) The M-test on the randomised data shows convergence for $M > 5000$.



(c) The absolute Gamma statistic $|\Gamma|$ for the randomised data M-test.



(d) A Gamma scatter plot generated from the data for the Windsor area model using level as the output. The scatter plot shows a high level of noise despite the choice of optimal lags and inputs.

Figure 7.18: M-test performed on the Windsor area model data. The red lines correspond to the Gamma statistic calculated for the river level at Windsor and the blue lines correspond to the flow.

the model quality and, if our assumptions about flow and level correlation are correct, the sensors themselves could be attributable.

A training data set was created from 6500 randomly selected data points using all of the available inputs (see Table 7.7 for the analysis). The target MSE for the models was 0.000627 for $M = 6500$.

Two types of model were constructed and tested. The first was a LLR model (with 30 near neighbours) and the second was a 13-20-15-1 BFGS neural network. The LLR model is shown in Figure 7.16 and the BFGS model is shown in Figure 7.20. Since the minimum lag used is the 4 hour lag from Bourne End to Windsor, these models give a four hour ahead prediction.

The neural network, trained using the BFGS method, reached a MSE of 0.00334 on the training set. On the unseen test set the MSE was a reasonable 0.0056. This is a similar result to the Theale area model where the test MSE was slightly higher than the training MSE and would seem to indicate that the model had not been overtrained. The LLR model produced a MSE of 0.00149 on the training data and 0.00862 on the unseen test set. These MSE figures for the scaled data are shown in Table 7.8 with the unscaled values.

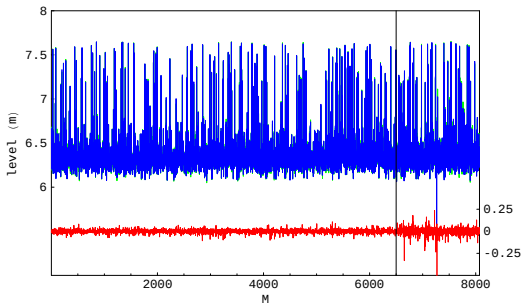
	Local-linear regression		Neural network	
	Scaled	Unscaled	Scaled	Unscaled
Training data	0.00149	0.000217	0.00334	0.000485
Test data	0.00862	0.00125	0.0056	0.000813

Table 7.8: A comparison of the MSE values of the two Windsor area models showing the scaled and unscaled data performance.

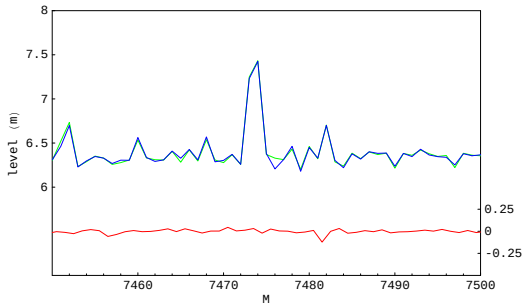
Table 7.8 shows that, like the results for the Theale area model in Table 7.6, the neural network performs better than the local-linear regression model on the unseen test data. If we compare the two models in unscaled units, the LLR model predicts the level with an average error of 0.035m and the neural network predicts with an average error of 0.029m.

7.9 Discussion of prediction results

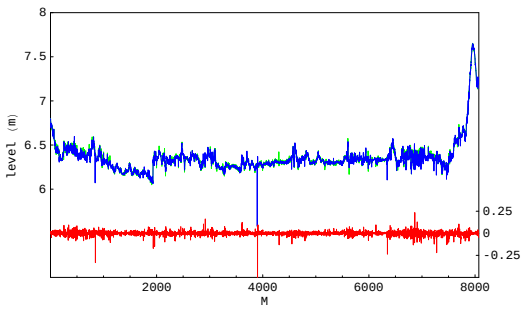
It would appear that there is no barrier in principle to producing very accurate forward predictions apart from the data quality issue. Some of the data problems may have arisen due to faulty sensors.



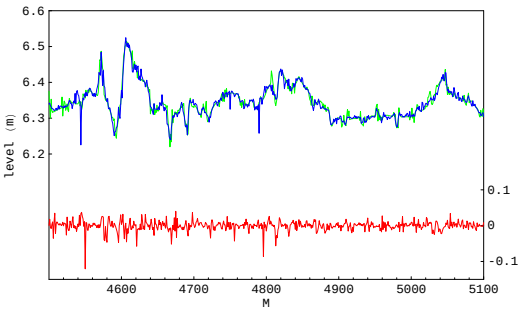
(a) The randomised river data is shown. The first 6500 points were used to construct the LLR model. The remaining points (after the vertical black line) were unseen and were used to test the model.



(b) A closer inspection of the LLR model performance on the unseen data shows an acceptable error level.

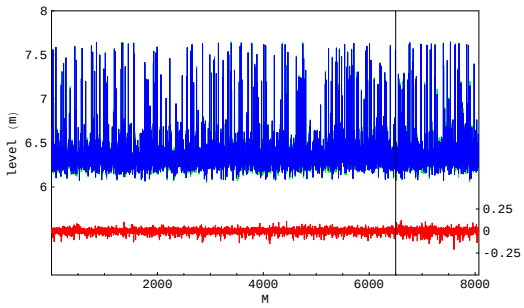


(c) The model response in chronological order (20% of points are unseen).

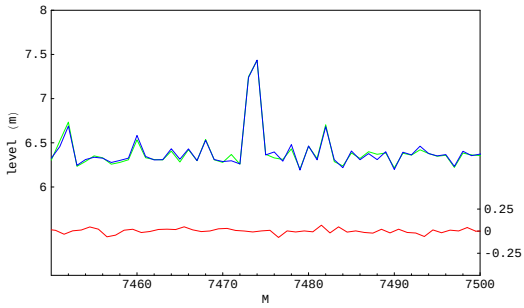


(d) A more detailed view of the chronologically ordered test (20% of points are unseen).

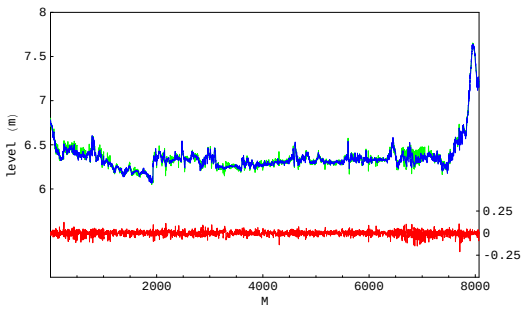
Figure 7.19: The performance of the 4 hour look-ahead LLR Windsor area model. The green line shows the actual river level at Windsor, the blue line shows the model prediction for the river level, and the red line shows the error between the actual and predicted level.



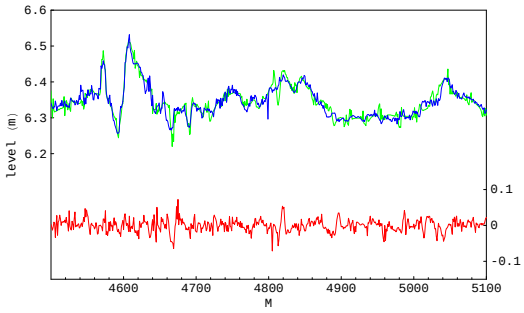
(a) The randomised river data is shown. The first 6500 points were used to train the BFGS model. The remaining points (after the vertical black line) were unseen and were used to test the model.



(b) A closer inspection of the BFGS model performance on the unseen data shows an acceptable error level.



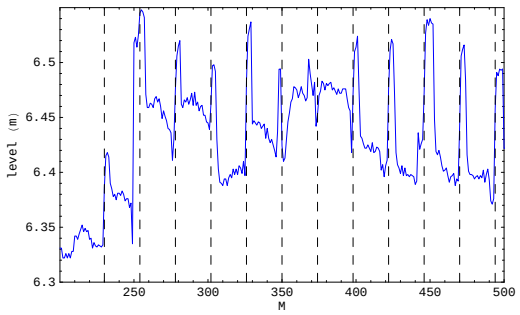
(c) The model response in chronological order (20% of points are unseen).



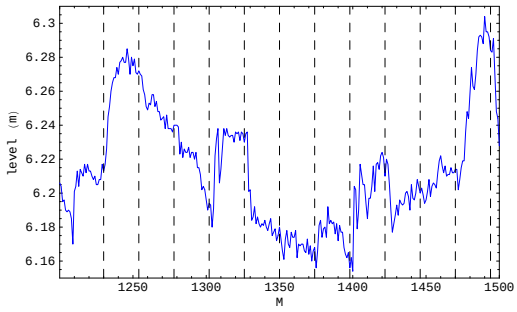
(d) A more detailed view of the chronologically ordered test (20% of points are unseen).

Figure 7.20: The performance of the 4 hour look-ahead BFGS Windsor area model. The green line shows the actual river level at Windsor, the blue line shows the model prediction for the river level, and the red line shows the error between the actual and predicted level.

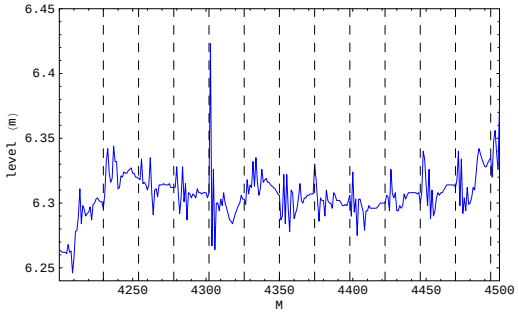
However, there are other issues that also need resolving. One problem is particularly evident in the Windsor data. In Figure 7.21 there are several plots of the hourly river levels over different 300 hour periods, each having an overlay of a series of vertical lines indicating 24 hour periods. It is apparent from the charts that there is daily activity affecting the river levels. The effect is too regular to be explained through coincidence, but instead is either an environmental effect or, perhaps more likely, has arisen through human intervention.



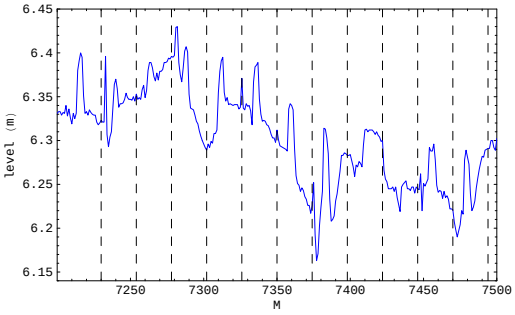
(a) There is evidence of daily activity affecting the river level. The sharp peaks occur on daily intervals. The blue lines indicate actual river level observations.



(b) The daily activity is not always visible.



(c) The periodic activity is not as dramatic as in Figure 7.21(a) but it is still distinctive.



(d) Daily activity is indicated with the series of peaks.

Figure 7.21: There are daily level fluctuations in the river level at Windsor. The spacing between each vertical bar is 24 hours.

We might attribute these fluctuations to periodic extractions and replacements, such as might be created by factory or human use of the river water. If indeed these fluctuations are due to industrial or agricultural use of the river, then our observations could be used to develop an automatic monitoring program for detecting the unlicensed use of river water.

Figure 7.22 shows that the Windsor area model does not predict the daily fluctuations. This sug-

gests that extraction and replacement activities are not captured by the inputs to the model. These activities could be detected by unexpected periodic fluctuations in the error signal, which is precisely analogous to the detection of a digitally encoded signal masked by a chaotic carrier through prediction errors [Tsui et al., 2001].

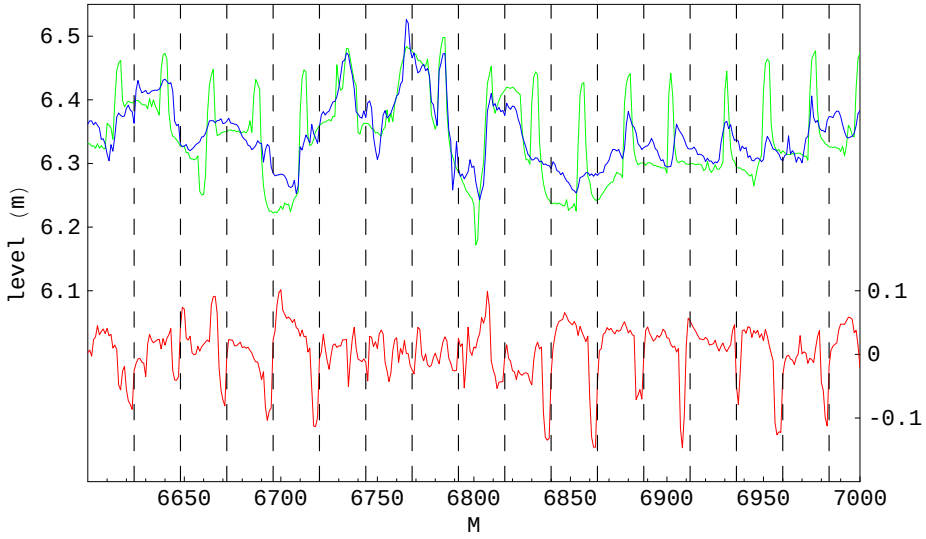


Figure 7.22: A comparison between the Windsor area model (blue) and the actual observations (green) shows that the model does not predict the daily fluctuations. The periodic fluctuations show up in the error (red).

7.10 Constructing a modular flood system

We constructed a modular prediction system of the River Thames at Windsor using the predicted behaviour of the River Kennet at Theale as an input to the Windsor area model. This approach demonstrates the feasibility of a modular flood prediction system, although in this case, because of the location of the tributaries and the placement of sensors sites, does not increase the look-ahead time of the prediction.

The Windsor area model takes the level and flow measurements at Theale as inputs. We know that there is a high correlation between the flow and level measured at Theale (see Figure 7.12(f)) so we built a neural network model to convert the modelled levels at Theale to flow measurements. The response of the model is shown in Figure 7.23.

Table 7.9 describes the performance of the Windsor area model using the modified data set and should be compared with Table 7.8. It can be seen that the LLR model is much less tolerant to

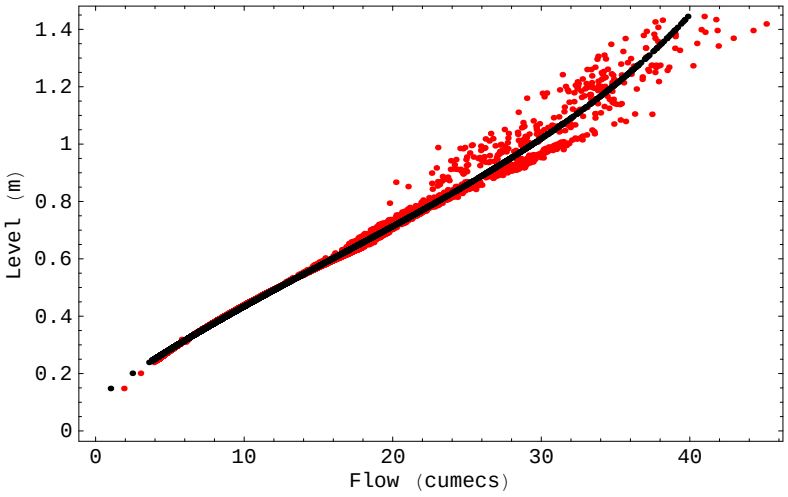


Figure 7.23: Theale level to flow conversion model.

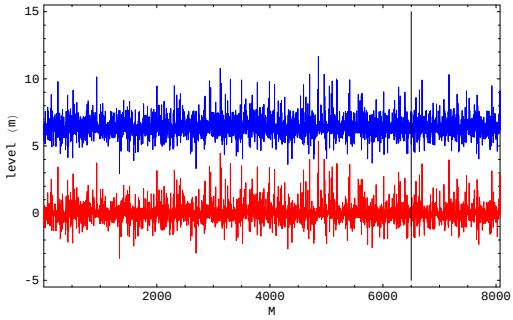
errors in the Theale predictions whereas the results for the BFGS neural network models are only marginally worse than those using the actual Theale values. We conclude that a modular system is feasible and, given an appropriate tributary structure and placement of sensors, could be used to enhance the look-ahead prediction interval.

	Local-linear regression		Neural network	
	Scaled	Unscaled	Scaled	Unscaled
Test data (based on original training data)	1.6057	0.22047	0.0045094	0.00065244
Test data (based on original test data)	1.4842	0.19361	0.0065209	0.00094235

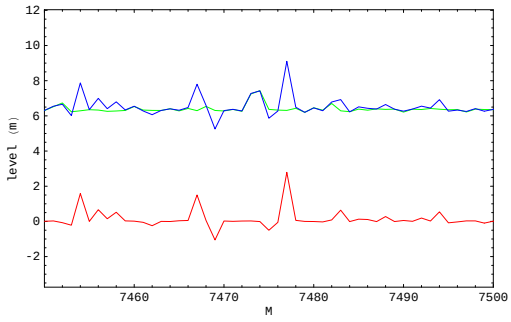
Table 7.9: A comparison of the MSE values of the two Windsor area models showing the scaled and unscaled data performance. The actual river measurements at Theale have been substituted by predicted values from the Theale area model. In all other respects, the test sets were identical to those used to train and test the Windsor area model.

If we compare the two models in unscaled units, the LLR model predicts the level with an average error of 0.44m and the neural network predicts with an average error of 0.031m.

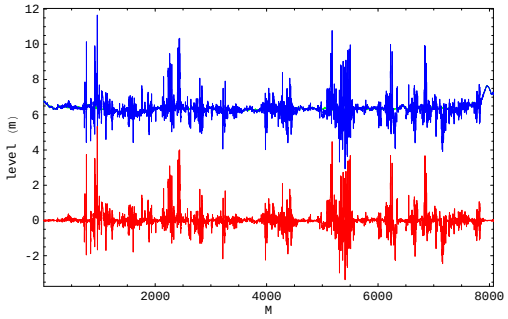
The performance of the modular system is illustrated in Figure 7.24 for the LLR model and in Figure 7.25 for the BFGS model. We can conclude from these results that neural network models would provide the most robust predictive models.



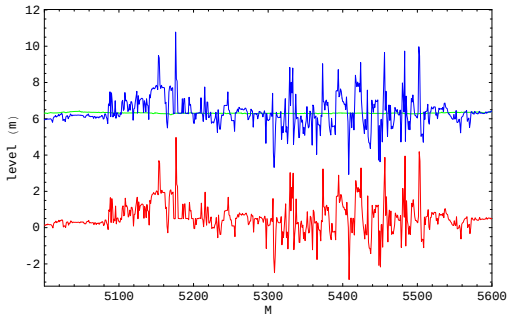
(a) The randomised river data is shown. The first 6500 points of the original data were used to construct the LLR model. The predicted flow and level values at Theale were then substituted for the actual values and this test was then run on the full data set. Thus the remaining points (after the vertical black line) were completely unseen.



(b) A closer inspection of the LLR model performance on the unseen data shows a marginally acceptable error level.

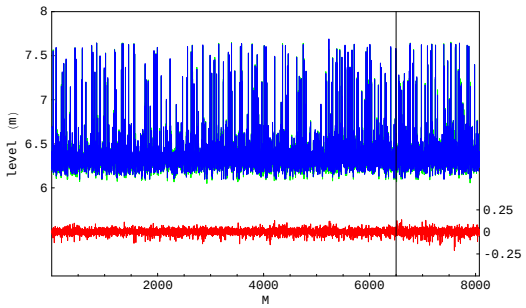


(c) The model response in chronological order (20% of points are completely unseen).

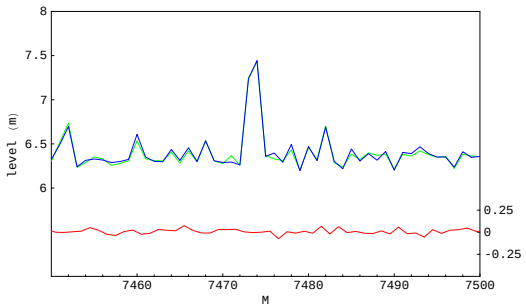


(d) A more detailed view of the chronologically ordered test (20% of points are completely unseen).

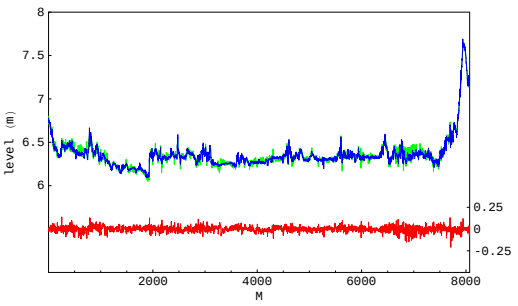
Figure 7.24: The performance of the 4 hour look-ahead modular LLR Windsor area model. The green line shows the actual river level at Windsor, the blue line shows the model prediction for the river level, and the red line shows the error between the actual and predicted level.



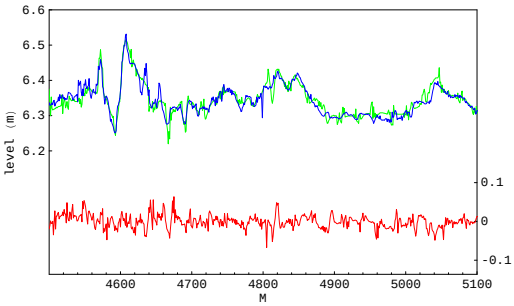
(a) The randomised river data is shown. The first 6500 points of the original data were used to train the BFGS model. The predicted flow and level values at Theale were then substituted for the actual values and this test was then run on the full data set. Thus the remaining points (after the vertical black line) were completely unseen.



(b) A closer inspection of the BFGS model performance on the unseen data shows an acceptable error level.



(c) The model response in chronological order (20% of points are completely unseen).



(d) A more detailed view of the chronologically ordered test (20% of points are completely unseen).

Figure 7.25: The performance of the 4 hour look-ahead modular BFGS Windsor area model. The green line shows the actual river level at Windsor, the blue line shows the model prediction for the river level, and the red line shows the error between the actual and predicted level.

7.11 Conclusions

Reliability of remote sensors and intelligent sensor placement, with appropriate levels of redundancy, are critical issues for a fully automated system. Given the capital expenditure of a fully automated data collection system complete with real-time data links, together with prior or prompt notification of river works (such as dredging or flow diversion) and regular extraction/replacement, it should be possible to rapidly identify sensor failure or progressive sensor degradation. This in turn carries the implication of a requirement for a rapid and responsive maintenance programme to maintain the integrity of the system.

In addition to providing predictions using observable data, the overall utility of the system could be enhanced by providing facilities to run what-if scenarios to predict what affect certain activities would have on the river. For example, a long-term weather forecast could be used to provide an advanced warning of the *possibility*[2] of flooding.

Provided these issues are adequately addressed then it would seem entirely feasible to develop an adaptive system for modelling and predicting river flow and levels over a time scale basically determined by the delay between the precipitation occurring and the water arriving at the prediction point. This is the intention of the *MAPFLOWS* (*Modular Automated Prediction and Flood Warning System*) project, a proposal currently under consideration.

The *MAPFLOWS* proposal stems from the following observations:

- Global warming is likely to lead to more unstable weather patterns. Indeed many meteorologists are of the opinion that this is already happening. Thus there is an urgent need for accurate water level prediction systems which have low set-up times.
- Once precipitation has occurred the process of runoff, although highly complex in any particular catchment area, is completely determined by smooth processes dependent on details of the flow system, the topography and underlying hydrology, the ambient conditions, and any flow control gating in the system.
- Appropriate data for a real-time prediction system can be provided by a number of suitably located standardised monitoring modules with telemetry to a central data processing location.

[2] Predicting the weather is a complicated undertaking, but if done with a suitable level of accuracy could enable the flood prediction system to provide early warning alerts.

One advantage of a system such as *MAPFLOWS* is the relatively low model construction time once appropriate data becomes available. Other advantages include the universality of the tools; once software, sensor technology and telemetry are constructed the system could be easily replicated, sold and installed at many sites around the world.

There is another important issue that surrounds data-derived modelling techniques arising from the extreme and infrequent events that we are often trying to capture. If the training data does not adequately contain this information then there is a possibility that the model will give the incorrect response when one of these events occurs. In order to avoid such pitfalls, the engineer of the system must extensively test it to ensure the model responds in an appropriate way.

Of course, there are other issues that become necessary to resolve. One notable failure of conventional level prediction techniques was the Columbia River (Oregon-Washington, USA) flood of February 1996, in which accumulated snow in the mountains melted rapidly when the temperature rose sharply accompanied by high rainfall. Upstream flooding was widespread and only extreme measures prevented the west coast city of Portland from serious flooding. Thus, apart from the obvious input variables previously listed, it would be wise to include the current depth of surface snow, ambient temperature, relative humidity etc. and to augment the modelling techniques to accommodate these variables. This would require further research which we were unable to undertake in the present investigation (primarily because of lack of data) but would seem to pose no insuperable obstacles.

7.12 Future work

As we have emphasised, we are not proposing a precipitation forecasting system at this stage. However the fusion of satellite cloud images and radar precipitation measurements cannot be ignored because it provides the greatest potential to improve the long term predictability of a river.

It would appear that sensor failure is relatively common. To counter these problems using software, we could construct models to predict the missing values. A useful by-product, if successful, would be that the same models could be used to automatically monitor each sensor to rapidly identify potential degradation and failure. This application could provide immediate benefits for existing river monitoring systems where telemetry provides continuous feedback from the sensors.

In our experiments we arbitrarily constructed average rainfall data for 1 day, 7 days and 28 days. It should be possible using the Gamma test to automatically identify the optimal period over which

to take the average since the Gamma statistic would reach a minimum.

It may be possible to adapt the false nearest neighbour algorithm for multiple time series which would then provide an additional tool to help identify the correct lags for the model inputs. We are not aware of existing work along these lines but, regardless of the potential application to river prediction, it would prove an interesting study in its own right.

The lag times chosen for our models have been constant. It may be possible to improve the quality of the models by bracketing the lag times since they will not necessarily be constant, but may vary with the amount of water moving downstream since higher flow rates produce a faster transfer between the two points in question. Thus once the average transfer time has been determined it may be necessary to bracket this lag by including one or two measurements both before and after the average lag. We can then use the Gamma test feature selection routines to determine the optimal variable combination.

CHAPTER 8

Conclusion

8.1 Introduction

Non-parametric, smooth non-linear modelling is being transformed from a somewhat hit and miss process into a precise science in which before the model is built the error associated with a model prediction can be quantified and, perhaps even more significantly, the relevant input variables can be selected *a priori*.

We have proposed a new approach to modelling level and flow in a water catchment area and performed experiments to illustrate the viability of these ideas. In another application James and Connellan [James and Connellan, 2000] have used the Gamma test to facilitate the construction of models for commercial property price prediction. The nature of these two applications differs in an interesting way. The first is based on general physical principles that flow in a river system cannot change arbitrarily but is constrained by physical laws. The second attempts to model a conglomerate of financial factors of which one is arguably sentiment.

8.2 *winGamma* lessons

The advent of *winGamma* has enabled many researchers to explore new applications of the Gamma test in diverse areas. We have mentioned the work of James and Connellan [James and Connellan, 2000] in modelling economic time series for the prediction of commer-

cial property prices, and might also mention that of [Chuzhanova et al., 1998] in feature extraction for the identification of DNA sequences. As it became easier to perform detailed Gamma test analyses the main barrier to finding new and exciting applications emerged as being the acquisition of appropriate data sets. Researchers understandably tend to adopt a quite propriety attitude towards their own data, commercial vendors tend to charge high prices for access to their data, and medical databases are fenced around with ethical restrictions that hamper access even to anonymised data. Thus it has not always been possible to acquire data which might be suitable and interesting for a Gamma test analysis.

As *winGamma* emerged as a commercially viable product it became apparent that most users were interested in economic or commercial time series analysis. Here the most promising approach seems to be to bring to bear user domain knowledge to determine which other available time series data can act as leading indicators for the target time series. Although *winGamma* can be used very successfully for time series analysis it was not designed specifically for this role. Most of the actual work for time series analysis involves the assembly and preparation of appropriate data sets; combining leading indicator time series and identifying suitable lags. For this we really require a separate software tool which can act as a time series editor. Moreover, we have shown that for activities such as determining the embedding dimension other, possibly faster, algorithms, for example False Nearest Neighbours, can also be profitably used. There is also the interesting question of extending the False Nearest Neighbour algorithm for multiple time series inputs. Ideally, all these analyses tools should be combined into a software platform specifically designed for time series analysis.

8.3 *autoGamma*

As we built up experience in *winGamma* analysis it became apparent that much of the interpretation of diagnostic results, for example the Gamma scatter plot, could be automated. *winGamma* was constructed as a non-linear analyst's workbench and, as with any such tool, there is a learning curve which must be ascended to acquire the necessary skills to apply the tool effectively. However, as we have gained more experience in the use of *winGamma*, and began to develop an analysis protocol, it has become apparent that the whole analysis process could be automated with relatively small loss in effectiveness. It thus appears quite practical to construct an automated tool, *autoGamma* which presented with a data set and some general data-semantics could perform a complete Gamma test analysis and return the results to the user in the form of a report and models in the form of for example *Excel* macros.

To produce *autoGamma* we would have to construct a rule base extracted from the experience of many *winGamma* experiments and this would require some further research. However, there would seem to be no major problems with this proposal and such a tool might well form a useful and commercial product.

8.4 *GammaMiner*

Because the Gamma test runs extremely quickly one can therefore envisage a more sophisticated program (*GammaMiner*) which automatically scans large databases looking for relationships between numerical fields which can be used for modelling and prediction. The user could define which attributes were of particular interest (the targets or outputs required to be predicted) and which other attributes the targets might reasonably depend on (these would form the set of potential inputs to the model). Designing such a program is not without pitfalls. For example, attribute values may not be time-stamped and one could easily find the program 'predicting' values which predate the attribute values used as inputs. There are consequently some problems regarding database semantics which need to be addressed. Because not all data falls into the category of numerical fields which might be modelled by a smooth function and because other types of tools (e.g. decision trees) may be more appropriate for constructing predictive models on discrete inputs or categorical outputs, one could also envisage engineering a subset of *GammaMiner* as a re-usable component designed to be integrated into existing or future data mining tools.

Nevertheless, it is possible to imagine such a program running continuously in the background and notifying its owner only when it found something interesting, e.g. "It is possible to build a predictive model for X for one month ahead that gives an accuracy of 0.5%, are you interested?". Whilst such program behaviour is arguably not intelligent in any real sense there is no doubt that such a tool would be useful, especially with the growing adoption of business intelligence tools that make use of data warehouse and data mining techniques.

Many users of *winGamma* are explicitly interested in time series prediction of economic data. We propose in the first instance to provide a set of time series editing tools which facilitate the alignment in time of attribute values from different time series and the selection of subsets of lagged data to be explored by *GammaMiner* in seeking to evaluate predictive capability.

The *GammaMiner* project seeks to prototype an automated model extraction capability possibly with special reference to time series. Whilst it is indeed possible that genuine new scientific knowledge might result from the use of such a program, it is worthwhile to reflect briefly on the

scientific value of such opportunistic model building.

8.5 The status of data-derived model predictions

When physicists try to make predictions they are following one of the basic principles of science:

- Postulate the basic laws - they are supposed to hold for all time and in all places.
- Calculate the consequences.
- Perform experiments or observations to verify the predictions.

Successful verification does not constitute a 'proof' of the law but failure to verify might constitute a disproof (if all the loopholes in the logic have been plugged).

The philosophical study of our sources of knowledge is known as *epistemology*. Since the laws of physics are supposed to be invariant over all time and space we could say loosely that physics espouses a Platonian view of knowledge in which the 'laws' are there and fixed and it is up to us to discover them: usually in some very pure mathematical form.

- The *advantage* of having such laws available is that because they are supposed invariant over time and space one may be able to make predictions for circumstances that have never been observed before - we call this extrapolation.
- The *disadvantage* is that sometimes the calculations directly from the laws (or 'first principles' as it is often called) may be so complicated or take so long as to be impractical.

The barrier which often presents itself is one of computational complexity. As a simple example consider the *protein folding problem*. A big protein has thousands of constituent atoms and we might know its atomic structure exactly. The biological action of the protein is what we would like to predict. Now if one were to hold the protein by both ends and let go it would collapse into something which, on the right scale, would look like a tangled ball of wool. The biological action of the protein is largely determined by what is left on the outside of the ball of wool. So the problem is simple: we know the effects of atomic bonds, we know the structure so let's just plug all this into a computer program and compute the folded structure. That sounds good, but except for fairly small molecules it can't be done - the program takes too long to run. But things are even worse than this!

Indeed even *without* the Heisenberg uncertainty principle (which says you cannot measure both the position and momentum of a particle with an arbitrary degree of precision) the universe *a la* Newton really contained the seeds of its own destruction.

- Even if you know all the initial conditions of some quite simple chaotic process exactly (which actually you can't) then the amount of computation required to predict a long way into the future with the fastest computer one could imagine would still *require a time greater than the estimated life expectancy of the universe*.

This is the first lesson of chaos. An example is the weather - where we know all the laws and can measure to our heart's content but we cannot even predict reliably several days into the future, let alone several months.

But there are other, more pragmatic, approaches. When we talk about 'predicting the future' in this context we have a rather cavalier approach in mind - a kind of *opportunistic epistemology* which runs more along the following lines:

- A model is 'good' just as long as it is predicting well. When it stops predicting well, we just try to build another model.

This is because we come at the question of prediction from an Artificial Intelligence perspective. What we need are predictive models which work and which can be computed rapidly. The extent to which economic or sociological models discovered by application of tools such as the Gamma test are truly scientific depends on the context but one should recognise that it is arguably a philosophical question.

8.6 Main contributions of this thesis

We have engaged in a software engineering exercise to produce a novel tool in non-linear analysis and modelling. This tool has already been widely used by researchers at Cardiff University and is now a commercial product with a small but rapidly increasing user base in the larger research community. The lessons learned from this development can now be applied to create more powerful tools which are simpler to use, requiring less expertise on the part of the user. Tools such as *GammaMiner* might eventually be used over large distributed databases to elicit genuinely new scientific knowledge.

In addition we have introduced a generalisation of the Gamma test, the *Higher Moments Gamma test* which shows how for symmetric noise a close approximation of the original unknown noise distribution can be reconstructed.

Finally, we have taken the software tools and theoretical techniques developed herein and shown how to adaptively construct non-linear predictive models for river system level and flow directly from the data. We believe that this application is the first detailed and extensive analysis of a river system aimed at constructing non-parametric, non-linear models and, as such, may represent a significant step forward in practical hydrology.

APPENDIX A

k-d Tree

The nearest neighbour problem involves finding the closest point (or points) to a query point from M points in a k-dimensional space. Considerable research has been undertaken to optimise the nearest neighbour search process particularly within the area of *vector quantisation* (VQ) where nearest neighbour encoding performs a fundamental role, [Ramasubramanian and Paliwal, 1992] and [Katsavounidis et al., 1996].

Our concern is to find the p_{max} nearest neighbours to a query point as efficiently as possible in order to minimise the execution time of the Gamma test (see algorithm 4). The brute force technique (which performs an exhaustive search) is only suitable for the simplest problems. The run-time complexity is $O(M^2)$ which becomes prohibitive for large M. We must therefore employ one of the fast near neighbour algorithms designed to deal with large data sets.

We can divide fast nearest neighbour algorithms into two categories

1. *Axis-partitioning algorithms*, e.g. the k-d tree algorithm [Friedman et al., 1979].
2. *Triangle inequality-based algorithms*, e.g. the FN [Fukunaga and Narendra, 1975] and FNM [McNames et al., 1999] algorithms.

We selected the k-d tree fast near neighbour search technique for its simple implementation and scalability, both in terms of low storage requirements and time complexity.

The scalability of the k-d tree is such that it can be implemented with minimal storage $O(M)$,

has a time complexity of construction $O(M \log M)$, and can be queried with time complexity $O(\log M)$. To perform a complete run of the Gamma test, or modelling using local-linear regression, using p_{max} near neighbours to each of the M query points has a time complexity of $O(M \log M)$. The dimensionality of the data k also affects the execution time. Most (if not all) alternative near neighbour techniques have the same time complexity, or worse.

A.1 k-d Tree

There are two components to a k-d tree: (1) the construction of the k-d tree data structure, and (2) the search algorithm for finding the nearest neighbours from the k-d tree.

We begin by describing the technique to build the data structure (the k-d tree) and then describe the search method to find the nearest neighbours of a query point.

A.1.1 k-d Tree construction

The k-d tree is a generalisation of the binary tree where the search space $\mathbb{R}^k$ is divided into two parts at each node. The root node represents the whole data set and each sub-node represents a subset of the parent's data. Maximally efficient information storage is encapsulated when the tree is balanced, such that each child node has an equal chance of being selected.

The partitioning of the search space occurs for the variable $K \in \{1 \dots k\}$ with the greatest range, where K is the *partitioning key*. The median value of the variable given by the partitioning key K provides the *partitioning value* V. Any data point $\mathbf{x}_i$ can then be located either into the left sub-tree L or the right sub-tree R with respect to the partitioning value V and the partitioning key K, such that $\mathbf{x}_i \in L$ if $\mathbf{x}_{i,K} \leq V$, otherwise $\mathbf{x}_i \in R$ (where $\mathbf{x}_{i,K}$ is the K^{th} component of $\mathbf{x}_i$).

This process of partitioning the data into sub-trees continues until no more than B points are stored at each node. These nodes are terminal and called *buckets* and contain 1 or more data points, up to the maximum *bucket size* B. Empirical evidence provided by [Friedman et al., 1979] suggests that between 4 and 32 points per bucket provides optimal performance. A bucket size of 4 was arbitrarily chosen for our implementation of *winGamma*.

k-d Tree time complexity analysis

The whole data set must be scanned at each level of the tree[1] to calculate the partitioning keys and the median values for each node. The calculation of the median for a list of numbers can be achieved with time complexity $O(M)$ (an algorithm to achieve this is given in [Press et al., 1992]). Therefore this computation has time complexity $O(M)$ at each level of the tree and must be performed for each of the $\log M$ levels. Therefore the total time complexity for k-d tree construction is $O(M \log M)$.

k-d Tree storage requirements

The storage requirements for a k-d tree is $O(M)$. In addition to storing the data set, very little extra information needs to be stored. In an efficient implementation where, in the process of finding the median value, the data is sorted in place, the storage requirements are: for each non-terminal node the location of the data subset, the partitioning key K, the partition value V, and the links to the child nodes need to be stored, and for each terminal node (bucket) the number of data points and the location of the bucket data needs to be stored.

k-d Tree construction

The k-d tree construction algorithm is shown in Algorithm 10.

The algorithm accepts data as input and returns the root node to the tree. If the number of data points in the current node does not exceed the bucket size, the node is made terminal and the algorithm finishes. If more data is available than can be accommodated in a single bucket, the data set is partitioned into two data sets according to the partitioning key and partitioning value. The $BuildTree$ function is then called for each of these two *left* and *right* data sets. This recursive process continues partitioning the data until all of the branches of the tree end with terminal nodes. As the recursive process unwinds the connections from the parent nodes to their immediate child nodes are made and stored.

The $CalcSpread$ routine should be implemented to return the range of the data for a particular variable. The $Median$ function returns the median of the data for a particular variable ([Press et al., 1992] describes one method to perform this with time complexity $O(M)$). The

[1] The scanning of the whole data set at each level of the tree arises from the accumulative effect of scanning the data subsets of each node.

```
node function BuildTree(data)
begin
  if Size(data) ≤ bucketSize then
    return (MakeTerminalNode(data))
  end if
  maxSpread = 0
  for j = 1 to k do
    if CalcSpread(data, j) > maxSpread then
      maxSpread = CalcSpread(data, j)
      partitioningKey = j
    end if
    j = j + 1
  end for
  partitioningValue = Median(data, partitioningKey)

  return (MakeNonTerminalNode(partitioningKey, partitioningValue,
    BuildTree(LeftSubSet(partitioningKey, partitioningValue, data)),
    BuildTree(RightSubSet(partitioningKey, partitioningValue, data)))
end


root = BuildTree(data)
```

Algorithm 10: k-d Tree Construction

$LeftSubSet$ and $RightSubSet$ functions create the subsets of the data for the left and right child nodes according to the partitioning key (given by the variable with the greatest spread) and the partitioning value (given by the median of the variable with the greatest spread).

$MakeNonTerminalNode$ and $MakeTerminalNode$ create the data structures for the nodes and store the appropriate supporting variables, i.e. the location of the data subset, partitioning key, partitioning value, and links to the child nodes for non-terminal nodes, and the number of points in the bucket and location of the bucket data for the terminal nodes.

A.1.2 Searching for Nearest Neighbours

If, for a given data set, the associated k-d tree is constructed in an optimal configuration (section A.1.1 describes how to do this) then the number of records required to be searched should be minimal. The k-d tree data structure enables the search to consider only those records closest to the query record, thus reducing the overall search time.

k-d Tree search

Some initialisation has to be done prior to the tree being searched and this is shown in Algorithm 11. Once initialised, the search, described in Algorithm 12, can begin.

```
{initialisation}
set queryPoint and pmax
nearNeighbours[1 : pmax]
nearNeighboursDistance[1 : pmax] = ∞
upperBound[1 : k] = ∞
lowerBound[1 : k] = ∞

{search the tree from the root node}
SearchTree(root)
```

Algorithm 11: k-d Tree search initialisation

The search is made for the $pmax$ points closest to the $queryPoint$ in the data set. A list of the near neighbours and their associated distances are maintained, called $nearNeighbours$ and $nearNeighboursDistance$ respectively. Initially the near neighbour distances are set to ∞ to enable nearer points to enter the list as the search progresses. A list of upper and lower bounds are maintained for each dimension ($upperBound$ and $lowerBound$ respectively). These describe the current bounds of the search space and are used to eliminate searching branches of the k-d tree that lie outside of the search space. Finally, a call to $SearchTree$ starts the search.

The search algorithm is shown in Algorithm 12. The initial call to $SearchTree$ is made from the root node. The algorithm then performs a depth-first recursive search through the tree.

The algorithm first checks whether the current node is terminal. If the node is terminal, all of the points in the bucket are checked against the current nearest neighbour distances list $nearNeighboursDistance$ to see if any points from the bucket are closer than those found so far. If a closer near neighbour is found, $nearNeighboursDistance$ is updated and the point is inserted into the $nearNeighbours$ list, displacing the furthest near neighbour point found so far. The algorithm then returns because there are no further branches to enter.

If the current node is non-terminal then the partitioning value (i.e. the median) and the partitioning key for that node are extracted. These are then used to determine which branch of the tree to examine from the current node. If $queryPoint[partitioningKey] \leq median$ then the search proceeds down the left-hand branch, otherwise the right-hand branch is searched.

The algorithm proceeds by descending the chosen branch, temporarily updating the upper and lower search boundaries for that branch, as determined by the $median$, and recursing down the

```
function SearchTree(node)
begin
  if node is terminal then
    Examine each point in bucket and
      update nearNeighbours and nearNeighboursDistance
    return
  end if

  {traverse the tree}
  median = Median(node)
  partitioningKey = PartitioningKey(node)

  if queryPoint[partitioningKey] ≤ median then
    {recurse on nearest child}
    temp = upperBound[partitioningKey]
    upperBound[partitioningKey] = median
    SearchTree(LeftChild(node))
    upperBound[partitioningKey] = temp

    {recurse on furthest child}
    temp = lowerBound[partitioningKey]
    lowerBound[partitioningKey] = median
    if BoundsOverlapBall then
      SearchTree(RightChild(node))
    end if
    lowerBound[partitioningKey] = temp
  else
    {recurse on nearest child}
    temp = lowerBound[partitioningKey]
    lowerBound[partitioningKey] = median
    SearchTree(RightChild(node))
    lowerBound[partitioningKey] = temp

    {recurse on furthest child}
    temp = upperBound[partitioningKey]
    upperBound[partitioningKey] = median
    if BoundsOverlapBall then
      SearchTree(LeftChild(node))
    end if
    upperBound[partitioningKey] = temp
  end if
end
```

Algorithm 12: k-d Tree search

tree to the bucket nodes. As the recursion unwinds, the bounds prior to the descent are reinstated followed by a call to $BoundsOverlapBall$. This is used to determine whether the opposite branch needs to be searched because it contains points closer to the query point than the furthest near neighbour found so far. If this is the case, the search bounds for the new branch are temporarily recorded and the descent is made.

Algorithm 13 describes the $BoundsOverlapBall$ routine. A *ball* can be imagined to surround the nearest neighbour points whose extent in each dimension is determined by the minimum and maximum values of the nearest neighbours in that dimension. If the extent of the ball lies outside of the search boundaries then there is no need to continue searching. If, however, the search boundaries overlap the *ball* (i.e. fully or partially contain it) then the search must continue down the opposite branch (the code that make this decision on the basis of the $BoundsOverlapBall$ routine is shown in Algorithm 12). This is because points in the opposite branch from the one searched already might be closer than the nearest neighbours found so far.

```
boolean function BoundsOverlapBall
begin
  sum = 0
  for d = 1 to k do
    if queryPoint[d] < lowerBound[d] then
      sum = sum + Distance(queryPoint[d], lowerBound[d])
      if Dissim(sum) > nearestNeighboursDistance(furthest) then
        return false
      end if
    else if queryPoint[d] > upperBound[d] then
      sum = sum + Distance(queryPoint[d], upperBound[d])
      if Dissim(sum) > nearestNeighboursDistance(furthest) then
        return false
      end if
    end if
  end for
  return true
end


{Euclidean distance function}
real function Distance(x1, x2)
begin
  return (x1 - x2)^2
end


{Euclidean dissimilarity function}
real function Dissim(x)
begin
  return sqrt(x)
end
```

Algorithm 13: k-d Tree bounds overlap ball

A.2 Conclusion and further work

In a recent study [McNames et al., 1999] the FN and FNM algorithms were shown to outperform the k-d tree algorithm for high dimensional data (for $k > 15$). If we assume that each technique was implemented to the same standard, although no evidence was given about the coding efficiency of each technique, then this also requires us to examine the FN and FNM algorithms as potential replacements for the k-d tree for high dimensional data.

An unpublished study by James McNames of Portland State University (February 2, 2000) demonstrates that, of 17 different near neighbour algorithms chosen for study, any of 5 algorithms could be chosen based on a variety of criteria. The data sets used to test the algorithms had variable dimensions, sample sizes, and distributions. For a wide variety of cases, the k-d tree algorithm produced a high ranking solution compared to the other algorithms (particularly for large sample sizes). The apparent disadvantage of the k-d tree comes when dealing with high dimensional data ($k > 15$ was mentioned in the report). Some care has to be taken when using the results of this study because implementation details were not fully detailed and it may be the case that some of the algorithms were more efficiently implemented than others. However, the document does seem to partially justify our choice of the k-d tree as a fast near neighbour algorithm, although improvements could be made, especially for high dimensional data.

One significant development of this unpublished study is the introduction of a new nearest neighbour algorithm called *PAT* (Principal Axis Tree). This report came to our attention after the development of *winGamma* and shows a new algorithm that is significantly faster than most of the algorithms in the study, including the k-d tree that is currently used.

Further work must be undertaken in order to maximise the performance of the nearest neighbour search. This part of the Gamma test algorithm contributes the overwhelming processing time and must be implemented with both the best algorithm and as efficiently as possible in order to minimise the execution time. In order to achieve this, the techniques that have been discussed briefly here, primarily FN, FNM, and PAT, require further investigation.

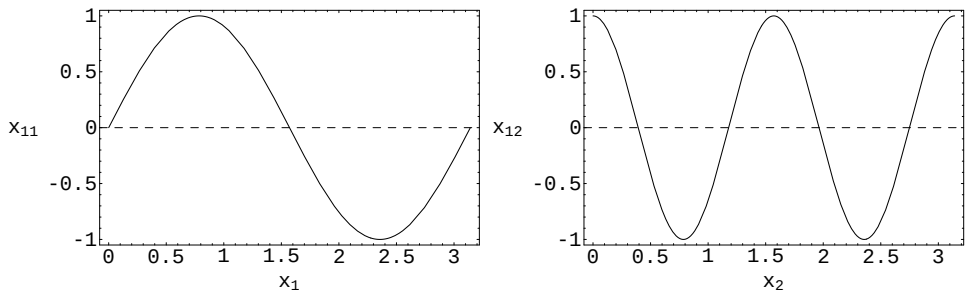
APPENDIX B

Feature Selection

This appendix provides additional information for the experiments described in Section 4.1.

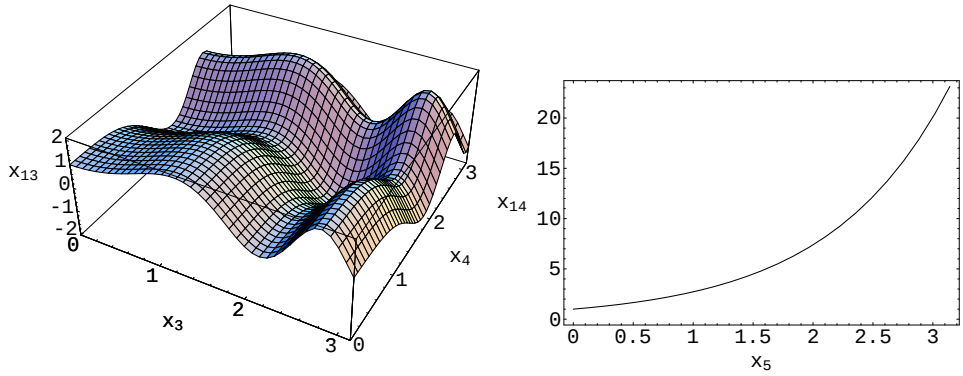
B.1 Full feature space search

These plots are for the 16-dimensional full feature space search described in Section 4.2.

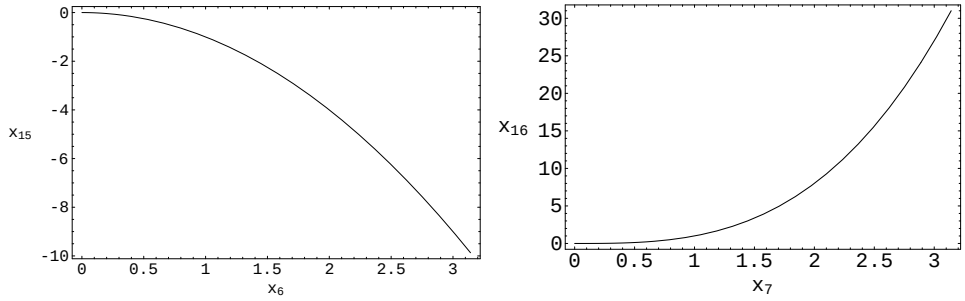


(a) $x_{11} = \sin(2x_1)$ (b) $x_{12} = \cos(4x_2)$

Figure B.1: Feature selection inputs (x_{11}, x_{12}).



(a) $x_{13} = \sin(x_3^2) + \cos(x_4^2)$ (b) $x_{14} = \exp(x_5)$



(c) $x_{15} = -x_6^2$ (d) $x_{16} = x_7^3$

Figure B.2: Feature selection inputs ($x_{13}, \ldots, x_{16}$).

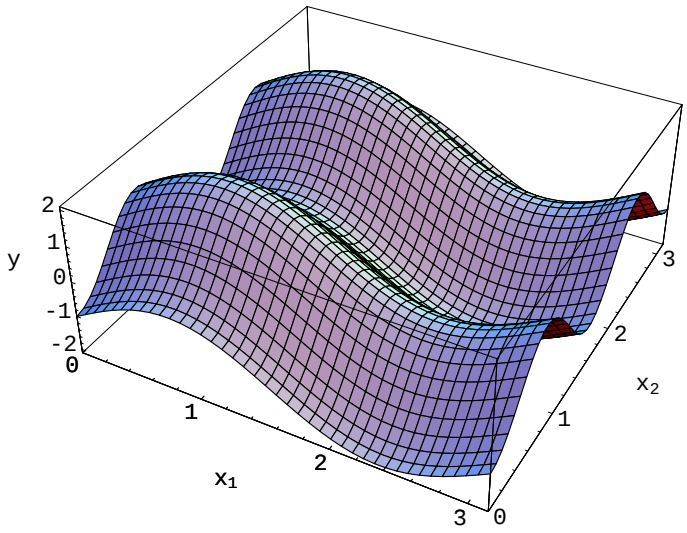


Figure B.3: Feature selection output $y = \sin(2x_1) - \cos(4x_2)$.

APPENDIX C

Non-symmetric Distributions

The non-symmetric noise distributions used within this thesis are based either on a pair of uniform distributions or a lognormal distribution. The techniques used to generate the distributions are described in the following sections.

C.1 Uniform distribution-pair

A simple non-symmetric distribution can be created from a pair of uniform distributions such that the distribution has zero mean and a pdf with area 1. An example of the distribution is shown in Figure C.1.

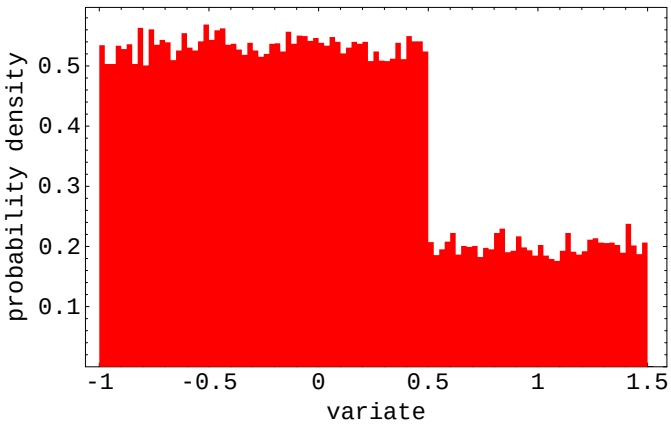


Figure C.1: An example uniform distribution-pair.

Algorithm 14 shows the technique used to generate the distribution. The bounds of the two distributions (`minLeft`, `maxLeft`, `minRight`, `maxRight`) and the proportion of points required in each distribution (`proportionRight`) must be defined to ensure that the algorithm has ability to generate a distribution with mean zero.

```
(* set widthLeft, widthRight, meanRight, proportionRight, numPoints
*)

minRight = meanRight - (widthRight/2);
maxRight = meanRight + (widthRight/2);

rightPoints = numPoints*proportionRight;
leftPoints = numPoints - rightPoints;

rightMoment = meanRight*rightPoints;

meanLeft = -rightMoment/leftPoints;
minLeft = meanLeft - (widthLeft/2);
maxLeft = meanLeft + (widthLeft/2);

uniformDLeft = UniformDistribution[minLeft, maxLeft];
uniformDRight = UniformDistribution[minRight, maxRight];

distLeft = Table[Random[uniformDLeft], x, 1, leftPoints];
distRight = Table[Random[uniformDRight], x, 1, rightPoints];
dist = Join[distLeft, distRight];
```

Algorithm 14: The non-symmetric uniform distribution-pair for *Mathematica*.

The values used to generate Figure C.1 were `widthLeft` $= 1.5$, `widthRight` $= 1$, `meanRight` $= 1$, `numPoints` $= 50000$, and `proportionRight` $= 0.2$.

C.2 Lognormal distribution

From [Kendall and Stuart, 1963] the lognormal distribution has a probability density function (pdf) defined by

$$p(y) = \frac{\delta}{\sqrt{2\pi}y} \exp\left(-\frac{1}{2}(\gamma + \delta \log y)^2\right) \tag{C.1}$$

$0 \leq y \leq \infty$, where δ and γ are parameters[1]. This is derived by considering the variate y defined by

$$\xi = \gamma + \delta \log y \tag{C.2}$$

where ξ is normally distributed with mean zero and standard deviation one.

[1] Unrelated to δ and γ as used in the main text.

Let $\mu_0 = \int_0^\infty p(y)dy = 1$, $\mu = \mu_1 = \int_0^\infty yp(y)dy$, for $r \geq 2$

$$\mu_r[\mu] = \int_0^\infty (y-\mu)^r p(y)dy \tag{C.3}$$

and put $\sigma^2 = \mu_2$. Then, after some algebra, the relationship between (δ, γ) and (μ, σ) is seen to be

$$\delta = \left(\log\left(\left(\frac{\sigma}{\mu} \right)^2 + 1 \right) \right)^{-\frac{1}{2}} \tag{C.4}$$

$$\gamma = \frac{1}{2}\left(\log\left(\left(\frac{\sigma}{\mu} \right)^2 + 1 \right) \right)^{\frac{1}{2}} - \log\mu \left(\log\left(\left(\frac{\sigma}{\mu} \right)^2 + 1 \right) \right)^{-\frac{1}{2}} \tag{C.5}$$

We can also check from (C.1) that the r^{th} moment about zero is given by

$$\mu_r[0] = \exp\left[\frac{r^2}{2\delta^2} - \frac{r\gamma}{\delta} \right] \tag{C.6}$$

and since

$$\mu_r[\mu] = \sum_{j=0}^{r} (-1)^j \binom{r}{j} \mu_{r-j}[0]\mu_1[0]^j \tag{C.7}$$

this enables us to compute $\mu_r[\mu]$ for $r \geq 2$, when given δ and γ.

Algorithm 15 is a *Mathematica* module that returns a random number according to a lognormal distribution specified by $\mu > 0$ and σ.

```
DDelta[mu_, sigma_] := Log[(sigma/mu)^2 + 1]^(-1/2);


GGamma[mu_, sigma_] :=
  (1/2) * Log[(sigma / mu)^2 + 1]^(1/2) -
  Log[mu] / Log[(sigma / mu)^2 + 1]^(1/2);


ALogNormal[mu_, sigma_] := Module[
  { delta, gamma, Xi, y },

  delta = DDelta[mu, sigma];
  gamma = GGamma[mu, sigma];

  Xi = Random[NormalDistribution[0, 1]];
  y = Exp[(Xi - gamma) / delta];

  Return[y];
];
```

Algorithm 15: The lognormal distribution for *Mathematica*.

For the experiments described in Section 5.3, we chose $\mu = 2$ or $\mu = 0.5$ and $\sigma = \sqrt{0.4} \approx 0.632$. From (C.4) and (C.5) this gives $\delta \approx 3.23914$ and $\gamma \approx -2.09084$ for $\mu = 2$ and $\delta \approx 1.02302$ and $\gamma \approx 1.19785$ for $\mu = 0.5$.

Finally since we require a noise distribution with mean zero we translate the random numbers generated by $-\mu$.

From (C.3) we have for $\mu = 2$

$$\begin{aligned} \mu_2[\mu] &= 0.4 & \mu_5[\mu] &= 1.47 \\ \mu_3[\mu] &= 0.248 & \mu_6[\mu] &= 4.36534 \\ \mu_4[\mu] &= 0.760976 & \ldots \end{aligned} \tag{C.8}$$

and for $\mu = 0.5$

$$\begin{aligned} \mu_2[\mu] &= 0.4 & \mu_5[\mu] &= 397.684 \\ \mu_3[\mu] &= 1.472 & \mu_6[\mu] &= 24951.2 \\ \mu_4[\mu] &= 15.7007 & \ldots \end{aligned} \tag{C.9}$$

We have described this implementation in some detail since the $LogNormalDistribution$ function supplied in *Mathematica* is defective.

APPENDIX D

winGamma Overview

The final *winGamma* design included the existing C++ components and incorporated elements and ideas generated during the prototyping cycle. This process lead to seven key areas that needed to be explored:

1. Application interface.
2. Data file management.
3. Data analysis using the Gamma test.
4. Model building using neural networks and local-linear regression.
5. Results visualisation.
6. Results exportation.
7. Project management.

The design and implementation of each of these components is discussed in the following sections. Screenshots are used where appropriate to show the state of the current implementation.

D.1 Application interface

The application interface was designed to have a similar 'look-and-feel' to most other Microsoft Windows compatible software. We decided to use a *multiple document interface* (MDI) design to provide an application window that can contain many sub-windows. In this case a MDI application allowed us to provide a sub-window to view the data sets (*data set manager*) and another sub-window from which experiments could be performed (*analysis manager*). Figure D.1 shows the *winGamma* interface after a data set has been loaded.

The application includes a standard menu structure with the commonest commands replicated on a toolbar for speed of use. The menu is formatted in much the same way as any other Microsoft Windows applications, including a *file* menu for loading data sets, loading and saving projects and exiting the program. There is also an *edit* menu for copying and pasting data, exporting results and deleting Gamma test experiments and models. A *transform* menu provides access to data manipulation routines, for example to scale a data set or to rename variables. The *options* menu can be used to control threads or to change the basic settings of *winGamma*. The *window* menu is a standard menu to control the display of the windows and a *help* menu provides application help and copyright information. The menu structure is shown in Figure D.2.

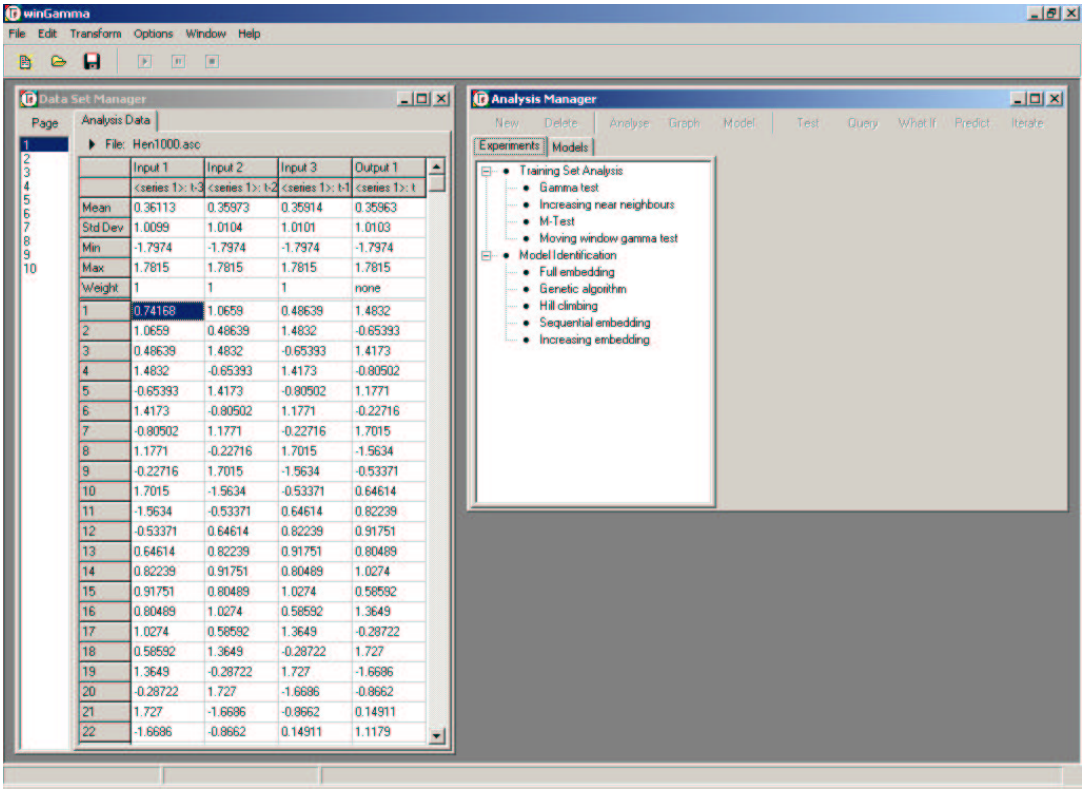
D.2 Data file management

winGamma was designed to load text data files in the formats described in Appendix E. In addition to loading a data file for analysis, we decided to implement *winGamma* with routines to display and manipulate data files in order to reduce the need for secondary software to perform these tasks. For example, it seemed unnecessary for data exported from a database to be manipulated in a spreadsheet prior to analysis if *winGamma* could perform the task.

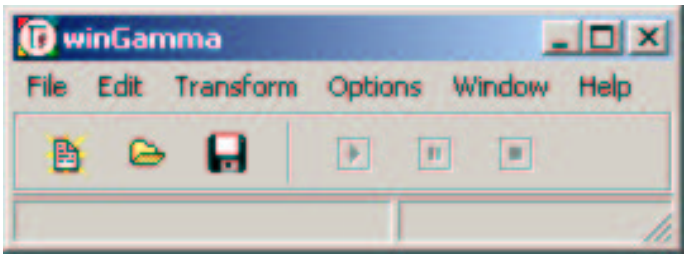
The data file management consists of loading, transforming, viewing and exporting data files. The data can be viewed in, and exported from, the data set manager, whereas loading and transforming a data set is managed by the main *winGamma* interface.

First we must consider how a data set is loaded into *winGamma*. The process to load a data set for analysis is shown in Table D.1. The data set is only loaded if all 6 steps are completed.

In step 1 the entire data file is loaded into memory and parsed to identify invalid formatting and illegal characters.

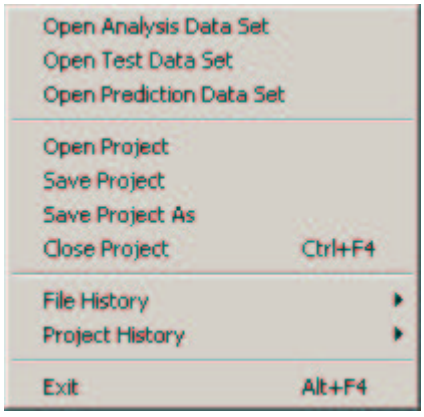


(a) The *winGamma* application interface. The main MDI application window contains the two child windows: (1) the *data set manager*, and (2) the *analysis manager*.

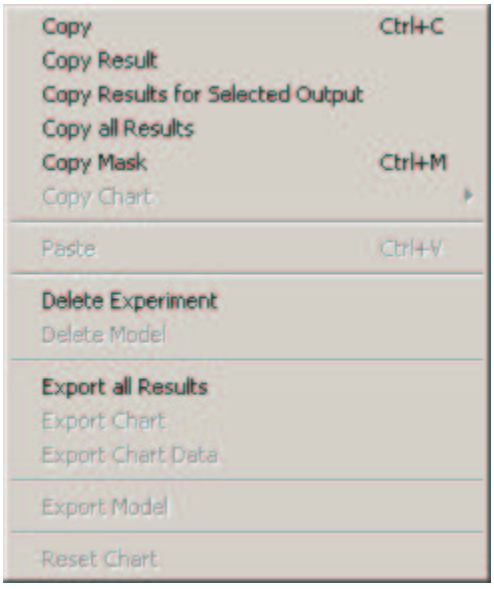


(b) The menu, toolbar and status bar. The toolbar buttons (from left to right) are load new data set, open project, save project, resume active process, pause active process and terminate active process.

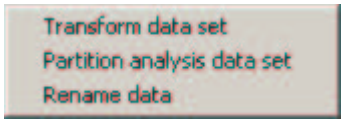
Figure D.1: *winGamma*.



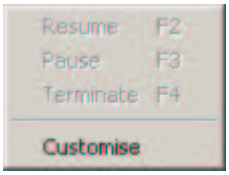
(a) The file menu for loading and saving projects, loading data sets and exiting *winGamma*.



(b) The edit menu for copying and pasting data, deleting experiments and models, exporting results and resetting charts.



(c) The transform menu for manipulating data sets.



(d) The options menu for controlling the active process and setting basic application information.

(e) The window menu for manipulating the active windows.

(f) The help menu for getting application help and copyright information.

Figure D.2: The *winGamma* menu structure.

	Process
1	Load and verify the data file
2	Determine the file type
3	Transform the data†
4	Scale the data‡
5	Partition the data‡
6	View the data

† Transform time series data to input-output form and/or randomise the order of the data.

‡ Optional.

Table D.1: The processes required to load a data set into *winGamma*. Each stage must be successfully completed for the file to load.

If the data file is correctly formatted, *winGamma* performs step 2 to determine whether the file is formatted as a time series, input-output or comma separated value (CSV) file. These formats and file types are detailed in Appendix E. If the file is in CSV format (as typically generated by a spreadsheet or database) then the user must decide whether to convert it to time series or input-output format. Figure D.3 shows the dialog designed for this where the user selects which variables are inputs and which are outputs. Selecting all of the variables as inputs (the default) converts the file into time series format.

Step 3 applies a number of transforms to the data. The order of the data can be randomised[1] for an input-output file as shown in Figure D.4(a). A time series data set must be formatted with the correct number of lags required for analysis by specifying the number of inputs and outputs. The option to randomise the order of the data is also available, but this can only be applied after the data has been formatted as a time series. Figure D.4(b) shows the dialog box designed for time series transformations.

Step 4 allows the data to be scaled (normalised) to put all of the inputs into the same range, which (theoretically) gives all of the inputs the same 'numerical significance'. If normalisation is selected then the input variables are scaled to mean zero and standard deviation 0.5. Another scaling option is called the *heuristic scaling* and is an experimental feature that uses the Gamma test to automatically scale the data according to the significance of each input. In the latest version of *winGamma* this has been disabled while more work is done to perfect the algorithm. This is one

[1] It is only necessary to randomise a data set if a subset of the data is being used for analysis, otherwise this has no effect on the analysis.

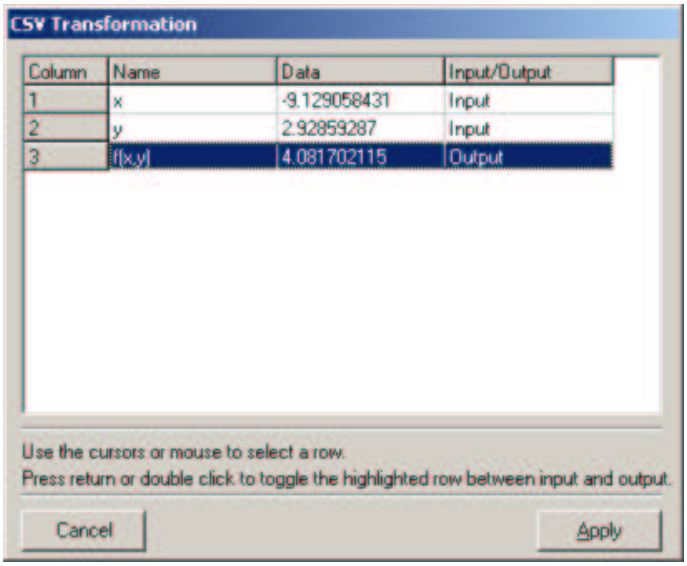
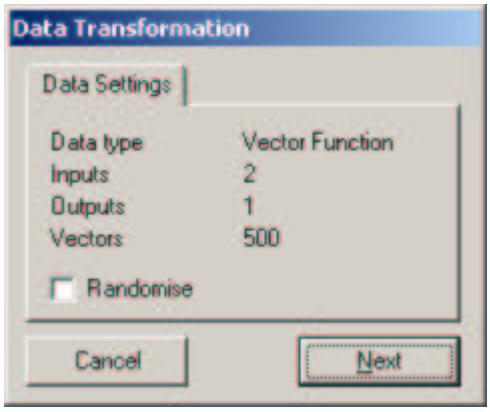
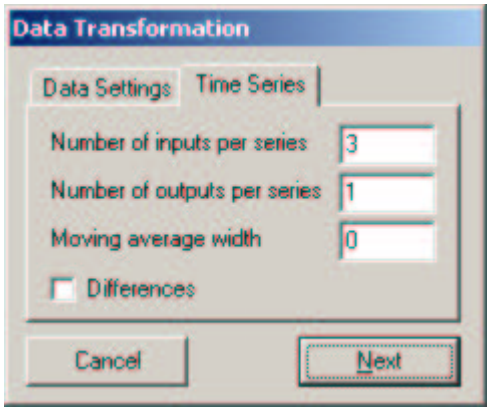


Figure D.3: The CSV transformation dialog allows the user to specify whether a CSV file is converted into time series or input-output format. This example shows a file with 3 variables formatted to input-output format with 2 inputs and 1 output. The values from the first row in the data set are shown as a guide to aid the user when the variable names are undefined.



(a) The data transformation dialog summarise the 'raw' data file. The option to randomise the order of the data is available at this stage. This example shows the dialog for an input-output format data set, but the same summary and randomisation option is provided for time series data, see Figure D.4(b).



(b) The time series transformation dialog box enables the user to specify the number of inputs and outputs required and whether to calculate a moving average and differences for each time series.

Figure D.4: The data transformations for input-output and time series data sets.

of the features discussed for future work in Section 6.6. The scaling dialog box is shown in Figure D.5.

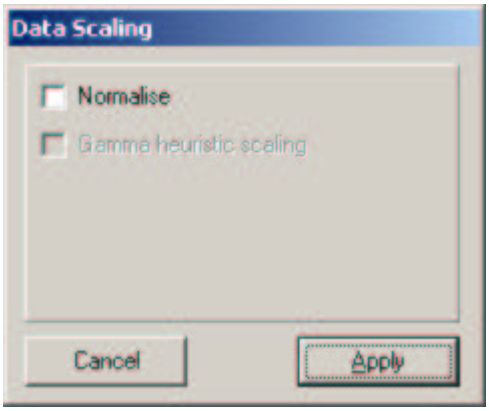


Figure D.5: The data scaling dialog box provides access to routines to normalise the data. Heuristic scaling has been disabled in *winGamma* until the algorithm is completed (see Section 6.6).

At step 5, after the user has selected how the data should be transformed, re-ordered and scaled, the data set can be partitioned. This facility is provided to enable a subset of data to be selected, either for preliminary analysis or where a subset of the data is sufficient for analysis and modelling. The dialog box created for selecting a range of data (and used throughout *winGamma* for partitioning any of the loaded data sets) is shown in Figure D.6.

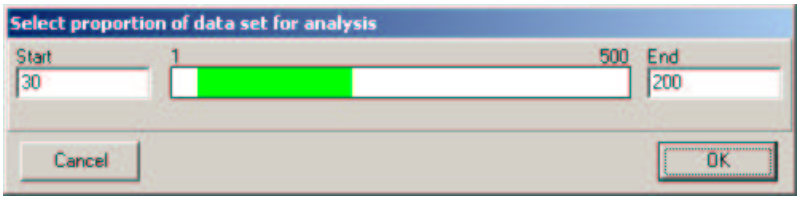
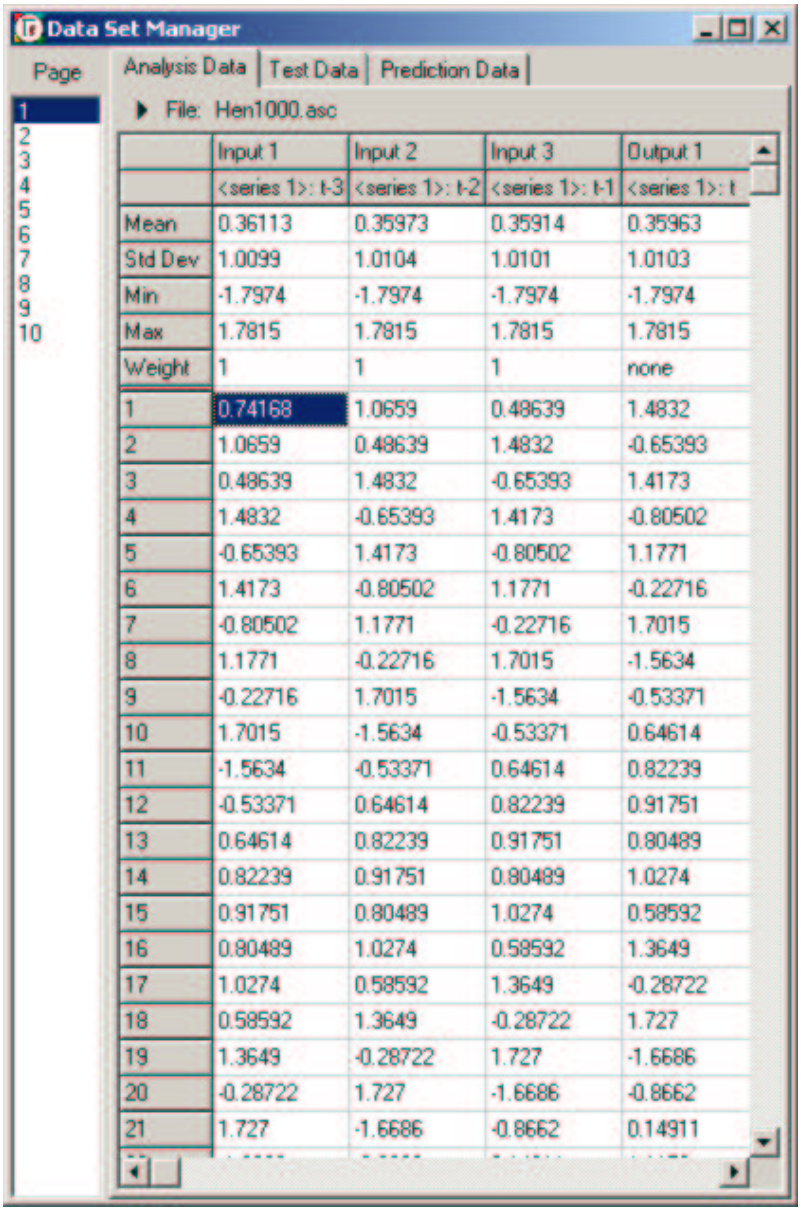


Figure D.6: The data partitioning dialog box enables a subset of the data to be selected for analysis or modelling. The whole data is represented by the white bar (in this case 500 vectors), and the selected data is represented by the green bar (vectors 30-200).

After a data set has been loaded, verified and the file type determined then the data set can be manipulated at any time according to steps 3-5 in Table D.1.

The final step is to display the transformed data set. The prototyping stage highlighted several ways to display data where an implementation using a text grid was chosen as the best solution. There are performance limitations of this technique because it requires an excessive amount of memory to display a large data set. We decided to work around this problem by dividing large data sets into *pages* to maintain fast scrolling and efficient memory management. The current implementation uses the data viewer shown in Figure D.7 using this paged method. 100 vectors

are placed on each page in the current implementation.



	Input 1	Input 2	Input 3	Output 1
	<series 1>: t-3	<series 1>: t-2	<series 1>: t-1	<series 1>: t
Mean	0.36113	0.35973	0.35914	0.35963
Std Dev	1.0099	1.0104	1.0101	1.0103
Min	-1.7974	-1.7974	-1.7974	-1.7974
Max	1.7815	1.7815	1.7815	1.7815
Weight	1	1	1	none
1	0.74168	1.0659	0.48639	1.4832
2	1.0659	0.48639	1.4832	-0.65393
3	0.48639	1.4832	-0.65393	1.4173
4	1.4832	-0.65393	1.4173	-0.80502
5	-0.65393	1.4173	-0.80502	1.1771
6	1.4173	-0.80502	1.1771	-0.22716
7	-0.80502	1.1771	-0.22716	1.7015
8	1.1771	-0.22716	1.7015	-1.5634
9	-0.22716	1.7015	-1.5634	-0.53371
10	1.7015	-1.5634	-0.53371	0.64614
11	-1.5634	-0.53371	0.64614	0.82239
12	-0.53371	0.64614	0.82239	0.91751
13	0.64614	0.82239	0.91751	0.80489
14	0.82239	0.91751	0.80489	1.0274
15	0.91751	0.80489	1.0274	0.58592
16	0.80489	1.0274	0.58592	1.3649
17	1.0274	0.58592	1.3649	-0.28722
18	0.58592	1.3649	-0.28722	1.727
19	1.3649	-0.28722	1.727	-1.6686
20	-0.28722	1.727	-1.6686	-0.8662
21	1.727	-1.6686	-0.8662	0.14911

Figure D.7: The data set manager window can show up to three loaded data sets (all of which are visible in this example). The data is divided into pages containing 100 vectors each. The pages are listed in the left pane of the window and the right tabbed-pane shows the active data set page. In this example, the data set contained 1000 vectors requiring 10 pages.

Additional data files can be loaded for testing and prediction (see Sections D.4.4 and D.4.7 respectively). These files must be in the same format as the analysis data set (except that prediction data sets do not contain output variables). The testing and prediction data sets then automatically undergo the same transformations that occurred to the analysis data set.

D.3 Data analysis using the Gamma test

Once the data set has been loaded (described in Section D.2) the *analysis manager* window is displayed, shown in Figure D.8. The experiments are displayed in the left pane, where the Gamma test experiment types are divided into the categories *data set analysis* and *model identification*.

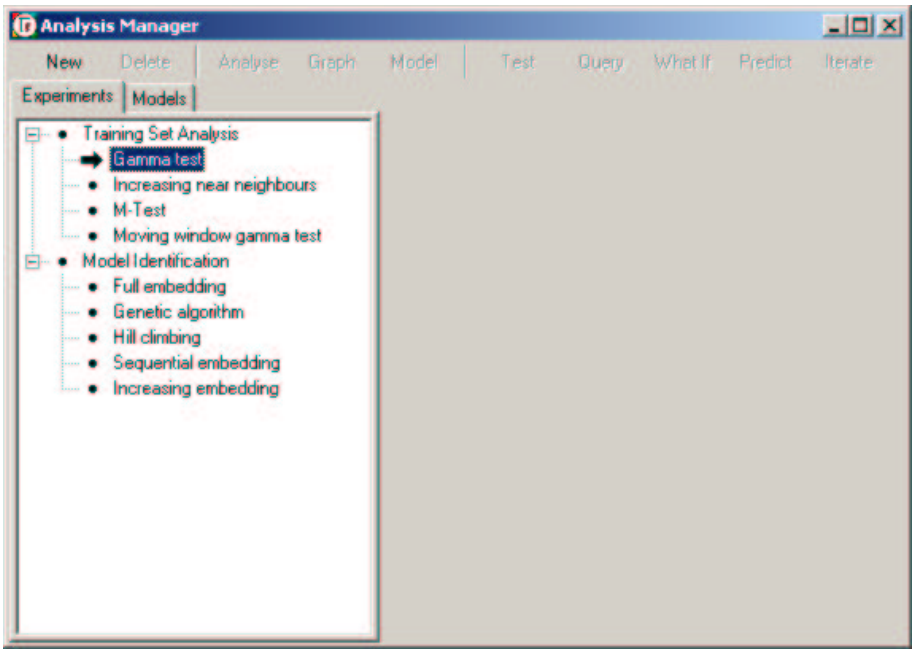


Figure D.8: The analysis manager window shows the Gamma test experiment types.

D.3.1 Experiment types

There are nine standard Gamma test experiments included within *winGamma*. These are divided into two distinct groups: *data set analysis* and *model identification*. The data set analysis experiments are (1) the Gamma test, (2) increasing near neighbour test, (3) M-test, and (4) the moving window Gamma test. These experiments perform basic tests on the data set to estimate, for example, whether there is sufficient data, or how sensitive a solution is to the number of near neighbours.

The model identification experiments perform feature selection or find the optimal embedding dimension. These techniques are (1) full search, (2) genetic algorithm search, (3) hill climbing, (4) sequential search, and (5) increasing embedding.

It is important that a record of past experiments is maintained for each data set to enable the user to easily compare results from many experiments. The tree structure listing the experiment types was developed to do this. The experimental results are designed to ‘hang’ under the appropriate

experiment type as shown in Figure D.9. This approach arose from the desire to maintain a project containing analysis and modelling experiments for a particular data set.

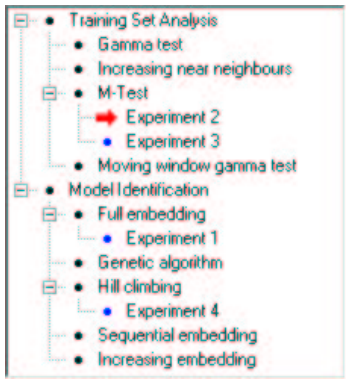


Figure D.9: The tree structure lists the available experiment types and records the results of experiments.

An experiment is created by selecting the appropriate experiment type from the tree structure and pressing *new* on the button bar at the top of the analysis manager, as shown in Figure D.10.

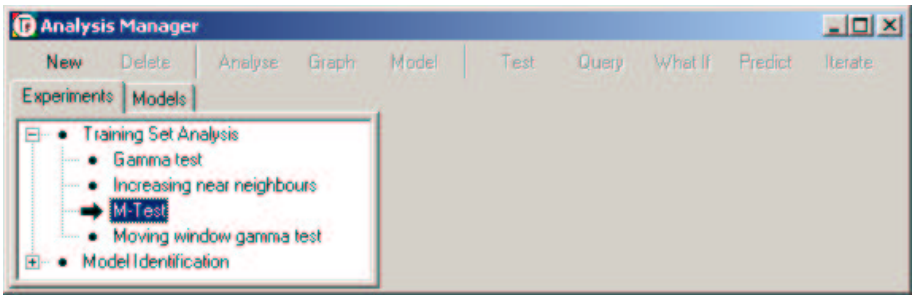


Figure D.10: A new experiment is created by selecting the experiment type from the tree (as indicated by the arrow) and pressing the *new* button on the button bar.

Once the experiment is complete, the results are recorded in the tree under the appropriate experiment type, as shown in Figure D.11.

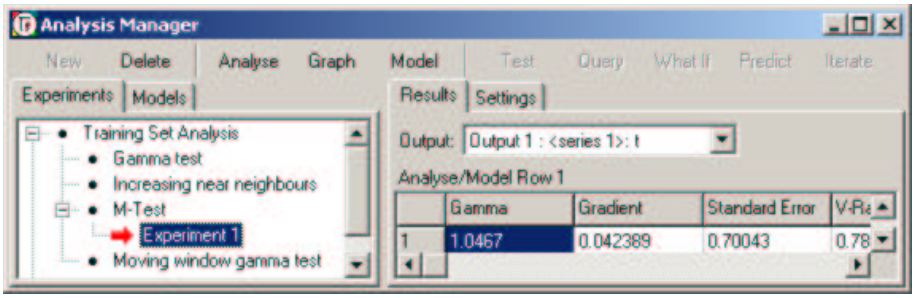


Figure D.11: The results for an experiment are stored in the tree under the appropriate experiment type. The results for the selected experiment are shown in the right pane.

D.3.2 Experiment options

We have briefly discussed how to create an experiment: the user selects the experiment type from the *analysis manager* window and clicks *new* on the button bar. However, before the results can be computed, certain parameters must be set for the particular experiment. The parameters that must be specified for each of the nine experiments are shown in Table D.2 (indicated by ○).

	Experiment type								
Interface	**T1**	**T2**	**T3**	**T4**	**M1**	**M2**	**M3**	**M4**	**M5**
Near neighbours	○		○	○	○	○	○	○	○
Mask	○	○	○	○			○	○	
Number of results					○		○		
Histogram size					○	○			
Evaluated output						○	○	○	
Additional input		○	○	○		○			

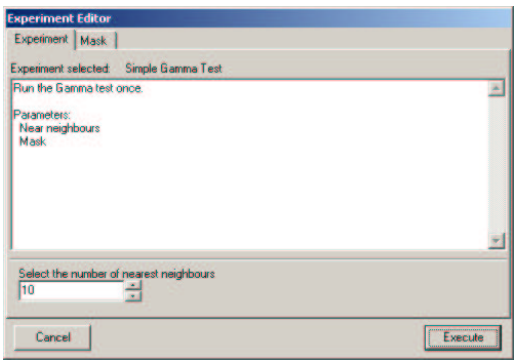
Code	Experiment
T1	Gamma test
T2	Increasing near neighbour test
T3	M-test
T4	Moving window test

Code	Experiment
M1	Full search
M2	Genetic algorithm search
M3	Hill climbing search
M4	Sequential search
M5	Increasing embedding

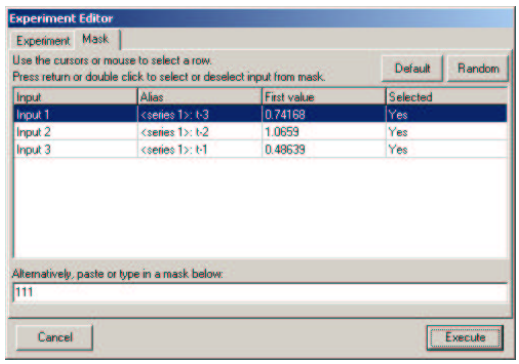
Table D.2: The highlighted parameters (○) must be specified for the Gamma test experiments. Note that additional input is required for some experiments beyond the generic parameters tabulated here.

We have provided a description of each experiment type within the *winGamma* interface to remind the user of the purpose of the selected experiment. Generic interfaces have been created to reduce the number of specific interface components. These are shown in Figure D.12. Several unique interfaces were developed for experiments requiring *additional input* (shown in Table D.2) and these are shown in Figure D.13.

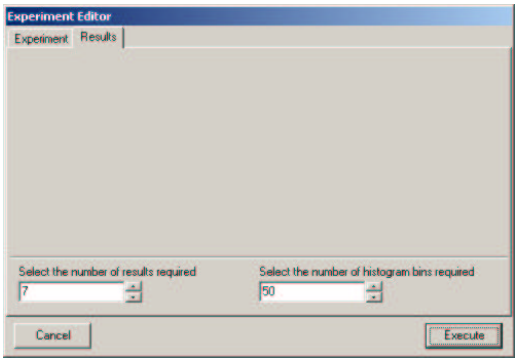
The required interfaces (as shown in Table D.2) for each experiment type are combined into a tabbed display and displayed within a single dialog box. The user then has the option to set any of the parameters or *execute* the experiment immediately using the default values.



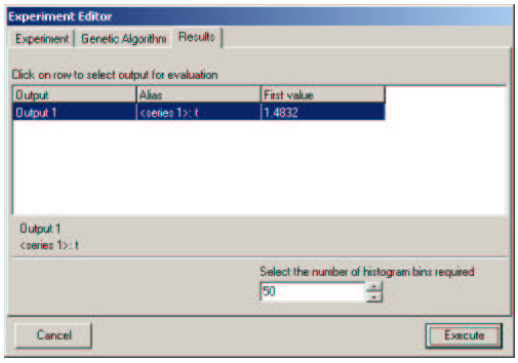
(a) The initial page provides an experiment description and provision to specify the number of near neighbours, which is hidden if not required.



(b) The mask dialog allows an input mask to be specified. A mask can be randomly generated or set by either typing a mask in or by toggling the inputs on or off using the mask grid.

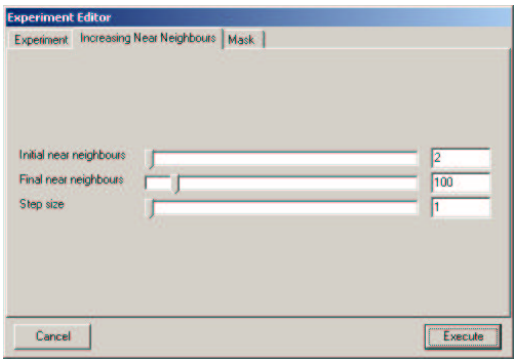


(c) The results dialog allows the total number of experimental results returned to be limited. The number of *buckets* required for the Gamma histogram can also be set. This is the same dialog as shown in Figure D.12(d) with the output selector hidden.

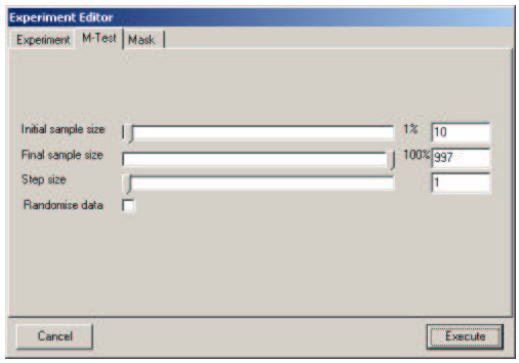


(d) Some heuristic experiments are evaluated for a specific output (e.g. the GA search). This dialog allows the evaluated output to be specified. The user can also set the number of *buckets* required for the Gamma histogram. This is the same dialog as shown in Figure D.12(c) with the irrelevant options hidden.

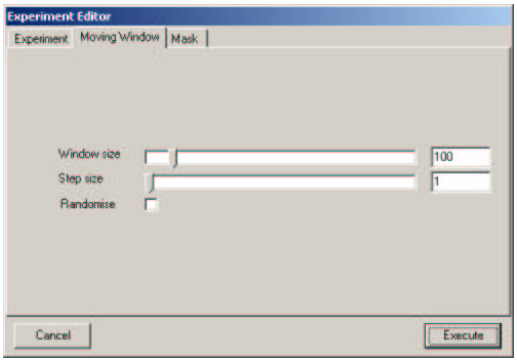
Figure D.12: Generic dialog boxes used for Gamma test experiments.



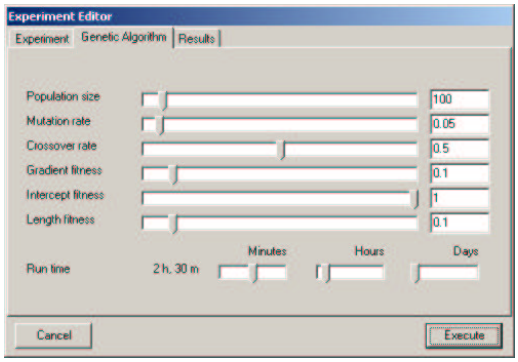
(a) The increasing near neighbours test performs a Gamma test for a range of near neighbours. The dialog allows the user to select the range of near neighbours and the step size to take between successive near neighbour values.



(b) The M-test requires the user to specify the range of data points to be used from the smallest data set size to a maximum size. Gamma tests are then performed for increasingly large sample sizes within the specified range, incremented according to the step size. The order of the data can be randomised and the results averaged over several M-tests (not shown).



(c) The moving window Gamma test performs a Gamma test within a set window size (number of data points). The window size is kept at a constant size and moved through the data set in fixed steps determined by the step size. The order of the data can be randomised and the results averaged over several moving window tests (not shown).



(d) The genetic algorithm search has many options to tune the performance of the algorithm including the maximum run-time.

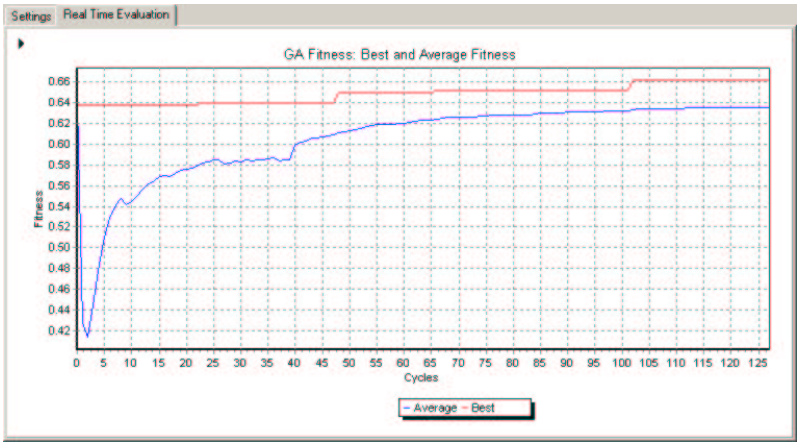
Figure D.13: Specific dialog boxes used for Gamma test experiments.

D.3.3 Experiment execution

Once a user has started an experiment the application prevents the user running any other experiments until the current one completes or is terminated. This is handled through the thread management routines developed during the prototyping stage.

Each experiment provides feedback to the user during execution. The form of reporting varies between experiments and data sets as required. By default the nearest neighbour algorithm reports via a progress meter which, if the data set is small, is automatically turned off to reduce the processing overhead[2]. In particular the heuristic experiments provide a lot of feedback to enable the user to examine how the experiment is running. Figure D.14 shows two examples of reporting.

(a) The status bar provides the primary source of feedback during the experiment. The first panel on the status bar describes the overall progress of the current experiment. The second and third panels describe the current operation and display its progress using a progress meter.



(b) The genetic algorithm performs an optimisation that can be measured in terms of the overall population fitness and best individual solution fitness. This chart is produced in real-time to provide continuous feedback during the experiment. This feedback can be used to determine when to stop the algorithm, for example when there is convergence between the best individual fitness and the overall population fitness.

Figure D.14: In-experiment feedback.

[2]Presenting visual information is computationally intensive and is only used where it consists only a small fraction of the computation time or is of relevance.

D.3.4 Experiment results

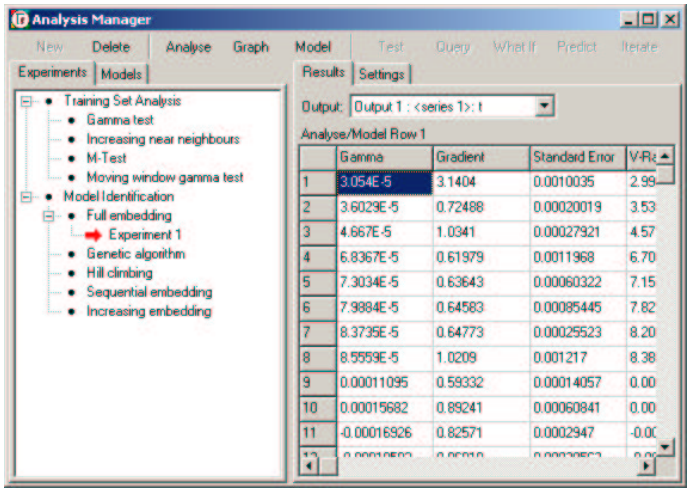
All of the Gamma test experiments return the results in the same structure as shown in Table D.3.

Result components
Gamma statistic, Γ
Gradient, A
Standard error of the regression line fit, SE
Noise variance to signal variance ratio, $V\text{-}ratio = \Gamma/\text{var}(y)$
Number of near neighbours, p_{max}
Start vector
Number of vectors
Evaluated output
Number of zero nearest neighbours
Lower 95% confidence on noise estimate using the zero nearest neighbour samples
Upper 95% confidence on noise estimate using the zero nearest neighbour samples
Mask

Table D.3: The results structure.

Where a data file contains multiple instances of the same input vector, together with non-identical corresponding outputs, these zeroth nearest neighbours might be construed as measurements of the same output variable with an identical input vector. In these, perhaps unusual, circumstances we have the opportunity to compute the variance of the noise directly, i.e. without recourse to the Gamma test. In this case we compute the variance of the output corresponding to identical input vectors, assume a normal distribution and, in addition to the Gamma test result, return estimates for the noise variance at the Upper and Lower 95% confidence level as calculated from Student's T-test. Student's T-test is used because under most normal circumstances there will not be very many zeroth near neighbours. These Upper and Lower estimates for the noise variance can then be compared with the Gamma statistic returned by the Gamma test. If all three are in close agreement this is strong evidence that the Gamma statistic is estimating the noise variance accurately.

The reporting of results is shown in Figure D.15. Figure D.15(a) shows a completed experiment listed in the left pane of the analysis manager window with the corresponding results for the experiment shown in the right pane. The method of displaying the results in the analysis manager is shown in Figures D.15(b)-(d).



(a) The analysis manager displays the completed experiment in the left pane and shows the corresponding results in the right pane.

	Gamma	Gradient	Standard Error	V-Ratio	Near Neighbour	Start Vector	U
1	3.054E-5	3.1404	0.0010035	2.9932E-5	10	1	9
2	3.6029E-5	0.72488	0.00020019	3.5311E-5	10	1	9
3	4.667E-5	1.0341	0.00027921	4.5738E-5	10	1	9
4	6.8367E-5	0.61979	0.0011968	6.7005E-5	10	1	9
5	7.3034E-5	0.63643	0.00060322	7.1578E-5	10	1	9
6	7.9884E-5	0.64583	0.00085445	7.8292E-5	10	1	9
7	8.3735E-5	0.64773	0.00025523	8.2066E-5	10	1	9

(b) The left hand side of the results grid.

	Near Neighbour	Start Vector	Unique Points	Evaluated Outp	Zero Nearest Ne	Upper 95% Conf	Lo
1	10	1	994	1	0	-	-
2	10	1	994	1	0	-	-
3	10	1	994	1	0	-	-
4	10	1	994	1	0	-	-
5	10	1	994	1	0	-	-
6	10	1	994	1	0	-	-
7	10	1	994	1	0	-	-

(c) The centre of the results grid.

	Zero Nearest Ne	Upper 95% Conf	Lower 95% Conf	1	2	3	4	5	6
1	0	-	-	0	0	1	0	1	0
2	0	-	-	0	0	1	0	1	1
3	0	-	-	0	0	0	1	0	1
4	0	-	-	1	0	0	0	1	1
5	0	-	-	0	1	0	0	1	1
6	0	-	-	1	1	1	1	1	1
7	0	-	-	0	0	1	1	1	1

(d) The right hand side of the results grid.

Figure D.15: The results reporting format is illustrated using some example results. (b)-(d) show the statistics recorded in the results table for each Gamma test.

D.3.5 Experiment analysis

Several standard types of analysis have been described within this thesis and it was appropriate to include them in *winGamma* where possible. The analysis can be divided into two main types: (1) analysis of a single result, and (2) the analysis of all the results generated by an experiment.

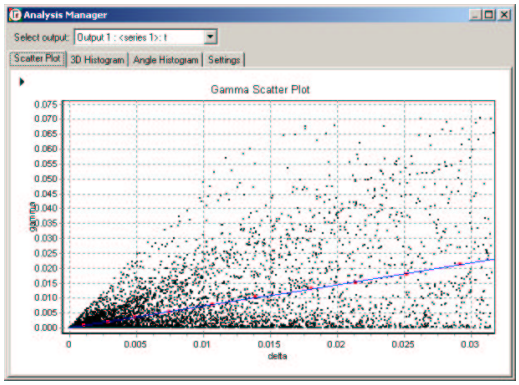
Figure D.16 shows the type of analysis possible for an individual result. The Gamma scatter plot is shown in Figure D.16(a) with an interpretation in 3-dimensions in Figure D.16(b). The Gamma angle histogram shown in Figure D.16(c) is an analysis not previously discussed. The histogram is produced by counting the number of points in the scatter plot at a particular angle from the intercept. This can be used as a guide to determine whether there are any points in the scatter plot in the crucial region at low δ and high γ where *noise* can be most evident (at an angle approximately $> 80°$). The final screenshot in Figure D.16(d) shows a summary of the result being analysed.

The analysis of all results is performed within the *results visualiser*, shown in Figure D.17, and is only available for an experiment that contains more than one result. The visualiser can plot any of the available statistics for all results as demonstrated in Figure D.17(a). Initially the visualiser automatically determines which statistics should be plotted for each experiment type, for example the analysis of an increasing embedding plots the lags against the Gamma statistic to show what the optimal embedding dimension should be. Figure D.17(b) shows the analysis available for feature selection using full and GA searches.

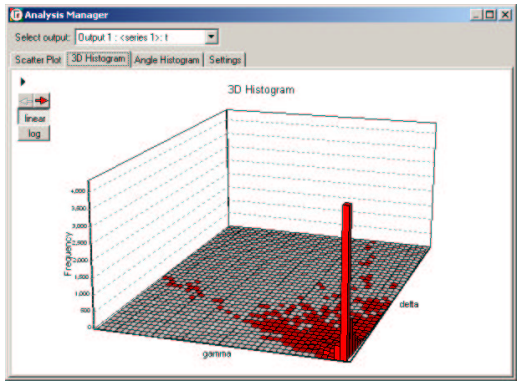
D.4 Model building

A model can be constructed in *winGamma* only after a Gamma test analysis has been completed. This is to ensure that the user does not attempt to blindly model the data, but instead uses any insights gained from the data to improve the model performance. A model is created from a single result generated from any Gamma test experiment.

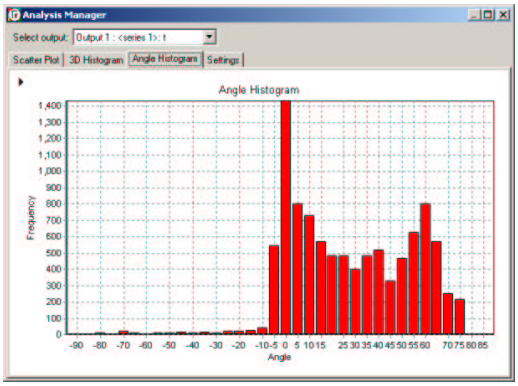
The result selected as the basis for modelling contributes several parameters to the design of the model. The primary factors for modelling are the Gamma statistic, which determines optimal model performance, and the mask, which provides the best combination of input variables. Secondary consideration must be given to the gradient, which provides an indication of the required model complexity and roughly determines how many nodes are required in a neural network model. The way in which the data was scaled during the analysis will also affect how the model



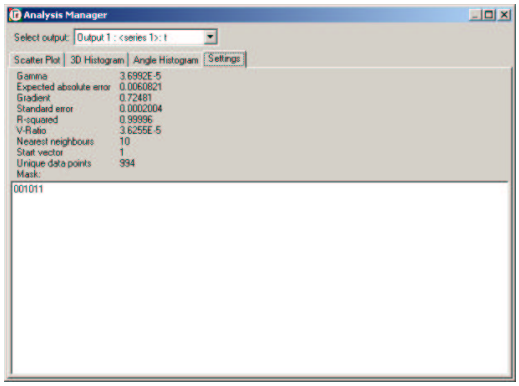
(a) The Gamma scatter plot is interactive allowing the user to zoom in for a closer look at different regions of the graph.



(b) The 3-dimensional Gamma scatter plot provides an indication of the distribution of points in the scatter histogram. The graph can be rotated and plotted with a linear or logarithmic scale.

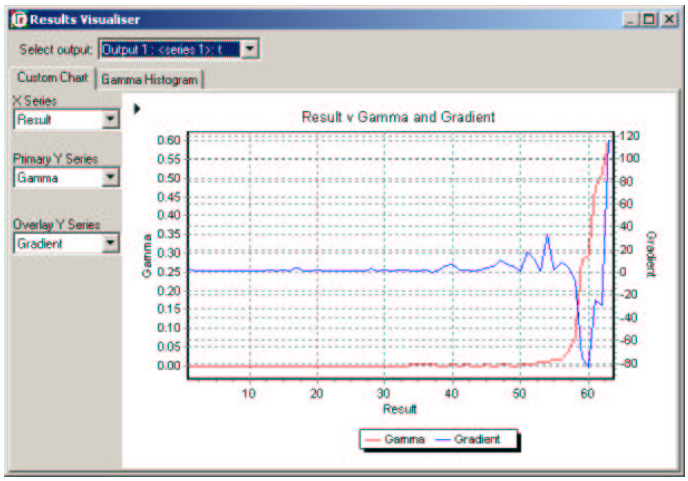


(c) The angle histogram summarises where the points lie in the Gamma scatter plot in relation to the intercept at $(0, \Gamma)$. A positive angle indicates points that lie above the intercept.

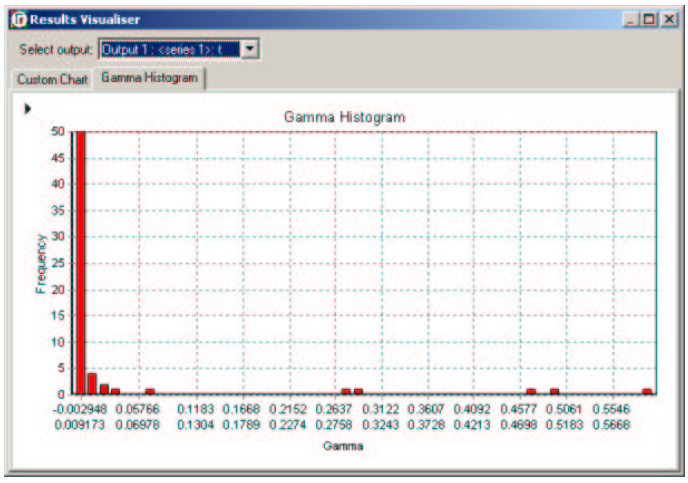


(d) The experiment result is summarised for reference.

Figure D.16: An individual Gamma test result can be analysed using variations of the Gamma scatter plot.



(a) Any combination of statistics can be plotted for a single experiment that contains more than one result. This example shows results from a full embedding plotted in ascending order of Gamma value.



(b) The results from the feature selection experiments are used to construct a Gamma histogram.

Figure D.17: All of the results for a single experiment can be analysed together. The *results visualiser* can graph any of the available statistics and provides facilities to copy, print or save the chart data.

will perform.

Figure D.15(a) illustrates the method by which a model is constructed from experimental results. Firstly, the experiment is selected from the tree structure in the left hand pane of the analysis manager. *winGamma* shows all of the results for the selected experiment in the right hand pane, where the result that forms the basis of the model design is selected. The modelling process is started by selecting the *model* button on the analysis manager toolbar.

The proportion of analysis data to be used for creating the model needs to be selected, as shown in Figure D.18. By default the proportion of data used for modelling is the same as was used to generate the experimental result. This default behaviour ensures that the Gamma statistic, to be used as the target MSE, is passed to the modelling routines. However, if a different proportion of data is selected then there is an option to recalculate the Gamma statistic to give a more appropriate estimate of the target MSE for the data. Once the proportion of data has been selected *winGamma* displays the model editor.

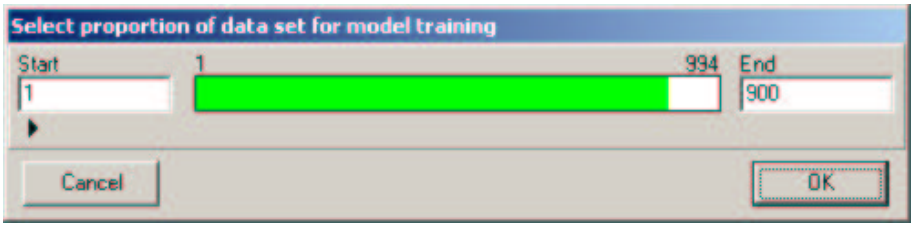


Figure D.18: The user selects how much data is used to create the model. The default is to use the same data as was used for the analysis.

D.4.1 Model types

winGamma can be used to construct neural network and local-linear regression models (technical details of these model types can be found in Chapter 2). The selection of model type is made using the *modelling editor*. Figure D.19 shows the editor and the choice of models available.

D.4.2 Model options

Table D.4 describes the parameters that must be set for each type of model.

The corresponding interface dialogs which enable the user to supply input parameters are shown in Figure D.20. In cases where there is a duplicate interface, for example linear regression and

	Model type				
Interface	**L1**	**L2**	**N1**	**N2**	**N3**
Near neighbours	○	○			
Regression constant	○	○			
Local flow threshold	○	○			
Number of nodes in first layer			○	○	○
Number of nodes in second layer			○	○	○
Target MSE			○	○	○
Learning rate			○		
Momentum			○		
Regularisation			○		
Initialisation time			○		
Training time			○		

Code	**Model**
L1	Local-linear regression
L2	Dynamic local-linear regression

Code	**Model**
N1	Backpropagation trained network
N2	Conjugate gradient trained network
N3	BFGS trained network

Table D.4: The highlighted parameters (○) must be specified for each model type.

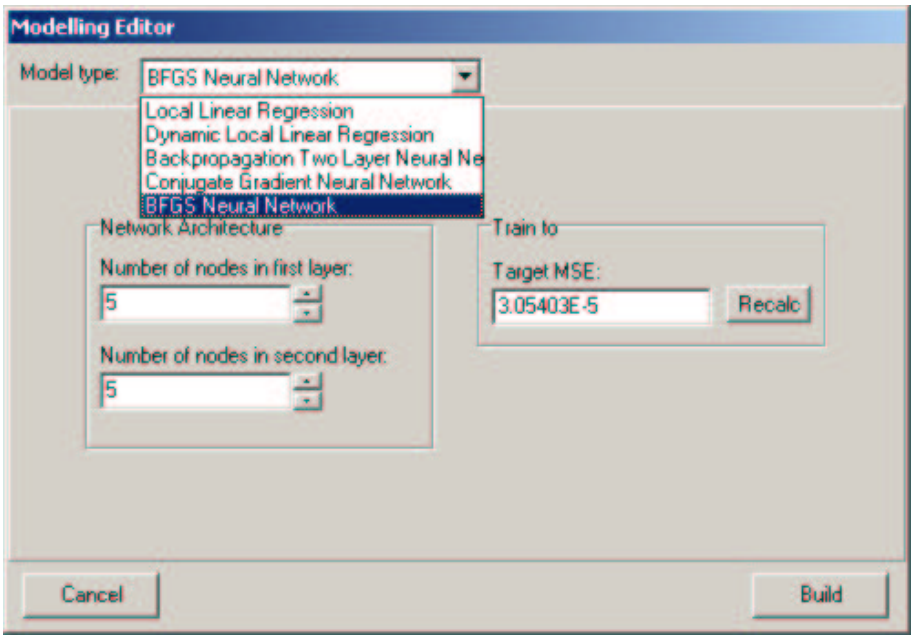


Figure D.19: The modelling editor provides a choice of models to the user. The model type is selected from the pull-down menu. There are two main model types (neural networks and local-linear regression) with several variants of each. The editor changes appearance depending on the parameters required for the chosen model type.

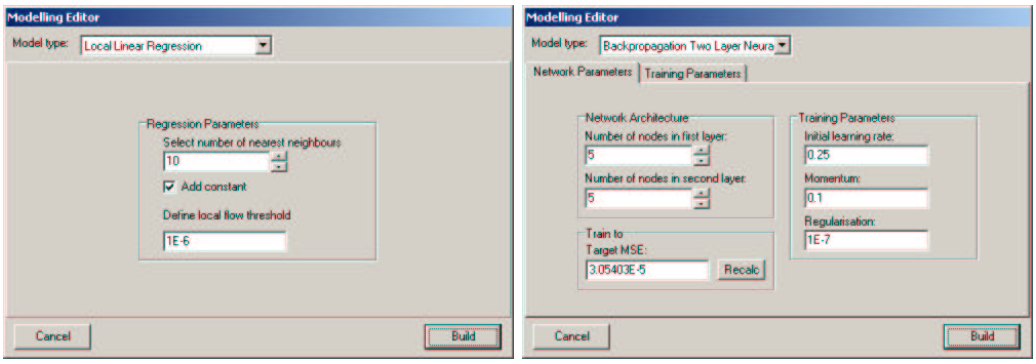
dynamic local-linear regression have identical parameters, only one example is shown.

The neural network dialog boxes allow the user to independently set the target MSE. By default the target MSE is set to the Gamma statistic of the experiment result used as a basis for the model. If the target MSE is changed, or the proportion of data selected for modelling was not the same as was used for the analysis, then it can be recalculated.

D.4.3 Model training

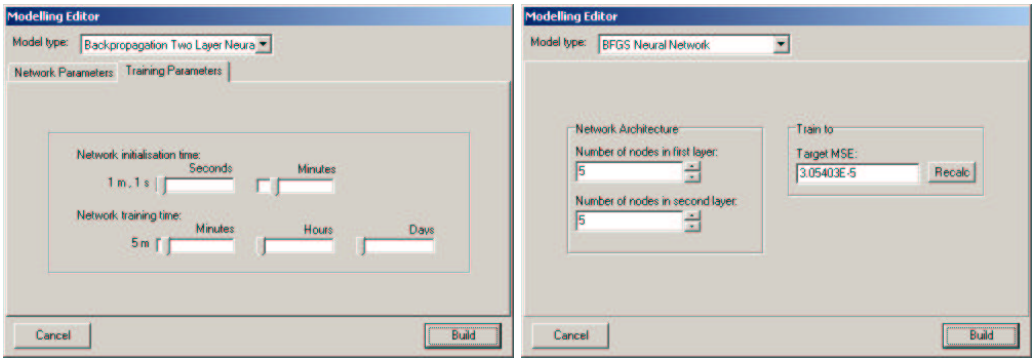
The local-linear regression models (at their most basic level) consist of only data organised as a k-d tree, and subsequently are trivial to produce. In contrast, a neural network model requires training time to learn from the data.

To provide feedback to the user during training, the neural network training algorithms produce a real-time chart showing the progress of the training measured in terms of the model MSE. Also shown on the chart is the target MSE because that is the model performance goal that determines when the training algorithm will terminate.



(a) The dialog box for local-linear regression and dynamic local-linear regression.

(b) The dialog box for backpropogration showing the network parameters.

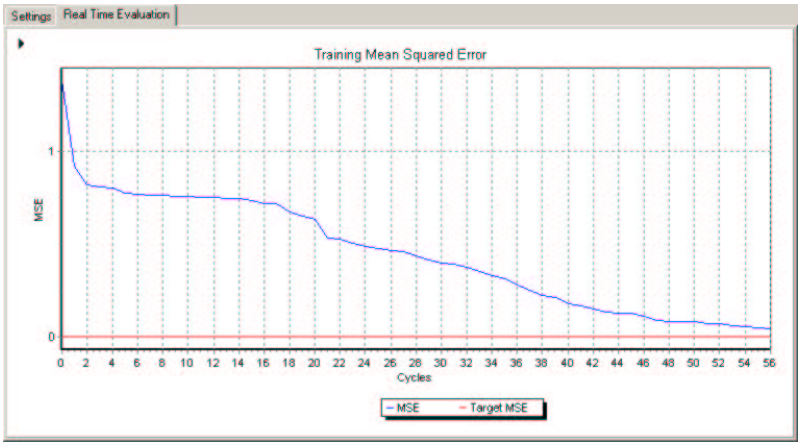


(c) The dialog box for backpropogration showing the training time parameters.

(d) The conjugate gradient and BFGS dialog box.

Figure D.20: The dialog boxes for setting modelling options.

Figure D.21 shows the feedback given during a typical training run. In addition to the real-time chart, feedback is also provided on the status bar. This gives numerical information regarding the training performance that can seen by the user even when they are using other features of *winGamma* which may obscure the real-time chart.



(a) The model performance is charted in real-time during network training. The model MSE is shown in comparison to the target MSE (in most cases this will be the Gamma statistic measured on the training data).

(b) The status bar provides numerical feedback on the training process. The progress meter shows the progress of the current training operation (in *winGamma* the primary network training operation is punctuated at regular intervals with a simulated annealing routine to avoid the confinement of the network weights to a local minima.).

Figure D.21: In-training feedback.

Once model construction has been completed, either because the training algorithm completed or was terminated early by the user, then the model is added to the record of models held by the analysis manager. The models are viewed using essentially the same tree structure used to hold experimental results, although the models cannot be constructed directly in the same way that experiments can. The constructed model is 'hung' under the appropriate model type and automatically numbered to keep a record of the order in which the models were created. Figure D.22 shows an example of a trained model in the analysis manager window.

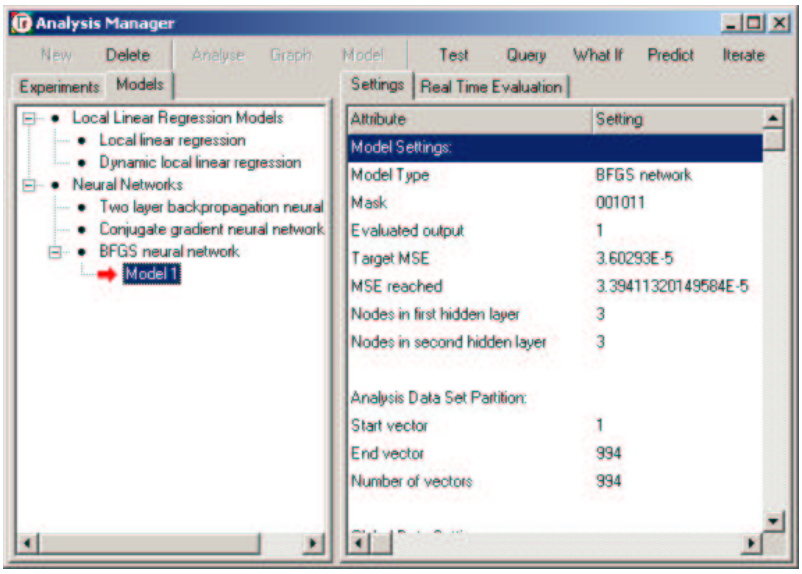


Figure D.22: Constructed models are shown on the analysis manager window. The models page has been selected in the left hand pane. The constructed models are shown in the tree structure in the corresponding model type branch. The buttons on the button bar (test, query, what if, predict and iterate) are activated for modelling.

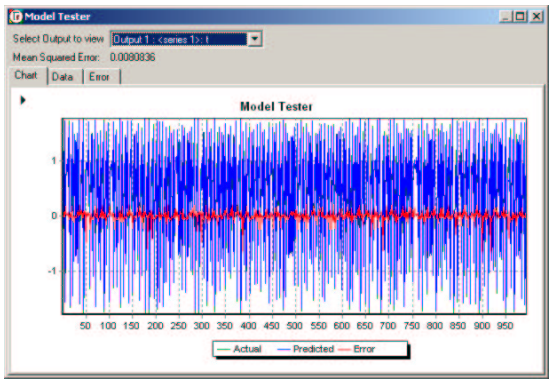
D.4.4 Model testing

All of the model types produce the same format of output when tested. The main difference between the models is that the neural network models require a feedforward calculation to compute the output, whereas the local-linear regression models require a more complicated (and hence more time consuming) computation involving finding the p_{max} nearest neighbours and then performing a singular value decomposition to obtain the least squares fit. Hence, local-linear regression models take much longer to test.

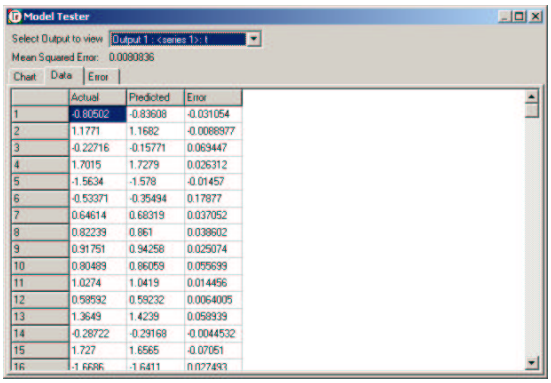
The form of the output generated through testing a model is shown in Figure D.23. The standard output is a graph showing actual output versus model output, an example of which is shown in Figure D.23(a). As shown in Figure D.23(b), the corresponding values can also be displayed and exported. Figure D.23(c) shows the error distribution plot as produced by the model for a particular test data set.

D.4.5 Model querying

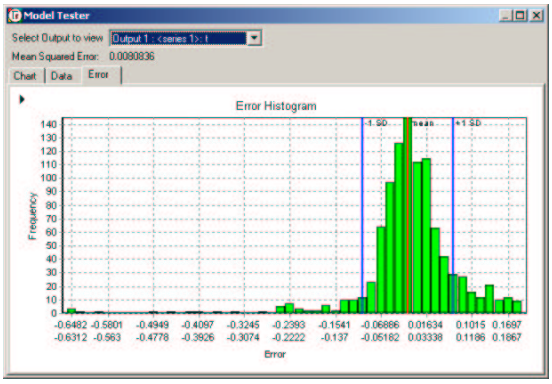
When a small number of queries are required of a model it may be more efficient to run a query as opposed to a model test or prediction. The query model routine was designed to handle these



(a) The graphical representation of the model performance, shown by the blue line, compared to the actual data observation (obscured), shown by a green line. The error is shown by the red line and the overall MSE value is shown above the chart.



(b) These are the data values corresponding to those plotted on the chart in Figure D.23(a).



(c) The error histogram shows the error distribution given by the model for the particular test set.

Figure D.23: The model testing output is represented in three ways.

situations. The user types in the inputs to the model and the corresponding output is calculated.

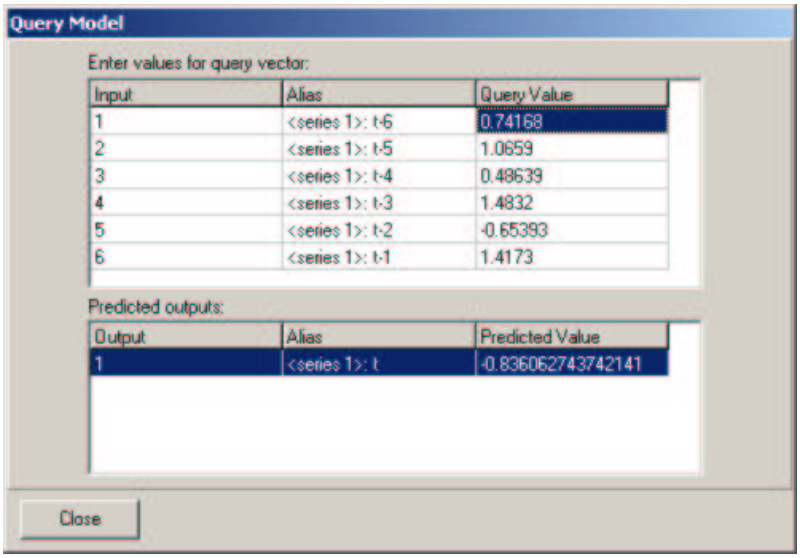


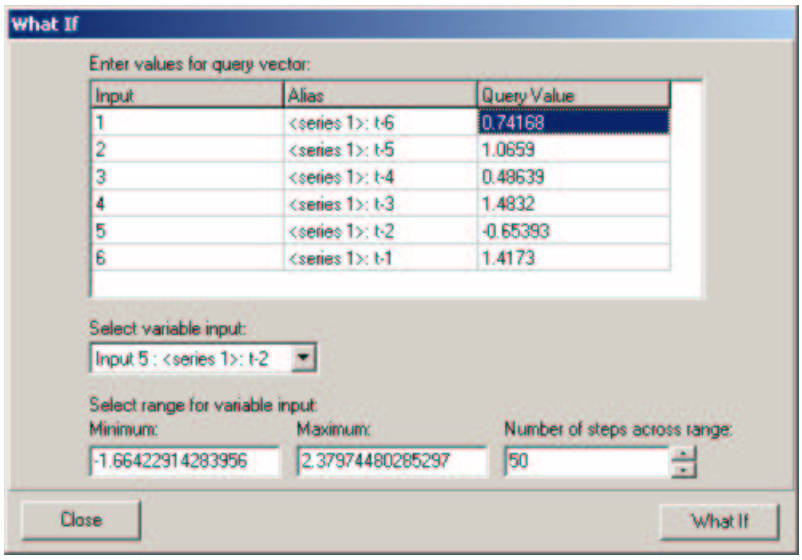
Figure D.24: The query model interface allows the user to enter the inputs to the model from which the output is calculated and displayed.

D.4.6 Model what-if (scenarios)

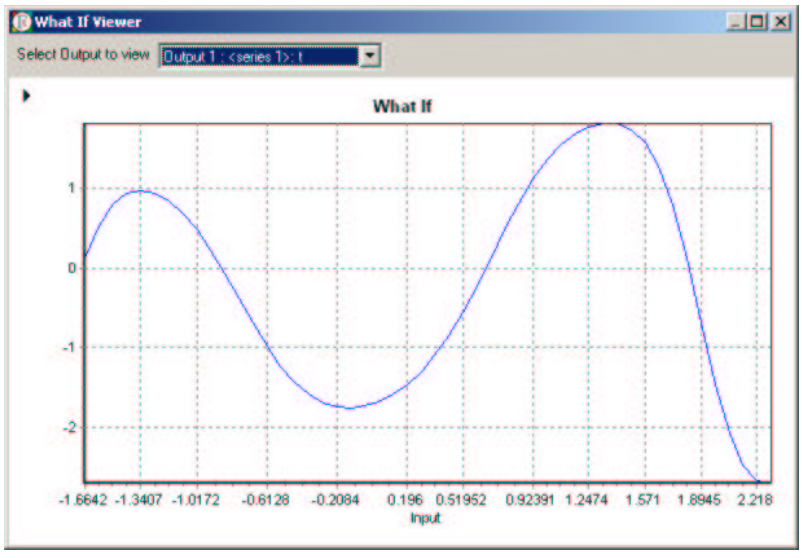
The what-if model query is a more advanced form of a model query. The user enters the inputs to the model, but instead of returning just a single value for the output in the way that a query works, the model performs a series of queries across a range of values for a specified input. This allows the user to see how a model responds to a change in stimulus on a given input. Figure D.25 shows the what-if dialog and the resulting graphical output.

D.4.7 Model predict

The prediction routine is similar to the test routine, but with one crucial difference. The prediction routine works when the output is unknown. This can be a useful technique for testing scenarios or performing a batch of queries. The model cannot provide an estimate of the prediction accuracy because a reference is unavailable.



(a) The what-if dialog requires the user to specify the point at which the what-if query takes place. The input to be varied and the range over which it is to be varied must also be specified by the user.



(b) The what-if query produces a chart that displays how the model varies with respect to the changing input stimulus.

Figure D.25: The what-if query allows the user to examine how a model responds to the change in stimulus on a single input.

Figure D.26: The predict routine is used when the output for a particular data set is unknown. A judgement of the prediction quality remains with the modeller because the prediction routine works without comparison to a known output.

D.4.8 Model iteration

An artificial system can often be successfully modelled using a n-step ahead model, where n is low. An extension to this form of modelling is to iterate a n-step ahead model a number of times to predict the outcome.

The model iteration routine works by first testing the model on a fixed number of points to provide a starting point for the iteration. The iteration routine then takes over and takes successive predictions of the output to generate the new inputs. If the model is successful then iterations will produce solutions similar to the expected values.

D.5 Application information

An on-line help system has been included with *winGamma* to provide assistance. There is an electronic user manual created using the Microsoft HTML help system, an example is shown in Figure D.28.

In addition to the on-line help, we have implemented an *about* dialog that displays the version number and licensing information specific to the user. This also provides contact e-mail addresses and the provides the address of the *winGamma* website. A second dialog box describes the *copyright* information. This also provides a disclaimer regarding the usage of the software to protect

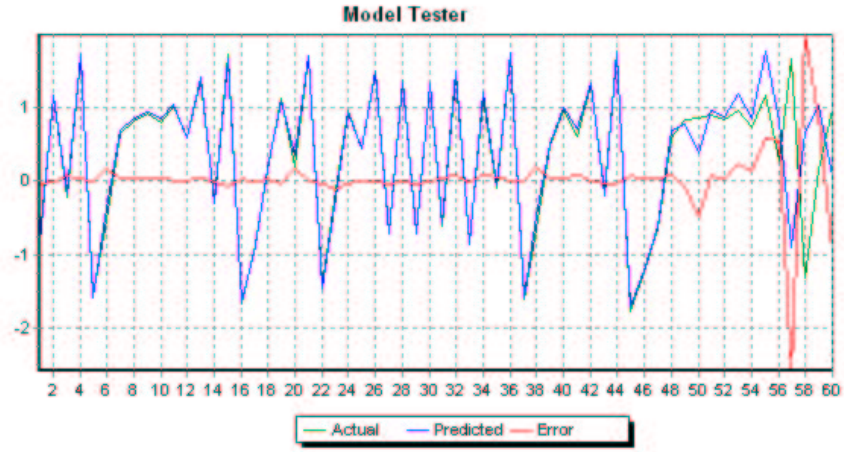


Figure D.27: Model iteration tests the model for a number of points then iterates forward for a specified number of time steps. In this example, the model was tested with 50 points followed by an iteration to generate the next 10 points. The blue line shows the test involving first the 50 points and then the 10 iterated points. The green line shows the actual output and the red line shows the error.

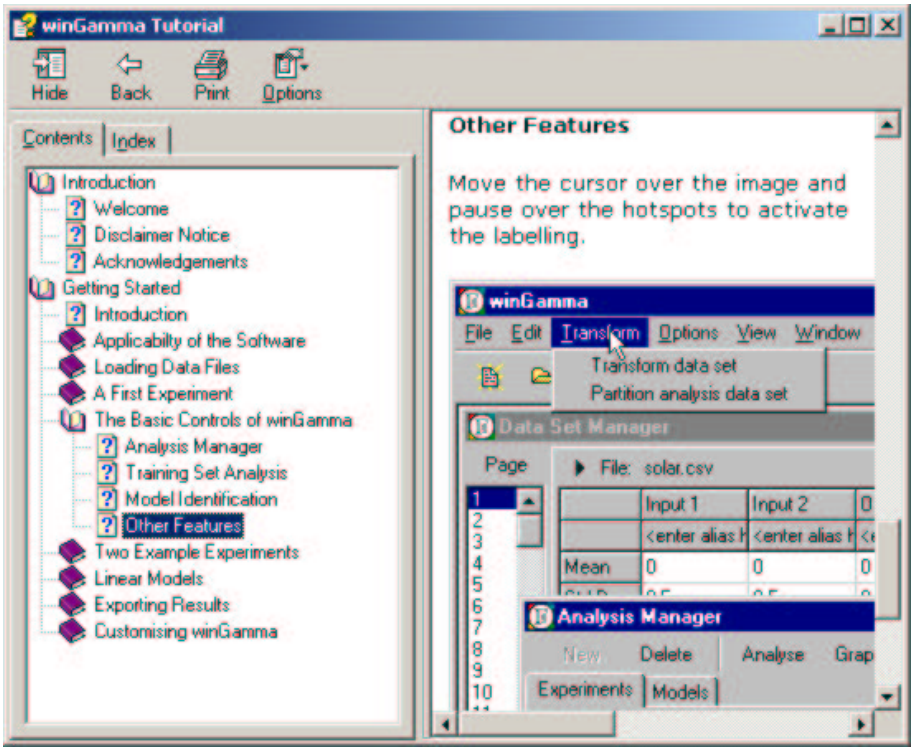
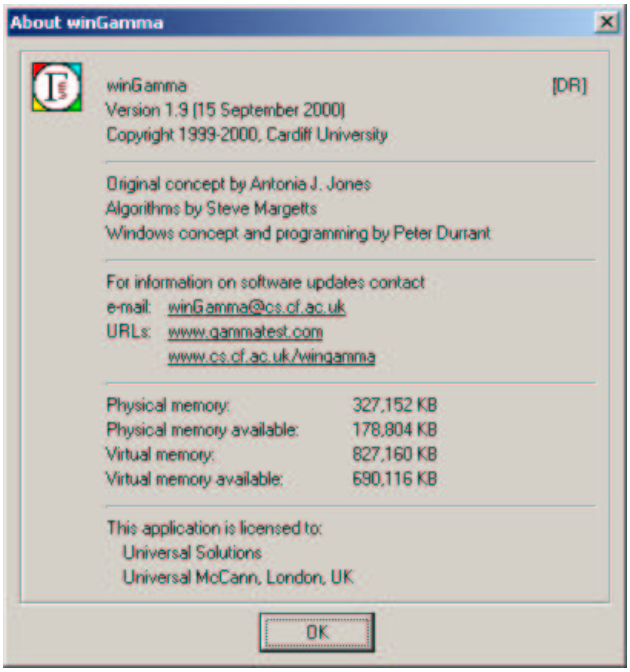
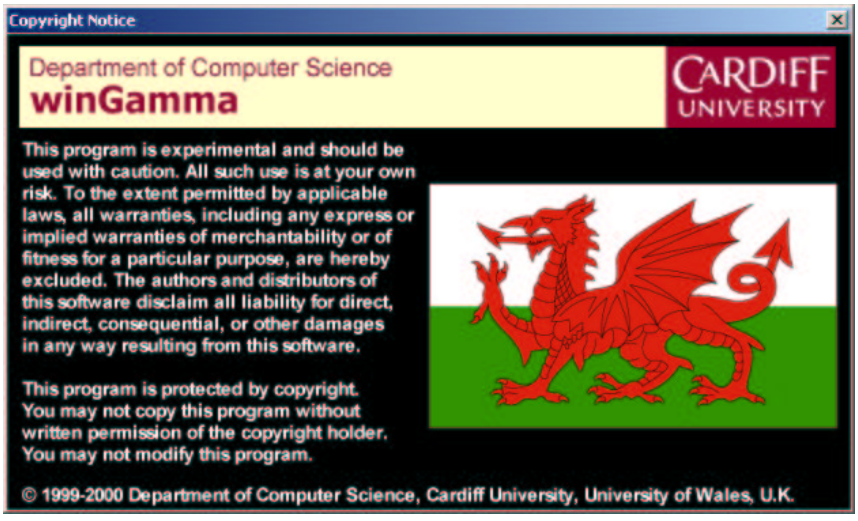


Figure D.28: The help system provides information about how to use *winGamma* and how to interpret the results.

the authors from any legal action that may arise from events caused through the use of the software. Figure D.29 shows these dialog boxes.



(a) The about dialog provides application information, including e-mail and internet addresses for contact and real-time system monitoring of memory (useful to examine system performance when analysing large data sets).



(b) The copyright dialog box exists to detail which organisation owns the copyright and to give notice that the authors accept no liability for any losses incurred during the use of the software.

Figure D.29: *winGamma* has the standard *about* and *copyright* dialog boxes to give information about the *winGamma* application.

APPENDIX E

winGamma Data and File Formats

This section provides supplementary information about *winGamma*.

E.1 File structures

All file formats use the ASCII representation. This enables files to be generated and read any application that supports ASCII. Examples include text editors, spreadsheets, databases and web repositories.

E.1.1 winGamma format (asc)

The file format native to the original Gamma test components is the asc format. This is an ASCII file format that describes either time series or input-output format data files.

Time series format

A single time series consists of a single column of numbers. A multiple time series file consists of several columns of numbers separated by white-space[1]. Each row in the data set corresponds to the next set of readings in the time series and is terminated by a comma followed by a new-line

[1] White-space consists of space and tab characters but not new-line characters.

character. A multiple time series file is shown below.

$$1.2\ 5.4\ 0.1\ 7.3,$$
$$1.4\ 5.7\ 0.0\ 6.4,$$
$$1.6\ 6.2\ 0.3\ 6.0,$$

Vector format

Vector (input-output) data is a natural format for the Gamma test and neural network supervised training algorithms.

Each row in the file describes an input vector and the corresponding output vector. The input and output vectors are separated by a comma. The values in each vector are separated by white-space. An example of an vector format file is shown below. Each vector can be optionally terminated with a comma (not shown).

$$1.2\ 5.4\ 0.1,\ 7.3$$
$$1.4\ 5.7\ 0.0,\ 6.4$$
$$1.6\ 6.2\ 0.3,\ 6.0$$

E.1.2 Comma-separated file format (csv)

The csv file format is an industry standard file format. It is commonly used with spreadsheet and database applications as a way of exporting and importing data in a platform independent manner.

The file consists of a number of rows and a fixed number of columns. Each value is separated from the next with a comma. The end of a row is marked with a new-line character instead of a comma. An example is shown below.

$$1.2,\ 5.4,\ 0.1,\ 7.3$$
$$1.4,\ 5.7,\ 0.0,\ 6.4$$
$$1.6,\ 6.2,\ 0.3,\ 6.0$$

winGamma can also handle csv files that have each row terminated with a comma.

E.2 Data formats

There are two types of data format:

1. Vector format data.

2. Time series data.

All data must be in vector format for the Gamma test and modelling algorithms. Time series data must be converted to this input-output form.

E.2.1 Vector file format

The input-output format consists of M pairs of vectors. Each vector pair consists of an input vector $\mathbf{x}$ (size n) and an output vector $\mathbf{y}$ (size m).

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2} & x_{M3} & \cdots & x_{Mn} \end{pmatrix} \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{M1} & y_{M2} & \cdots & y_{Mm} \end{pmatrix}$$

E.2.2 Time series file format

Time series data x consists of m variables sampled at regular time intervals t for T time steps.

$$\begin{pmatrix} x_{1,t=1} & x_{2,t=1} & x_{3,t=1} & \cdots & x_{m,t=1} \\ x_{1,t=2} & x_{2,t=2} & x_{3,t=2} & \cdots & x_{m,t=2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{1,t=T} & x_{2,t=T} & x_{3,t=T} & \cdots & x_{m,t=T} \end{pmatrix}$$

Time series data must be transformed into vector format (see Section E.2.1) using a window of past measurements to generate the input vectors and window of future measurements to generate the output vectors. The concept is illustrated below for a single time series transformed to vector format.

$$\begin{pmatrix} x_{t=1} \\ x_{t=2} \\ \vdots \\ x_{t=T} \end{pmatrix} \rightarrow \begin{pmatrix} x_{t=i-3} & x_{t=i-2} & x_{t=i-1} \\ x_{t=(i+1)-3} & x_{t=(i+1)-2} & x_{t=(i+1)-1} \\ \vdots & \vdots & \vdots \\ x_{t=N-3} & x_{t=N-2} & x_{t=N-1} \end{pmatrix} \begin{pmatrix} x_{t=i} & x_{t=i+1} \\ x_{t=(i+1)} & x_{t=(i+1)+1} \\ \vdots & \vdots \\ x_{t=N} & x_{t=N+1} \end{pmatrix}$$

where $i = steps\_back + 1$ and $N = T - steps\_ahead + 1$ and the number of vectors is $T - (steps\_back + steps\_ahead) + 1$.

Bibliography

[Aðalbjörn Stefánsson et al., 1997] Aðalbjörn Stefánsson, Končar, N., and Jones, A. J. (1997). A note on the gamma test. *Neural Computing & Applications*, 5(3):131–133. ISSN 0-941-0643.

[Anderson, 1991] Anderson, M. B. (1991). Which costs more: prevention or recovery? In Kreimer, A. and Munasinghe, M., editors, *Managing Natural Disasters and the Environment*, pages 17–27. Environment Department. The World Bank, Washington, D.C.

[Beck, 2000] Beck, K. (2000). *Extreme programming explained: embrace change*. Addison-Wesley. ISBN 0-201-61641-6.

[Bishop, 1996] Bishop, C. M. (1996). *Neural Networks for Pattern Recognition*. Oxford University Press. ISBN 0-19-853864-2.

[Cerf et al., 1997] Cerf, N. J., Boutet de Monvel, J., Bohigas, O., Martin, O. C., and Percus, A. G. (1997). The random link approximation for the Euclidean traveling salesman problem. *Journal de Physique I*, 7:117–136.

[Cherkassky and Mulier, 1998] Cherkassky, V. and Mulier, F. (1998). *Learning From Data: Concepts, Theory, and Methods*. John Wiley and Sons. ISBN 0-471-15493-8.

[Cherkauer and Shavlik, 1995] Cherkauer, K. J. and Shavlik, J. W. (1995). Rapidly estimating the quality of input representations for neural networks. In *IJCAI-95 Workshop on Data Engineering for Inductive Learning*.

[Chuzhanova et al., 1998] Chuzhanova, N. A., Jones, A. J., and Margetts, S. (1998). Feature selection for genetic sequence classification. *Bioinformatics*, 14(2):139–143.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signal and Systems*, 2:303–314.

[Evans, 2001] Evans, D. (2001). *Data Derived Estimates of Noise using Near Neighbour Asymptotics*. PhD thesis, Cardiff University, Wales, UK.

[Friedman, 1994] Friedman, J. H. (1994). An overview of predictive learning and function approximation. In Cherkassky, V., Friedman, J. H., and Wechsler, H., editors, *Computer and Systems Sciences*, volume 136, pages 1–61. Springer-Verlag.

[Friedman et al., 1979] Friedman, J. H., Bentley, J. L., and Finkel, R. A. (1979). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226.

[Fukunaga and Narendra, 1975] Fukunaga, K. and Narendra, P. M. (1975). A branch and bound algorithm for computing k-nearest neighbours. *IEEE Transactions on Computers*, C(24):750–753.

[Georgakakos, 1986a] Georgakakos, K. P. (1986a). A generalized stochastic hydrometeorological model for flood and flash-flood forecasting. 1. Formulation. *Water Resources*, 22(13):2083–2095.

[Georgakakos, 1986b] Georgakakos, K. P. (1986b). A generalized stochastic hydrometeorological model for flood and flash-flood forecasting. 2. Case studies. *Water Resources*, 22(13):2096–2106.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. MIT Press.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:356–366.

[Hunt and Thomas, 2000] Hunt, A. and Thomas, D. (2000). *The pragmatic programmer*. Addison-Wesley. ISBN 0-201-61622-X.

[James and Connellan, 2000] James, H. and Connellan, O. (2000). Forecasts of a small feature in a property index. RICS Cutting Edge Conference, London.

[Katsavounidis et al., 1996] Katsavounidis, I., Kuo, C.-C. J., and Zhang, Z. (1996). Fast tree-structured nearest neighbor encoding for vector quantization. *IEEE Transactions on Image Processing*, 5(2):398–404.

[Kendall and Stuart, 1963] Kendall, M. G. and Stuart, A. (1963). *The Advanced Theory of Statistics, Volume 1: Distribution Theory*. Griffin.

[Kennel et al., 1992] Kennel, M. B., Brown, R., and Abarbanel, H. D. I. (1992). Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical Review A*, 45(6):3403–3411.

[Kerr, 1985] Kerr, T. H. (1985). The proper computation of the matrix pseudoinverse and its impact in MVRO filtering. *IEEE Transactions on Aerospace and Electronic Systems*, 21(5):711–724.

[Kitadinis and Bras, 1980a] Kitadinis, P. K. and Bras, R. L. (1980a). Real-time forecasting with a conceptual hydrological model 1 - analysis of uncertainty. *Water Resources*, 16(6):1025–1033.

[Kitadinis and Bras, 1980b] Kitadinis, P. K. and Bras, R. L. (1980b). Real-time forecasting with a conceptual hydrological model 2 - applications and results. *Water Resources*, 16(6):1034–1044.

[Končar, 1997] Končar, N. (1997). *Optimisation methodologies for direct inverse neurocontrol*. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London.

[Kumoluyi et al., 1995] Kumoluyi, A., Daltaban, T. S., Končar, N., Jones, A. J., and Archer, J. S. (1995). Well reservoir model identification using translation and scale invariant higher order networks. *Neural Computing & Applications*, 3(3):128–138. ISSN 0-941-0643.

[Le Cun, 1986] Le Cun, Y. (1986). Learning processes in an asymmetric threshold network. In Bienenstock, E., Souli, F. F., and Weisbuch, G., editors, *Computer and Systems Sciences*. Springer.

[McCulloch and Pitts, 1943] McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

[McNames et al., 1999] McNames, J., Suykens, J. A. K., and Vandewalle, J. (1999). Winning entry of the K.U. Leuven time-series prediction competition. *International Journal of Bifurcation and Chaos*, 9(8):1485–1500.

[Michalewicz and Fogel, 2000] Michalewicz, Z. and Fogel, D. B. (2000). *How to Solve It: Modern Heuristics*. Springer-Verlag. ISBN 3-540-66061-5.

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). *Perceptrons*. MIT Press.

[Mutreja et al., 1987] Mutreja, K. N., Yin, A., and Martino, I. (1987). Flood forecasting model for Citandy river. In Singh, V. P., editor, *Flood Hydrology*, pages 211–220. Reidel, Dordrecht, The Netherlands.

[Parker, 1985] Parker, D. (1985). Learning logic. *Technical report TR-47, MIT Center for Research in Computational Economics and Management Science, Cambridge, MA.*

[Penrose, 1955] Penrose, R. (1955). A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, 51:406–413.

[Penrose, 1956] Penrose, R. (1956). On best approximate solution of linear matrix equations. *Proceedings of the Cambridge Philosophical Society*, 52:17–19.

[Pfleger et al., 1994] Pfleger, K., John, G., and Kohavi, R. (1994). Irrelevant features and the subset selection problem. In Cohne, W. W. and Hirsh, H., editors, *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129.

[Pi and Peterson, 1994] Pi, H. and Peterson, C. (1994). Finding the embedding dimension and variable dependencies in time series. *Neural Computation*, 5:509–520.

[Press et al., 1992] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C*. Cambridge University Press, second edition. ISBN 0-521-43108-5.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

[Ramasubramanian and Paliwal, 1992] Ramasubramanian, V. and Paliwal, K. K. (1992). Fast k-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding. *IEEE Transactions on Signal Processing*, 40(3):518–531.

[Rosander, 1957] Rosander, A. C. (1957). *Elementary Principles of Statistics*. D. Van Nostrand, third edition.

[Rosenblatt, 1962] Rosenblatt, F. (1962). *Priciples of neurodynamics: perceptrons and the theory of brain mechanics*. Spartan.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Orientated Modeling and Design*. Prentice-Hall International. ISBN 0-13-630054-5.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

[Sauer et al., 1991] Sauer, T., Yorke, J. A., and Casdagli, M. (1991). Embedology. *Journal of Statistical Physics*, 65(3):579–616.

[Scherf and Brauer, 1997] Scherf, M. and Brauer, W. (1997). Feature selection by means of a feature weighting approach. Technical Report FKI-221-97, Forschungsberichte Künstliche Intelligenz, Institut für Informatik, Technische Universität München.

[Smith, 1996] Smith, K. (1996). *Environment Hazards: Assessing Risk and Reducing Disaster*. Routledge. ISBN 0-415-12204-X.

[Sommerville, 1996] Sommerville, I. (1996). *Software Engineering*. Addison-Wesley. ISBN 0-201-42765-6.

[Stephens and Stapleton, 1983] Stephens, H. S. and Stapleton, C. A., editors (1983). *International Conference on the Hydraulic Aspects of Floods & Flood Control*. BHRA Fluid Engineering. ISBN 0-906085-86-1, ISSN 0265-0894.

[Suykens and Vandewalle, 1998] Suykens, J. A. K. and Vandewalle, J., editors (1998). *Nonlinear modelling: advanced black-box techniques*. Kluwer Academic Publishers. ISBN 0-7923-8195-5.

[Takens, 1981] Takens, F. (1981). Detecting strange attractors in turbulence. In Rand, D. A. and Young, L. S., editors, *Dynamical Systems and Turbulence*, volume 898, pages 366–381. Lecture Notes in Mathematics, Springer-Verlag.

[Thirumalaiah and Deo, 1988] Thirumalaiah, K. and Deo, M. C. (1988). River stage forecasting using artificial neural networks. *Journal of Hydrologic Engineering*, 3(1):26–31. ISSN 1084-0699.

[Tsui, 1999] Tsui, A. (1999). *Smooth Data Modelling and Stimulus-Response via Stabilisation of Neural Chaos*. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London.

[Tsui et al., 2001] Tsui, A. P. M., Jones, A. J., and de Oliveira, A. G. (2001). The construction of smooth models using irregular embeddings determined by a gamma test analysis. *Forthcoming publication in Neural Computing & Applications*.

[Valença and Ludermir, 2000] Valença, M. and Ludermir, T. (2000). Neural networks vs. PARMA modelling: Case studies of river flow prediction. In Ribeiro, C. H. C. and França, F. M. G., editors, *Proceedings Volume I of VIth Brazilian Symposium on Neural Networks SBRN 2000*. IEEE Computer Society, Los Alamitos, CA. ISBN 0-7695-0856-1.

[Werbos, 1974] Werbos, P. (1974). *Beyond regression: new tools for prediction and analysis in the behavioural sciences*. PhD thesis, Harvard University, Boston, MA.

[Widrow and Hoff, 1960] Widrow, B. and Hoff, M. (1960). Adaptive switching circuits. In *IRE WESCON Convention Record*, pages 96–104.