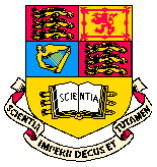
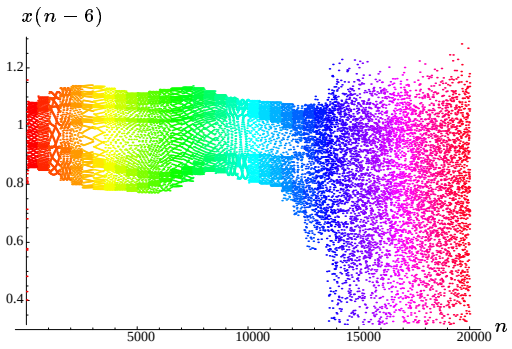


Smooth Data Modelling and Stimulus-Response via Stabilisation of Neural Chaos

A Thesis submitted for the degree of Doctor of Philosophy
in the University of London and for the Diploma of
Membership of Imperial College

by

Alban Pui Man Tsui



Department of Computing
Imperial College of Science, Technology and Medicine
University of London

Abstract

On the basis of studies of the olfactory bulb of a rabbit Freeman suggested that in the rest state the dynamics of this neural cluster is chaotic, but that when a familiar scent is presented the neural system rapidly simplifies its behaviour and the dynamics becomes more orderly, more nearly periodic than when in the rest state. This suggests an interesting model of recognition in biological neural systems. To realise this in an artificial neural system, some form of control of the chaotic neural behaviour is necessary to achieve periodic dynamical behaviour when a stimulus is presented.

In this thesis we first study the general problem of modelling smooth systems and introduce a number of useful techniques relevant to the problem of modelling chaotic dynamics. After a preliminary review of chaotic dynamical systems and their control, and discussing several examples of neural chaos, we then construct a chaotic neural model. We show how this model can be successfully controlled using several different parametric control methods. However, such methods of control are *external* to the network and we are interested in the control of higher dimensional networks using a technique which is *intrinsic* to the neural dynamics.

Using a higher dimensional system we investigate several methods of control and conclude that control using *delayed feedback* is a feasible mechanism for producing the retrieval behaviour described by Freeman. Delayed feedback provides a mechanism for stabilisation onto unstable periodic behaviours. The particular unstable periodic orbit which is stabilised depends quite strongly on the precise character of the applied stimulus. Thus the system can act as an associative memory in which the act of recognition corresponds to stabilising onto an unstable periodic orbit which is characteristic of the applied stimulus. The entire artificial system therefore exhibits an overall behaviour and response to stimulus which precisely parallels the biological neural behaviour observed by Freeman.

Acknowledgement

I would like to *thank* Prof. Antonia J. Jones for all her supervision and support towards the success of this piece of research and her continuous inspiration. I would also like to thank all the staff and members in Prof. Jones' research group at University of Wales, Cardiff, and staff, especially Tina, at the *Reseach Farm* in Cwm Cadlan.

Thanks are also due to Steve Margetts and Peter Durrant at Cardiff, who have provided many software solutions. I would also like to give special thanks to Nenad Končar and Ana Guedes de Oliveira for their cooperation in achieving many inspiring works. Thanks to Mr. & Mrs. D Blessley for putting me up for my 'research stay' at Brynawel. Thanks to those whom I have forgotten to mention.

I own a great debt to my family for their trust and emotional and financial support. A special thanks to them.

I also acknowledge the support by EPSRC, U.K., Studentship number: 96307552.

Alban P.M. Tsui

徐佩文

March 1999

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This work draws its inspiration from [Freeman 1991]. On the basis of studies of the olfactory bulb of a rabbit Freeman suggested that in the 'rest state' the dynamics of this neural cluster is chaotic, but that when a familiar scent is presented the neural system rapidly simplifies its behaviour and the dynamics becomes more orderly, more nearly periodic than when in the rest state. We call this the 'retrieval behaviour' since it is analogous to the act of recognition . This suggests an interesting model of recognition in biological neural systems which is quite different from earlier attempts to use neural networks for pattern recognition or as associative memories. To create an artificial neural network which behaves in the manner described by Freeman we have to investigate several fields of study which at first sight are far removed from the conventional study of neural networks.

To construct such a system we have to consider how best to construct neural models which exhibit chaotic dynamics. Neural network models which are dynamical systems are not (of course) new. The classical example is the Hopfield network, for which the simplest case considers nodes whose outputs are zero or one and where memories are associated with specified (preferably uncorrelated) point attractors. However, such a model cannot meet our needs. The state space is finite, consisting of fixed length vectors whose components are zero or one, and hence 'chaos' in the classical sense of dynamical systems, with its infinitely rich variety of modalities will never be exhibited. Indeed for a symmetric Hopfield network the dynamics are essentially trivial: starting from any initial state the network will simply iterate to a fixed point.

In contrast if the dynamics are chaotic then unstable periodic orbits are dense on the chaotic attractor and there are infinitely many of them. Thus an associative memory such as described by Freeman, for which the computations are performed to an *arbitrary* precision, could in principle accommodate infinitely many memories. At any rate such a system is not subject to the conventional Hopfield upper bound of $0.15n$, where n is number of neurons [Amit *et al.* 1987]. Of course, for the Hopfield net the situation is rather different. In the Hopfield model memories are associated with specified point attractors, whereas in the Freeman paradigm memories would be associated with unstable periodic behaviours which could not be specified *ab initio*. However, another great attraction of the Freeman approach is that it introduces the possibility of responding to stimuli over varying *time scales* using behaviours with different periodicities.

Plainly we need to work with network models having continuous node outputs rather than the discrete outputs of the classical Hopfield model.

One of the major developments of neural networks in the 1980's was the introduction of backpropagation which enabled the construction of smooth non-linear input/output models using multilayer

feedforward neural networks. As we shall see in Chapter 3 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 number of previous states. This observation is in fact a quite deep theorem due originally to [Takens 1981].

Historically there have been several interesting models of neural systems which generate chaos and in Chapter 5 we shall give a brief survey. An early example of such a study was the VCON oscillator neuron of [Hoppensteadt 1989]. However, building on the existing knowledge of smooth non-linear modelling techniques we choose to use an approach based on Takens theorem and construct feedforward networks which can form accurate iterative models of any given system.

We therefore start in Chapter 2 by examining some recent developments in data analysis and modelling under noise, the Gamma test [Končar 1997; Stefánsson *et al.* 1997], which help us to construct smooth non-linear models with some degree of efficiency. Use of the Gamma test eliminates much of the tiresome process of trial and error often associated with training a feedforward neural network. In particular we study how much can be inferred about the required architecture for the feedforward network directly from the training data.

In Chapter 3 we digress slightly and investigate other possible modelling techniques which might provide alternative methods for prediction without the long training times often associated with feedforward neural network model building. These techniques are all based on estimating the corresponding output of a hitherto unseen input using the *local* near neighbour information of the input signal in the input space of the training data. Extraction of such local information is accomplished using a data structure known as a *kd-tree*. kd-trees are also basic to the Gamma test and are described in some detail in Chapter 3. In Chapter 4, more examples of applications which require smooth data modelling are given, as well as many essential pre-processing techniques introduced for improving the modelling.

Once we have developed diverse and relatively efficient techniques for modelling smooth non-linear functions we are then able to meet the requirement of constructing feedforward networks that approximate iterative chaotic maps with a very small mean-squared error (of the order of 10^{-6}).

Having seen how to exhibit neural chaos the next question becomes how to control it? We approach this issue by considering a range of existing techniques which since 1989 have been used to control an enormous range of different types of chaotic systems. Historically the first of these was a technique due to Ott, Grebogi and Yorke, known as the *OGY method* [Ott *et al.* 1990] and we describe this method in some detail in Chapter 5. The basic idea is that a chaotic system exhibits numerous unstable periodic orbits and, having located one such behaviour, the OGY method seeks to stabilise this orbit using small variations of some accessible system parameter.

Many such methods require careful and systematic analysis of the chaotic dynamical behaviour, which is usually difficult and computationally expensive, before successful control can be achieved. Moreover, such control techniques are *external* to the system being controlled, whereas for a neural system to behave as described by [Freeman 1991] the control should be *intrinsic* to the neural dynamics. Nevertheless, such preliminary studies serve as a useful starting point for studying the control of neural chaos. In Chapter 6, many simple examples of controlling chaotic artificial neural networks are given.

The dynamics of large neural ensembles are high-dimensional, and whilst the OGY technique is an effective tool for the control of low dimensional chaos it needs further elaboration for effective control of higher dimensional systems. Indeed, for higher dimensional systems it may be that other types

of control procedures will prove far more effective. To investigate the control of higher dimensional chaos as a starting point for the discussion of higher dimensional neural chaos we have chosen a well studied dynamical system described by a modified form of the Euler equations, the so called chaotic satellite attitude control problem. In Chapter 7 we apply various techniques to a variation of the chaotic satellite attitude control problem and show that it is possible to stabilise the system in a situation where five of the six sensors (three angular velocities and three attitude angles) and two of the three thrusters are inoperative. It emerges from the work of this chapter that a remarkably simple and effective method of stabilisation onto an unstable periodic behaviour can be effected by the application of delayed feedback. Delayed feedback to control *continuous* dynamical systems exhibiting chaos was first suggested in [Pyragas 1992]. We use a modified version of this approach to stabilise an iterative neural model (previously trained to generate chaotic behaviour in the 'rest state') in the presence of an input stimulus. We determine that the response to a particular stimulus is remarkably robust in the face of noise. A result which we found to be rather surprising whilst at the time extremely encouraging.

Little theoretical analysis is available for the Pyragas method of continuous delayed feedback control, let alone for the discrete form of the method used here. However, a discrete version of a variation of Pyragas' method has already successfully been applied to the *synchronisation* of two identical iterative chaotic maps in [Oliveira and Jones 1998]. The version used there for *synchronisation* is similar to but not identical to the method used here for *stabilisation*. [Oliveira and Jones 1998] also contained a suggestive discussion of the local stability properties of the method used. For both the Hénon map and the chaotic neural network used here it was shown that whilst the synchronisation control method used by [Oliveira and Jones 1998] was not locally stable it was nevertheless *probabilistically locally stable*.

We provide a similar empirical analysis for the method of stabilisation proposed here in the case where no external stimulus is present.

One of the attractions of delayed feedback stabilisation is that it has a very low computational overhead and so is extremely easy to implement in hardware. It would also be very easy to implement in biological neural circuitry and so offers one plausible mechanism whereby such stabilisation might occur.

The particular unstable periodic orbit which is stabilised depends quite strongly on the precise character of the applied stimulus. Thus the system can act as an associative memory in which the act of recognition corresponds to stabilising onto an unstable periodic orbit which is characteristic of the applied stimulus. The entire artificial system therefore exhibits an overall behaviour and response to stimulus which precisely parallels the biological neural behaviour observed by Freeman.

Chapter 2

Feedforward neural network modelling & the Gamma test

Feedforward artificial neural networks (FANN for short) with a smooth sigmoidal have been commonly chosen as the choice for modelling smooth input-output systems. A suitable architecture for the feedforward network and an effective learning algorithm can often be used to model data or predict time series with good accuracy [Dracopoulos and Jones 1993]. In this chapter we sketch these fundamental results on the modelling capability of feedforward neural networks. An alternative graphical explanation of feedforward networks based on ideas due to Lapedes [Lapedes and Farber 1988] is also presented.

A data noise estimation technique called the *Gamma test* is then presented and discussed in terms of its usefulness and relevance to automating neural network construction for data modelling. Basically, this is an introduction to the Gamma test which will be continuously used and exploited in the rest of this work.

2.1 Feedforward neural network approximation

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 $f : \mathbb{R}^n \to \mathbb{R}^m$ over a compact subset of $\mathbb{R}^n$, provided sufficiently many hidden units are available [Hornik *et al.* 1989; 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. These results depend essentially on the *Stone-Weierstrass theorem* which asserts that an algebra $\mathcal{A}$ of real continuous functions, that separates points on a compact set $\mathcal{K}$ and does not vanish at any point of $\mathcal{K}$, is *dense* in the space of real *continuous* functions on $\mathcal{K}$.

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.

Here we present a simple theorem on feedforward neural network modelling to illustrate the idea that a feedforward neural network can be viewed as an approximation function. In this example we

show that any continuous function on a square in $\mathbb{R}^2$ can be approximated by a 3-layer[1] feedforward neural network [Blum and Li 1991]. The result requires the use of a special case of the Stone-Weierstrass theorem which says that any continuous function on such a square can be approximated by a sum of cosine functions. Here is the theorem we need without the proof:

Theorem 2.1.1 (Stone-Weierstrass theorem special case). *Let $f : [0, \pi]^2 \to \mathbb{R}$ be continuous. For any given $\epsilon > 0$, there is $N \in \mathbb{N}$ and constants a_{mn}, with $0 \leq m, n \leq N$, such that*

$$\left| f(x, y) - \sum_{m,n=0}^{N} a_{mn} \cos(mx) \cos(ny) \right| < \epsilon \tag{2.1}$$

for all $(x, y) \in [0, \pi]^2$.

Stated in this form the result bears a strong resemblance to Fourier's theorem, but we are not interested here in performing Fourier analysis. Our plan is simply to prove an approximation theorem for feedforward networks using threshold neurons. Define

$$\text{step}\,(x) = 1 \quad \text{if} \quad x \geq 0, \quad \text{otherwise} \quad \text{step}\,(x) = 0. \tag{2.2}$$

We first establish that $\cos(t)$ can be uniformly approximated by a suitable linear combination of such step functions, i.e.

Lemma 2.1.1. *Let*

$$\gamma(t) = \sum_{j=1}^{M} w_j \text{step}\,(t - \theta_j) \tag{2.3}$$

Given $\delta \geq 0$ and $X > 0$, $X \in \mathbb{N}$ we can choose $M = M(\delta, X)$, $M \in \mathbb{N}$, sufficiently large and real numbers w_j and θ_j $(1 \leq j \leq M)$ so that

$$|\gamma(t) - \cos(t)| < \delta \quad \textit{for} \quad |t| \leq 2X\pi \tag{2.4}$$

Sketch proof. The idea is simply that we approximate $\cos(t)$ by a sequence of $M/2$ small horizontal line segments, where each line segment is composed from a pair of step functions. This process is illustrated in Figure 2.1. ∎

We can use this lemma to establish

Theorem 2.1.2. *Let $f : [0, \pi]^2 \to \mathbb{R}$ be continuous. For any given $\epsilon > 0$, there is a 3-layer feedforward neural network with McCulloch-Pitts neurons in the hidden layer and a linear output unit which approximates f on $[0, \pi]^2$ to within ϵ.*

Proof. According to Theorem 2.1.1 above, there is N and constants a_{mn}, $0 \leq m, n \leq N$, such that

$$\left| f(x, y) - \sum_{m,n=0}^{N} a_{mn} \cos(mx) \cos(ny) \right| < \frac{\epsilon}{2} \tag{2.5}$$

for all $(x, y) \in [0, \pi]^2$. Let $K = \max |a_{mn}|$. We express

$$\cos(mx) \cos(ny) = \frac{\cos(mx + ny) + \cos(mx - ny)}{2} \tag{2.6}$$

[1] This includes the input layer, one hidden layer and one output layer.

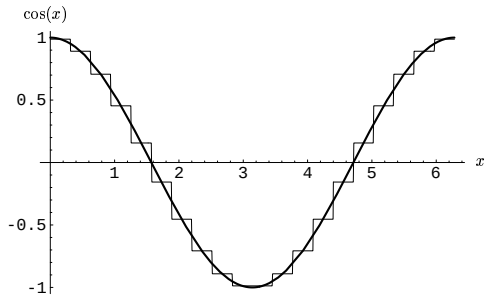


Figure 2.1: $\cos(x)$ can be approximated by n line segments for which each line segment is composed from a pair of step functions.

and also note that $|mx \pm ny| \leq 2N\pi$ for any $(x, y) \in [0, \pi]^2$. Writing

$$h(x,y) = \sum_{m,n=0}^{N} \frac{a_{mn}}{2} \left(\cos(mx+ny) + \cos(mx-ny)\right) \tag{2.7}$$

we have from (2.5)

$$|f(x,y) - h(x,y)| < \frac{\epsilon}{2}. \tag{2.8}$$

We next use Lemma 2.1.1, taking $\delta = \epsilon/(4(N+1)^2K)$, to approximate the sum of cosines and obtain

$$\begin{aligned} &\left| h(x,y) - \sum_{m,n} \frac{a_{mn}}{2}(\gamma_1(mx+ny) + \gamma_2(mx-ny)) \right| \\ &\leq \sum_{m,n} |a_{mn}| \left(\frac{\epsilon}{4(N+1)^2K} + \frac{\epsilon}{4(N+1)^2K} \right) \\ &< \sum_{m,n} \frac{\epsilon}{2(N+1)^2} = \frac{\epsilon}{2}. \end{aligned} \tag{2.9}$$

If we now examine the feedforward network illustrated in Figure 2.2 we see that the output $g(x, y)$ of the network is precisely a linear combination of step functions,

$$\begin{aligned} g(x,y) &= \sum_{m,n=0}^{N} \frac{a_{mn}}{2} \left(\gamma_1(mx+ny) + \gamma_2(mx-ny)\right) \\ &= \sum_{m,n=0}^{N} \frac{a_{mn}}{2} \left(\sum_{j=1}^{M} u_j \text{step}\left(mx+ny-\mu_j\right) + \sum_{j=1}^{M} v_j \text{step}\left(mx-ny-\nu_j\right) \right). \end{aligned} \tag{2.10}$$

Inequalities (2.8) and (2.9) give

$$\begin{aligned} |f(x,y) - g(x,y)| &\leq |f(x,y) - h(x,y)| + |h(x,y) - g(x,y)| \\ &< \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon \end{aligned} \tag{2.11}$$

for $(x, y) \in [0, \pi]^2$. ∎

More detailed examples and more general theorems for different type of activation functions can be found in [Hornik *et al.* 1989; Cybenko 1989]. The main point concluded from these results is that multilayer feedforward networks are a class of *universal approximators*. In general any $f \in C[\mathcal{S}]$, the space of continuous real functions on a compact subset, $\mathcal{S}$ of $\mathbb{R}^n$, can be arbitrarily closely

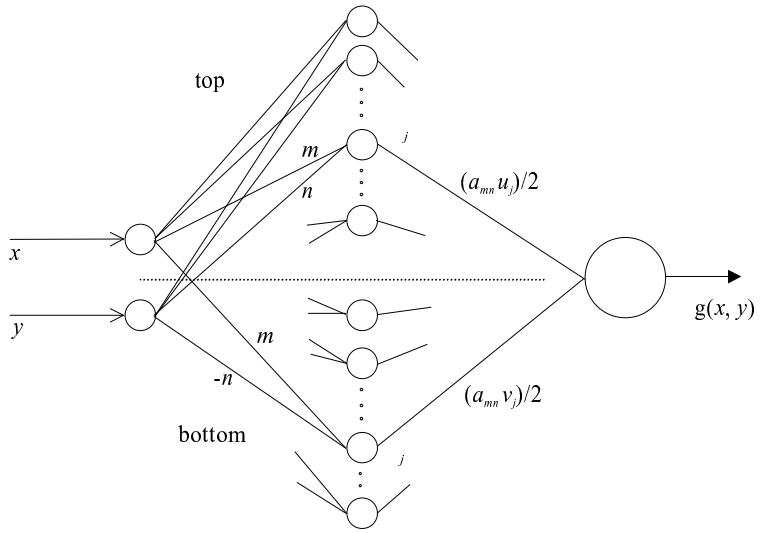


Figure 2.2: The 3-layer neural network implementing f to within ϵ. The network has $(N+1)^2 \times M + (N+1)^2 \times M$ hidden neurons (doubly labelled by m, n, j). The two weights from the inputs to the hidden neuron labelled m, n, j in the top half are m and n, whereas those to the hidden neuron m, n, j in the bottom half are m and $-n$.

approximated in the uniform norm by a two-layer feedforward network with semi-linear hidden units using a sigmoidal threshold function, g, and one linear output unit [Hornik *et al.* 1989]. Thus

$$f(x) \approx_\epsilon \sum_{i=1}^{m} w_i g\left(\sum_{j=1}^{n} a_{ij}x_j - c_i\right) \tag{2.12}$$

where the weights w_i and a_{ij} and thresholds c_i are real numbers. The symbol $\approx_\epsilon$ denotes the approximation with error $\leq \epsilon$. The function g is a monotone real function with $g(z) \to 0$ as $z \to -\infty$ and $g(z) \to 1$ as $z \to \infty$.

2.2 Graphical understanding of FANNs

For a clearer understanding of the functionality of a feedforward network, Lapedes [Lapedes and Farber 1988] has opted for a graphical and modular approach to construct a feedforward network to model a surface, i.e. $f : [0,1]^2 \to \mathbb{R}$.

Lapedes' basic recipe will produce a 'hill' by constructing a network with 2 hidden layers of sigmoidal nodes and a summing output layer. Here is a simple recipe for a neural unit of 2 inputs and 1 output to produce a hill:

1. Construct a 'sigmoidal surface' with a single node, see Figure 2.3. The weights set the orientation of the facing slope and the threshold value positions the slope on the input space.

2. Construct another 'sigmoidal surface' with another node with the same set of weights (i.e. the same orientation) and a threshold with a suitable slight offset from the previous threshold value. The difference of the outputs of the two nodes can produce a 'ridge' surface (Figure 2.4).

3. Similarly produce another 'ridge' perpendicular to the previous 'ridge' using the previous two steps but with different set of weights and thresholds. The sum of the two 'ridges' will produce a 'gentle' hill as shown in Figure 2.5.

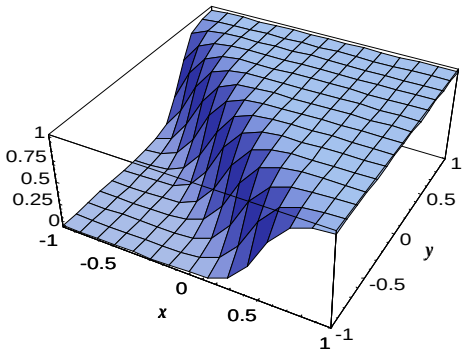


Figure 2.3: Single sigmoidal node ($\sigma(20x + 20y + 10)$ with $T = 2$) can produce a 'sigmoidal surface' for this 2 inputs and 1 output system.

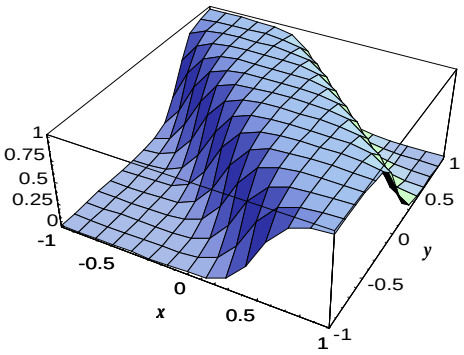


Figure 2.4: Adding another sigmoidal node with suitable weights ($\sigma(20x + 20y + 10) - \sigma(20x + 20y + 20)$ with $T = 2$) can produce a ridge.

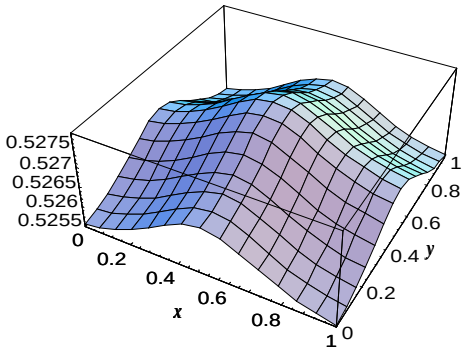


Figure 2.5: The intersection of two 'ridges' placed perpendicularly to each other forms a gentle 'hill' surface.

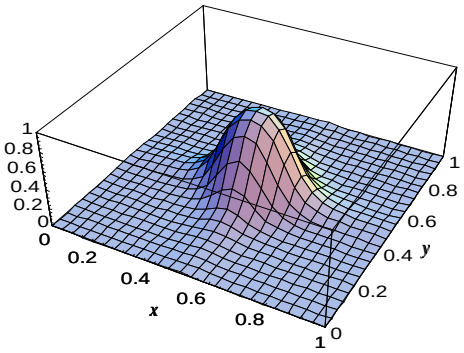


Figure 2.6: Suitable choice of weights, thresholds for the final layer can smooth off the unwanted trailing ridges.

4. Passing the sum into another sigmoidal node to flatten the unwanted elevations to form a perfect round hill surface, Figure 2.6.

Further hills on the same surface can be produced by constructing different units and then their outputs can be passed through a summing node. One should note that the steepness of hills depends on the temperatures, T (i.e. slopes) of the sigmoidal nodes.

Based on this recipe, we can produce a hill using the architecture shown in Figure 2.7. The sections **A** and **B** correspond to the two 'ridges' described in step 1 and 2 of the recipe. The node at **C** is the node which smoothes off the trailing 'ridges' of the intersection of the two 'ridges' as in step 4. The final node at **D** is a sum node for summing all the hills together.

This graphical approach inspires an interesting idea, that of using the 'slope' estimate in the *Gamma test* [Stefánsson *et al.* 1997] to construct suitable feedforward neural networks to model data of input and output pairs. Early investigation shows how the second parameter A returned by the Gamma test can be used to estimate the number of hidden nodes for a single hidden layer feedforward neural network which are required to attain the best achievable performance.[2] Without any further discussion of this new idea which can be found later in the chapter, let us introduce the Gamma test.

[2] A joint investigation with Nenad Kŏncar

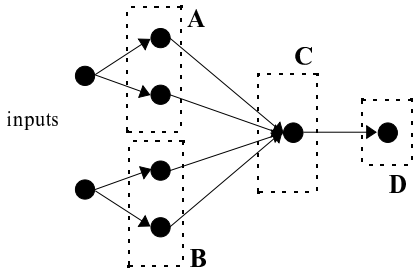


Figure 2.7: Connectivity of a unit to produce a hill. Add 4 more nodes to hidden layer 1 and one more node to hidden layer 2 for each additional hill.

2.3 The Gamma test - an introduction

In a recent paper [Stefánsson *et al.* 1997] a simple test (the *Gamma test*) which, in many situations, can accurately estimate from the available input/output data the best achievable performance of a smooth data model was developed. For non-linear modelling applications, and in particular for feedforward neural networks trained by backpropagation, such a test is extremely useful because it enables us to predict the best achievable performance of the model without the time consuming necessity of estimating this empirically by creating, training and testing a number of networks.

The Gamma test is a data analysis routine, that (in an optimal implementation) runs in time $O(M \log M)$ as $M \to \infty$, where M is the number of sample data points, and which offers an estimate of the best Mean Squared Error (MSE) that can be achieved by any continuous or smooth (bounded first partial derivatives) data model constructed using the data without over-fitting. For completeness we briefly describe the Gamma test but here we are interested in how the complexity of the modelling task can be estimated rather than the best achievable MSE.

Let a data sample be represented by

$$((x_1, x_2, \ldots, x_d), y) = (\boldsymbol{x}, y) \tag{2.13}$$

in which we think of the vector $\boldsymbol{x} = (x_1, ..., x_d)$ as the input, confined to a closed bounded set C, and the scalar y as the output. In the interests of simplicity the following explanation is presented for a single scalar output y which is assumed, whenever y is the output of a neural network, to lie in the interval $(0, 1)$. But the same algorithm can be applied to the situation where y is a vector with very little extra complication or time penalty.

We focus on the case where samples are generated by an unknown continuous function $f : C \subseteq \mathbb{R}^d \to \mathbb{R}$ and

$$y = f(x_1, x_2, \ldots, x_d) + r \tag{2.14}$$

where r represents an indeterminable part, which may be due to real noise or might be due to lack of functional determination in the posited input/output relationship i.e. an element of 'one $\to$ many-ness' present in the data. Over-training can eliminate the first of these for a particular training data set, but no amount of over-training can eliminate the second. In the case of applications to a time series $s(t)$, when the data $\boldsymbol{x}$ might represent a number d' of successive samples $(s(t-1), \ldots, s(t-d'))$ in time and y represent $s(t)$, the indeterminable quantity r may result from an insufficient embedding[3]

[3] An embedding is a technique for reconstructing dynamics using delay coordinates. Further discussion in Section 4.1.1.

dimension d'. For the present we treat the issue as a data processing problem in which r is statistical noise uncorrelated with $\boldsymbol{x}$ or y and $\mathrm{Mean}(r) = 0$.

In essence the Gamma test returns two numbers ($\bar{\Gamma}$, A), in which $\bar{\Gamma}$ is an estimate of the MSE of an output y, and in the case where the data is uniformly distributed in input space A is approximately $\frac{1}{4}\langle|\nabla f|^2\rangle$ [Končar 1997], where the angle brackets denote expectation with respect to the sampling distribution. Thus provided $y \in [0, 1]$ (or some fixed bounded interval) A is a rough measure of the complexity of the surface to be modelled. In constructing feedforward neural networks the ability to quickly estimate from the training data the surface complexity we seek to model is useful because one would expect such a measure to be correlated with the architecture of the required neural network. In particular it can be used to give us some idea of how many hidden layer units are required for the network to be capable of producing the MSE suggested by the Gamma test. Parsimony of hidden units is important because we are seeking to interpolate the *simplest* higher dimensional surface which can be generated by a feedforward neural network through the data set without over-fitting.

First a series of experiments are performed to investigate and/or validate the functionality of the Gamma test. Then the rest of this chapter will discuss how the second parameter returned by the Gamma test can be used to estimate the number of hidden nodes for a single hidden layer feedforward neural network which are required to attain the best achievable performance. Other statistics might easily be used to estimate surface complexity from the data, however a significant consideration here is that whatever method is used the algorithm should have a reasonable run time if we expect to be processing a large data set. Since the Gamma test runs in $O(M \log M)$ time and we already need this algorithm to estimate the best achievable MSE without over-fitting it seemed natural to begin by investigating how the slope parameter A is correlated with the required number of hidden units.

The later section will discuss a good correlation between the value of A returned by the Gamma test and the number of hidden layer neurons required to attain a good model of the data using a feedforward neural network with one hidden layer. In order to enable simple visualisation we have restricted the number of inputs in these experiments to 2 or 3 but the same principles can be applied regardless of the number of inputs provided that sufficient data is available. The purpose here is to convince the reader that the approach has some promise rather than to describe a precise formula for automated neural net construction.

The results presented in Section 2.4 are preliminary but indicate that the method is quite practical. The approach offers the possibility that the entire process of performance prediction and constructing a feedforward neural network which attains the best achievable performance on the basis of given training data can be automated with a fair degree of reliability.

For the Gamma test to be applicable the following assumptions are required. We assume that training and testing data are different sample sets in which:

- **Assumption A**

    1. the training set inputs are non-sparse in input-space.
    2. Each output is determined from the inputs by a deterministic process which is the same for both 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.

Given samples such as (2.13), in which the underlying continuous function f is unknown, we cannot hope to estimate the mean μ of r, since a non-zero mean will create a bias which could just

as easily be incorporated into the data model by considering f to be replaced by $f + \mu$. We therefore assume in what follows that $\mu = 0$.

In point of fact it is the variance of r, $\mathrm{Var}(r)$, which is of real interest. For example, if we were using a number of samples such as (2.13) to train a neural network then $\mathrm{Var}(r)$ provides a lower bound for the mean squared error of the output y (i.e. the variance of $y - f(x)$), beyond which, if the estimate is accurate and **Assumption A** holds, any attempt to improve the neural network model by further training would at best result in over-training. Indeed, this is true for *any* continuous or smooth data modelling technique.

The Gamma test is a very simple method for estimating $\mathrm{Var}(r)$. The most time consuming part of the process is to compute near neighbour lists for each point $\boldsymbol{x}$, but assuming that a bounded number of near neighbours are required (we found that $p_{\max} = 20$ or 30 near neighbours is typically adequate) this can be done in $O(M \log M)$ time using kd-trees [Bentley 1975]. Suppose $(\boldsymbol{x}(i), y(i))$ and $(\boldsymbol{x}(j), y(j))$, $i \neq j$, are two data samples. The basis of the idea is the observation that if $\boldsymbol{x}(i)$ and $\boldsymbol{x}(j)$ are near neighbours in input space and f is continuous then $y(i)$ and $y(j)$ should be near in the output space. Thus, for example, we should not expect the test to work well on d-bit parity, where the input vectors (being the vertices of an d-cube) are sparse and the output values (1 or 0) are uncorrelated for input-space near neighbours.

Suppose $(\boldsymbol{x}, y)$ is a data sample. Let $(\boldsymbol{x}', y')$ be a data sample such that $|\boldsymbol{x}' - \boldsymbol{x}| > 0$ is minimal. Here $|\cdot|$ denotes Euclidean distance and the minimum is taken over the set of all sample points different from $\boldsymbol{x}$. Thus $\boldsymbol{x}'$ is the nearest neighbour to $\boldsymbol{x}$ (in any ambiguous case we create a list of all equidistant points and incorporate them into the averaging).

The Gamma test (or near neighbour technique) is based on the statistic

$$\gamma = \frac{1}{2M} \sum_{i=1}^{M} \left(y'(i) - y(i)\right)^2 . \tag{2.15}$$

Let δ be the mean-squared first near neighbour distance. One can show that under reasonable conditions

$$\lim_{\delta \to 0} \gamma = \mathrm{Var}(r) \tag{2.16}$$

where the convergence is *convergence in probability*. For a finite set of data samples we cannot have arbitrarily small nearest neighbour distances. However, in practice even the crude measure provided by (2.15) often proves very useful.

If one is prepared to assume that f is smooth with bounded first partial derivatives we can obtain a more precise estimate than (2.15) by using a regression line fit on the statistic γ. Now given data samples $(\boldsymbol{x}(i), y(i))$, where $\boldsymbol{x}(i) = (x_1(i), \ldots, x_m(i))$, $1 \leq i \leq M$, let $\mathcal{N}[i, p]$ be the list of (equidistant) p^{th} nearest neighbours to $\boldsymbol{x}(i)$. We write

$$\delta(p) = \frac{1}{M} \sum_{i=1}^{M} \frac{1}{L(\mathcal{N}[i,p])} \sum_{j \in \mathcal{N}[i,p]} |\boldsymbol{x}(j) - \boldsymbol{x}(i)|^2 = \frac{1}{M} \sum_{i=1}^{M} |\boldsymbol{x}(\mathcal{N}[i,p]) - \boldsymbol{x}(i)|^2 \tag{2.17}$$

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

$$\gamma(p) = \frac{1}{2M} \sum_{i=1}^{M} \frac{1}{L(\mathcal{N}[i,p])} \sum_{j \in \mathcal{N}[i,p]} (y(j) - y(i))^2 \tag{2.18}$$

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

Under reasonable conditions one can show that

$$\gamma \approx \mathrm{Var}(r) + A\delta + o(\delta) \qquad \text{as} \qquad M \to \infty \tag{2.19}$$

where the convergence is in probability. From which it follows that $\lim \gamma = \mathrm{Var}(r)$ (in probability) as $\delta \to 0$.

This Gamma test computes the mean-squared p^{th} nearest neighbour distances $\delta(p)$ ($1 \le p \le p_{\text{max}}$, typically $p_{\text{max}} \approx 10$) and the corresponding $\gamma(p)$. The regression line of ($\delta(p)$, $\gamma(p)$) is computed and the vertical intercept $\bar{\Gamma}$ is returned as the Gamma value. Effectively this is the limit $\lim \gamma$ as $\delta \to 0$ (i.e. $M \to \infty$) which in theory is $\mathrm{Var}(r)$.

The original version of the Gamma test in [Stefánsson *et al.* 1997; Končar 1997] used smoothed versions of $\delta(p)$ and $\gamma(p)$ given by

$$\Delta(p) = \frac{1}{p}\sum_{h=1}^{p}\frac{1}{M}\sum_{i=1}^{M}\left|\boldsymbol{x}(\mathcal{N}[i,h]) - \boldsymbol{x}(i)\right|^2 \tag{2.20}$$

and

$$\Gamma(p) = \frac{1}{p}\sum_{h=1}^{p}\frac{1}{2M}\sum_{i=1}^{M}\left(y(\mathcal{N}[i,h]) - y(i)\right)^2 \tag{2.21}$$

The idea being that these equations rolled off the significance of more distant near neighbours. Thus taking p_{max} large in such an implementation often does not significantly alter the resulting $\bar{\Gamma}$ value. Indeed all the values of $\bar{\Gamma}$ reported in this chapter (and the discussion of the Gamma-minimum-predictor in Chapter 3) are based on the original version of the algorithm. However, later experience showed that provided p_{max} is kept small the extra complication of computing $\Delta(p)$ and $\Gamma(p)$ is largely unnecessary (although this form of the Gamma test can sometimes produce better $\bar{\Gamma}$ estimates when M is small) and the later implementations are based on equations (2.17) and (2.18).

An implementation of the Gamma test is given in Algorithm 2.1. The method used to construct the near neighbour lists can be $O(M^2)$ or $O(M \log M)$ depending on the sophistication of the coding.

Procedure: Gamma Test(data)
{data is an array of points $(\boldsymbol{x}(i), y(i))$, $(1 \le i \le M)$, in which $\boldsymbol{x}$ is a real vector of dimension d and y is a real scalar}
for $i = 1$ to M **do**
  {compute $\boldsymbol{x}$ nearest neighbours}
  for $p = 1$ to $p_{\max}$ **do**
    $\mathcal{N}[i,p] = t$ where $\boldsymbol{x}(t)$ is the p^{th} nearest neighbour to $\boldsymbol{x}(i)$.
  end for
end for
for $p = 1$ to $p_{\max}$ **do**
  compute $\Delta(p)$ as in (2.20) for original version (or replaced with (2.17) for later version)
  compute $\Gamma(p)$ as in (2.21) for original version (or replaced with (2.18) for later version)
end for
Perform least squares fit on coordinate $(\Delta(p), \Gamma(p))$ (or replaced with $(\delta(p), \gamma(p))$ for later version) $(1 \le p \le p_{\max})$ obtaining (say) $y = Ax + \bar{\Gamma}$
return ($\bar{\Gamma}$, A)

Algorithm 2.1: The Gamma test algorithm.

2.3.1 Some supporting analysis for the Gamma test

Since the full justification of the theoretical background of the Gamma test is not completely published and available, we outline some supporting analysis and present two illustrative experiments.

Consider the term

$$\begin{aligned}\frac{1}{2}(y'-y)^2 &= \frac{1}{2}(f(\boldsymbol{x}')+r'-f(\boldsymbol{x})-r)^2 \\ &= \frac{1}{2}\left((r'-r)+(f(\boldsymbol{x}')-f(\boldsymbol{x}))\right)^2.\end{aligned} \tag{2.22}$$

Using the smoothness hypothesis on the unknown function f we can expand the term

$$f(\boldsymbol{x}')-f(\boldsymbol{x}) = (\boldsymbol{x}'-\boldsymbol{x})\cdot f'(\boldsymbol{x}) + O(|\boldsymbol{x}'-\boldsymbol{x}|^2) \tag{2.23}$$

and substitute back into (2.22) to obtain

$$\begin{aligned}\frac{1}{2}(y'-y)^2 &= \frac{1}{2}\left((r'-r)+(\boldsymbol{x}'-\boldsymbol{x})\cdot f'(\boldsymbol{x})+O(|\boldsymbol{x}'-\boldsymbol{x}|^2)\right)^2 \\ &= \underbrace{\frac{1}{2}(r'-r)^2}_{[a]} + \underbrace{(r'-r)(\boldsymbol{x}'-\boldsymbol{x})\cdot f'(\boldsymbol{x})}_{[b]} \\ &\quad + \underbrace{\frac{1}{2}\left((\boldsymbol{x}'-\boldsymbol{x})\cdot f'(\boldsymbol{x})\right)^2}_{[c]} + \underbrace{(r'-r)O(|\boldsymbol{x}'-\boldsymbol{x}|^2)}_{[d]} + O(|\boldsymbol{x}'-\boldsymbol{x}|^3)\end{aligned} \tag{2.24}$$

If we now average both sides over $1 \le i \le M$ we can consider each term separately. Let us write

$$\mathcal{A} = \left|\mathrm{Var}(r) - \frac{1}{2M}\sum_{i=1}^{M}(r'_i - r_i)^2\right|, \tag{2.25}$$

$$\mathcal{B} = \left|\frac{1}{M}\sum_{i=1}^{M}(r'_i - r_i)(\boldsymbol{x}'_i - \boldsymbol{x}_i)\cdot\nabla f_{\boldsymbol{x}=\boldsymbol{x}_i}\right|, \tag{2.26}$$

$$\mathcal{C} = \left|\frac{1}{2M}\sum_{i=1}^{M}((\boldsymbol{x}'_i - \boldsymbol{x}_i)\cdot\nabla f_{\boldsymbol{x}=\boldsymbol{x}_i})^2\right| \tag{2.27}$$

and

$$\mathcal{D} = \left|\frac{1}{M}\sum_{i=1}^{M}(r'_i - r_i)|\boldsymbol{x}'_i - \boldsymbol{x}_i|^2\right|. \tag{2.28}$$

To justify (2.15) it is sufficient as $M \to \infty$ that each of these terms tends to zero. For $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ this is fairly clear by virtue of the assumption that there are no isolated points in the $\boldsymbol{x}$-space sampling distribution, i.e. that $|\boldsymbol{x}'-\boldsymbol{x}| \to 0$ as $M \to \infty$, and the fact that ∇f is assumed bounded.

That $\mathcal{A} \to 0$ as $M \to \infty$ follows from the assumption that r'_i and r_i are uncorrelated. In fact, since $\mathrm{Mean}(r) = 0$, we would expect that, with probability one $\mathcal{A} = O(M^{-1/2})$ as $M \to \infty$.

To establish (2.19) is more demanding. We have to show not only that these terms tend to zero but that the term $\mathcal{C}$, which corresponds to the term $A\delta$ in (2.19), does so more *slowly* than the terms $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{D}$. In other words we could justify (2.19) if we could prove that

$$\left.\begin{aligned}&\mathcal{A} = O(M^{-1/2}) = o(\mathcal{C}) \\ &\mathcal{B} = o(\mathcal{C}) \\ &\mathcal{D} = o(\mathcal{C})\end{aligned}\right\} \quad \text{as} \quad M \to \infty. \tag{2.29}$$

What should we expect for the order of $\mathcal{C}$? Heuristic considerations suggest that $(\boldsymbol{x}'_i - \boldsymbol{x}) \approx M^{-1/d}$ as $M \to \infty$, where d is the dimension of the support of the sampling distribution in $\boldsymbol{x}$-space, see for example [Melzak 1979] for more formal results. In which case we expect that $\mathcal{C} \approx M^{-2/d}$ as $M \to \infty$. So that $\mathcal{A} = o(\mathcal{C})$ provided $1/2 > 2/d$, i.e. $d > 4$. We should recognise that in seeking to justify (2.19) by treating each of these terms separately we loose useful cancellation between error terms. In practice the Gamma test based on (2.19) seems to work very well even for $d = 1$.

Now consider the requirement that $\mathcal{B} = o(\mathcal{C})$. We regard the individual terms of the sum in $\mathcal{B}$ as noise $r'_i - r_i$, which has mean zero, multiplied by terms $(\boldsymbol{x}'_i - \boldsymbol{x}_i) \cdot \nabla f$ in which ∇f is bounded and $(\boldsymbol{x}'_i - \boldsymbol{x}_i) \approx M^{-1/d}$ as $M \to \infty$. We should therefore expect $\mathcal{B}$ to be $O(M^{-(1/d+1/2)})$ with probability one. Thus $\mathcal{B} = o(\mathcal{C})$ provided $1/d + 1/2 > 2/d$, i.e. $d > 2$.

The final requirement is that $\mathcal{D} = o(\mathcal{C})$. The individual terms $|\boldsymbol{x}'_i - \boldsymbol{x}_i|^2$ of the sum in $\mathcal{D}$ are similar in magnitude to those of $\mathcal{C}$ except that each is multiplied by a noise term $r'_i - r_i$. The net effect on the sum is to introduce cancellation. We should expect that $\mathcal{D} = O(M^{-(2/d+1/2)})$ with probability one as $M \to \infty$. Thus $\mathcal{D} = o(\mathcal{C})$ provided $2/d + 1/2 > 2/d$, which is automatically satisfied.

We next give two simple experiments to check if these terms behave along the lines predicted. We take $d = 6$ and use the function

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 \tag{2.30}$$

with $-1 \leq x_i \leq 1$ and a uniform sampling distribution in $\boldsymbol{x}$-space. First we try with a uniform noise distribution with $\mathrm{Var}(r) = 0.09$. By increasing M we can repeatedly calculate the corresponding term $\mathcal{A}$ and we can then perform a linear regression to fit a line to the plot of $\log \mathcal{A}$ against $\log M$. The gradient of this line is then the value g of $\mathcal{A} \approx M^g$. Similarly this procedure can be repeated for the terms $\mathcal{B}$ and $\mathcal{C}$. We then plot the gradients g of the "loglog" plots for the terms $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ against M to check the asymptotic values of g. The result for the uniformly distributed noise experiment is shown in Figure 2.8. The asymptotic behaviour of the critical terms is approximately $\mathcal{A} \approx M^{-0.57}$, $\mathcal{B} \approx M^{-0.53}$, and $\mathcal{C} \approx M^{-0.39}$, so that $\mathcal{C}$ is the dominant term as required.

The experiment is repeated for Gaussian noise with mean $\mathrm{Mean}(r) = 0$ and variance $\mathrm{Var}(r) = 0.09$. The corresponding result is shown in Figure 2.9. Here the asymptotic behaviour of the critical terms is approximately $\mathcal{A} \approx M^{-1.03}$, $\mathcal{B} \approx M^{-1.11}$, and $\mathcal{C} \approx M^{-0.39}$, so that once again $\mathcal{C}$ is dominant.

In practice such large values of M are not necessary when using the Gamma test on data generated

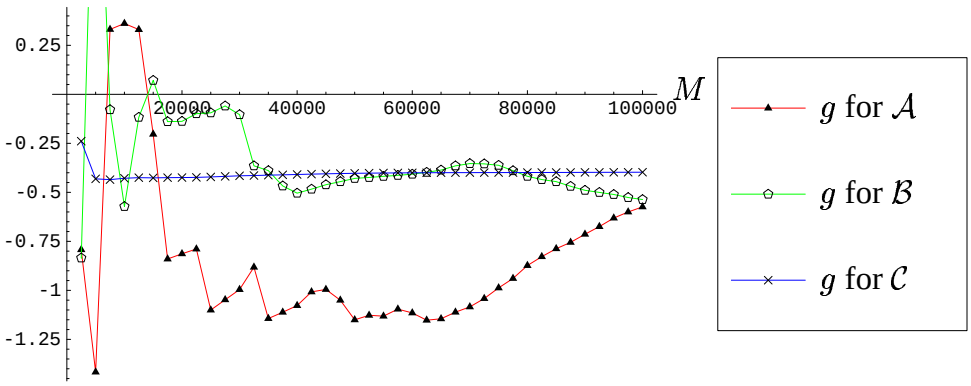


Figure 2.8: Gradients g of the "loglog" plots for the terms $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ against varying M for uniformly distributed random noise with variance about 0.09

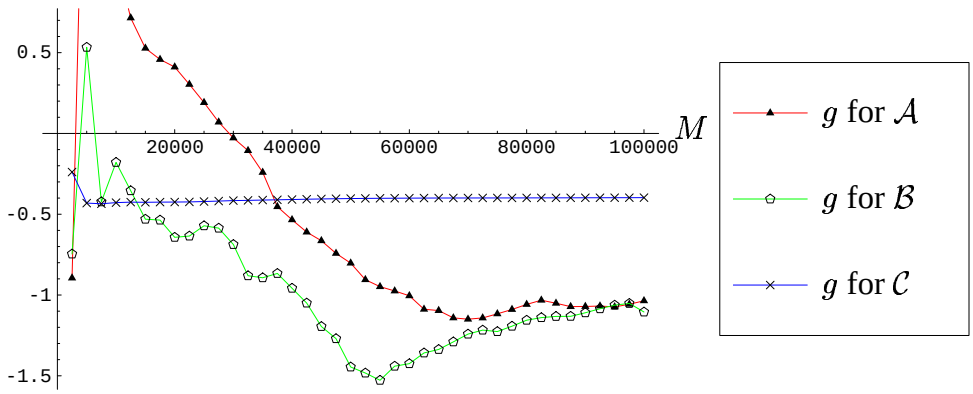


Figure 2.9: Gradients g of the "loglog" plots for the terms $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ against varying M for Gaussian noise, with mean 0 and $\mathrm{Var}(r) = 0.09$.

by such simple surfaces f.

2.3.2 Gamma test in higher-dimensional input space

A simple experiment is set up to investiage how the Gamma test (original form) behaves when the input vectors are higher-dimensional. First define a function

$$f(\boldsymbol{x}) = \sum_{i=1}^{d} \Psi_i \sin^2(a_i x_i + b_i), \tag{2.31}$$

where x_i is the i^{th} component of vector $\boldsymbol{x}$ with $0 \le x_i \le 1$, $\Psi_i = \cos^2(i)$, $a_i = 0.5\cos^2(i\sqrt{2})$ and $b_i = \sin^2(i\sqrt{6})$, for generating sample data. The choice of these parameters was arbitrary, subject to the requirement of maintaining model complexity. In fact, this is the underlying model to be implicitly approximated by the Gamma test in order to measure the level of noise of the data. For each dimension d, M data points are generated by this function and random noise, of 20% of the size of the output range, $|\max f(\boldsymbol{x}) - \min f(\boldsymbol{x})|$, of the function f in (2.31), is then added to the outputs of the sampled data.

For each d, we increase M and measure the true variance of the noise, $\mathrm{Var}(r)$ and also calculate $\bar{\Gamma}$ for comparison. M is increased until approximately

$$\frac{\left|\bar{\Gamma} - \mathrm{Var}(r)\right|}{\mathrm{Var}(r)} < 0.05, \tag{2.32}$$

i.e. the percentage error of $\bar{\Gamma}$ is less than 5 percent. This is done for d from 2 to 11.

Some of the results are shown in Figures 2.10–2.12. To achieve the desired 5% error, the Gamma test only requires about 700 data for $d = 3$ whereas for $d = 11$, it requires over 10000 data. In fact, the error still fluctuates after 10000 data for $d = 11$. Therefore it appears that for higher d, the larger M is required to approximate the variance of noise. However, very much larger values of M are required to stabilise the gradient A.

Of course, this is for *uniformly* distributed input data. For many of the examples of interest the support of the sampling distribution in input space has small Hausdoff dimension. In such cases far fewer data samples are required to get an accurate estimate.

To summarise, if there are enough data available, the Gamma test can capture very well the variance of noise, $\mathrm{Var}(r)$ as shown in Figures 2.10–2.12. To have a better estimate of A for the same problem, a

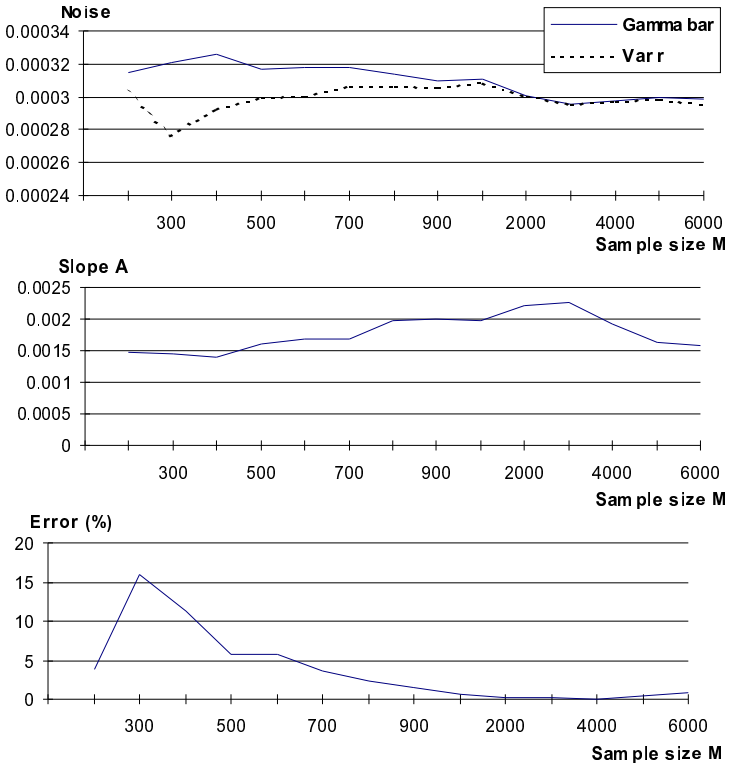


Figure 2.10: Gamma test experiment result for $d = 3$.

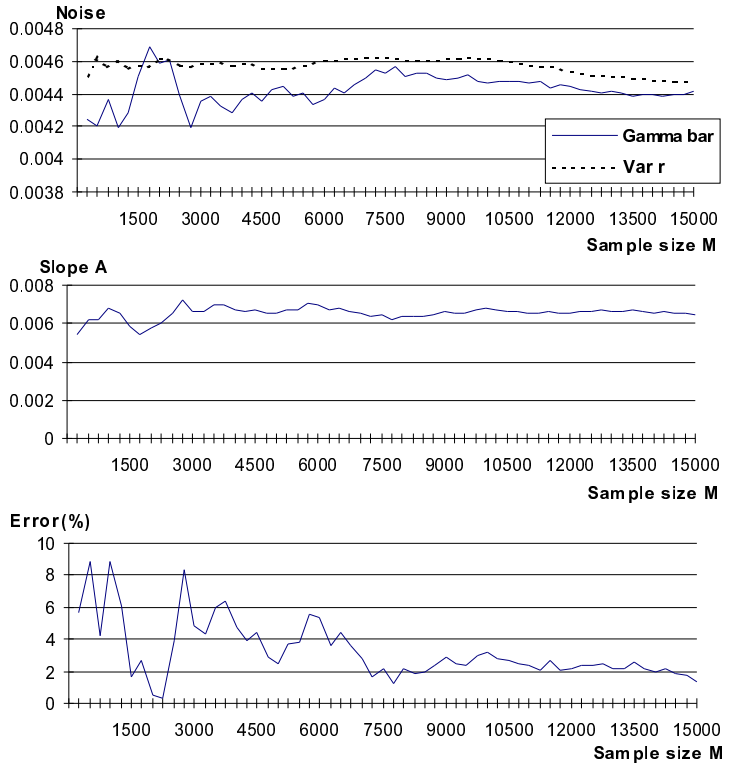


Figure 2.11: High-dimensional Gamma test result for $d = 9$.

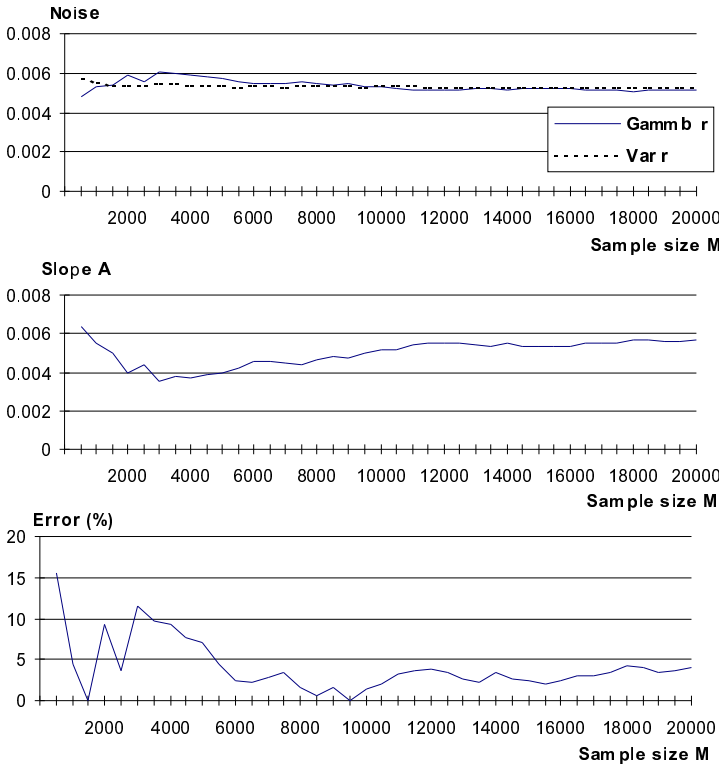


Figure 2.12: High-dimensional Gamma test result for $d = 11$.

much larger M is required. For simpler surfaces the number M required is much smaller. The practical usefulness of an accurate A is discussed next.

2.4 Analysis of relationship between A and neural models

In what follows, we try to discuss the strong relationship between the data and the neural architecture required to model the data, based on experimental observations. The main results and data from these experiments have been presented in [Končar 1997], therefore we only provide brief descriptions and general discussion.

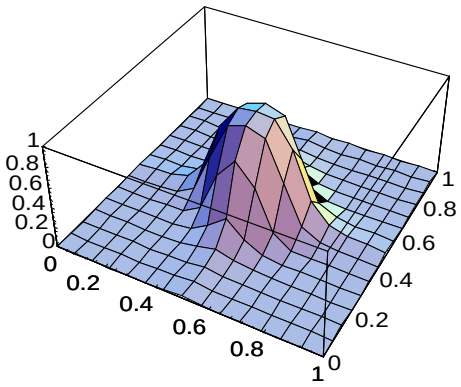


Figure 2.13: One hill (2-4-1).

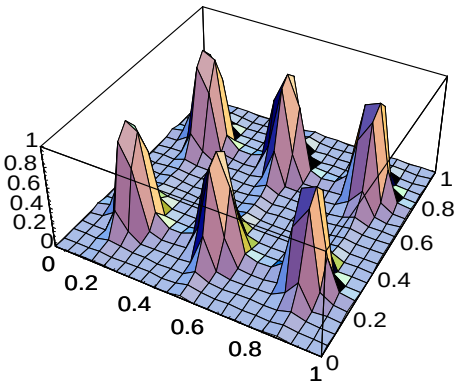


Figure 2.14: Six hills (2-10-1).

We try to demonstrate the correlation between A, calculated from a data set in which the underlying

model was supposed unknown, and the ideal minimum number of hidden layer neurons required to attain the best (neural) model possible without over-fitting the data. We assume A is already accurately estimated using large M (the number of data for the Gamma test). Thus we first create a routine which constructs a 2-h-1 neural network with an input/output surface for which the number of hills is pre-specified. This is very simply done and is based on the ideas described in [Lapedes and Farber 1988] as described in Section 2.2, but with only one hidden layer for ease of comparison with the neural network architecture. By placing k ridges in parallel vertically and similarly l ridges horizontally we can obtain an input/output surface, with kl hills using $h = 2(k + l)$ hidden units, for which the output neuron performs the flattening of unwanted elevations. This is a parsimonious use of hidden units to create maximal surface complexity and represents in some sense a 'worst case' scenario. (See Figures 2.13 – 2.14)

The next step is to create an experimental input/output data set by selecting an input vector at random and then propagating these inputs through the ideal network to obtain the associated output. In this way we can generate as many points in our data set as we wish, so that we can perform the Gamma test on these data to obtain A. Then the relationship between the number of hidden units of the neural networks and the value A for a fixed size of sampled data can be compared. We concluded that there was a reasonably linear correlation between the slope A and the number of hidden units.

Similarly, we also extend the experiment with 3 inputs and 1 output neural network. Instead of using the neural network to construct surfaces, we construct 3D density field as illustrated in Figure 2.15, which employs both a grey scale and variations in point-size to indicate the variation of the output. Again, we could see a near linear correlation between A and the number of hidden nodes.

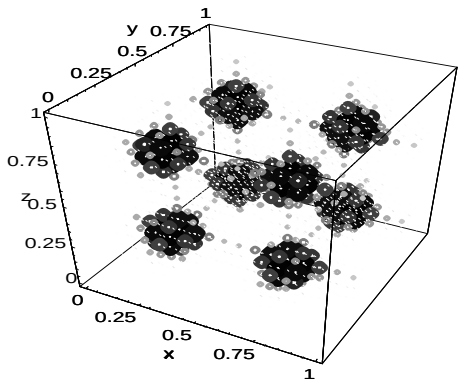


Figure 2.15: Discrete approximation to the 3D density field output of a 3-12-1 neural network.

We next define a class of surfaces for which a single parameter can be varied so as to increase or decrease the complexity. This is easily done by writing

$$f(x, y) = 0.15 + \frac{1 + \sin(p(2x + 3y))}{3.5 + \sin(p(x - y))}. \tag{2.33}$$

Increasing the value of p causes the surface to become progressively more complex. For each surface generated by varying p, 650 data points were sampled and then a fixed amount of additive noise (Gaussian) with a variance of 0.0014 was added. We then performed the Gamma test to determine $\bar{\Gamma}$ and A for this particular p. Using the neural network training software from [Masters 1993], an implementation of conjugate gradient method training algorithm (see also Appendix D), with each node using sigmoidal $1/(1 + \mathrm{Exp}(x))$, and an architecture with one hidden layer and no cross connections, we

trained the feedforward network on the 650 data points in an effort to determine the least number of hidden units required so that the network error could be reduced to the corresponding $\bar{\Gamma}$. Figure 2.16 and Figure 2.17 compare the original surface with that learnt by the neural network using the 650 data points. Considering the added noise the surface shown in Figure 2.17 is a very good approximation to the surface in Figure 2.16. From a number of such experiments, we also observed that there is a near linear relationship between A and the number of hidden units in the 'best' neural network.

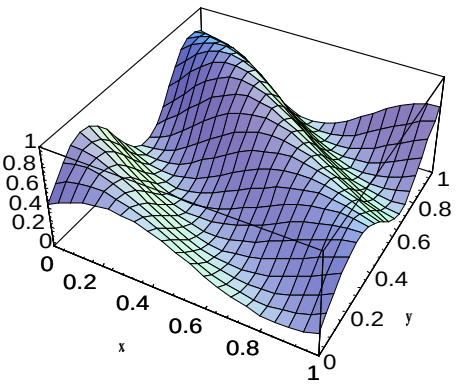


Figure 2.16: $f(x, y)$ when $p = 2.75$.

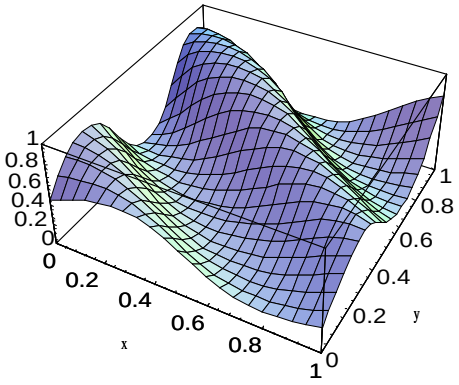


Figure 2.17: Trained neural network output on noisy data for $p = 2.75$.

The potential application of these results to the automated construction of problem specific neural networks is fairly plain. Firstly, if possible one ensures the accuracy of the Gamma test values ($\bar{\Gamma}, A$) by increasing M until these values asymptote to a stable value. Second, using the Gamma test value $\bar{\Gamma}$ enables one to predict beforehand a useful performance metric for the model, which may or may not be a neural network. If a neural network is the chosen tool for constructing the non-linear input/output model then using A we can predict with fair accuracy how many hidden units will be needed in a single hidden layer network. This is discussed further in the next section. Finally, we use the $\bar{\Gamma}$ value to indicate when backpropagation should cease. If the MSE obtained by backpropagation cannot be reduced to $\bar{\Gamma}$ then the number of hidden units should be increased.

The observed strong link between the Gamma test value A and the architecture of the simplest feedforward neural network, which accurately represents the data model, leads to an attempt to use A to estimate the minimal neural network architecture required to model the data. Assuming that we have a uniform distribution of the $\boldsymbol{x}$ data [Končar 1997], then approximately

$$A \approx \frac{1}{4} \left\langle |\nabla f|^2 \right\rangle . \tag{2.34}$$

In principle, we can choose f explicitly and compute $|\nabla f|^2$ at any point and thus the average can be evaluated by

$$\frac{1}{|C|} \int |\nabla f|^2 dx_1 \ldots dx_m, \tag{2.35}$$

where $|C|$ is the volume of the closed bounded data set region.

The idea of using A is to assume that the data will produce a smooth surface of hills and that there is a real function $f : [0, 1]^d \to [0, 1]$ (with bounded output) describing this surface whose complexity can be modified by varying a parameter.

We can choose the function f which is defined as follows:

$$f(x_1, x_2, \ldots, x_d) = \frac{1}{d}\left[\sin^2(a\pi x_1) + \sin^2(a\pi x_2) + \cdots + \sin^2(a\pi x_d)\right]. \tag{2.36}$$

For $d = 1$, the function is

$$f(x) = \sin^2(a\pi x) \tag{2.37}$$

which has one hill for $a = 1$. By increasing a, more hills can be produced (See Figure 2.18). In fact the positive number a is the number of hills within the range input domain $[0, 1]$. This is also true for any $d > 1$. If $d = 2$, this function describes a^2 hills which are distributed evenly on this 2-dimensional surface. (See Figure 2.19).

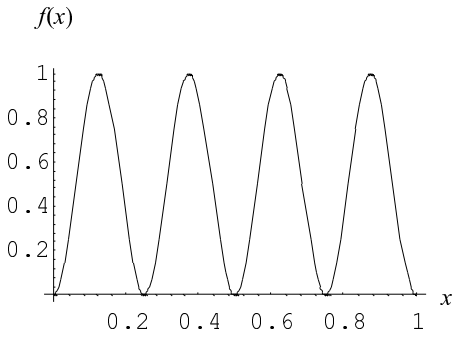


Figure 2.18: $f(x)$ with $a = 4$.

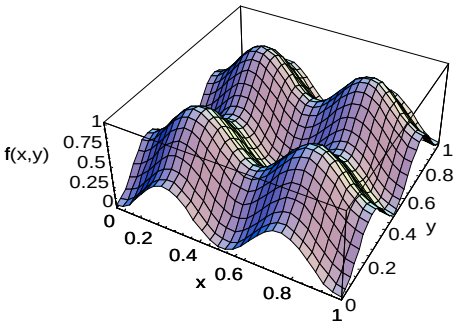


Figure 2.19: Surface of $f(x, y)$, hills produced by Sine.

By expressing the definition of A explicitly with this function f we get

$$\begin{aligned} A \approx \frac{1}{4}\left\langle |\nabla f|^2 \right\rangle &= \frac{\displaystyle\int_0^1 \cdots \int_0^1 \left[\left(\frac{\partial f}{\partial x_1}\right)^2 + \cdots + \left(\frac{\partial f}{\partial x_d}\right)^2 \right] dx_1 \cdots dx_d}{\displaystyle 4 \int_0^1 \int_0^1 dx_1 \cdots dx_d} \\ &= \frac{1}{4}\int_0^1 \cdots \int_0^1 \left[\left(\frac{\partial f}{\partial x_1}\right)^2 + \cdots + \left(\frac{\partial f}{\partial x_d}\right)^2 \right] dx_1 \cdots dx_d. \end{aligned} \tag{2.38}$$

The partial derivatives are given by

$$\frac{\partial f}{\partial x_i} = \frac{a\pi}{2}\sin(2a\pi x_i), \tag{2.39}$$

and as a result we finally get the following equation

$$A \approx \frac{a^2\pi^2}{8d}\left(1 - \frac{\sin(4a\pi)}{4a\pi}\right). \tag{2.40}$$

If a is an integer then $\sin(4a\pi)$ = 0. From (2.40) we get

$$A \approx \frac{a^2\pi^2}{8d} \qquad \text{or} \qquad a \approx 2\sqrt{2}\frac{\sqrt{Ad}}{\pi}. \tag{2.41}$$

For the d inputs case, the number a gives us a^d hills. If we use the recipe for our previous experiments, the number of hidden neurons (for our one hidden layer architecture) required for a given value a is

$$n = D(2d)a \approx \frac{4\sqrt{2}Dd}{\pi}\sqrt{Ad} \tag{2.42}$$

from (2.41), where D is a constant. This can also be used with Lapedes' recipe of architecture for hills [Lapedes and Farber 1988] and the required number of neurons is

$$n = D(2d+1)a^d \approx D(2d+1)\left(2\sqrt{2}\frac{\sqrt{Ad}}{\pi}\right)^d. \tag{2.43}$$

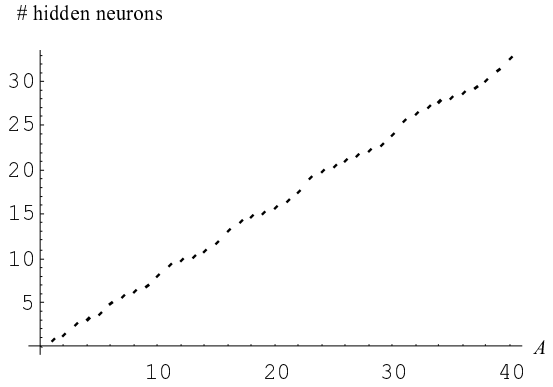


Figure 2.20: Theoretical relationship between #hidden neurons and A.

However, one should notice that the practical results from the experiments do not necessarily match the theoretical one shown in Figure 2.20. This is mainly caused by the choice of the hill function which is *not* exactly the same as the hill generated by 2 neurons (in 1-dim case), therefore it only gives us an approximation of the neural network architecture and A itself is only an approximation. There are several limitations to this technique:

- The choice of f as defined in (2.36) is arbitrary and one does not know which function should be used.
- Every surface to be modelled is assumed to be regularly placed hills and A cannot give the true nature of the surface.
- Although this can be used for higher dimensional input space, the resulting a, from (2.40), degrades accordingly because of the assumption of regular distribution of hills.

However, these limitations can be overcome in certain situations. The choice of f can be replaced by a similar sine function with a higher power than 2 which may give us a stronger similarity with our hill in the experiments. The surface can be assumed to be of some other form with a periodic nature but it should be able to reflect the necessary number of neurons required for a neural network architecture to model the data.

These results are only a preliminary presentation of this modelling technique. Whilst for outputs normalised to $[0,1]$ (or bounded) the slope A returned by the Gamma test gives an estimate of the surface complexity, this value gives no idea how difficult it will be to approximate the surface by a feedforward network using particular sigmoidals. Thus the number of hidden units actually required depends on both surface complexity and the difficulty of approximating the surface using particular sigmoidals. It might be interesting if some quantitative estimate for this second factor could be readily derived from the data, as is the case with the slope A.

2.5 Automated neural network modelling strategy

Using the idea from the last section, on using A from the Gamma test to estimate the complexity of surfaces, Končar [Končar 1997] himself later further developed the application of adaptive feedforward neural network construction to train the network to learn a particular surface based on a given set of data. The technique is named the *Metabackpropagation* and is concisely summarised in Algorithm 2.2.

Procedure: Metabackpropagation(data)
{data is a list of training data, input-output pairs $(\boldsymbol{x}(i), y(i))$, $(1 \leq i \leq N)$, in which $\boldsymbol{x}$ is a real vector of dimension d and y is a real scalar}
Perform initial Gamma test on data to calculate $\bar{\Gamma}$ and A. {Assuming the format of the data will give a good model - further discussion in the next chapter.}

Set $\bar{\Gamma}$ as the training target MSE (mean squared error).
Create an initial feedforward neural network that has number of hills (using Lapedes's recipe in Section 2.2) specified by A using equation (2.43).

First randomise the weights of the whole network.

Initialise each hill by doing a few backpropagation training cycles on subset of the data. Subset of points is chosen from the near neighbour list of the point (information that is required in the Gamma test calculation and is now available) that give the largest error identified by feeding every point through the network. Therefore each hill is trained on its own exclusive set so that each hill is positioned in the right place in the input space.

Set flag $done = false$.
while $done \neq true$ **do**
  Perform backpropagation training on the whole network until either a specified number of cycles is reached or MSE is achieved by the training algorithm.
  if MSE is achieved **then**
    Set $done = true$.
  else
    Create an extra hill.
    Initialise the new hill by backpropagation on subset of points which give the largest MSE.
    Append the new hill into the network.
  end if
end while
return the network

Algorithm 2.2: The Metabackpropagation neural network construction using the Gamma test.

2.6 Discussion

Together with the graphical explanation of the modelling capability of a feedforward neural network [Lapedes and Farber 1988] and the Gamma test [Stefánsson *et al.* 1997], a new idea of using a heuristic on the input data to estimate the necessary feedforward neural network architecture to model such data is introduced. The results presented for this technique so far are preliminary but indicate that the method is quite practical. The approach offers the possibility that the entire process of performance prediction and constructing a feedforward neural network which attains the best achievable performance on the basis of given training data can be automated with a fair degree of reliability.

It is essential to study the Gamma test to understand and improve the *Metabackpropagation* tool

which constructs a neural network with given desired properties as well as in other data modelling applications which are discussed later.

Chapter 3

Smooth data modelling

As we have seen a significant disadvantage of using feedforward neural networks to construct global models of smooth mappings is the often rather drawn out process of training. Whilst it is true that the Gamma test and Metabackpropagation significantly help in this respect, it remains a fact that training a neural network can be a time consuming and somewhat uncertain process. Moreover, it is difficult to use a neural network for *dynamic* modelling, for example in time series prediction we may wish to incorporate newly arrived information into our predictive model. This is often not practical using a neural network model because at every prediction step it requires an unquantifiable amount of time to perform further backpropagation training before making the prediction.

Apart from neural networks there are a wealth of alternative non-linear modelling and prediction techniques available. Putting to one side conventional parametric statistics, on the grounds that such an approach requires we postulate *a priori* the nature of the model, many of the alternatives are based on neighbourhood information elicited from the data.

In the case of time series, for example, we attempt to reconstruct the dynamics using an embedding technique; then the neighbourhoods are neighbourhoods in embedding space. This predictive technique is very intuitive and has an illustrious history in forecasting, Lorenz called it the 'method of analogies'. The idea is that we make a prediction based on historical evidence by asking 'what happened in the past when we saw a *similar* sequence of events'? To implement this idea efficiently we simply recognise that finding sequences of historically similar events exactly corresponds to finding near neighbours in the embedding space i.e. to the construction of a kd-tree.

The advantages of using kd-trees combined with some *local* prediction method are considerable. We can build the kd-tree quite quickly and unlike neural network backpropagation we do not have to worry about long training times, becoming trapped in a local minimum, or over-fitting. We can also update the kd-tree with new information very rapidly. Moreover, once the current near neighbours are known, techniques such as local linear regression can build accurate local models very quickly.

In this chapter we first examine geometrical surface reconstruction techniques, which can be very effective when we are dealing with the low noise case. Here we will exploit the fact that there is no real requirement for a global model. When making a prediction for a *particular* data point it is sufficient to perform a local reconstruction of the surface in the vicinity of the query point. Provided the density of previously seen data points is high near the query point interpolative local reconstruction of the surface can provide an accurate estimate of the output value. The point here is that whilst the scaling properties of a higher dimensional geometrical reconstruction technique may be poor if one is seeking a *global* reconstruction, the same algorithm applied on a *local* basis may be very fast - sufficiently

fast to do the reconstruction 'on the fly' for each query point as it is presented. If the response time required for the final application permits this approach then it has considerable advantages: the set-up (i.e. training) times are reduced to the construction of near neighbour lists and no large data structures need to be retained.

Of course, if the density of data points near the query point is low the problem of producing an accurate prediction becomes much harder. At one level it can be regarded as a problem of *interpolation/extrapolation*, alternatively we might argue that what is truly required is the *recovery of the underlying law/function* which governs the input-output relationship - only then can interpolation/extrapolation by undertaken with any degree of confidence. Global models constructed from data, such as neural networks or geometrical surface reconstructions, cannot make any great claim to handle such situations effectively, although neural networks may on occasions perform better in this situation than straightforward surface reconstruction. The ability of such data models to *generalise* has historically been approached most often by trying to provided some confidence measure associated with a prediction. Indeed, such measures, if reliable, provide an extremely useful enhancement of a predictive model, but in truth do not really address the basic issue of effective generalisation.

After a prior discussion of kd-trees we first examine the area of geometrical surface reconstruction algorithms with particular reference to higher dimensional reconstruction and computational complexity. We select the algorithm which seems most appropriate as a general purpose interpolative surface reconstruction technique and implement it for comparison with alternative techniques designed explicitly to deal with higher noise levels.

To deal with noisy data we need to develop techniques which employ statistical cancellation in some manner. Two rather obvious points need to be made. Firstly, if the data is modelled by

$$y = f(x_1, \ldots, x_d) + r \tag{3.1}$$

where f is smooth and r is statistical noise with $\mathrm{Mean}(r) = 0$, then any particular prediction made using f is going to have an error statistically determined by the variance $\mathrm{Var}(r)$. If $\mathrm{Var}(r)$ is large then the expected error will be large, no matter how accurately f is known. Second, if despite this we seek to extract an accurate approximation for f, then the only viable way to proceed is to attempt to reconstruct f by statistically cancelling the high noise level. Put plainly:

- High noise levels require more data.

One reason feedforward neural networks take a long time to train under high noise levels (in the data) is that backpropagation only cancels noise as a by-product of the algorithm. Such cancellation is not efficiently performed and becomes relatively less efficient as the complexity of the surface increases.

We shall examine one noise cancelling technique based directly on the Gamma test: the Gamma-minimum-predictor. A detailed examination of the underlying rationale shows the Gamma-minimum-predictor is effectively performing local linear regression on the squared distance of near neighbours. Once this is appreciated it seems natural to ask: why do local linear regression of the squared distances (and thereby throw away directional information in the input space) when one can almost as easily do local linear regression on the data coordinates themselves? In retrospect this seems quite obvious, but we include the work on the Gamma-minimum-predictor out of interest (since it performed surprisingly well under the circumstances) and because it provides a useful comparison with the more general local linear regression technique which we have found to be very useful.

Local linear regression, although well known, has no been much exploited in the neural network community. This seems rather surprising since one might expect that it is the natural *computer science*

algorithm with which to compare a feedforward network - at least in terms of performance. For very large data sets the kd-tree construction and extraction of near neighbours could easily be parallelised (although in practice this is not usually necessary) so that neural networks offer little or no advantage in that particular respect.

One intriguing possibility which emerges from this study is that, if one is determined to build a neural network model under high noise levels, it might well be advantageous to pre-process the training data using local linear regression to effect noise cancellation and hence generate new (much smoother) training data for the network - thereby substantially reducing the training time.

3.1 Extracting local near neighbours information

In order to perform *local analysis* of data in the vicinity of a particular query point, it is necessary to first extract a list of near neighbours of the point. This enables us to examine the output values associated with the neighbours, and other information such as the distances of the near neighbours from the query point, and hence which neighbour is the nearest etc.

Finding such neighbours can be regarded as the computational geometry problem of *range searching*. The typical way to solve this problem is to process the data into some type of data structure. One of the first data structures for range searching was the *quadtree* [Finkel and Bentley 1974]. Shortly after quadtrees, *kd-trees* [Bentley 1975; Finkel *et al.* 1977] were developed as an improvement in terms of worst-case behaviour. Later, another data structure, the *range tree* was discovered [Lueker 1978; Bentley 1979; Lee and Wong 1980].

The kd-tree is the structure that has been implemented for the Gamma test and is used there to extract lists of near neighbours. Since we probably want to examine our data using the Gamma test it is natural to use the already created kd-tree as the basis for our local data modelling. Therefore, in what follows we shall describe in some detail the construction and use of kd-trees and omit any discussion of alternative methods, descriptions of which are readily available in the literature.

3.1.1 Fast nearest neighbours search using a kd-tree

The *kd-tree* is a data structure for storing M data points allowing for logarithmic expected time searching for the nearest neighbours of the given query point from these M points distributed in a d-dimensional space [Bentley 1975; Finkel *et al.* 1977]. Originally the name kd-tree stood for k-dimensional tree but we will instead use d, in line with the rest of this work, to denote the dimension of the input space. Our discussion and implementation of kd-tree construction and querying is based on [Margetts 1996].

Building the kd-tree

Before we can find the $p_{\max}$ nearest neighbours of a query point, we first need to construct the kd-tree from the set of M input data points. The kd-tree can be built recursively. The data points are stored at the leaves of the tree. Each leaf node, or *bucket* can contain a maximum number (called the *bucket size*) of points. The bucket size is used to determines whether or not to call the building procedure recursively.

If the number of data points being processed at this instance is less than the bucket size, the data set is simply returned to be a leaf node. If it is larger, we create an internal node by splitting this data set

into two subsets and each will become a subtree for this node. The split is performed by determining which component of the data points gives the largest "spread", found by searching through all the values of the i^{th} component of the data and recording the maximum and the minimum values for $1 \leq i \leq d$. Then the median of the values for the i^{th} component which gives the largest spread is extracted. We can then partition the data around this median value so that the data points will be evenly distributed. This kind of fair splitting allows the tree to be constructed in time $O(M \log M)$ provided that p_{max} remains bounded. Of course, if $p_{\text{max}} = O(M)$ the time complexity is unavoidably $O(M^2)$.

The structure of the kd-tree can be formatted as follows

kd-tree ::= {*component index, median value,* **kd-tree**, **kd-tree**}
[{*data point, identifier*}, . . .]

where the two **kd-tree** references in the definition are the left and right branches of the tree. We need to give an identifier for each data point in order to distinguish them, because the order of the data points will be disrupted by the tree building process. A summary of the building procedure is given in Algorithm 3.1.

Procedure: buildkdTree(d, *dataPoints***)**
{Assuming that *bucketsize* is set to a fixed value.}
if sizeof(*dataPoints*) $\leq$ *bucketsize* **then**
  {a leaf node}
  return *dataPoints*
else
  {a non-terminal node}
  i = index of component of *dataPoints* which gives the largest spread
  median = the median of the i^{th} component values of all *dataPoints*
  if *median* == the lowest or the highest i^{th} components of *dataPoints* **then**
    {repeated values exist, return data without splitting}
    return *dataPoints*
  end if
  split *dataPoints* into two sets, *dataA* (i^{th} component $<$ *median*) and *dataB* (i^{th} component $\geq$ *median*)

  return {i, *median*, **buildkdTree(**d, *dataA***)**, **buildkdTree(**d, *dataB***)** }
end if

Algorithm 3.1: Build kd-tree of d-dimensional data recursively.

Searching for nearest neighbours

Once the kd-tree has been built, we can use it to find the nearest neighbours for any query point. The simple search routine first searches through the tree to find the closest leaf node to the query, it then repeatedly searches outward from this leaf node to the next nearest terminal node until the required number of nearest neighbours is reached.

The searching routine is recursive and it relies on several important global variables used within the searching procedure:

nearest A priority queue of the nearest neighbours found so far. It is basically a list of pairs containing the data identifier and its corresponding distance from the query point. These pairs are sorted in

increasing distance from the query point, so that the furthest near neighbour always lies at the end of the list. Before calling the search procedure, all distances are initialised to ∞;

lowerBounds This is a set of d lower bounds currently defining the *lower* edges of the search region. These bounds are set to $-\infty$ before each nearest neighbour search;

upperBounds This is a set of d upper bounds currently defining the *upper* edges of the search region. These bounds are set to ∞ before each nearest neighbour search.

It is also necessary to define a boolean test to determine if the search can terminate. This is performed by checking if the geometric boundaries of the branch of the kd-tree under consideration are closer than the furthest nearest neighbour found so far. The branch being considered can only be ignored if the "spherical" region covered by the furthest nearest neighbour distance centred at the query point does not overlap with the potential search region defined by the **lowerBounds** and **upperBounds**. Otherwise, that branch has to be searched. This procedure is described by the pseudo code in Algorithm 3.2. This ability to ignore sections of the tree allows us to perform the search in time proportional to $O(\log M)$.

Procedure: boundsOverlapBall(d, *query***)**
total $= 0$
for each i^{th} component, $i \leq d$ **do**
  if *query*$[i]$ $<$ **lowerBounds**$[i]$ **then**
    total = *total* + $(\textit{query}[i] - \textbf{lowerBounds}[i])^2$
    if *total* $>$ the square of the furthest distance in **nearest then**
      return FALSE
    end if
  else
    if *query*$[i]$ $>$ **upperBounds**$[i]$ **then**
      total = *total* + $(\textit{query}[i] - \textbf{upperBounds}[i])^2$
      if *total* $>$ the square of the furthest distance in **nearest then**
        return FALSE
      end if
    end if
  end if
end for
return TRUE

Algorithm 3.2: The nearest neighbours search termination boolean test.

The kd-tree search algorithm is divided into the case for handling the leaf node and the case for the non-terminal node. Whenever a leaf node is encountered, the data points within it are added to the priority queue, **nearest**, if their distances from the query node are less the current furthest nearest neighbour. If the query node itself is encountered, we may or may not, depending on the user's need, include it in the list of nearest neighbours. If the node is non-terminal, we search the branch closer to the query node. Then the further branch is searched on backtracking if it is required by using **boundsOverlapBall** boolean test in Algorithm 3.2.

The searching routine is described in detail in Algorithm 3.3 which is similar to Bentley's pseudo-code [Bentley 1975]. The building and searching of the kd-tree is a frequently used and important technique for extracting nearest neighbours of a given query from a set of points. This fast extraction of local near neighbour information enables the Gamma test to run in $O(M \log M)$. The near neighbour information from the kd-tree is the essential building element of various local modelling techniques which are discussed in what follows.

```
Procedure: searchkdTree(d, query, tree)
if tree is a terminal node/bucket then
   for each point p at tree do
      calculate distance of p from query
      if distance is less the distance of the last point of nearest then
         update nearest with p in increasing order of distances
      end if
   end for
else
   {non-terminal node; refer to the main for its structure}
   i = tree[1] {the index i for splitting at this node}
   median = tree[2] {median value for splitting at this node}
   {recursive call on closer branch}
   if query[i] < median then
      t = upperBounds[i]
      upperBounds[i] = median
      searchkdTree(d, query, tree[3]){left branch}
      upperBounds[i] = t
   else
      t = lowerBounds[i]
      lowerBounds[i] = median
      searchkdTree(d, query, tree[4]){right branch}
      lowerBounds[i] = t
   end if
   {recursive call on further branch if necessary}
   if query[i] < median then
      t = lowerBounds[i]
      lowerBounds[i] = median
      if boundsOverlapBall(d, query) then
         searchkdTree(d, query, tree[4]){right branch}
      end if
      lowerBounds[i] = t
   else
      t = upperBounds[i]
      upperBounds[i] = median
      if boundsOverlapBall(d, query) then
         searchkdTree(d, query, tree[3]){left branch}
      end if
      upperBounds[i] = t
   end if
end if
```

Algorithm 3.3: The nearest neighbours search recursive procedure.

3.2 Geometrical local data modelling

We can think of modelling a section of some 'surface' as constructing a terrain, this is often encountered in computer flight simulation, geographical visualisation of the Earth's surface and many other 3-dimensional object modelling applications. Such a *terrain* is basically a 2-dimensional surface in 3-dimensional space with the property that every vertical line intersects the surface at a point, if it intersect it at all. Alternatively in mathematical language, the terrain is the graph of a *function* $f : \mathcal{A} \subset \mathbb{R}^2 \rightarrow \mathbb{R}$ that assigns a height $f(p)$ to every point p in the domain $\mathcal{A}$ of the terrain. For modelling purposes we only know the value of the function f at a finite set $P \subset \mathcal{A}$ of sample points. From the heights of the sample points we need to approximate the height at other points in the domain in such a manner as to give a 'smooth' surface.

The technique already commonly used by 3D graphics programmers is to first determine a *triangulation* of P, a planar subdivision whose bounded faces are triangles and whose vertices are the points of P, assuming that the sample points are such that we can make the triangles cover the domain of the terrain. We can then lift each sample point to its correct height, thereby mapping every triangle in the triangulation to a triangle in 3-space. This results in a *polyhedral terrain* - a graph of a continuous function that is piecewise linear - as an approximation of the original terrain, as illustrated in Figure 3.1. The remaining problem is, given only the heights of the sample points, to determine the appropriate triangulation, such that the terrain looks 'natural'.

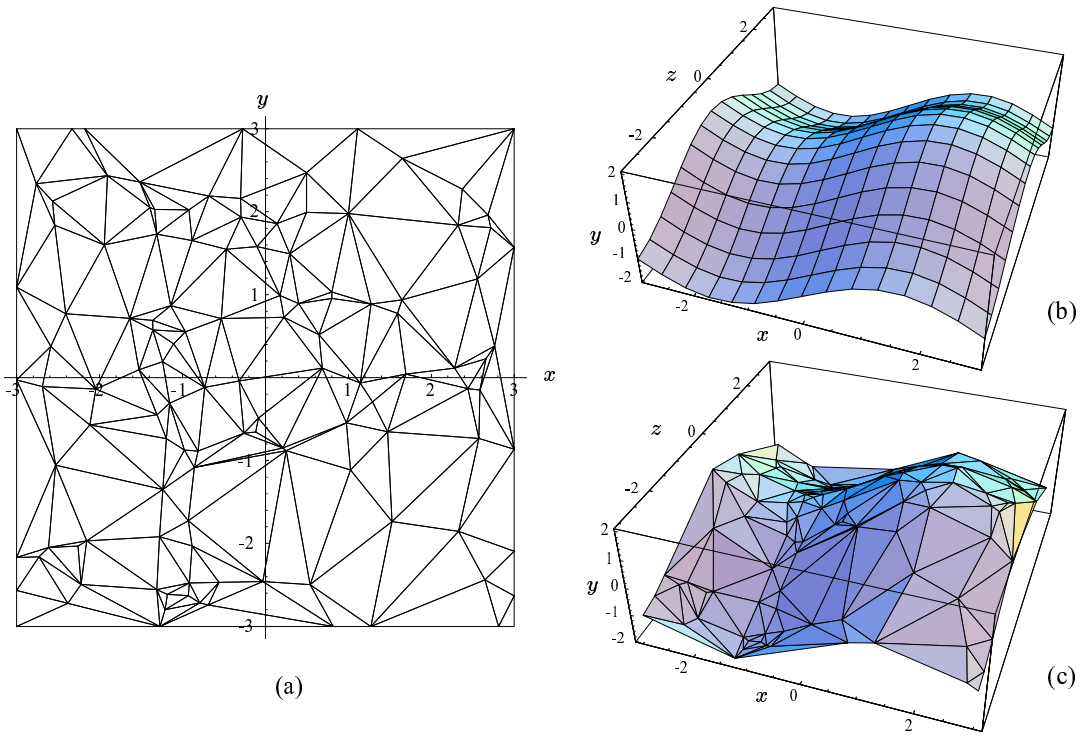


Figure 3.1: Reconstruction of a terrain with triangulation: (a) triangulation of 120 sample points on the xy-plane; (b) the surface $z = \sin(x) + \cos(y)$ to be modelled; (c) the reconstructed terrain based on the triangulation by lifting each sample point to its correct height.

It turns out that a triangulation that contains small angles is bad, because that means we are using 'skinny' triangles to give the approximation of the height of a point. For example, consider the point q, as shown in Figure 3.2, determined by two points that are relatively far away. In fact the same point

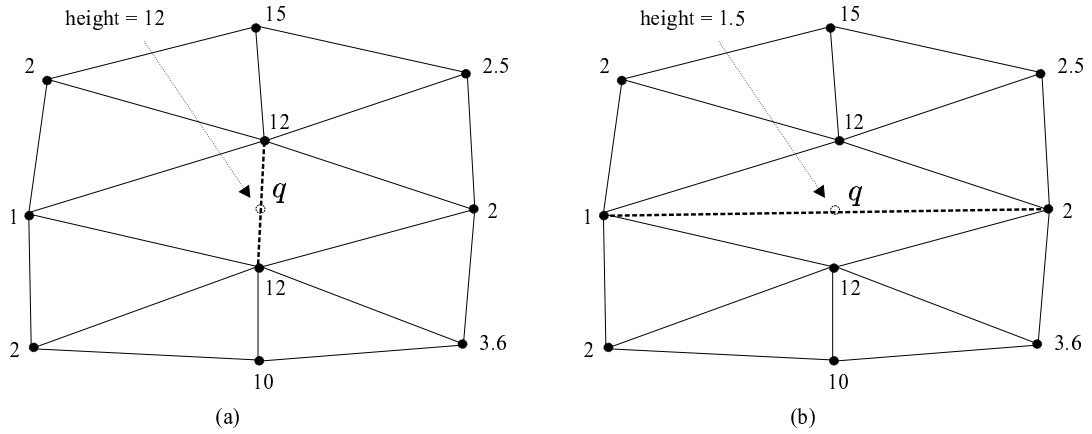


Figure 3.2: Difference of approximate height at q on a high ridge running from North to South by flipping one edge: (a) a near approximation to the true height by two near sample points; (b) a bad approximation of height by two sample points that are relatively far apart.

q can be better approximated by using two nearer sample points. Therefore we can rank triangulations by comparing their smallest angles. If the minimum angles of two triangulations are identical, then we can look at the second smallest angle and so on. Since there are only a finite number of different triangulations of a given set of points P, this means that there must be a optimal triangulation - this is the triangulation we seek and it maximises the minimum angle. This optimal triangulation is called the *Delaunay triangulation* and we shall shortly discuss it in some detail.

3.2.1 Triangulations of point sets

Before we fully explain our geometrical approach to modelling and prediction, a brief but formal introduction to some fundamentals of computational geometry is necessary. For simplicity we shall introduce the ideas in the planar space of 2 dimensions and then extend the definitions to higher dimensions where necessary.

First, let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane and define a *maximal planar subdivision* as a subdivision S such that no edge connecting two vertices can be added to S without destroying its planarity. So a *triangulation* of P is defined as a maximal planar subdivision whose vertex set is P.

Delaunay triangulation attempts to achieve small and approximately equilateral triangles. If we then use the triangles as the basis for interpolating the height for an unknown point we can expect to get better accuracy using the Delaunay triangulation than by simply selecting some arbitrary enclosing small triangular region.

We next provide a more precise mathematical definition of Delaunay triangulation. Let n be the number of triangles in a triangulation and consider the sequence of the angles of the triangulation, T given as $(\alpha_1, \ldots, \alpha_{3n})$, and sorted in order from the smallest to largest. Let a sequence of angles of some other triangulation T^* be given as $(\alpha_1^*, \ldots, \alpha_{3n}^*)$. Then we can define the relationship $T \geq T^*$, between any two triangulations to indicate that the angle sequence of T is lexicographically greater than the angle sequence of T^*.

We then obtain the following theorem, stated here without proof (see [Edelsbrunner 1987]).

Theorem 3.2.1. *The* Delaunay triangulation *is* maximal *over all possible triangulations in the sense that* $T \geq T^*$.

Before we provide an alternative definition of the Delaunay triangulation let us introduce the *Voronoi diagram*. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n distinct and not all collinear points in the Euclidean plane, and let dist(p_i, p_j) be the Euclidean distance between point p_i and p_j. Then the *Voronoi diagram* Vor(P) of P is defined as the subdivision of the plane into n cells for each *site* (point) in P, with the property that a point q lies in the cell corresponding to a site p_i if and only if dist$(q,p_i) <$ dist(q, p_j) for each $p_j \in P$ with $j \neq i$. Very often these cells are referred as the *Voronoi polygons*.

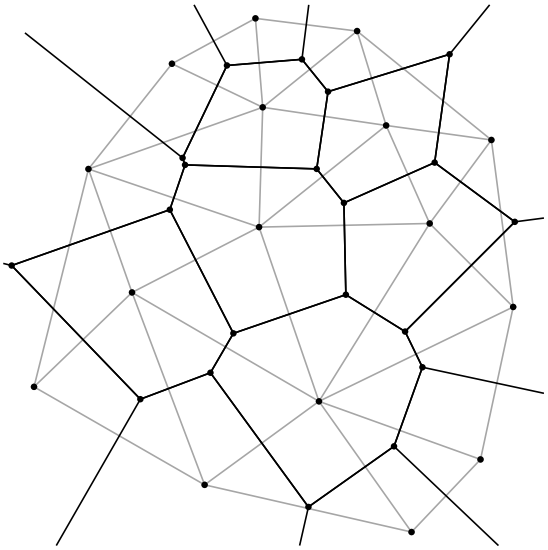


Figure 3.3: Voronoi diagram with Voronoi polygons indicated by black connected lines and its dual Delaunay triangulation indicated by grey/lighter connected lines.

Delaunay [Delaunay 1934] has shown that the dual of the Voronoi polygons is a triangulation of the n points. If we draw line segments between every two points in P whose Voronoi polygons have a common border of length greater than zero, see Figure 3.3, then, under the assumption that no four points are co-circular (as a result that all vertices of the Voronoi diagram have degree three), this triangulation is the uniquely defined Delaunay triangulation.

Therefore, an alternative definition of a Delaunay triangulation is a triangulation where the circumscribed circle of any triangle contains no point of P in its interior. Define a set of points to be in *general position* if it contains no four points on a circle. Then the Delaunay triangulation of P is *unique* if and only if the resulting graph of P is a triangulation, which is the case if P is in general position. See Figure 3.4 for an example. Without such uniqueness of the definition, we will start having problems similar to Figure 3.2 due to the flipping of an edge in reconstructing the surfaces.

Several definitions are required. A set of points S is a *convex set* if the line segment joining any pair of points in S is wholly contained in S. The *convex hull* of a set of points is the smallest convex set that contains the points. In a d-dimensional space, d non-collinear points define a *facet*. A *d-simplex* is a d-polytope forming the convex hull of $d+1$ affinely independent points. In fact, a d-simplex can be thought of as a polytope constructed by $d+1$ facets, with each facet defined by d points from the set of $d+1$ points specifying the simplex. The points defining the simplex are called the *vertices*. The boundary elements of a facet are called the *ridges*. Each ridge is basically an element defined by $d-1$ points. A ridge in fact signifies the adjacency of two facets. In the space $\mathbb{R}^3$ generally, facets are triangles and ridges are edges and therefore a 3-simplex is a tetrahedron. A Delaunay triangulation in

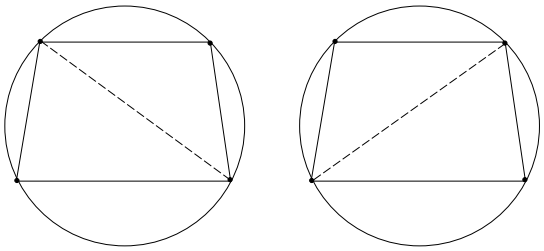


Figure 3.4: An example of two different maximal triangulations of the same point set of four points on a circle, therefore no unique Delaunay triangulation exists.

$\mathbb{R}^3$ is a therefore a subdivision of the space into tetrahedrons, as opposed to triangles, in $\mathbb{R}^2$.[1]

Thus the idea of a Delaunay triangulation of a set of points may be generalised for any d-dimensional real space.

Our requirement is for an accurate and correct Delaunay triangulation algorithm in higher dimensions. In fact, triangulation in higher dimension is still an actively researched area. Most higher dimensional Delaunay triangulation algorithms may occasionally lead to a triangulation which is not quite the Delaunay triangulation - this can occur when almost collinear points are misclassified due to fixed precision floating point calculations.

In our case, for data modelling, having a precise Delaunay triangulation of points in input space is desirable but is not always essential. The triangulation will be used to infer the value we are trying to predict. If the triangulation is not optimal it may have the effect of making the prediction slightly less accurate, but in most cases the effect will be negligible. As long as the input sample points available are dense and/or evenly distributed in some sense, this technique seems to perform very well even without the assurance of a perfect Delaunay triangulation of points in the input space.

3.2.2 Computing the Delaunay triangulation

Because we interested in having Delaunay triangulation incorporated into our data modelling scheme, we seek a readily available but efficient Delaunay triangulation algorithm. In this section, we give a brief introduction to the computational geometry, without going into details for every algorithm mentioned. However, we will provide a clear description of the algorithm finally selected to provide the Delaunay triangulation module used in our data modelling.

There are many alternative Delaunay triangulation algorithms, but most of them are designed specifically for 2D or 3D problems. However, for our purposes we need a quite general Delaunay triangulation algorithm that can work in any dimension d. It emerges that with this constraint the choice is very limited. In fact finding such an efficient algorithm is still an actively researched area.

Su [Su 1994; Su and Drysdale 1997] has given an excellent survey of a variety of the standard 2D and 3D algorithms. Examples are *divide-and-conquer* methods [Guibas and Stolfi 1985; Dwyer 1987] which use special data structures to break down the problem into smaller sub-problems and then combine the sub-solutions to obtain the required solution; the *sweepline algorithm* by Fortune [Fortune 1987]; *incremental techniques* based on adding sites to the diagram one by one and updating the diagram after each site is added [Clarkson and Shor 1989; Guibas and Stolfi 1985; Guibas *et al.* 1992]; and *gift wrapping* algorithms which start with a single Delaunay triangle and then incrementally

[1] However, regardless of the dimension we may still refer to such a d-simplex in a Delaunay triangulation as a triangle.

discover valid Delaunay triangles one at a time [Dwyer 1991; Maus 1984]. We shall not discuss these algorithms further.

Algorithms for calculating Voronoi diagrams can also be used. In fact, many Delaunay triangulation algorithms to which we have referred are derived from the corresponding Voronoi diagram algorithms. These algorithms were taking the advantage of the dual relationship between Delaunay triangulation and the Voronoi diagram.

We have chosen to base the calculation technique on the use of a convex hull algorithm - the *Quickhull* (sometimes *Qhull* for short) by [Barber *et al.* 1996]. This is described by Su [Su and Drysdale 1997] as a stable but relatively slow algorithm if used in planar Delaunay triangulation. Barber's Qhull, is essentially a combination of the classical 2-dimensional divide-and-conquer Quickhull algorithm and the general dimension *Beneath-beyond algorithm* (to be described shortly), an algorithm that works in any dimension. Using Qhull we can construct a higher dimensional Delaunay triangulation algorithm based on the following observation.

The Delaunay triangulation in $\mathbb{R}^d$ may be computed from a convex hull in $\mathbb{R}^{d+1}$ [Brown 1979]. To determine the Delaunay triangulation of a set of points $\boldsymbol{x} \in \mathbb{R}^d$, we first lift the points to a paraboloid using the transformation

$$\boldsymbol{x} = (x_1, x_2, \dots, x_d) \mapsto \boldsymbol{x}' = (x_1, x_2, \dots, x_d, x_1^2 + x_2^2 + \cdots + x_d^2) \tag{3.2}$$

We then compute the convex hull of these transformed coordinates $\boldsymbol{x}' \in \mathbb{R}^{d+1}$. The set of *ridges* of the *lower convex hull* is the Delaunay triangulation of the original points in $\mathbb{R}^d$. This means that the wealth of convex hull algorithms can be directly applied to compute Delaunay triangulations (as well as high-dimensional Voronoi diagrams).

3.2.3 Qhull - a convex hull algorithm

Quickhull (or Qhull for short and not to be confused with the classical quickhull in 2D) is the convex hull algorithm [Barber *et al.* 1996] selected as the basis of our method to compute Delaunay triangulations. The motivations for this choice were: algorithmic stability, availability of source code library [Barber *et al.* 1996] and applicability in any number of dimensions.

This algorithm uses two main operations, *oriented hyperplane through d points* - a hyperplane represented by its outward-pointing (pointing away from the convex hull) unit normal and its offset from the origin, and *signed distance to hyperplane* - the 'signed distance' of a point to a hyperplane is the inner product of the point and the normal plus the offset. The hyperplane defines a halfspace of points that have distances from the hyperplane with an extra attached *sign*. If the attached sign is *negative* we shall say the point is *below* the hyperplane. If the sign is *positive* we shall say the point is *above* the hyperplane. See Figure 3.5 for illustration.

Not unlike other randomised incremental algorithms, Qhull's incremental processing technique is based on the theorem by [Grünbaum 1961, Theorem 5.2.1]. It uses a simplified version given below.

Theorem 3.2.2 (Simplified Beneath-Beyond). *Let H be a convex hull in $\mathbb{R}^d$ and let p be a point in $\mathbb{R}^d - H$. Then F is a facet of the convex hull* $\operatorname{conv}(p \cup H)$ *if and only if*

1. *F is a facet of H and p is* below *F; or*
2. *F is not a facet of H and the vertices of F are p together with the vertices of a ridge R of H, such that there is one facet of H which contains R (H is incident to R) lying* below *p and all other facets of H containing R (i.e. facets incident to R) lie* above *p (see Figure 3.6).*

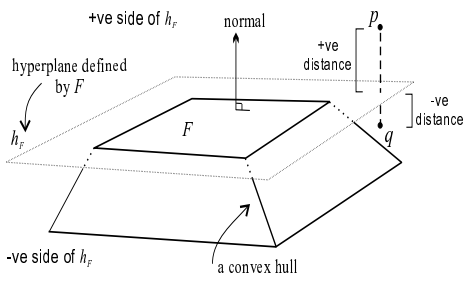


Figure 3.5: p is *above* F (or p is visible to F and vice versa) and q is *below* F (not visible to F) because q is on the *negative* side of the hyperplane h_F defined by *outward pointing* normal of F.

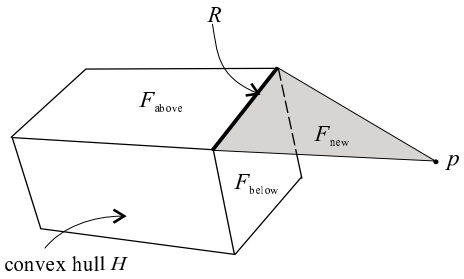


Figure 3.6: A new facet F_{new} defined by a new point p of the convex hull contains ridge R. One facet F_{below} that is incident to R is below p and the other facet F_{above} that is incident to R is above p.

An example of a change to the convex hull effected by introducing a new point is illustrated in Figure 3.7. Efficiently determining the facets which are *visible* from a point (the point is *above* the facets) is the central problem of the Beneath-Beyond theorem. A clever technique which sets Qhull apart from the other incremental algorithms is that after initialisation, it assigns each un-processed point to an *outside set* of a facet, or by definition, the corresponding facet is visible from the point.

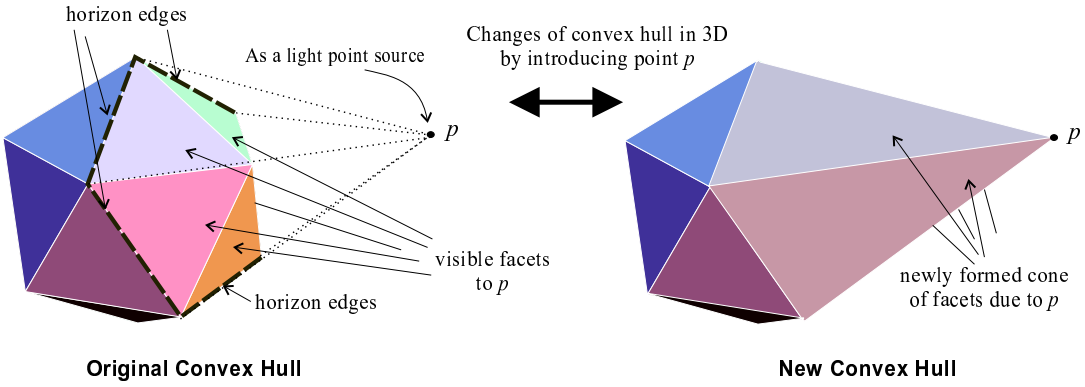


Figure 3.7: Incremental construction of a convex hull in 3D.

When Qhull constructs a cone of new facets, it uses a *partitioning* technique to build new outside sets from the outside sets of the visible facets by locating a new visible facet for each point. If the point is below all of the new facets, the point is inside the convex hull and can be removed. At the same time, partitioning records the furthest point of each outside set. At initialisation, Qhull selects a non-degenerate set of points, which if possible should be far apart, for the initial starting simplex. The full outline of Qhull is given in Algorithm 3.4.

Thus Qhull algorithm works in $\mathbb{R}^d$ and via the lower facets of the convex hull in $\mathbb{R}^{d+1}$ of the *transformed* coordinates, the Delaunay triangulation of the points in the input space can easily be calculated.

Although Qhull cannot be described as a fast algorithm for low dimensional problems, it can work in any dimension d and produces results comparable with other similar algorithms. Now let M be the number of input points in $\mathbb{R}^d$, r be the number of processed points (i.e. the number of the randomly selected points used for the 'cone' construction process of the convex hull construction) and $f_r = O(r^{\lfloor d/2 \rfloor}/\lfloor d/2 \rfloor!)$ the maximum number of facets of r vertices [Klee 1966]. We also define

Procedure: Convex hull in $\mathbb{R}^d$
{Given a set P of M ($M > d$) data points.}
select $d+1$ points to construct a starting simplex S
for each facet F of S **do**
  for each unassigned point p of P **do**
    if p is *above* F **then**
      assign p to F's *outside set*
    end if
  end for
end for
for each facet F with non-empty *outside set* **do**
  select furthest point $\boldsymbol{p}$ of F's *outside set*
  initialise *visible set* V
  assign F into *visible set* V
  for all unvisited neighbours B of facets in V **do**
    if $\boldsymbol{p}$ is *above* B **then**
      add B into V
    end if
  end for
  create H, a set of horizon ridges of boundary of V
  for each ridge R in H **do**
    create a new facet from R and $\boldsymbol{p}$
    link the new facet to its neighbours
  end for
  for each newly created facet F' **do**
    for each unassigned point q in an *outside set* of a facet in V **do**
      if q is above F' **then**
        assign q to F''s *outside set*
      end if
    end for
  end for
  remove the facets in V
end for

Algorithm 3.4: The Qhull algorithm for convex hull construction in $\mathbb{R}^d$

Definition 3.2.1. *An execution of Qhull is* balanced *if*

- *the average number of new facets for the j^{th} processed point is df_j/j and*
- *the average number of partitioned points for the j^{th} processed point is $(M-j)d/j$.*

Then if the balance condition holds, the worst-case complexity of Qhull is $O(M \log r)$ for $d \leq 3$ and $O(Mf_r/r)$ for $d \geq 4$ [Barber *et al.* 1996]. If $r = M$ and the balance conditions hold, the cost of Qhull is $O(M \log M)$ for $d \leq 3$ and $O(f_M)$ otherwise. This is the same as the expected cost of the randomised incremental algorithms [Clarkson *et al.* 1993].

3.2.4 Query estimation using local Delaunay triangulation

Now that we have the Delaunay Triangulation of the input space of sample points we can easily model a surface. The original problem of 'terrain' modelling is basically to model a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, usually in a restricted closed bounded domain $\mathcal{C} \subset \mathbb{R}^2$. Given an unseen data point $q = (x_0, y_0) \in \mathcal{C}$, we can easily calculate q's corresponding output z_0 (or height in the 'terrain' problem) by performing linear interpolation using the 3 sample points, say, $p_1 = (x_1, y_1, z_1)$, $p_2 = (x_2, y_2, z_2)$ and $p_3 = (x_3, y_3, z_3)$ corresponding to the triangle which is calculated via Delaunay triangulation and which *encloses* the query data in the input space as shown in Figure 3.8.

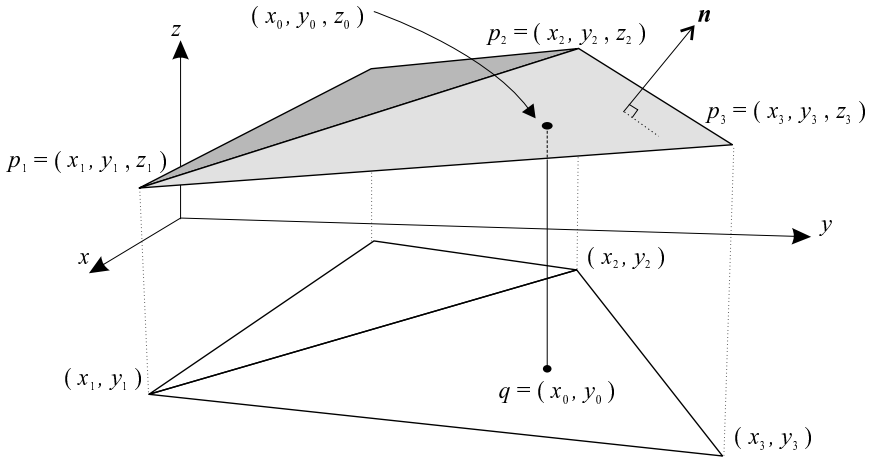


Figure 3.8: Estimating the height z_0 at the query point $q = (x_0, y_0)$ by linear interpolation given that the sample points p_1, p_2 and p_3 form a triangular and enclosing q in 2D input space and a hyperplane in 3D.

In effect, these 3 points p_1, p_2 and p_3 define a triangular 'linear' surface which can be easily determined by calculating its normal using these 3 points. Suppose the surface (or hyperplane in 3D) is given by

$$n_1 x + n_2 y + n_3 z = c, \tag{3.3}$$

where $\boldsymbol{n} = (n_1, n_2, n_3)$ is the normal to this triangle in $\mathbb{R}^3$ and c is a constant. We can substitute $q = (x_0, y_0)$ into (3.3) to estimate the output by

$$z_0 = \frac{c - n_1 x_0 - n_2 y_0}{n_3}, \tag{3.4}$$

which is a simple linear interpolation in $\mathbb{R}^3$.

This whole calculation can be generalised for data modelling in $\mathbb{R}^d$. Given a set of M ($\geq d$) sample data points $\{\{\boldsymbol{x}_1, y_1\}, \{\boldsymbol{x}_2, y_2\}, \ldots, \{\boldsymbol{x}_M, y_M\}\}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, we can estimate y_q for the query point $\boldsymbol{x}_q$ as in Algorithm 3.5. We can refer this to be *local Delaunay triangulation modelling* or LDT for short.

1. From the set of M sample points $\boldsymbol{x}_i$, find the first $p_{\max}$ nearest neighbours of the query $\boldsymbol{x}_q$ by constructing the kd-tree of the M sample points in the input space.
2. Perform Delaunay triangulation of these $p_{\max}$ nearest neighbours in the input space using a technique such as the one described earlier.
3. Locate the simplex S which encloses the query $\boldsymbol{x}_q$ in the d-dimensional input space.
4. Use the $d+1$ $(d+1)$-dimensional sample points $\boldsymbol{X}_i = (\boldsymbol{x}_i, y_i)$ defining the simplex S in d-space to calculate the normal $\boldsymbol{n} = (n_1, n_2, \ldots, n_{d+1})$ in the $(d+1)$-space and get the equation of this hyperplane
$$\boldsymbol{X} \cdot \boldsymbol{n} = n_1 x_1 + n_2 x_2 + \cdots + n_d x_d + n_{d+1} y = c, \tag{3.5}$$
where c is a constant.
5. Substitute the query $\boldsymbol{x}_q$ into (3.5), therefore
$$n_1 x_{q1} + n_2 x_{q2} + \cdots + n_d x_{qd} + n_{d+1} y_q = c \tag{3.6}$$
We can solve for y_q, the only unknown in this equation.

Algorithm 3.5: Data modelling of $f : \mathbb{R}^d \to \mathbb{R}$ using local linear interpolation via Delaunay triangulation.

The most expensive step is the Delaunay triangulation of the sample points. Instead of processing the whole set of points, we can calculate only the Delaunay triangulation of the $p_{\max}(> d+1)$ nearest neighbours of the query point to improve the speed, because we know that such a simple local linear interpolation step involves only $d+1$ *near* sample data points.

The other time-consuming step is to locate the triangle/simplex which encloses the query point. In computational geometry, a planar point location problem can be solved in $O(\log M)$ time, but the optimal time for the point location problem in three and higher dimensions is still essentially an open question. For a subdivision induced by a set of M hyperplanes in d-dimensional space, it is known to be $\Theta(M^d)$ in the worst case [Edelsbrunner 1987]. However, many specific subdivisions can lead to much more efficient point location, e.g. convex polytopes [Clarkson 1987; Matoušek and Schwarzkopf 1993] and arrangements of triangles [de Berg and Schwarzkopf 1995] in low dimensions, or rectangular subdivisions [Edelsbrunner *et al.* 1986; de Berg *et al.* 1995] in higher dimensions. In our case, since we have the nearest neighbour distance information about each point, and if we assume that the triangle/simplex enclosing the query point must be formed by *some* selection of $p_{\max}$ near neighbours, then by labelling the simplices with their vertices of near neighbours and sorting those simplices lexicographically in ascending order of the near neighbour distances defining the simplex, the search time can be much improved. In many cases, the first simplex in the sorted list is usually the required simplex.

During the linear interpolation step, the required normal $\boldsymbol{n}$ is a vector which is mutually orthogonal to all vectors formed by the data points defining the simplex. One of the basic calculations is shown in Appendix B.

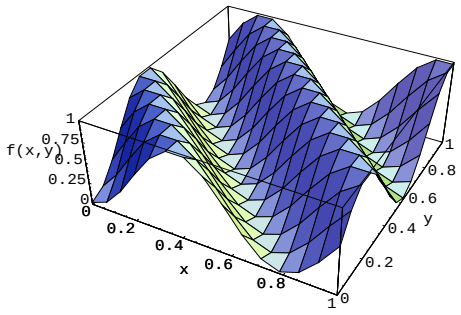


Figure 3.9: Surface of Equation (3.7) in the bounded input space $[0,1]^2 \subset \mathbb{R}^2$ for the modelling experiments.

3.2.5 Some simple prediction experiments using local Delaunay triangulation

To demonstrate this predictor, we perform a simple surface reconstruction experiment. The surface to be modelled is given by

$$f((x,y)) = \sin^2[4(x+y)] \tag{3.7}$$

with the bounded input space $[0,1]^2 \subset \mathbb{R}^2$, see Figure 3.9.

$M = 200$ data points are randomly sampled from input-space with a uniform distribution which together with their corresponding y values provide a set of training data for the predictor. We then define a test data set as 'a grid of data' equally spaced in the input space, i.e. a set of $((x,y), f(x,y))$ where $0 \leq x, y \leq 1$ and sampled at every 0.0625 along the x and y axes to have 256 (16^2) test data. In this way a surface can be reconstructed from these test data.

The result of this experiment using $p_{\max} = 12$ is shown in Figure 3.10 with MSE of 0.0118274. In this experiment, if the query point is not in any triangle, its nearest neighbour's output is used as the estimate for the query. The distribution of the squared error is shown in Figure 3.11. Note the high concentration of error at the boundary of the convex hull of the input training data. The error is also more pronounced in regions where the gradient of the surface is large. A more detailed illustration of how this predictor copes with test data at the boundary is given in the next section.

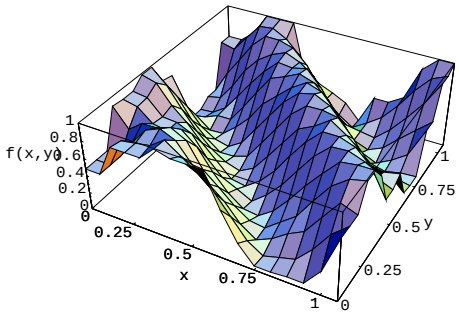


Figure 3.10: Reconstructed surface with $p_{\max} = 12$. MSE = 0.0118274.

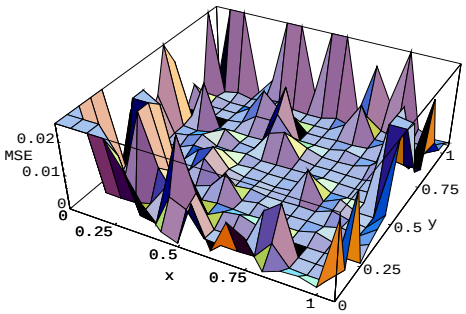


Figure 3.11: Distribution of the squared errors of the prediction.

3.2.6 Outside query prediction

When the convex hull of the sample points (training data) does not enclose the query point in the input space, this means that the query is not contained in any one of the triangles from the Delaunay

triangulation of the sample points. We call this an *outside query*. As we discussed in the chapter introduction performing (linear) interpolation to predict/estimate the output at an outside query is a difficult issue (as the previous experiment illustrates for that particular modelling technique). We examine several strategies which can sometimes improve the estimation in such cases, but the accuracy of each technique depends heavily on the distribution of the sample points and how *far* the query is from the *convex hull of neighbouring sample points* in the input space.

The following techniques rely on the availability of the local near distance information found by the kd-tree. The simplest method (used above) is to take the output value of the closest near neighbour point of the query as the output of the outside query point, e.g. point B's z value for the query at $\boldsymbol{q}$ in Figure 3.12. This only works well if the near distance from the query point is relatively *small*.

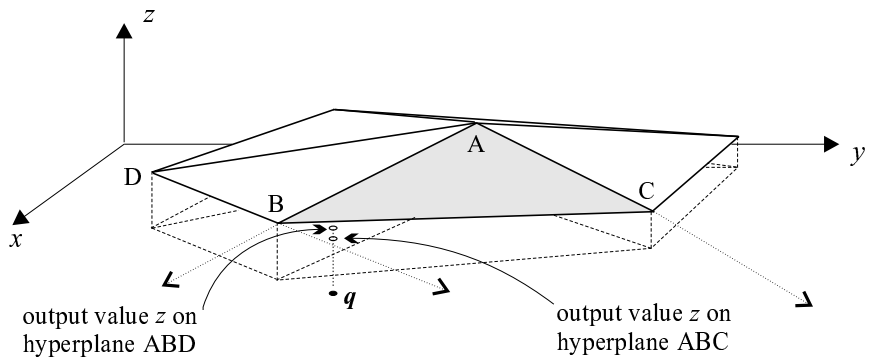


Figure 3.12: The difference of estimating the output value z at $\boldsymbol{q}$, an outside query point, between using the hyperplane given by $\triangle ABD$ formed by $\boldsymbol{q}$'s 3 nearest neighbours A, B and D and using the hyperplane formed by $\triangle ABC$. The hyperplane given by $\triangle ABC$ is in fact a better choice for this estimation.

Another method is to take the first $d+1$ nearest neighbours and form a hyperplane which can be extended to cover the outside query. Using this hyperplane, a linear interpolation can be performed to estimate the output value. Although this seems to offer the opportunity of taking the *slope* information into account for the estimation, the chosen hyperplane is not necessarily the hyperplane in the 'correct direction', as demonstrated in Figure 3.12. Moreover, sometimes these neighbours may not be the vertices of a triangle which is given by the Delaunay triangulation of the sample points at all. This approach also is dependent on the assumption that the query point is not too distant from the near neighbours.

Instead of taking the hyperplane formed by the first $d+1$ nearest neighbours and since the triangles calculated are lexicographically sorted in the ascending order of nearest distances of their vertices, we can take the first simplex for our hyperplane linear interpolation. This simplex is not necessarily the hyperplane formed by the $d+1$ nearest distance neighbours. In general, this performs better than the previous technique because it removes the assumption that the first $d+1$ nearest neighbours forms a valid Delaunay triangle from the d-triangulation of the samples.

We can also take several nearest triangles, perform linear interpolation for each triangle, and then average the results to estimate the output value. Our early experiments shows that there is not much improvement taking averages, because if the local area to be modelled is very hilly the angles between those hyperplanes are large and will result a poor estimation which does not reflect the 'slope' information of the nearest triangle.

Finally we can combine several technique together for such outside query estimation to compen-

Strategy No.	Method	MSE
1	Output value of the nearest neighbour	0.0118274
2	Hyperplane formed by the first $d+1$ nearest neighbours	0.00625806
3	First simplex from sorted list	0.00541832
4	Averaging estimates from several (d) simplices	0.00408974
5	OQCS ($\zeta = 0.45$)	0.00553535

Table 3.1: MSE of different outside query strategies used in the experiment.

sate for each technique's limitations and exploit the available information reflecting near neighbour distances from the query. We define a heuristic measure to estimate the *relative closeness* of the query point from the cluster of the $p_{\max}$ near neighbours by looking at the ratio

$$\nu = \frac{\delta_1}{\delta_{p_{\max}}}, \tag{3.8}$$

where δ_i is the distance of the i^{th} nearest neighbours from the query point. If $\nu \approx 1$, this means the query is very 'far' away from its nearest neighbour. On the other hand if ν is close to zero, this means the query is near the cluster of near neighbours. Using this ratio we can define the following strategy: if ν is small (say below some threshold ζ) we can pick the first triangle from the sorted list to perform linear interpolation, if ν is large and close to one ($\nu > \zeta$), we do not perform any linear interpolation but simply pick the first nearest neighbour's output as the estimate. Surprisingly, this technique gives a rather competitive performance compared with the other techniques, at least on the particular problems examined, provided a suitable value of the threshold ζ is chosen (usually taken to be about 0.45 in the experiments). We simply refer to this as the *outside query combined strategy* (OQCS).

A comparison of the MSEs of the various strategies described is shown in Table 3.1. This comparison only provides a general guidance of usage for tackling the outside query problem. Some strategies might perform better than others on different problems. Table 3.1 shows that the fourth strategy performed best in this experiment. But from general experience, OQCS seems to be a more robust practical choice, particularly if the data are sparsely and unevenly distributed in the input space.

3.2.7 Performance analysis

Intuitively, since this predictor is trying to reconstruct the surface based on the underlying geometrical properties, it might be expected to degrade very rapidly in terms of performance under noisy data.

To investigate this issue we use the same experimental setup, but with normally distributed noise r having $\mathrm{Mean}(r) = 0$ added to the output of the test data. The predictor is set up with $p_{\max} = 12$ and using OQCS for outside query. We then measure the MSE of the estimation on the test data for varying variance of the added noise $\mathrm{Var}(r)$ starting from 0.02 to 0.5 in steps of 0.02. A graph of MSE against $\mathrm{Var}(r)$ can then be plotted, as in Figure 3.13. As expected, the performance reduces as the level of noise increases, although the graph shows that the predictor may still perform moderately well at high noise levels.

These experiments are merely illustrative. In general, for a particular problem, we should need to consider in far more detail the interrelationship between surface complexity, noise level and the variation of $p_{\max}$ which we shall discuss later.

The size M of the training data set obviously also plays an important role in the accuracy of the final prediction. More training data can improve the estimation. Figure 3.14 is a graph of MSE against

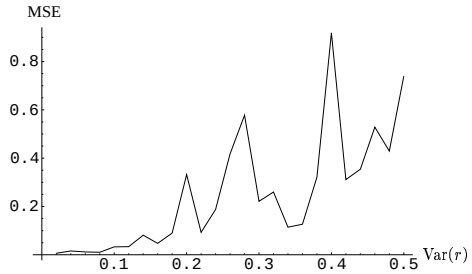


Figure 3.13: MSE error against $\mathrm{Var}(r)$ of normally distributed noise r with $\mathrm{Mean}(r) = 0$ added to the output. $\mathrm{Var}(r)$ goes from 0.02 to 0.5 in steps of 0.02, with OQCS ($\zeta = 0.45$) and $p_{\max} = 12$.

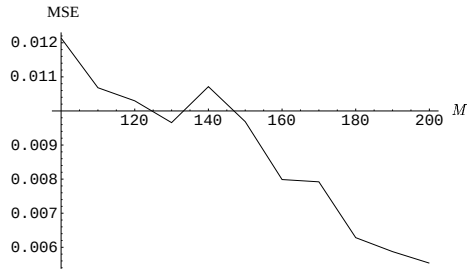


Figure 3.14: MSE error of the test data against the number of sample data M used for the training, size from 100 to 200 in steps of 10 using OQCS ($\zeta = 0.45$) with $p_{\max} = 12$.

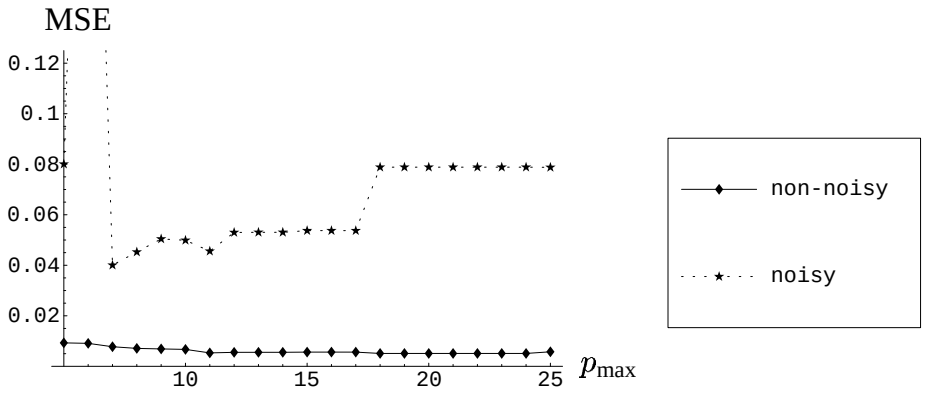


Figure 3.15: MSE against $p_{\max}$, the number of nearest neighbours, varying from 5 to 25 for the local Delaunay triangulation and using OQCS ($\zeta = 0.45$) without noise and with noise r ($\mathrm{Mean}(r) = 0$, $\mathrm{Var}(r) = 0.15$).

M, with the same surface estimation experimental setup as before. We shall have more to say about estimating the necessary size of a data set.

The accuracy of this predictor also depends on $p_{\max}$, the number of nearest neighbours. Using the same experiment setup (without noise and using OQCS), we increase $p_{\max}$ in the range from 5 to 25 and plot against the MSE on the testing data (see Figure 3.15).

By examining the distribution of training data in relation to the distribution of the squared errors of the test data (see Figure 3.16) of the experiment with $p_{\max} = 12$ and using OQCS ($\zeta = 0.45$), we see that even in some regions with a low concentration of training data, the errors are still small. The large errors are along two ridges, where the generating surface has a large gradient combined with sparse training data in the local region.

Having a large $p_{\max}$, in the non-noisy output data case, certainly improves the chance of having a 'more correct' local Delaunay triangulation, and hence a better local estimation. At the same time, it is neither desirable nor necessary to have $p_{\max}$ too large, because the local triangulation eventually will not change when more points further away are included. Also the estimation may degrade by increasing $p_{\max}$ because the assumption of local linearity becomes less likely to be true over a larger region, see for example Figure 3.15 at about $p_{\max} = 25$. Of course, the distribution of the training data can also affect our choice of $p_{\max}$, since a large concentration of data points in a small region may only require a small $p_{\max}$ to produce an accurate estimation.

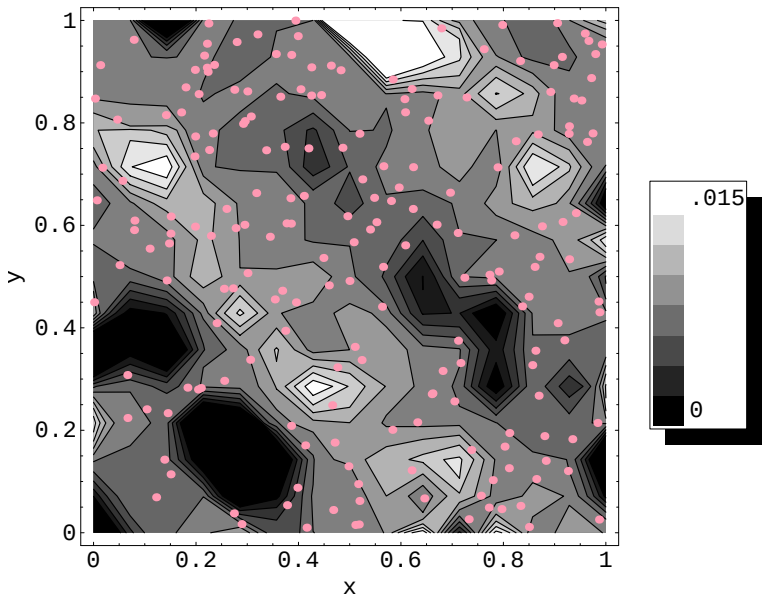


Figure 3.16: Distribution of the squared errors (contour plot) of the test data and the distribution of the training data (point plot) in the input space of the local Delaunay triangulation surface modelling experiment with $p_{\max} = 12$ and using OQCS ($\zeta = 0.45$).

On the other hand with noisy data, it is more desirable to have a small $p_{\max}$ as illustrated in Figure 3.15. Small amounts of noise can significantly affect the true nature of the output surface, thereby, reducing the accuracy of estimation using piecewise linear interpolation.

3.3 The Gamma predictor

One attractive idea might be to extend the Gamma test itself so that the underlying ideas could be used for *modelling* rather than simply noise estimation. The suggestion is that since the Gamma test estimates noise, maybe we can use a extension of the idea to choose a predicted value for a query point $\boldsymbol{x}$ so as to *minimise* the expected noise when the new data pair $(\boldsymbol{x}, y)$ is added to the training set. We have called the resulting algorithm the *Gamma-minimum-predictor*, or GMP for short. In this section we describe the derivation which is based on the original Gamma test, give illustrative results, and eventually explain why there is a well known and better technique for local modelling under noise.

3.3.1 Derivation of the Gamma-minimum-predictor

Suppose we are given M points $(\boldsymbol{x}(i), y(i))$, where $\boldsymbol{x}(i) = (x_1(i), \ldots, x_d(i))$, $1 \leq i \leq M$, and that $\bar{\Gamma}$ is computed as described in section 2.3. Now suppose that we are given a new point $(\boldsymbol{x}(M+1), y)$ for which $y = y(M+1)$ is unknown. We want to choose y so that the recomputed value of $\bar{\Gamma} = \bar{\Gamma}(y)$ is minimal.

Consider now

$$\Delta(p) = \frac{1}{p} \sum_{h=1}^{p} \frac{1}{M} \sum_{i=1}^{M} |\boldsymbol{x}(\mathcal{N}(i,h)) - \boldsymbol{x}(i)|^2 \tag{3.9}$$

and

$$\Gamma(p) = \frac{1}{p}\sum_{h=1}^{p}\frac{1}{2M}\sum_{i=1}^{M}\left(y(\mathcal{N}(i,h)) - y(i)\right)^2. \tag{3.10}$$

Notation: In this section we replace the previous used symbol $\bar{\Gamma}$ by Γ_R, since we wish to use

$$\bar{\Gamma} = \frac{1}{p_{\text{max}}}\sum_{k=1}^{p_{\text{max}}}\Gamma(k). \tag{3.11}$$

Similarly we write

$$\bar{\Delta} = \frac{1}{p_{\text{max}}}\sum_{k=1}^{p_{\text{max}}}\Delta(k). \tag{3.12}$$

The regression line formulates for computing Γ_R (the Γ-axis intercept) is

$$\Gamma_R = \bar{\Gamma} - \frac{S_{\Delta\Gamma}}{{S_\Delta}^2}\cdot\bar{\Delta}, \tag{3.13}$$

where

$$S_{\Delta\Gamma} = \frac{1}{p_{\text{max}}}\sum_{p=1}^{p_{\text{max}}}(\Delta(p)-\bar{\Delta})(\Gamma(p)-\bar{\Gamma}), \tag{3.14}$$

and

$${S_\Delta}^2 = \frac{1}{p_{\text{max}}}\sum_{p=1}^{p_{\text{max}}}(\Delta(p)-\bar{\Delta})^2. \tag{3.15}$$

We have to consider what happens when a new point $\boldsymbol{x}(M+1)$ is added. One effect is that the neighbourhood structure in input-space changes. Computationally this is straightforward: we merely have to add the new point $\boldsymbol{x}(M+1)$ to the kd-tree, a procedure that already exists and was used to build the tree. (Of course, after the prediction is made we shall should also need to remove the new point from the kd-tree if we plan to use it again. In practice, for data sets that are not too large, it is often easiest to maintain a copy of the tree, add the new point to the copy, use the augmented tree for the prediction and then replace it by the original copy!)

Let us assume that this has been done and that the new neighbourhood structure is described by $\mathcal{N}_{\text{new}}[i,p]$ ($1 \le i \le M+1$, $1 \le p \le p_{\text{max}}$). Corresponding to the new neighbourhood structure we can use (3.9) and (3.10) to compute new values for $\Delta(p)$ and $\Gamma(p)$. Thus

$$\Delta_{\text{new}}(p) = \frac{1}{p}\sum_{h=1}^{p}\frac{1}{M+1}\sum_{i=1}^{M+1}\left|\boldsymbol{x}(\mathcal{N}[i,h]) - \boldsymbol{x}(i)\right|^2 \tag{3.16}$$

and this is easily calculated and does not involve y. For $\Gamma_{\text{new}}(p)$ we write

$$\Gamma_{\text{new}}(p) = \Gamma(p,y) = U(p,M) + \frac{1}{p}\sum_{h=1}^{p}\frac{1}{2(M+1)}\left[(y(\mathcal{N}_{\text{new}}[i,h]) - y)^2 + \sum_{i\in S(h)}(y-y(i))^2\right], \tag{3.17}$$

where $U(p,M)$ is independent of y and $S(h) = \{i|\mathcal{N}_{\text{new}}[i,h] = M+1\}$ which is easily determined from the new neighbourhood structure.

From (3.13) with $\Delta(p)$ and $\Gamma(p)$ replaced by their values we now have

$$\Gamma_R(y) = \bar{\Gamma}_{\text{new}} - \frac{S_{\Delta_{\text{new}}\Gamma_{\text{new}}}}{S_{\Delta_{\text{new}}}^{\ 2}} \cdot \bar{\Delta}_{\text{new}}. \tag{3.18}$$

In this expression only the first term and the term with subscript Γ_{new} involves y. This equation will determine the new value of Γ_R.

If we apply the criterion then the value of y is that the value which minimises $\Gamma_R(y)$. We proceed by evaluating $\partial\Gamma_R(y)/\partial y$. From (3.18) we have

$$\frac{\partial}{\partial y}\Gamma_R(y) = \frac{\partial}{\partial y}\bar{\Gamma}_{\text{new}} - \frac{\bar{\Delta}_{\text{new}}}{S^2_{\Delta_{\text{new}}}}\frac{\partial}{\partial y}(S_{\Delta_{\text{new}}\Gamma_{\text{new}}}). \tag{3.19}$$

The next step is to evaluate the partial derivatives. For the last term in (3.19) we proceed as follows. Note that the terms $\bar{\Delta}_{\text{new}}$ and $S_{\Delta_{\text{new}}}$ can be considered as known (i.e. they are easy to compute) and do not involve y. Hence it remains to consider

$$\begin{aligned}
\frac{\partial}{\partial y}\left(S_{\Delta_{\text{new}}\Gamma_{\text{new}}}\right) &= \frac{\partial}{\partial y}\left(\frac{1}{p_{\max}}\sum_{p=1}^{p_{\max}}(\Delta_{\text{new}}(p) - \bar{\Delta}_{\text{new}})(\Gamma(p,y) - \bar{\Gamma}_{\text{new}})\right) \\
&= \frac{1}{p_{\max}}\sum_{p=1}^{p_{\max}}(\Delta_{\text{new}}(p) - \bar{\Delta}_{\text{new}})\left(\frac{\partial}{\partial y}\Gamma(p,y) - \frac{\partial}{\partial y}\bar{\Gamma}_{\text{new}}\right) \\
&= \frac{1}{p_{\max}}\sum_{p=1}^{p_{\max}}C(p)\left(\frac{\partial}{\partial y}\Gamma(p,y) - \frac{\partial}{\partial y}\bar{\Gamma}_{\text{new}}\right),
\end{aligned} \tag{3.20}$$

where

$$C(p) = \Delta_{\text{new}}(p) - \bar{\Delta}_{\text{new}} \tag{3.21}$$

and these are readily computed.

We determine the first term in the sum of (3.20) and obtain

$$\begin{aligned}
\frac{\partial}{\partial y}\Gamma(p,y) &= \frac{1}{p(M+1)}\sum_{h=1}^{p}\left(y - y(\mathcal{N}[M+1,h]) + \sum_{i\in S(h)}(y - y(i))\right) \\
&= \frac{1}{M+1}\left[1 + \frac{1}{p}\sum_{h=1}^{p}|S(h)|\right]y - \lambda,
\end{aligned} \tag{3.22}$$

where

$$\lambda = \lambda(p,M) = \frac{1}{p(M+1)}\sum_{h=1}^{p}\left[y(\mathcal{N}_{\text{new}}[M+1,h]) + \sum_{i\in S(h)}y(i)\right]. \tag{3.23}$$

The first term in (3.19), i.e. $\partial\bar{\Gamma}_{\text{new}}/\partial y$, is also the second term in the sum of (3.20). To evaluate this we replace $\Gamma(h)$ by the new value in (3.17) and obtain from the updated version of (3.11),

$$\begin{aligned}
\frac{\partial}{\partial y}\bar{\Gamma}_{\text{new}} &= \frac{1}{p_{\max}}\sum_{h=1}^{p_{\max}}\frac{1}{h}\sum_{k=1}^{h}\frac{1}{M+1}\left[-(y(\mathcal{N}_{\text{new}}[M+1,k]) - y) + \sum_{i\in S(k)}(y - y(i))\right] \\
&= \frac{1}{p_{\max}(M+1)}\sum_{h=1}^{p_{\max}}\frac{1}{h}\sum_{k=1}^{h}\left[y - y(\mathcal{N}_{\text{new}}[M+1,k]) + \sum_{i\in S(k)}(y - y(i))\right].
\end{aligned} \tag{3.24}$$

Collecting together the terms in y we have

$$\frac{\partial}{\partial y}\bar{\Gamma}_{\text{new}} = \frac{1}{M+1}\left[1 + \frac{1}{p_{\max}}\sum_{h=1}^{p_{\max}}\frac{1}{h}\sum_{k=1}^{h}|S(k)|\right]y - \mu \tag{3.25}$$

where

$$\begin{aligned}\mu = \mu(M) &= \frac{1}{p_{\max}(M+1)}\sum_{h=1}^{p_{\max}}\frac{i}{h}\sum_{k=1}^{h}\left[y(\mathcal{N}_{\text{new}}[M+1,k]) + \sum_{i\in S(k)} y(i)\right] \\ &= \frac{1}{p_{\max}}\sum_{h=1}^{p_{\max}}\lambda(h,M).\end{aligned} \tag{3.26}$$

Hence

$$\frac{\partial\Gamma(p,y)}{\partial y} - \frac{\partial\bar{\Gamma}}{\partial y} = \frac{1}{M+1}\left[\frac{1}{p}\sum_{h=1}^{p}|S(h)| - \frac{1}{p_{\max}}\sum_{h=1}^{p_{\max}}\frac{1}{h}\sum_{k=1}^{h}|S(k)|\right]y - \lambda(p,M) + \mu. \tag{3.27}$$

For $1 \le p \le p_{\max}$, let

$$A(p) = \frac{1}{M+1}\left[\frac{1}{p}\sum_{h=1}^{p}|S(h)| - \frac{1}{p_{\max}}\sum_{h=1}^{p_{\max}}\frac{1}{h}\sum_{k=1}^{h}|S(k)|\right] \tag{3.28}$$

and

$$B(p) = \lambda(p,M) - \mu \tag{3.29}$$

then

$$\frac{\partial}{\partial y}(S_{\Delta_{\text{new}}\Gamma_{\text{new}}}) = \frac{1}{p_{\max}}\sum_{p=1}^{p_{\max}}C(p)\left(A(p)y - B(p)\right). \tag{3.30}$$

Finally let

$$\nu = \frac{1}{M+1}\left[1 + \frac{1}{p_{\max}}\sum_{h=1}^{p_{\max}}\frac{1}{h}\sum_{k=1}^{h}|S(k)|\right]. \tag{3.31}$$

Then equating (3.19) to zero and solving for y we obtain

$$y = \frac{\mu - \dfrac{\bar{\Delta}_{\text{new}}}{S^2_{\text{new}}}\dfrac{1}{p_{\max}}\displaystyle\sum_{p=1}^{p_{\max}}C(p)B(p)}{\nu - \dfrac{\bar{\Delta}_{\text{new}}}{S^2_{\text{new}}}\dfrac{1}{p_{\max}}\displaystyle\sum_{p=1}^{p_{\max}}C(p)A(p)}. \tag{3.32}$$

The final algorithm is given in Algorithm 3.6. But with careful examination, the whole process would appear to be simply performing a local linear regression on the squared distances of the near neighbours from the query point.

3.3.2 Performance analysis

To investigate the GMP, we perform a series of experiments based on the experimental setup for surface modelling (defined by (3.7)) as described in Sections 3.2.5, 3.2.7). We first investigate how the number of nearest neighbours $p_{\max}$ used for GMP affects the modelling, given the same set of test data. The

Procedure: Gamma-minimum-predictor(kd-tree, $\boldsymbol{x}$, $p_{\max}$, M**)**
Add point x to kd-tree to get $\mathcal{N}_{\text{new}}[i,p]$ ($1 \leq i \leq M+1$, $1 \leq p \leq p_{\max}$)
{i.e. new nearest neighbour list}
for $p = 1$ to $p_{\max}$ **do**
  Compute $\Delta_{\text{new}}(p)$ from (3.16)
  Compute $S(p) = \{i | \mathcal{N}_{\text{new}}[i,p] = M+1\}$ {these sets are small}
  Compute $\lambda(p, M)$ from (3.23)
end for
for $p = 1$ to $p_{\max}$ **do**
  Compute $A(p)$ from (3.28)
  Compute $B(p)$ from (3.29)
end for
Compute $\bar{\Delta}_{\text{new}}$
for $p = 1$ to $p_{\max}$ **do**
  Compute $C(p)$ from (3.21)
end for
Compute $S_{\Delta_{\text{new}}}$
Compute μ from (3.26)
Compute ν from (3.31)
Compute y from (3.32)
Remove point x from kd-tree {if not copied}
End

Algorithm 3.6: The Gamma-minimum-predictor given a new query x to estimate output y.

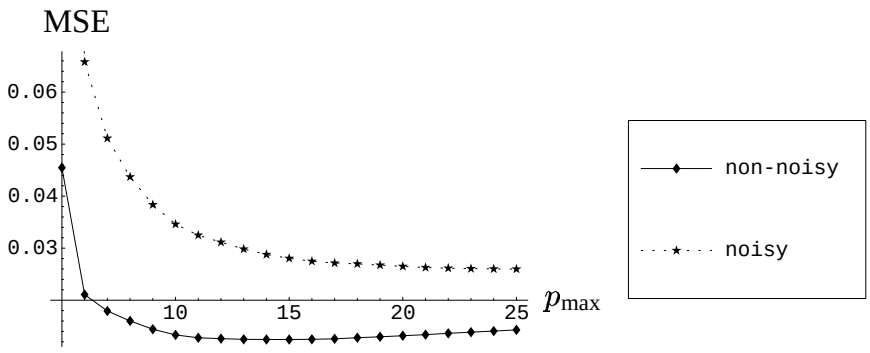


Figure 3.17: MSE against $p_{\max}$, the number of nearest neighbours, varying from 5 to 25 for GMP without noise and with noise ($\text{Mean}(r) = 0$, $\text{Var}(r) = 0.15$).

result is shown in Figure 3.17 which indicates that the MSE of the estimation is optimal when $p_{\max} = 14$ when no noise is present. Not surprisingly if $p_{\max}$ is taken too large, the performance of GMP degrades. Notice that even at the optimal value, the MSE is still higher than the Delaunay triangulation predictor, LDT. When noise is present, it is advisable to have a large $p_{\max}$ as shown in Figure 3.17.

We then examined how GMP copes with varying noisy data when $p_{\max} = 12$. Normally distributed noise r with $\mathrm{Mean}(r) = 0$ and variance $\mathrm{Var}(r)$ from 0.02 to 0.5, increasing in steps of 0.02, is added to the outputs of the training data. The MSE of the test data is then plotted against the noise variance as shown in Figure 3.18. As expected, GMP degrades 'gracefully' as the level of noise increases. This experiment also illustrates that the GMP can cope with noise well, unlike Delaunay triangulation predictor, LDT.

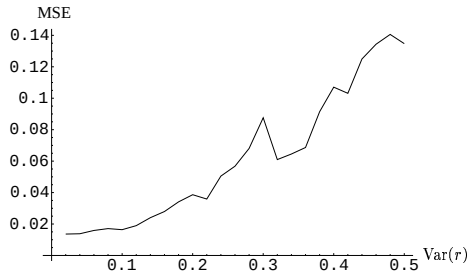


Figure 3.18: MSE against normal distributed noise r with $\mathrm{Var}(r)$ from 0.02 to 0.5 in steps of 0.02 and $\mathrm{Mean}(r) = 0$ using GMP with $p_{\max} = 12$.

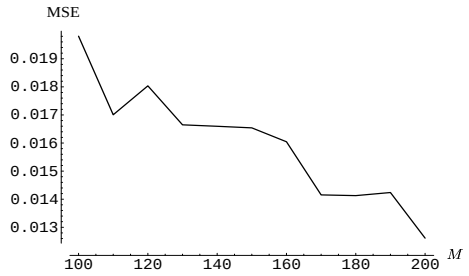


Figure 3.19: MSE against the number of sample data used, M for the training, from 100 to 200 in step of 10, of GMP with $p_{\max} = 12$.

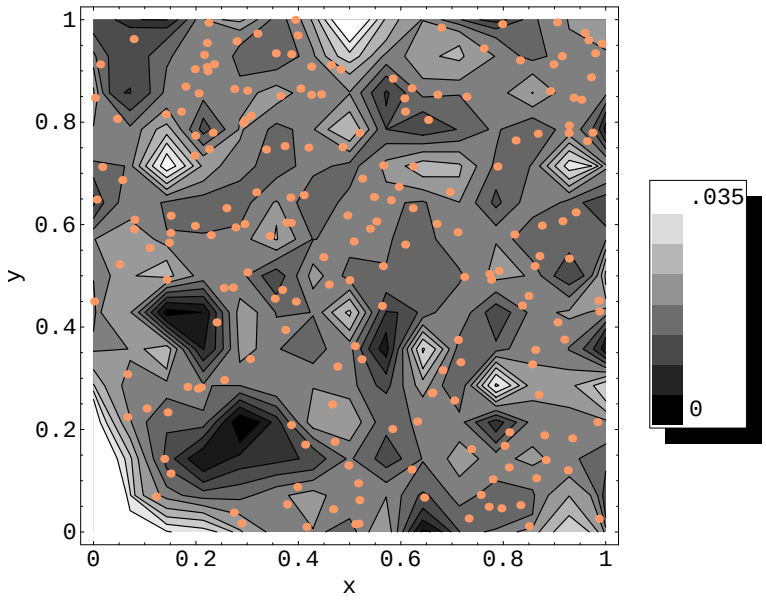


Figure 3.20: Distribution of the test data squared error (by contour plot) for $p_{\max} = 12$ and the distribution of training data (points).

The MSEs of the test data are also measured for varying size of the set of training data with $p_{\max} = 12$, see Figure 3.19. The result shows that for this experiment the accuracy of estimation by GMP depends on M, i.e. $M >= M(\epsilon)$ for given MSE ϵ as expected. We then generate the result again for the full set of training data, i.e. $M = 200$ and look at the distribution of the squared error of the test

data in relation to the distribution of the training data in the input space. This is shown in Figure 3.20. The MSE is 0.0126141. Some regions with low concentration of training data, especially at the lower left corner of the plot, have much higher squared error. This plot essentially implies that for fixed M the modelling can be improved if the training data are more evenly distributed in the input space.

In fact, the main determinant of model accuracy seems to be that the maximum nearest neighbour distances approach zero as M becomes large; regions where nearest neighbour distances are large give larger predictive errors. This should not be too surprising: it is intuitively obvious for any interpolation-based technique and is, moreover, one of the fundamental assumptions behind the Gamma test itself.

In any event examination of the basic formula (3.32) for the GMP shows that, by a somewhat devious route, we have eventually arrived at an algorithm which is effectively performing local linear regression on the squared distances of nearest neighbours of the query points. But why just do local linear regression on the squared distances? Why not do it on the data coordinates themselves? In the next section we shall see that this simple approach works extremely well - especially at high noise levels provided sufficient data is available. Once the kd-tree has been constructed, local linear regression is very fast for any reasonable input dimension, certainly for d up to several hundred. Moreover, local linear regression based on kd-trees can be done using *dynamic updating*, i.e. new data can augment the kd-tree as it becomes available and the next prediction can use this information. This is something that is very difficult to accomplish with feedforward neural network modelling using backpropagation.

3.4 Prediction using local linear regression

Based on the local information given by the kd-tree, we can take the $p_{\max}$ nearest points to perform a *least squares fit* to estimate the query point by assuming its underlying model is 'locally linear' and we simply call this *local linear regression* or LLR for short.

We first review the idea of 'least-squares-fit' and its relationship to the pseudoinverse of a matrix as introduced by Penrose [Penrose 1955; 1956].

3.4.1 Least squares fit

The simple *least squares fit* (LSF) problem is defined by:

Definition 3.4.1 (Least squares fit). *Given a set of M data points*

$$(y_i, (x_{i1}, x_{i2}, \ldots, x_{im})), \qquad (1 \leq i \leq M) \tag{3.33}$$

where M is large and m is fixed, the least squares fit *is the point $(A_1, A_2, \ldots, A_m)$ such that*

$$F_M(A_1, A_2, \ldots, A_m) = \sum_{i}^{M} \left(y_i - (A_1 x_{i1} + A_2 x_{i2} + \ldots + A_m x_{im})\right)^2 \tag{3.34}$$

is minimised.

The conventional LSF considers F_M as a function of $(A_1, A_2, \ldots, A_m)$, which (since $F_M \geq 0$ for all $(A_1, A_2, \ldots, A_m)$ is minimised by the expedient of equating the partial derivatives to zero and solving the corresponding equations. Treating $(A_1, A_2, \ldots, A_m)$ as unknowns in these equations the coefficients are therefore determined by the M data points (3.33). The amount of data to be manipulated in solving these linear equations therefore becomes large with M.

There are many algorithms already available for solving this problem [Press *et al.* 1992] and also there are many adaptive algorithms available, such as a recursive generalised-least-squares procedure described in [Hastings-James and Sage 1969] and methods for updating pseudoinverse iteratively [Maeda and Mutata 1984; Telfer and Casasent 1989] and for estimating the parameters which we are not going to discuss. The idea of pseudoinverse will be described later. However we investigate a simple iterative method (suggested by A.J. Jones in 1991) as a detailed illustration of how to solve such LSF problem. Although an iterative step method is unnecessary for our local data modelling when only a finite, fixed set of data is available, it could be useful in situation when the data set to be modelled requires constant updating due to the arrival of new data.

3.4.2 Iterative least squares algorithm

In studying dynamical systems it is often necessary to approximate the dynamics or mapping by just observing a small finite set of data. This is facilitated using LSF. Here we investigate an iterative technique and how it can be used to improved to minimise computational cost without losing the accuracy of the approximation, especially for real-time systems which requires constant updating due to the constant arrival of new data.

The LSF algorithm described here involves, for fixed M, an $m \times m$ matrix of numbers $U_{ij}(M)$ $(1 \leq i, j \leq m)$ and a vector of elements V_i $(1 \leq i \leq m)$ which, as M varies, satisfy the following recursion relations

$$\begin{aligned} U_{ij}(M+1) &= U_{ij}(M) + x_{(M+1)i}x_{(M+1)j} \qquad (1 \leq i, j \leq m) \\ V_i(M+1) &= V_i(M) + y_{M+1}x_{(M+1)i} \qquad (1 \leq i \leq m) \end{aligned} \tag{3.35}$$

The point to note about these recursion relations is that the $m^2 + 1$ numbers for $M + 1$ only depend on the new data point

$$\big((x_{(M+1)1}, x_{(M+1)2}, \ldots, x_{(M+1)m}), y_{M+1}\big) \tag{3.36}$$

and so can be computed recursively as new data points are added.

Theorem 3.4.1. *If $U_{ij}(1) = 0$, $V_i(1) = y_1 x_{1i}$ and for $M \geq 1$ the numbers $U_{ij}(M)$ $(1 \leq i, j \leq m)$, $V_i(M+1)$ are defined by (3.35), then for any given M the solution $A^* = (A_1^*, A_2^*, \ldots, A_m^*)$ to the LSF problem satisfies*

$$\begin{bmatrix} U_{11} & U_{12} & \cdots & U_{1m} \\ U_{21} & U_{22} & & U_{2m} \\ \vdots & & \ddots & \vdots \\ U_{m1} & \cdots & & U_{mm} \end{bmatrix} \begin{bmatrix} A_1^* \\ A_2^* \\ \vdots \\ A_m^* \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_m \end{bmatrix}. \tag{3.37}$$

Thus provided the distribution of data points is such that the matrix (U_{ij}) is invertible, one inversion of the $m \times m$ matrix will solve the LSF problem for the given data points.

Proof. To minimise F_M we require

$$\frac{\partial F_M}{\partial A_k} = 0 \ \text{ for } \ 1 \leq k \leq m. \tag{3.38}$$

This gives

$$\sum_i^M [-2(y_i - (A_1 x_{i1} + A_2 x_{i2} + \ldots + A_m x_{im}))x_{ik}] = 0 \tag{3.39}$$

or

$$\sum_i^M (y_i x_{ik}) - A_1 \sum_i^M (x_{i1} x_{ik}) - \cdots - A_m \sum_i^M (x_{im} x_{ik}) = 0 \tag{3.40}$$

for $0 \leq k \leq m$. If we use the definitions

$$U_{jk} = U_{jk}(M) = \sum_i^M x_{ij} x_{ik} \qquad \text{and} \qquad V_j = V_j(M) = \sum_i^M y_i x_{ij} \tag{3.41}$$

for $0 \leq j, k \leq m$ as in (3.35) we can then express (3.40) as

$$\begin{bmatrix} U_{11} & U_{12} & \cdots & U_{1m} \\ U_{21} & U_{22} & & U_{2m} \\ \vdots & & \ddots & \vdots \\ U_{m1} & \cdots & & U_{mm} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_m \end{bmatrix} \tag{3.42}$$

which is (3.37). ∎

Obviously by Theorem 3.4.1, this immediately demonstrates that for very large M, the iterative LSF problem (normally solving M simultaneous equations) reduces to solving m simultaneous equations.

Extension for more than one y

It would be interesting to know that similar recursion relations can be defined for the LSF problem with more than one y in a data point. The original problem then becomes:
Given a set of M data points

$$((x_{i1}, x_{i2}, \ldots, x_{im}), (y_{i1}, y_{i2}, \ldots, y_{in})) \tag{3.43}$$

for $1 \leq i \leq M$ where M is large and m and n are fixed, find the matrix

$$(A_{ij}) = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{m1} & \cdots & & A_{mn} \end{bmatrix} \tag{3.44}$$

such that

$$F_M((A_{ij})) = \sum_i^M \sum_k^n (y_{ik} - (A_{1k} x_{i1} + A_{2k} x_{i2} + \ldots + A_{mk} x_{im}))^2 \tag{3.45}$$

is minimised.

In other words, we need to solve

$$\begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & & y_{2n} \\ \vdots & & \ddots & \vdots \\ y_{M1} & y_{M2} & \cdots & y_{Mn} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & & x_{2m} \\ \vdots & & \ddots & \vdots \\ x_{M1} & x_{M2} & \cdots & x_{Mm} \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} \tag{3.46}$$

for (A_{ij}) by mean of matrix algebra such as the pseudoinverse of a matrix (described later). This would be a quite inefficient technique if M were large. One approach to tackle this is to split up the problem into smaller sub-problems. Therefore we can arrange the data points as n sets of data points

$$((x_{i1}, x_{i2}, \ldots, x_{im}), y_{ik}) \tag{3.47}$$

and solve for $(A_{1k}, A_{2k}, \ldots, A_{mk})$ for $1 \leq j \leq n$ individually using recursion relations (3.35) described above. Ideally we should seek an alternative iterative technique for solving this problem.

To derive a similar iterative LSF algorithm for this problem, we adhere closely to the proof of Theorem 3.4.1. To minimise F_M, we require

$$\frac{\partial F_M}{\partial A_{jk}} = 0 \text{ for } 1 \leq j \leq n, 1 \leq k \leq m. \tag{3.48}$$

Now we consider (3.48) for a particular k and then we get a set of equations

$$\begin{aligned}
\frac{\partial F_M}{\partial A_{1k}} = 0 &= \sum_i^M (x_{i1}(y_{ik} - A_{1k}x_{i1} - A_{2k}x_{i2} - \ldots - A_{mk}x_{im})) \\
\frac{\partial F_M}{\partial A_{2k}} = 0 &= \sum_i^M (x_{i2}(y_{ik} - A_{1k}x_{i1} - A_{2k}x_{i2} - \ldots - A_{mk}x_{im})) \\
&\vdots \\
\frac{\partial F_M}{\partial A_{mk}} = 0 &= \sum_i^M (x_{im}(y_{ik} - A_{1k}x_{i1} - A_{2k}x_{i2} - \ldots - A_{mk}x_{im}))
\end{aligned} \tag{3.49}$$

and rearranging these equations (3.49) we get

$$\begin{bmatrix} \sum_i^M x_{i1}y_{ik} \\ \sum_i^M x_{i2}y_{ik} \\ \vdots \\ \sum_i^M x_{im}y_{ik} \end{bmatrix} = \begin{bmatrix} \sum_i^M x_{i1}x_{i1} & \sum_i^M x_{i1}x_{i2} & \cdots & \sum_i^M x_{i1}x_{im} \\ \sum_i^M x_{i2}x_{i1} & \sum_i^M x_{i2}x_{i2} & & \sum_i^M x_{i2}x_{im} \\ \vdots & & \ddots & \vdots \\ \sum_i^M x_{im}x_{i1} & \sum_i^M x_{im}x_{i2} & \cdots & \sum_i^M x_{im}x_{im} \end{bmatrix} \begin{bmatrix} A_{1k} \\ A_{2k} \\ \vdots \\ A_{mk} \end{bmatrix} \tag{3.50}$$

For the other k's, equations similar to (3.50) can be derived and then can be gathered together and formulated as

$$\begin{bmatrix} \sum_i^M x_{i1}y_{i1} & \sum_i^M x_{i1}y_{i2} & \cdots & \sum_i^M x_{i1}y_{in} \\ \sum_i^M x_{i2}y_{i1} & \sum_i^M x_{i2}y_{i2} & & \sum_i^M x_{i2}y_{in} \\ \vdots & & \ddots & \vdots \\ \sum_i^M x_{im}y_{i1} & \sum_i^M x_{i1}y_{i2} & \cdots & \sum_i^M x_{im}y_{in} \end{bmatrix} =$$
$$\begin{bmatrix} \sum_i^M x_{i1}x_{i1} & \sum_i^M x_{i1}x_{i2} & \cdots & \sum_i^M x_{i1}x_{im} \\ \sum_i^M x_{i2}x_{i1} & \sum_i^M x_{i2}x_{i2} & & \sum_i^M x_{i2}x_{im} \\ \vdots & & \ddots & \vdots \\ \sum_i^M x_{im}x_{i1} & \sum_i^M x_{im}x_{i2} & \cdots & \sum_i^M x_{im}x_{im} \end{bmatrix} \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{m1} \\ A_{12} & A_{22} & & A_{m2} \\ \vdots & & \ddots & \vdots \\ A_{1n} & A_{2n} & \cdots & A_{mn} \end{bmatrix} \tag{3.51}$$

Equation (3.51) forms the basis of our new relations which involves the two matrices V $(m \times n)$ and U $(m \times m)$ and their components recursive relations are

$$\begin{aligned}
U_{ij}(M+1) &= U_{ij}(M) + x_{(M+1)i}x_{(M+1)j} \quad (1 \leq i, j \leq m) \\
V_{ij}(M+1) &= V_{ij}(M) + x_{(M+1)i}y_{(M+1)j} \quad (1 \leq i \leq m, 1 \leq j \leq m)
\end{aligned} \tag{3.52}$$

with $U_{ij}(0) = 0$ and $V_{ij}(0) = 0$. Therefore the whole original problem, in terms of these matrices, reduces to solving

$$V = U(A_{ij}) \tag{3.53}$$

and the solution (A_{ij}) can be found by pre-multiplying both sides of (3.53) by U^{-1}, the inverse matrix of U if U is non-singular i.e.

$$(A_{ij}) = U^{-1}V \tag{3.54}$$

The equation (3.53) is then a reduced form of the original LSF problem, commonly referred to as the *normal equation*. In fact, if we formulate (3.46), the original problem equivalently into

$$Y = XA. \tag{3.55}$$

In this simplified notation, its normal equation is simply given by

$$X^T Y = X^T X A \tag{3.56}$$

where superscript T denotes the transpose of a matrix. Then the solution of A is simply given by

$$A = (X^T X)^{-1} X^T Y. \tag{3.57}$$

However, it is important to recognise such adaptive properties of the normal equation. Only a simple inversion of a square matrix and a matrix multiplication are required to recalculate the LSF whenever a new data point arrives, providing the recursive relations U_{ij} and V_i are kept. [One should be aware that this is not necessarily the best or only adaptive algorithm and there are many alternative algorithms already available [Hastings-James and Sage 1969; Telfer and Casasent 1989; Maeda and Mutata 1984; Weigend *et al.* 1990].] The other immediate advantage is that there is no need to calculate the inverse of a large matrix if M is large. In the case of singular U, the optimal solution of (3.53) is given by $U^{\#}V$ as stated in Theorem 3.4.2 in the next section, where $U^{\#}$ is the pseudoinverse of U.

3.4.3 Pseudoinverse of a matrix

First we need to introduce the precise definition of *pseudoinverse* of a matrix [Penrose 1955; 1956]. For any given matrix $X \in \mathbb{R}^{m\times n}$, the matrix $X^{\#} \in \mathbb{R}^{n\times m}$ is said to be a pseudoinverse of X if the following conditions are satisfied:

1. $XX^{\#}X = X$,
2. $X^{\#}XX^{\#} = X$,
3. $(XX^{\#})^T = XX^{\#}$,
4. $(X^{\#}X)^T = X^{\#}X$,

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

The pseudoinverse of a matrix is important in solving the LSF problem. The reason becomes obvious with the following theorem [Penrose 1955].

Theorem 3.4.2. *Let $X \in \mathbb{R}^{n\times p}$ and $Y \in \mathbb{R}^{m\times p}$ be given. Then $A = X^{\#}Y \in \mathbb{R}^{m\times p}$ is the unique best approximate solution of the equation $XA = Y$.*

One important property for the pseudoinverse which is implied by this theorem is that the pseudoinverse exists even for singular $X \in \mathbb{R}^{p\times p}$. Before demonstrating how this pseudoinverse can help to solve the LSF, we first need to introduce some definitions. The phrase *best approximate solution* means that the quantity $\|XA - Y\|_F$ is minimised and it is further explained as follows.

Definition 3.4.2. *The $\|\cdot\|_F$-norm on $\mathbb{R}^{m\times n}$ is defined by*

$$\|X\|_F = \mathrm{Trace}(X^T X) \tag{3.58}$$

*for $X \in \mathbb{R}^{m\times n}$, where $\mathrm{Trace}(Y)$ is the trace of the square matrix Y. This norm is called the **Frobenius norm**.*

Lemma 3.4.1. *For $X \in \mathbb{R}^{m\times n}$, $X^T X X^{\#} = X^T$.*

Proof. We have

$$\begin{aligned} X^T(XX^{\#}) &= X^T(XX^{\#})^T && \text{(by pseudoinverse condition 3)} && (3.59)\\ &= ((XX^{\#})X)^T && && (3.60)\\ &= X^T && \text{(by pseudoinverse condition 1)} && (3.61) \end{aligned}$$

as required. ∎

Theorem 3.4.3. *Let $X \in \mathbb{R}^{n\times p}$ and $Y \in \mathbb{R}^{m\times p}$ be given. Then $A = X^{\#}Y$ is an element of $\mathbb{R}^{m\times n}$ which minimises the quantity $\|XA - Y\|_F$.*

Proof. We have

$$\begin{aligned} \|XA - Y\|_F^2 &= \|X(A - X^{\#}Y) + (XX^{\#} - I_p)Y\|_F^2 && (3.62)\\ &= \|X(A - X^{\#}Y)\|_F^2 + \|(XX^{\#} - I_p)Y\|_F^2 \\ &\quad +2\ \mathrm{Trace}\left((A - X^{\#}Y)^T X^T (XX^{\#} - I_p)Y\right), && (3.63) \end{aligned}$$

where $I_p \in \mathbb{R}^{p\times p}$ is the identity matrix. Since $X^T(XX^{\#} - I_p) = 0$ by Lemma 3.4.1, the last term disappears and we get

$$\|XA - Y\|_F^2 = \|X(A - X^{\#}Y)\|_F^2 + \|(XX^{\#} - I_p)Y\|_F^2 \tag{3.64}$$

which achieves its minimum $\|(XX^{\#} - I_p)Y\|_F^2$ when $A = X^{\#}Y$ ∎

The derivation of the calculation of $X^{\#}$ depends on the following proposition.

Proposition 3.4.1. *For any $X \in \mathbb{R}^{n\times p}$, the $p \times p$ matrix $X^T X$ is invertible if and only if the columns of X are linearly independent in $\mathbb{R}^n$.*

Proof. Consider that the square matrix $X^T X$ is invertible if and only if the equation $X^T X v = 0$ has the unique solution $v = 0$, $v \in \mathbb{R}^p$.

Suppose that the columns of X are linearly independent and that $X^T X v = 0$. Then it follows that $v^T X^T X v = 0$ and so $Xv = \mathbf{0}$, since $v^T X^T X v = \sum_{i=1}^n (Xv)_i^2 = |Xv|^2$, the square of the Euclidean length of the n-dimensional vector Xv. Since Xv is also a linear combination of the columns of

X, we can express $Xv = v_1x^{(1)} + v_2x^{(2)} + \cdots + v_px^{(p)}$, where $x^{(i)}$ is the i^{th} column of X and v_i is the i^{th} component of v. Because the columns of X are linearly independent, then $Xv = \mathbf{0} = v_1x^{(1)} + v_2x^{(2)} + \cdots + v_px^{(p)}$ implies that $v_1 = v_2 = \cdots = v_p = 0$, i.e. the vector $v = \mathbf{0}$. Hence X^TX is invertible.

On the other hand, if X^TX is invertible, then $Xv = \mathbf{0}$ implies that $X^TXv = 0$ and so $v = \mathbf{0}$. Hence the columns of X are linearly independent. ∎

Given $X \in \mathbb{R}^{m\times n}$, then $X^\#$ can be computed using

Proposition 3.4.2. *Let $X \in \mathbb{R}^{m\times n}$.*

- *If* $\operatorname{rank} X = n$, *then* $X^\# = (X^TX)^{-1}X^T$.
- *If* $\operatorname{rank} X = m$, *then* $X^\# = X^T(XX^T)^{-1}$.

Proof. If rank $X = n$, then X has n linearly independent columns and we know that (from Proposition 3.4.1) this implies that X^TX is invertible in $\mathbb{R}^{n\times n}$. Then it is only a matter of verifying that $(X^TX)^{-1}X^T$ satisfies the four defining properties of the pseudoinverse, which completes the first part of the proof.

If rank $X = n$, we simply consider the transpose instead by letting $Y = X^T$. Then rank $Y = m$, since X and X^T have the same rank, and so by the argument above, $Y^\# = (Y^TY)^{-1}Y^T$. However, $X^{T\#} = X^{\#T}$, as is easily checked again from the defining conditions. Hence

$$\begin{aligned} X^\# &= X^{\#TT} = (X^T)^{\#T} && (3.65)\\ &= Y^{\#T} = Y(Y^TY)^{-1} && (3.66)\\ &= X^T(XX^T)^{-1} && (3.67) \end{aligned}$$

which establishes the second part. ∎

In practice, the computation of $X^\#$ is modestly demanding for large matrices. There are many algorithms for approximating pseudoinverses [Kerr 1985; Penrose 1955]. The most common technique involves performing the *Singular Value Decomposition* (SVD) of a matrix [Press *et al.* 1992; Golub and Van Loan 1996], which is a computationally expensive but widely accepted technique for its accuracy. A general discussion of pseudoinverse and SVD is given in Appendix A

For a LSF problem with large M, the direct approach in solving it is to work out the pseudoinverse of a $M \times m$ matrix if $M \neq m$. However, by formulating the normal equation of the problem, we can solve the same LSF problem equivalently by calculating the inverse of a square $m \times m$ matrix if it is non-singular or the pseudoinverse of a matrix if the matrix is singular. One interesting relationship observed is that the optimal solution for A of the LSF problem $XA = I$ (the identity matrix) is in fact the pseudoinverse of X.

3.4.4 Local linear regression

The local linear regression (LLR) algorithm is simply explained in Algorithm 3.7. The advantage is that this statistical modelling is performed locally with a small amount of sample data, usually within a small region in the input space, on the assumption of local linearity. The LSF technique is a widely studied method with many efficient algorithms readily available. In fact this technique is more statistically sound than the previously discussed geometrical modelling technique, which relies

on many heuristic assumptions and suffers the problem of outside query, as opposed to the linear regression technique which does not require extra computational analysis and effort for such outside query prediction.

Given a set of M sample data points $(\boldsymbol{x}_i, y_i)$ representing a mapping of $\mathbb{R}^d \to \mathbb{R}$, estimate the output value y_q of the query $\boldsymbol{x}_q$.

1. Select $p_{\max}$ the number of nearest neighbours used for the linear regression.
2. Use the input vectors $\boldsymbol{x}_i$ to construct a kd-tree.
3. Find the $p_{\max}$ nearest neighbours of $\boldsymbol{x}_q$ from the kd-tree.
4. Construct

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & & x_{2d} \\ \vdots & & \ddots & \vdots \\ x_{p_{\max}1} & x_{p_{\max}2} & \cdots & x_{p_{\max}d} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{p_{\max}} \end{bmatrix} \iff X\boldsymbol{p}^T = Y$$

   via the iterative technique described earlier or simply let $U = X^T X$ and $V = X^T Y$ to construct the normal equation

$$U\boldsymbol{p}^T = V.$$

5. Perform a LSF to estimate the parameters $\boldsymbol{p} = (p_1, p_2, \ldots, p_d)$ by

$$\boldsymbol{p}^T = U^{\#} V$$

   where $U^{\#}$ is the pseudoinverse of U.
6. The output y_q is estimated by $y_q = \boldsymbol{x}_q \cdot \boldsymbol{p}$

Algorithm 3.7: Data modelling using local linear regression

Basically Algorithm 3.7 as it stands assumes that the linearity is passing through the origin, i.e. $p_1 x_1 + p_2 x_2 + \cdots + p_d x_d = y$ but in many cases it would be better to have an extra term to have an affine model

$$p_1 x_1 + p_2 x_2 + \cdots + p_d x_d + c = y, \tag{3.68}$$

where c is a constant. Therefore for the input of the algorithm, we can assume that the input vector becomes (x_{i1}, x_{i2}, …, x_{id}, 1) for estimating the parameters (p_1, p_2, …, p_d, c).

The only problem with LLR is to decide the size of $p_{\max}$, the number of near neighbours to be included for the local linear modelling. Although having more sample data points intuitively can improve the estimation, at the same time this may have the effect of assuming, say, a 'hilly' surface to be linear. Choosing $p_{\max}$ is usually a trial and error process as it depends heavily on the nature of the underlying actual model and of course M. Analysis of the choice of $p_{\max}$ for linear regression is called *influence statistics*. It examines how influential having extra near neighbours is on the accuracy of the linear regression using methods, such as COVRATIO, which measures the effect on the variance-covariance matrix of the parameter estimates [Rawlings 1988] or a method of simply studying the MSE. We made some initial attempts to use influence statistics to make the choice of $p_{\max}$ dynamically adaptive, however, the results were poor and more in-depth study is needed. Nevertheless, we can

always perform a increasing-near-neighbour-test by increasing the size of $p_{\max}$ and study the error of prediction on a set of test data (with known output values) to choose a 'suitable' $p_{\max}$, which minimises the prediction error on the test data, before fixing $p_{\max}$ for further estimation.

3.4.5 Performance analysis

As before, we use the same experimental setup as in Section 3.2.7 to predict points on the surface defined by (3.7) using the same set of training data, but this time we use LLR.

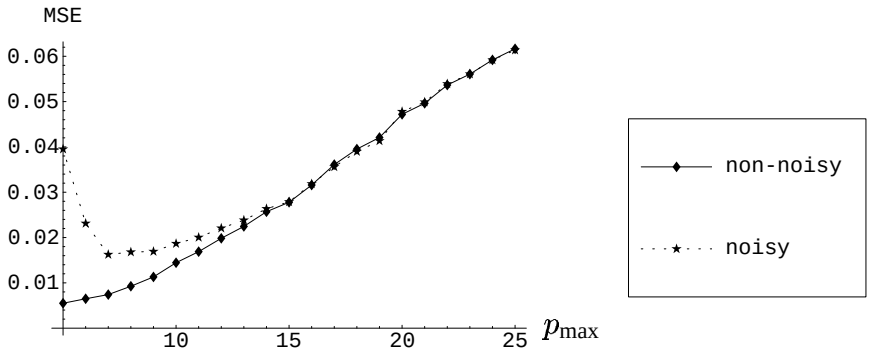


Figure 3.21: MSE against $p_{\max}$, the number of nearest neighbours, varying from 5 to 25 for the local linear regression with affine model without noise and with noise ($\mathrm{Mean}(r) = 0$, $\mathrm{Var}(r) = 0.15$) and $M = 200$.

We first demonstrate the surprising result of varying the size $p_{\max}$, the number of nearest neighbours of the query point for LLR. Interestingly, by varying $p_{\max}$ from 5 to 25 (using a affine model), the MSE of the non-noisy test data seems to increase proportionally as shown in Figure 3.21, contrary to the immediately intuitive idea that having more near neighbours should improve the prediction. Note in this case M is fixed so that, increasing $p_{\max}$ means that the assumed local linear region is larger. The distributions of surface estimation squared error for $p_{\max} = 12$ and $p_{\max} = 5$ are shown in Figure 3.22 and Figure 3.24 respectively. Clearly, for $p_{\max} = 12$ the error (MSE = 0.0198301) is much higher than for $p_{\max} = 5$ (MSE = 0.00550315). The estimated surface for $p_{\max} = 5$ shown in Figure 3.23 looks almost as smooth as the original surface which is surprising, considering so few nearest neighbours are used for the local data modelling. Most errors are concentrated at places with large curvature of the surface. This implicitly signifies that if $p_{\max}$ is too large, the estimation at regions with fast changes of gradient is much poorer, due to the fact that such regions are highly "non-linear". Therefore,

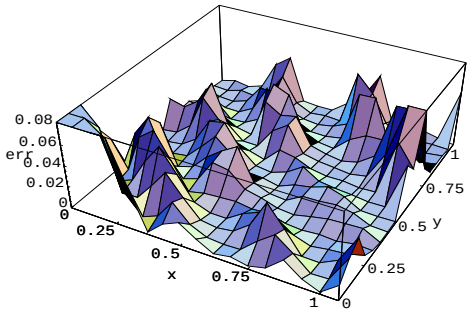


Figure 3.22: Distribution of squared error of surface estimation using LLR with $p_{\max} = 12$ and affine model with MSE = 0.0198301.

having fewer near neighbours will in effect approximate a smaller region, so that the assumption of local linearity is more likely to be valid. However, for data with added normally distributed noise r ($\mathrm{Mean}(r) = 0$, $\mathrm{Var}(r) = 0.15$) the optimal $p_{\max}$ for minimum MSE is higher. Therefore the effect of noise can be 'ironed' out as expected by taking bigger $p_{\max}$ as also shown in Figure 3.21. In general this may required that we increase M.

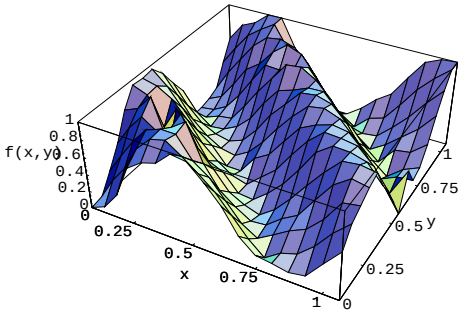


Figure 3.23: Estimated surface using LLR with $p_{\max} = 5$ using affine model with MSE = 0.00550315.

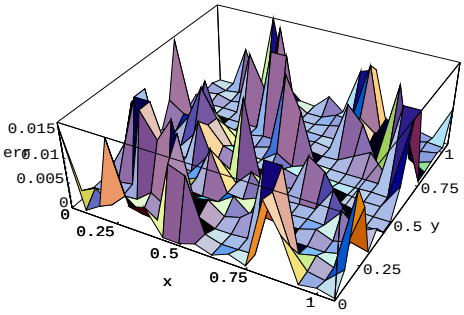


Figure 3.24: Distribution of squared error of surface estimation with $p_{\max} = 5$ and MSE = 0.00550315.

LLR is also tested for performance under noisy data. We then measure the MSE of the estimation on the test data for varying variance $\mathrm{Var}(r)$, of normally distributed noise r added to the output values of the training data, starting from 0.02 to 0.5 in steps of 0.02 using the same experimental setup. The result is shown in Figure 3.25. The noise does have a significant effect on the accuracy of estimation, but in general LSF based LLR performs better than the Delaunay triangulation based technique described earlier.

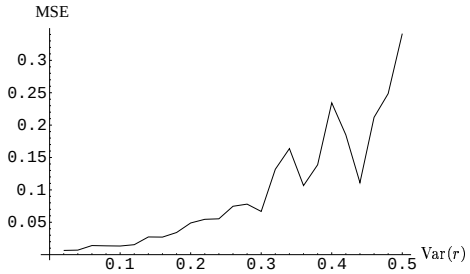


Figure 3.25: MSE error against normal distributed noise r added to output with $\mathrm{Var}(r)$ from 0.02 to 0.5 in steps of 0.02 and of $\mathrm{Mean}(r) = 0$ using LLR ($p_{\max} = 5$, affine model).

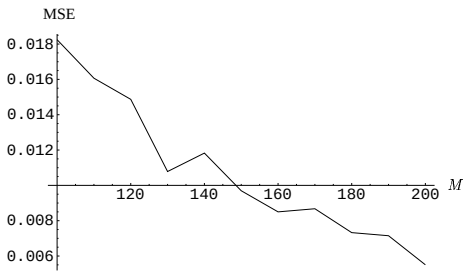


Figure 3.26: MSE of test data against the size of training data M from 100 to 200 in step of 10, using LLR ($p_{\max} = 5$, affine model).

As in the experiment in Section 3.2.7, we also investigate how the size of training data set can affect the MSE of the test data. The expected result is in Figure 3.26, showing that having a reasonably large training data set is essential for better estimation. In fact, there seems to be an inversely proportional relationship between the size of the training data set and the testing MSE.

The effect of the distribution of the training data input space on the prediction squared error of the test data of the same experiment setup with affine LLR with $p_{\max} = 5$ is shown in Figure 3.27. The precise distribution of the training data seems not to be the main factor affecting the modelling performance. However, less dense input data in regions with sharp changes of gradient at the ridges results in slightly degraded estimations. In general, LLR works well with non-evenly distributed training data

in the input space.

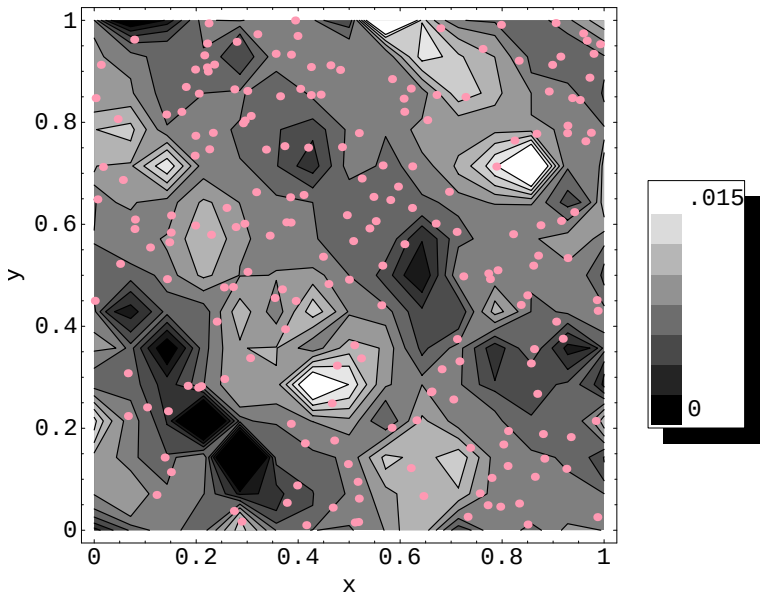


Figure 3.27: Distribution of the squared errors (contour plot) of the test data and the distribution of the training data (point plot) in the input space of LLR surface modelling experiment with $p_{\max} = 5$ using an affine linear model.

3.5 Direct comparison

Using the experimental results from Sections 3.2.7, 3.3.2 and 3.4.5, we make a direct comparison between each modelling technique described so far as a summary. We also discuss other aspects such as the complexity of the underlying model etc. which may also affect the modelling process. Although there are still many other possible factors indirectly determining the effectiveness of our modelling techniques, we shall present only a short discussion.

Figure 3.28 and Figure 3.29 are the combination of Figures 3.15, 3.17 and 3.21 in both the non-noisy and the noisy case respectively. In the non-noisy situation, we can see that having large $p_{\max}$ can improve the modelling for both LDT and GMP. Though by careful examination, having $p_{\max}$ too large will cause both LDT and GMP to degrade in terms of the precision of estimation. On the other hand for the LLR, it would be advisable to use small $p_{\max}$ when there is no noise in the data.

However, in the presence of noise, having slightly larger $p_{\max}$ for LLR can improve the prediction. In general, if $\mathrm{Var}(r)$ increases, the normal procedure is to increase the number of local data (assuming the size of local region is fixed) to obtain a better estimation. However, by increasing $p_{\max}$, we are basically including the number of data from the finite size of training data set for LSF as well as implicitly increasing the size of the local region assumed to be linear. Eventually this will have an adverse effect on the estimation, due to using linear approximation in a 'larger' local region as shown in Figure 3.29.

As $p_{\max}$ increases in the noisy case, both LDT and LLR degrade very rapidly as expected (Figure 3.29). Surprisingly, having very large $p_{\max}$ for GMP can definitely improve the result and the MSE seem to asymptotically approach to an optimal MSE value.

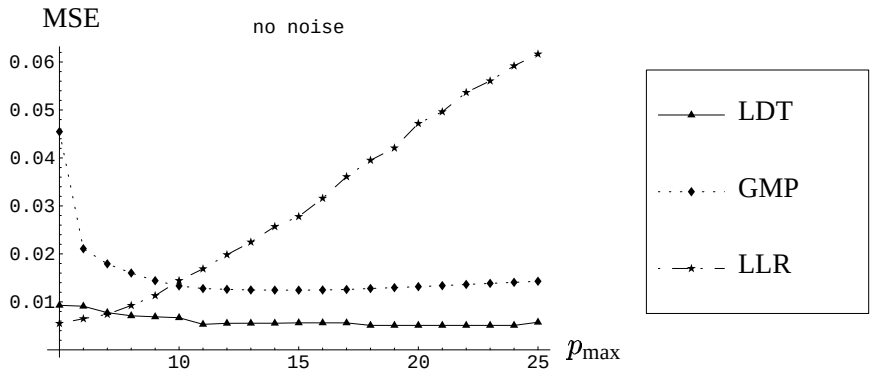


Figure 3.28: MSE against $p_{\max}$ on data without noise for LDT, GMP and LLR.

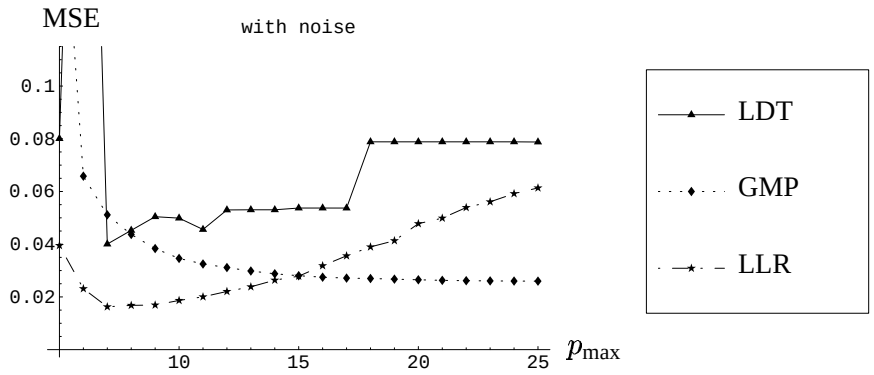


Figure 3.29: MSE against $p_{\max}$ on data with added noise for LDT, GMP and LLR.

Figure 3.30 is the combined result from Figures 3.13, 3.18 and 3.25. As the variance of noise $\mathrm{Var}(r)$ increases all modelling degrades as expected, although LDT and LLR degrade at a much faster rate, especially for LDT, due to the geometrical reconstruction of the surface using noisy output values for prediction. GMP performs well in high levels of noise, perhaps because it is a regression of data based on the distance rather than all the coordinates as used by LLR.

The collected results from Figures 3.14, 3.19 and 3.26, showing the changes of MSEs for each modelling technique in earlier experiments against varying M the number of training data without noise, are shown in Figure 3.31 and demonstrate that having a large M is necessary for accurate prediction for all modelling techniques. Surprisingly, we observe that LLR, compared with the other two, improves at a faster rate as M increases.

It would also be interesting to investigate how the *complexity* of the underlying model to be reconstructed and the sparseness of the training data, in other words the size of M, can affect the performance of each modelling technique. This experiment is very similar to Sections 3.2.5 and 3.2.7 by modelling the surface given some training data. The surface to be modelled is defined by

$$f((x,y),a) = \sin^2[a(x+y)] \tag{3.69}$$

with the bounded input space $[0,1]^2 \subset \mathbb{R}^2$, similar to (3.7). By varying a we can increase the complexity as shown in Figure 3.32.

For this experiment, we use the same sampling of the input space. The sample size M for the training runs from 100 to 200 using a step size of 10. The complexity of the surface is also varied by increasing a of (3.69) from 1 to 6 using a step size of 0.5. Then we use the trained model to reconstruct

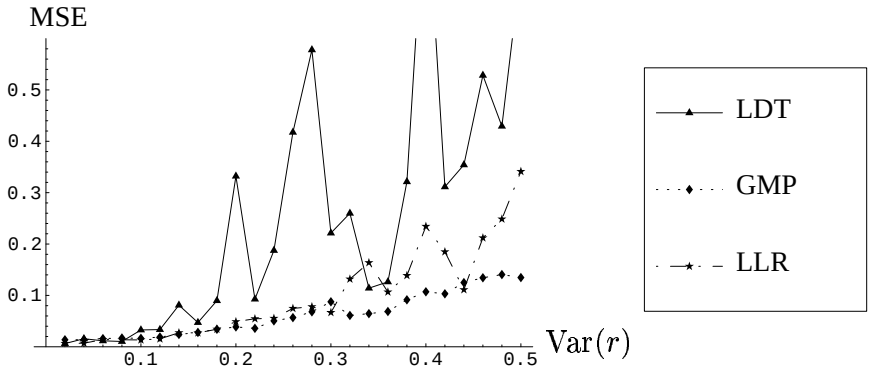


Figure 3.30: MSE against $\mathrm{Var}(r)$ variance of added noise r for LDT, GMP and LLR.

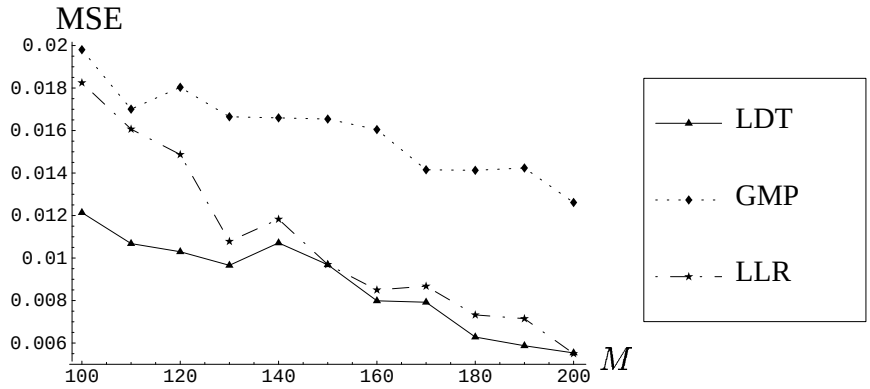


Figure 3.31: MSE against M the number of training data for LDT, GMP and LLR.

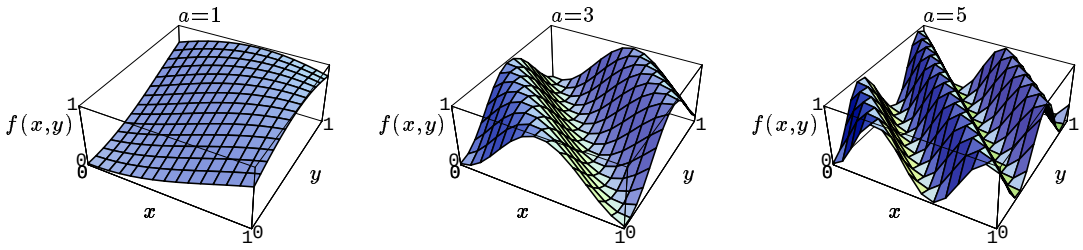


Figure 3.32: Surfaces defined by (3.69) for different values a.

the surface, using the same set of testing data, and calculate the MSE for each model. We also add normal distributed noise r ($\mathrm{Mean}(r) = 0$, $\mathrm{Var}(r) = 0.15$) to the output values of the training data for comparison with the non-noisy case. This is applied to all three modelling technique.

For LDT modelling, we use $p_{\max} = 12$ with OQCS ($\zeta = 0.45$). For the GMP, we set $p_{\max} = 12$ whereas for the LLR, we use an affine model with $p_{\max} = 5$. These choices of $p_{\max}$ etc. are chosen for each modelling technique at optimal or near optimal performance without the presence of noise. Although $p_{\max}$ can be varied, we believe that our choice should give a fair comparison between each modelling technique since different modelling techniques have optimal performance at different values of $p_{\max}$ which are very problem-dependent, especially when the data is noisy.

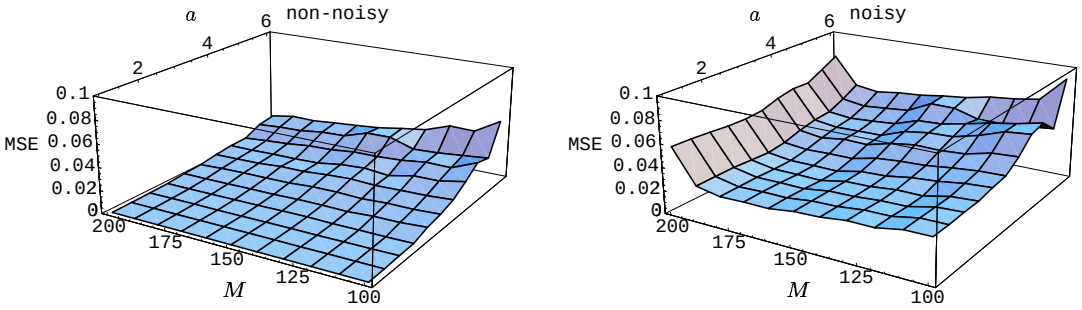


Figure 3.33: Relationship between the MSE of the surface reconstruction, the complexity of the underlying model a and the number of training data M for the LDT modelling experiment.

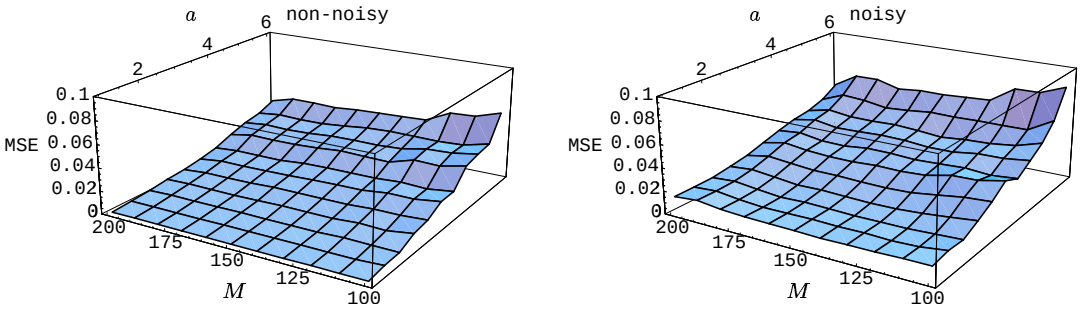


Figure 3.34: Relationship between the MSE of the surface reconstruction, the complexity of the underlying model a and the size of train data M for the GMP modelling experiment.

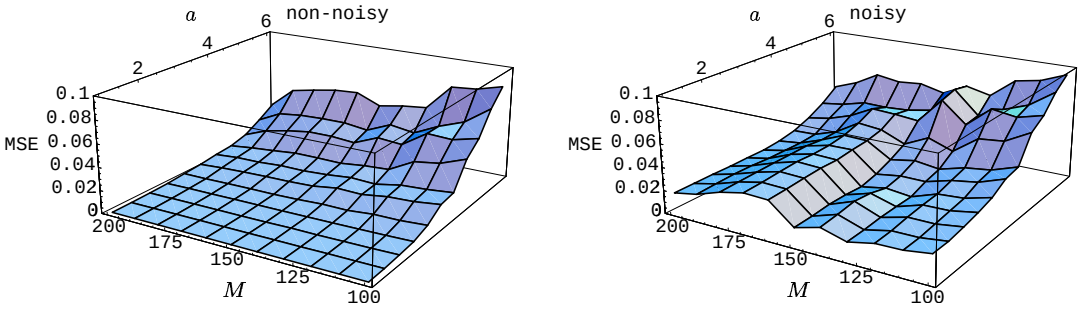


Figure 3.35: Relationship between the MSE of the surface reconstruction, the complexity of the underlying model a and the size of train data M for the LLR modelling experiment.

This result for LDT is shown in Figure 3.33. Without noise and even having high a, the modelling is still fairly good. Of course, having larger M will improve the modelling for large a. In the *noisy* case,

having large M and keeping $p_{\max}$ constant, as discussed earlier, reduces the quality of the prediction for LDT significantly, especially when a is large.

For the GMP modelling as shown in Figure 3.34, large a increases the difficulty of the modelling especially having fewer training data (i.e. smaller M) in both noisy and non-noisy cases.

Figure 3.35 shows the result for LLR. Obviously, in the noisy case, the MSE is high due to the choice $p_{\max} = 5$ only. But in terms of the relationship between a and M, this seems to perform very badly when a is large with M small in contrast with LDT and GMP which perform better in the same case with or without noise. In fact, as expected when the data is sparsely distributed, we would expect that LDT will outperform LLR in non-noisy and complex surface situations. Unusually, GMP performs better in the same case even though GMP is simply a variation of "LLR in squared distances". What we mean here is that the graphs for GMP are similar in terms of noisy and non-noisy situations, unlike the case for LLR, the graph for the noisy case fluctuates by a large amount by just varying M, especially when the surface is more complex. This implies that more data points are needed for the training.

3.6 Discussion

Out of the three modelling techniques, LLR seems to be an efficient and practical method. In terms of running speed, we have not performed any detailed analysis but here we would like to give a brief and general discussion and a very crude estimation of time for the steps involved for the techniques, especially the effect of the dimension d of the input space on the query time.

The GMP and LLR are both fairly fast modelling techniques. They both require the construction of a kd-tree which takes $O(M \log M)$ time. For the GMP, every query involves the restructuring of the near neighbour relationship which in the worst case should take only $O(M \log M)$ but in practice, the time taken is much less. The rest of the computation for the output value is therefore polynomial in time with respect to the selected $p_{\max}$ and M. It does not seem to be highly sensitive to the dimension d of the input space in terms of running speed.

The LLR similarly requires the kd-tree to extract the local near neighbours but it does not restructure such relationships for each query. The most expensive calculation step is to calculate the pseudoinverse and this will depend on the choice of technique for the computation. Using the SVD technique the time involved for such query should be in the order of $O(p_{\max} d^2) + O(d^3)$ (assuming we calculate the SVD of a $p_{\max} \times d$ matrix, see Appendix A) plus the near neighbour query of $O(\log M)$. Therefore, the running time is highly dependent on the dimension d but still polynomial. From our experience, this method seems to be the most efficient of the three techniques discussed.

From our experiments, the LDT seems to be the worst performer in terms of speed. The main drawback is due to the slow calculation of the Delaunay triangulation and the point location problem of finding the correct Delaunay cell containing the query point. In general the computational theory of Voronoi diagrams and the dual Delaunay triangulation is still not well understood, especially in high dimensions, but Seidel [Seidel 1991] has estimated the tight upper bound of the number of cells of the Voronoi (equivalently Delaunay) subdivision to be $O(n^{\lfloor (d+1)/2 \rfloor})$, for n given d-dimensional points. To determine whether a point is in a convex polytope in d-dimensional space, i.e. outside or the inside query, can take as much as $\Theta(n^{\lfloor d/2 \rfloor})$ for convex polytopes defined by the intersection of n half-spaces [Edelsbrunner 1987].

Therefore, given that we have $p_{\max}$ d-dimensional near neighbours we would expect in the worst

case, the query time would take about $O(p_{\max}^{\lfloor (d+1)/2 \rfloor}) + \Theta((d+1)^{\lfloor (d+1)/2 \rfloor})$, plus the time calculating the linear interpolation and the local near neighbours searching time. Of course, since we are using Qhull and the simplices are sorted in the order of the nearest points defining the simplices, the query should take less time. Nevertheless, the performance of LDT will suffer for a very large d. In particular, the running time is exponential in time in terms of d.

However, there is a method which *translates* the whole problem of locating the Delaunay cell which contains the query point into a problem of solving a linear programming problem [Fukuda 1998]. This method became available very late in the present account and it has not been tested, but the detail is explained in Appendix C. This technique, in general, exploits the relationship between the convex hull in the "lifted space" and the Delaunay triangulation as explained earlier in Section 3.2.2 and it seems to bypass many problems that have arisen in the calculation of high-dimensional Delaunay triangulations and the point location problem. Although, the problem of having the query point outside the convex hull of the set of training data points, our 'outside query' problem, remains.

In a recent article, Ekeland [Ekeland 1998] describes [Bombieri *et al.* 1969]'s work on Bernstein's theorem dealing with functions whose graphs are *minimal* surfaces, i.e. the function $f(x_1, \ldots, x_d)$ for which the surface $y = f(x_1, \ldots, x_d)$ minimises the area between all small closed curves drawn on it. Must such functions be linear or must their graphs be hyperplanes? In fact, the answer is yes if the underlying dimension is two, three and up to seven. [Bombieri *et al.* 1969] have proven that it is no longer the case in dimension eight or higher. Although we are not directly interested in differential geometry (but it may be a future research area), this result is very surprising and important for modelling techniques which use hyperplanes.

Therefore, one should be careful in extending any new idea which works in low-dimensional space into higher-dimensional space, e.g. applying a modelling technique assuming linearity in 8- and higher-dimensional problems, may not be necessarily correct as one might assume intuitively. However, we believe that using a small local region modelling technique, and assuming locally piecewise linearity, the error accumulated from such an assumption would not be that high. [Bombieri *et al.* 1969]'s result indirectly implies that using neural network modelling techniques, we may achieve a more accurate and general model for which no assumption of linearity is used within the model. Of course this still suffers from potentially long neural network training times before it can start to predict.

Chapter 4

Practical techniques & examples of modelling

We will present a series of examples of modelling on a few practical problems. These examples demonstrate that the Gamma test has been an invaluable preprocessing tool to aid the model identification process. Before we give these examples we first briefly discuss a few typical preprocessing techniques which are important and necessary in any model construction, as a contrast to our approach in using the Gamma test.

We shall also address the general problem of generating an iterative neural network which can model a given chaotic dynamical system with a high degree of precision. This we do in Section 4.2.3, where we apply the tools developed in the earlier parts of this chapter to produce a feedforward iterative neural network which closely models the Mackey-Glass time series. Although first introduced by means of a particular example, these techniques are quite general and allow is to construct a feedforward iterative network which can accurately model any (reasonable) chaotic dynamical system given only a sufficiently long times series of a single scalar variable of the system. This enables us to produce such networks extremely easily and in Chapter 8 we shall how such networks can be controlled.

4.1 Model identification and data preprocessing techniques

This section briefly introduces two basic but essential preprocessing techniques on given data before passing the data onto the modelling stage.

4.1.1 Embedding

Instead of being given a full state description of the system and a output to be modelled as assumed above, very often there is only one accessible state variable available in the system (especially in real applications) and we are required to use the available past values of this particular variable to predict the future state. In other words, we need to reconstruct the dynamics of the system from the available time series in order to predict the unseen states.

If h is the observable variable this reconstruction is normally done by using *delay coordinates* to construct d-dimensional vector $\boldsymbol{\xi} = (h(t), h(t-\tau), h(t-2\tau), \ldots, h(t-(d-1)\tau))$. If d is chosen large enough and the underlying dynamics is finite dimensional, then there exists a dynamical system describing the evolution of $\boldsymbol{\xi}$ which can be used for our modelling purpose. Assume that the actual

system is described by

$$\frac{d\boldsymbol{x}}{dt} = F(\boldsymbol{x}) \tag{4.1}$$

where $\boldsymbol{x}$ is, say, d'-dimensional. (Further discussion on dynamical systems is given in Chapter 5.) Then the quantity $h(t)$ may be regarded as a smooth function of the state variable $\boldsymbol{x}$. Hence, $\boldsymbol{\xi}$ can be related to $\boldsymbol{x}$ by some function G, i.e.

$$\boldsymbol{\xi} = G(\boldsymbol{x}). \tag{4.2}$$

The important issue now is to ensure that $\boldsymbol{\xi}$ should represent a dynamical system that evolves forward in time such that if $\boldsymbol{x}_0$ denotes a system state and $\boldsymbol{\xi}_0 = G(\boldsymbol{x}_0)$, then there is no state $\boldsymbol{x}_0' \neq \boldsymbol{x}_0$ satisfying $\boldsymbol{\xi}_0 = G(\boldsymbol{x}_0')$. Thus given the delay coordinate $\boldsymbol{\xi}_0 = G(\boldsymbol{x}_0)$, the state $\boldsymbol{x}_0$ is uniquely determined and can be evolved forward any amount in time by (4.1) to a new state, which can then be transformed to the $\boldsymbol{\xi}$ variable by the function G. This basically defines a dynamical system evolving $\boldsymbol{\xi}$ forward in time. Importantly the function G must satisfy the condition that $\boldsymbol{x} \neq \boldsymbol{x}'$ implies

$$G(\boldsymbol{x}) \neq G(\boldsymbol{x}'). \tag{4.3}$$

If this is true, we can then say that G is an *embedding* of the d'-dimensional $\boldsymbol{x}$-space into the d-dimensional $\boldsymbol{\xi}$-space.

Takens [Takens 1981] studied this problem and obtained the result that generically

$$d \geq 2d' + 1 \tag{4.4}$$

is sufficient to avoid the problem of intersections in the embedding space and we refer to this result as the Takens' embedding theorem.

Practically, we often construct the d-dimensional *embedding space vectors* as

$$\boldsymbol{\xi}_n = (x(nt_J), x(nt_J + t_D), x(nt_J + 2t_D), \ldots, x(nt_J + (d-1)t_D)) \tag{4.5}$$

by sampling the time series of a system variable x, to represent the original dynamics. Here t_D is the *delay time*, which is the time period between successive components of each of the embedding space vectors and t_J is the *jump time* which is the time interval between successive vectors. Careful choice of the delay time is essential for a good reconstruction of the chaotic attractor. Many techniques for constructing such embedding vector and determining the jump time and the delay time can be found in [Otani and Jones 1997b; Rosenstein *et al*. 1994]. Later in our experiments we show how the Gamma test can determine a 'good' embedding to aid the construction of an accurate model.

4.1.2 Principal component analysis (PCA) and dimension reduction

Given M data points $\boldsymbol{x}_i \in \mathbb{R}^n$ $(1 \leq i \leq M)$, the *principal component analysis* (PCA) is to find the $(n-m)$-dimensional hyperplane that best represents the data points. Mathematically, the problem is to find an $m \times n$ matrix A whose rows are orthonormal to each other and an n vector $\boldsymbol{a}$ that minimises $\sum_i \|A(\boldsymbol{x}_i - \boldsymbol{a})\|^2$.

The whole process of PCA can be briefly summarised into the following steps:

1. Form the $n \times n$ covariance matrix from the $n \times M$ data matrix.

2. Extract the eigenvectors and eigenvalues from the covariance matrix.

3. The eigenvectors are the *principal components* and the eigenvalues are their *magnitudes*.

Using PCA, we can perform a *feature selection* of the given set of data. It is a process that tramsforms a "data space" into a "feature space". It is designed so that the data may be represented by a reduced number of "significant" and "effective" features which retain most of the intrinsic information content of the data.

First we assume that the vector $\boldsymbol{x}$ has zero mean:

$$E[\boldsymbol{x}] = 0 \tag{4.6}$$

where E is the standard statistical expectation operator. (There is no loss of generalisation: if we have a non-zero mean, we can subtract the mean from the data vectors before proceeding with the analysis.)

Let a $M \times M$ matrix R be the *correlation matrix* of the data, defined as the expectation of the outer product of the vector $\boldsymbol{x}$ with itself,

$$R = E[\boldsymbol{x}\boldsymbol{x}^T]. \tag{4.7}$$

The whole problem of performing PCA is to solve for

$$R\boldsymbol{u} = \lambda\boldsymbol{u} \tag{4.8}$$

a standard *eigenvalue problem*. The precise justification can be found in [Haykin 1994]. Let the eigenvalues of$R$ be denoted by $\lambda_0, \lambda_1, \ldots, \lambda_{M-1}$ and the associated eigenvectors be denoted by $\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{M-1}$ respectively. We can then write

$$R\boldsymbol{u}_i = \lambda_i\boldsymbol{u}_i, \qquad (0 \leq i \leq M-1), \tag{4.9}$$

where we shall assume the eigenvalues are distinct. Let the corresponding eigenvalues be arranged in decreasing order

$$\lambda_0 > \lambda_1 > \cdots > \lambda_i > \cdots > \lambda_{M-1} \tag{4.10}$$

so that $\lambda_0 = \lambda_{\max}$ and let the associated eigenvectors be used to construct a $M \times M$ matrix

$$U = [\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_i, \ldots, \boldsymbol{u}_{M-1}]. \tag{4.11}$$

We can write (4.9) as a single matrix equation

$$RU = U\Lambda \tag{4.12}$$

where Λ is a diagonal matrix defined by the eigenvalues of matrix R,

$$\Lambda = \text{diag}[\lambda_0, \lambda_1, \ldots, \lambda_i, \ldots, \lambda_{M-1}]. \tag{4.13}$$

Note that the U is an *orthogonal matrix* ($U^T = U^{-1}$).

These vectors $\boldsymbol{u}_i$ basically give M possible *projections* of the data vector $\boldsymbol{x}$, i.e.

$$a_i = \boldsymbol{x} \cdot \boldsymbol{u}_i, \qquad (0 \leq i \leq M-1) \tag{4.14}$$

where the a_i are the projections of $\boldsymbol{x}$ onto the principal directions represented by the unit vectors $\boldsymbol{u}_i$ and are called the *principal components*. We can combine the set of projections into a single vector by

$$\boldsymbol{a} = [a_0, a_1, \ldots, a_{M-1}]^T = U^T\boldsymbol{x}. \tag{4.15}$$

Since U is orthogonal we also obtain

$$\boldsymbol{x} = U\boldsymbol{a} = \sum_{i=0}^{M-1} a_i \boldsymbol{u}_i. \tag{4.16}$$

If the vectors $\boldsymbol{u}_i$ are linearly independent, they form a *basis* of the data space. (4.16) is in fact a coordinate transformation, according to which a point $\boldsymbol{x}$ in the data space is transformed into a corresponding point $\boldsymbol{a}$ in the feature space.

The main practical value of PCA is that it provides an effective technique for *dimension reduction*. We may reduce the number of features needed for effective data representation by removing those linear combinations in (4.16) that have small variances and retain only those terms that have large variances [Oja 1983]. In fact the variance of a_i is directly related to the corresponding λ_i. Thus to perform dimensionality reduction on some input data, we can then project the data orthogonally onto the subspace spanned by the eigenvectors (calculated from the correlation matrix R) belonging to the largest eigenvalues. This technique is also referred to as *subspace decomposition*.

In our brief investigation, performing PCA with subspace decomposition on a local scale such as for our local modelling techniques, involving the Gamma test, can only slightly improve the accuracy of modelling but the result is not always necessarily significant. Perhaps, this is due to the fact that for many problems, it is hard to determine whether a principal value is small enough to be removed for the subspace decomposition step. In all our modelling examples, the Gamma test is always used to determine the best embedding in the data preprocessing stage and subsequent PCA on such processed data does not seem to effect any significant improvement. We believe that using the Gamma test to choose the best embedding has already taken care of picking up the significant 'variables', as if performing the PCA dimensionality reduction. The result from our Gamma test approach is comparable with the PCA technique, if not better.

4.2 Practical examples on data modelling and prediction

In this section, we apply our modelling techniques to some practical experiments. At the same time, we introduce other practical techniques and ideas, involving the Gamma test, which help to perform the prediction of time series. First we look at sunspot activity prediction and then we look at how the modelling technique can be used for other applications.

4.2.1 Modelling of sunspot activity and prediction

The data used in this experiment consists of 280 points representing sunspot activity from over the period 1700 – 1979 shown in Figure 4.1. This was used and described in [Weigend *et al.* 1990]. The data was available from the ftp address: `ftp.santafe.edu` in the directory `/pub/Time-Series/data/`. The data has been scaled to $[0, 1]$ range and the variance is estimated to be 0.0410558.

In order to model such time series data, we need to construct the model by choosing an embedding to establish an input-output relationship. Basically, an embedding of a time series is a selection of past values which are used to predict the current value forming a 'mapping' relationship. The choice of a good embedding can produce a good model for quality prediction of future values. We can of course make use of Takens' embedding theorem as introduced in the last section. In our case of the sunspot activity time series, we do not know the exact dimensionality of the sunspot activity dynamics. So

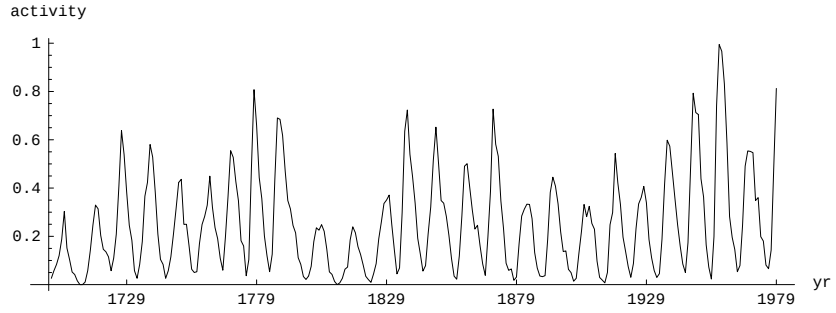


Figure 4.1: Time series of sunspot activity from over the period 1700 – 1979.

we turn to an alternative technique provided by the Gamma test to choose a suitable embedding and dimensionality.

First we define the notion of a *mask*. A mask of, say length 5, is a string of 0's and 1's with length 5. Each binary digit represents one particular past value of a time series x. The rightmost digit represent the most recent value. A "1" indicates to include that value for the embedding and "0" indicates not to include it. Therefore 11111 means that we have an embedding of $x(t-1), x(t-2), x(t-3), x(t-4), x(t-5)$ whereas 11001 represents an three-dimensional embedding of $x(t-1), x(t-4), x(t-5)$. This last type of embedding is called *irregular* or *non-uniform*. Non-uniform embedings were also considered by [Judd and Mees 1998], a paper which we only became aware of in the final phases of present work.

We can run the Gamma test on each different embedding to choose the one which gives us the gamma value $\bar{\Gamma}$ closest to zero. If the embedding dimension is m, then we have $2^m - 1$ embeddings to consider. If m is large, performing the Gamma test on all such embeddings is time consuming and sometimes impractical. Very often, we take at most 20 past data points for such an embedding search or use some heuristic searching techniques such as standard hill-climbing. The main reason why we can obtain a different $\bar{\Gamma}$ value using an irregular embedding is that in effect we are changing the near neighbour relationships between each data point in input-space. Leaving out one particular variable and obtaining a lower $\bar{\Gamma}$ is an indication this variable is either irrelevant or subject to a great deal of measurement noise. Therefore, that variable should not be included for our model reconstruction.

By doing a search on the possible best embedding using 15 past values, using 9 nearest neighbours for the Gamma test, the 8-dimensional embedding 001000100111111 was found with $\bar{\Gamma} = 0.0083971616$, $A = 0.13698427$. It is interesting to note that this embedding uses all past years for half an 11-year cycle and supplements this information with samples approximately bracketing a full cycle. Of course, there might possibly be a better embedding with lower $\bar{\Gamma}$ on using a different value of $p_{\max}$. Also using 15 past values for embedding search arises because we have a very limited size data set which may not be enough for accurate estimation of the true noise r from the data (or for data modelling).

One method to estimate the number of training data required is by the *M-test*. This is simply a series of Gamma tests varying M. If M is sufficiently large, $\bar{\Gamma}$ should be a close estimate of the true value of $\mathrm{Var}(r)$. In other words, M such data points should allow us reconstruct an accurate model. Using the above embedding, an M-test was performed on the sunspot data. Starting from $M = 15$ to $M = 265$ in steps of 5, M data were randomly chosen from the data set and the Gamma test was performed. This was repeated 5 times and then the average of $\bar{\Gamma}$ was taken and plotted against M. Due to the high dimension of the embedding and a small data set, we would not expect that $\bar{\Gamma}$ stabilises as

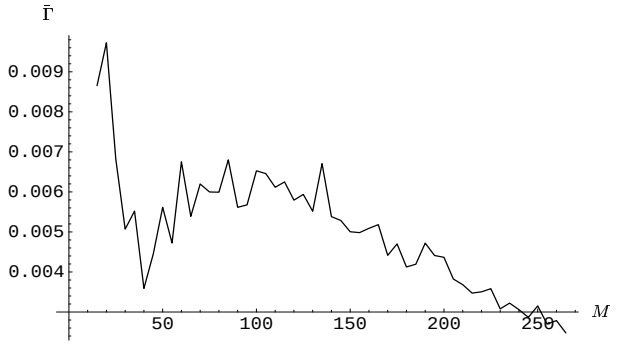


Figure 4.2: M-test on the sunspot embedding data.

shown in the resulting Figure 4.2. Hence, all available data are needed for testing $\bar{\Gamma}$.

Using this embedding, we obtain 267 data points and the first 208 data are used for training the model and the remaining 59 data points are used as test data. Various results using different modelling are shown in Table 4.1. Using LDT, the MSEs are high as expected due to few and sparse noisy training data. In fact, for the same reason it is not a good idea to use any clever heuristic techniques for the outside query problem and just taking the first near neighbour value for the outside query (strategy 1) can produce better result. Taking higher $p_{\max}$ is also helpful.

Model technique	Options	$p_{\max}$	MSE
LDT	strategy 1	40	0.015547
LDT	strategy 5, $\zeta = 0.35$	20	0.0192818
GMP	–	21	0.0146605
GMP	–	25	0.0148654
LLR	affine	59	0.007649749
LLR	non-affine	54	0.005806554
Neural Net	8-10-10-1 (train MSE = 0.0084)	–	0.013834

Table 4.1: Test data MSEs of various modellings of sunspot activity.

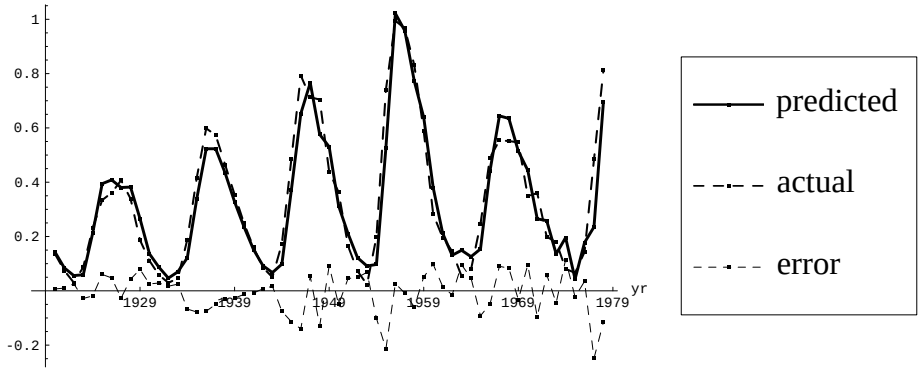


Figure 4.3: Sunspot prediction on test data using LLR with non-affine model with $p_{\max} = 54$.

For the GMP, we can obtain a better result by just having a larger $p_{\max}$. In fact, it is a much faster computation because for LDT, a higher $p_{\max}$ is needed which means performing the time consuming process of Delaunay triangulation of about 40 8-dimensional data. A surprisingly good result can be

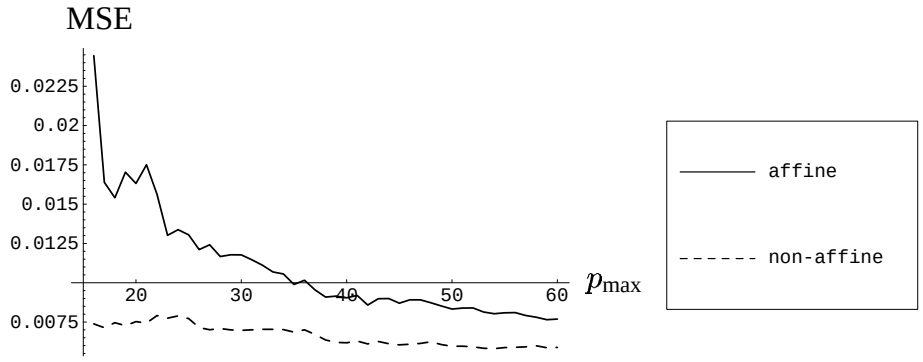


Figure 4.4: MSE against $p_{\max}$ for affine and non-affine LLR modellings on the sunspot activity.

produced by using LLR which is efficient and powerful in the way it can reduce the MSE to about 5×10^{-3}. The non-affine model performed better than the affine model and this was unexpected. The actual result for the non-affine LLR using $p_{\max} = 54$ is shown in Figure 4.3. The non-affine model seems to be a better model for this particular time series as shown in Figure 4.4. In general, other techniques suffer poor prediction for the last two big peaks around year 1958 and 1968 resulting the overall large MSE of the test data whereas LLR can predict it fairly accurately.

As a comparison, we trained a feedforward neural net on the same set of training data until the training MSE reached about 0.0084, the $\bar{\Gamma}$ of the data set. The network uses the sigmoidal function $f(x) = s_F(2/(1 + e^{-xT}) - 1)$, where the temperature $T = 1.2$ and the scale factor $s_F = 1.5$, as the activation function and it is trained by using the BFGS algorithm (see Appendix D). As indicated by the M-test, we would not expect the network to perform well. However, the MSE on the test data is rather pleasing and closely matches the MSEs for many modelling techniques studied, but it is still not as good as the LLR as shown in Table 4.1. In this small problem, the training time required is not long but for a larger data set, this would not be chosen as the ideal technique for a 'quick' answer.

We then tried a dimension reduction approach based on the idea of dimension reduction via PCA in a local scale for the LLR modelling technique as an improvement method. The idea is to take advantages of the calculation of the inverting matrix stage. To solve for a best solution for A in $XA = Y$, as shown in Section 3.4 typically, we need to calculate the pseudoinverse $X^{\#}$ of the matrix X, say it is a $m \times n$ matrix, so to obtain $A = X^{\#}Y$. Computationally, $X^{\#}$ can be calculated using the Singular Value Decomposition (SVD) of the matrix X by expressing

$$X = UDV^T \tag{4.17}$$

where $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices. $D \in \mathbb{R}^{m\times n}$ is a diagonal matrix with entries $D_{ii} = w_i$, $1 \le i \le r$, $w_1 \ge w_2 \ge \cdots \ge w_r > 0$, where $r = \operatorname{rank} X$ and all the other entries are zeroes. Further discussion of SVD is in Appendix A. Then

$$X^{\#} = V \underbrace{\begin{bmatrix} W^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{n\times m} U^T. \tag{4.18}$$

where W is a $r \times r$ diagonal matrix of w_i. The inverse W^{-1} is then a diagonal matrix of $1/w_i$.

We now define a tolerance value T_r, so that a threshold $T_{\text{thr}} = T_r \times w_1$ is defined, i.e. the threshold T_{thr} is a percentage of the largest w_i. If $w_i < T_{\text{thr}}$ then $1/w_i$ is set to zero in (4.18) for the calculation of the pseudoinverse $X^{\#}$. Setting $1/w_i$ to zero for small w_i is in fact important for practical numerical

computation since it reduces any floating point arithmetic accumulated error due to the reciprocal of *small* w_i, as well as removes the 'least important' information, e.g. noise, from our input data. This is the same as removing some of the basis set of vectors in the matrix U spanning the subspace.

Notice that $U = XX^T$ is the correlation matrix of X (see Section 4.1.2). The squares of the entries w_i are in fact the non-zero eigenvalues of XX^T, in order words, the magnitude of the principal vectors of the PCA of X. Zeroing the entries corresponding to small w_i effectively removes those principal components with small variances, although the data in the above case is not expressed in the feature space. Therefore this technique is not exactly a PCA dimension reduction. The relationship of SVD and PCA is studied and further demonstrated in [Gerbrands 1981].

We then apply our simple but 'crude' dimension reduction technique, by setting a suitable value T_r, on the $p_{\max}$ local data points before linear regression, with the same embedding and non-affine model using $p_{\max} = 54$. However, this does not improve the model in a significant way and occasionally, the modelling seems to be worse. The main difficulty in applying this technique is to determine the threshold value for removing unwanted spurious and insignificant components. As a control experiment, we used the full embedding 1111111111111 using non-affine LLR with $p_{\max} = 54$ and varied threshold value T_r to compare the test MSEs. The results are in Table 4.2. From these results, it seems that the Gamma test modelling approach for choosing a suitable embedding is comparable to using the locally dimensionality reduction technique for LLR modelling, but the Gamma test has provided a deterministic way to aid the modelling process, whereas this dimension reduction technique requires the careful choice of T_r. Too large a value of T_r could render the local linear regression inaccurate.

T_r	$p_{\max}$	MSE
0.1	54	0.012119
0.05	54	0.007819
0.005	54	0.0088599
1×10^{-6}	54	0.0088599

Table 4.2: MSEs of test data on non-affine LLR sunspot modelling using $p_{\max} = 54$ with various T_r for component removal.

Usefulness of the Gamma test embedding search

From the above example, we can clearly see that having an *irregular* embedding as opposed to the standard approach taking regular past time lags, as suggested by Takens, a good model can be reconstructed from a finite set of time series data.

To explicitly demonstrate the usefulness of this Gamma test approach for identifying the 'dependable' past lags, we construct a chaotic map for which the current system state does not depend on the immediately previous two states but directly depends on the values of the further past states. We construct the following system,

$$x_n = -1.4x_{n-5}^2 + 0.05x_{n-6}, \tag{4.19}$$

in a form similar to the Hénon map

$$x_n = -1.4x_{n-1}^2 + 0.3x_{n-2}, \tag{4.20}$$

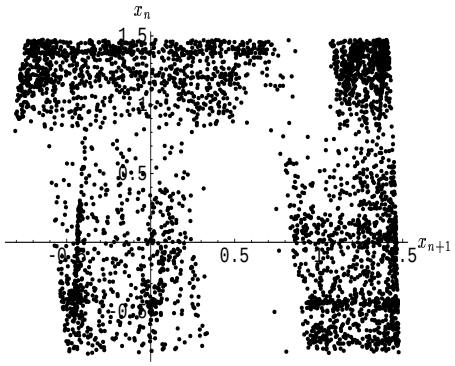


Figure 4.5: Chaotic attractor of the modified the map defined in (4.19).

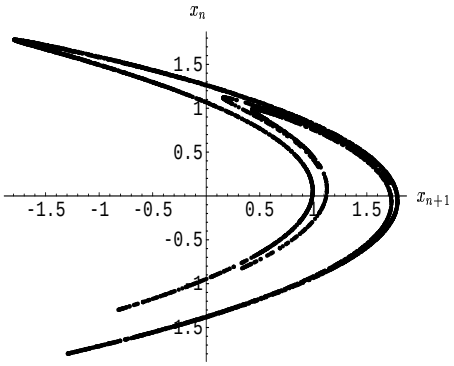


Figure 4.6: Chaotic attractor of the original Hénon map in (4.20).

but instead depending on the system's fifth and sixth lagged values. The chaotic attractor from (4.19) is shown in Figure 4.5, which is very different from the original Hénon map attractor as shown in Figure 4.6. If we use the standard embedding technique, we could take the embedding 111111 so as to include the fifth and the sixth delayed states. However, this would presumably pick up unnecessary values, the first delayed state, the second delayed state etc., for the system reconstruction. Very likely, the extra values may behave as noise in the dynamic reconstruction.

Using (4.19) we generate a time series of 6000 samples for this experiment. By having a fixed length of 10 lags, we can construct 2^{10} different embedding vector data sets. For each data set, we can perform the Gamma test on the embedding vectors. By comparing the returned $\bar{\Gamma}$ values, we can obtain the best embedding which gives the least $|\bar{\Gamma}|$. Table 4.3 show the 10 best embeddings from this full search. Of course, other searching techniques such as using genetic algorithm or hill-climbing can approximate a best embedding without needing to compute $\bar{\Gamma}$ for all embeddings. In fact, simply having increasing embeddings and computing the $\bar{\Gamma}$ values is good enough for a fast solution.

Order	$\bar{\Gamma}$	A	Embedding
1	6.879299×10^{-6}	0.338208	0100110001
2	7.484070×10^{-6}	0.314507	0101111001
3	1.521949×10^{-5}	0.174325	1001111010
4	-1.821650×10^{-5}	0.379670	0000110000
5	1.848148×10^{-5}	0.287325	0011110010
6	1.891623×10^{-5}	0.426509	1000110000
7	-2.002271×10^{-5}	0.276795	0000111011
8	2.034062×10^{-5}	0.541767	0000110000
9	-2.106169×10^{-5}	0.343622	0000110011
10	2.990317×10^{-5}	0.381624	0001110100

Table 4.3: A list of 'good' embeddings sorted in ascending order of $|\bar{\Gamma}|$.

As seen in the result in Table 4.3, some of the $\bar{\Gamma}$ values are negative which is probably caused by statistical noise, especially when the $\bar{\Gamma}$ are small and close to zero. We could simply ignore those embeddings which give negative $\bar{\Gamma}$, but if the asymptotic $\bar{\Gamma}$ is sufficiently small the resulting embeddings should produce a good model. As expected, the embedding 0000110000 is in the list, because this is the recurrence relationship used to define the system (4.19). Surprisingly, there are five better embed-

dings with positive $\bar{\Gamma}$ values. Indeed, such embeddings often produce better models. Note that the fifth and the sixth past lag states also appear in those five better embeddings. This experiment demonstrates that the Gamma test can easily help us to determine an appropriate embedding constructed from a reasonably sized set of time series data.

4.2.2 Detecting a message buried in a chaotic carrier

This section is basically a summary report on the joint mini-project with Ana Oliveira who is interested in secure communication via synchronisation of chaos. This experiment is to use our modelling technique in an attempt to model a digitised chaotic signal to detect and retrieve hidden binary messages within such a carrier signal, i.e. another demonstration of the usefulness of this simple modelling scheme (See [Oliveira *et al.* 1999] for the main result).

Using synchronisation to secure communication has been an actively researched area [Cuomo and Oppenheim 1993; Oketani and Ushio 1996; Parlitz *et al.* 1992; Pecora *et al.* 1997]. The method used is to assume that we have two identical chaotic systems, one in the transmitter and one in the receiver and select one of the chaotic system variables of the transmitter as a carrier for the transmitted message. Using a suitable synchronisation technique, the message can then be decoded from the chaotic carrier.[1]

We are interested in decoding the message without using any synchronisation techniques and no knowledge of the dynamical system used to generate the carrier. There have been several attempts such as a forecasting approaching (one-step predictor) involving filtering in the frequency domain in [Short 1994] and a technique without filtering in the frequency domain in [Short 1997]. Nevertheless, our method appears to be simpler, using the combined Gamma test and LLR strategy. By modelling the carrier and using a one-step predictive model, the binary message should appear as noise or a large error signal when a model prediction is compared with the received signal. We have tried the method on two different message encoding schemes, a binary message masked by adding it to a chaotic carrier and a binary message modulated in one of the system bifurcation parameters.

Masked message

First we masked a binary message as a square wave signal as in Figure 4.8 into the y variable (Figure 4.7) of the Chua circuit which is defined as

$$\begin{cases} \dot{x} = \alpha(y - x - f(x)) \\ \dot{y} = x - y + z \\ \dot{z} = -\beta y \end{cases}, \tag{4.21}$$

where

$$f(x) = bx + \frac{1}{2}(a - b)(|x + 1| - |x - 1|) \tag{4.22}$$

and α, β, a, b are constants and set to be: $\alpha = 10$, $\beta = 14.87$, $a = -1.27$, $b = -0.68$. One digit of the binary message is a 'square' peak for a duration of 0.8 time unit. The message is completely hidden by masking as in Figure 4.9.

As before the Gamma test was used to find a good embedding for the modelling of the time series of y sampled at every 0.01 second (see Figure 4.9). The embedding 0011110001 was found with $\bar{\Gamma} = 1.8219 \times 10^{-6}$ and $A = 0.29044$ using about 10000 data points. As suggested by the M-test,

[1] The detail of the synchronisation is omitted here - see [Oliveira *et al.* 1999].

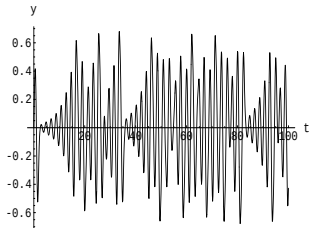


Figure 4.7: y of the Chua circuit defined in (4.21).

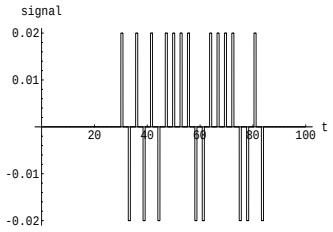


Figure 4.8: Binary message to be encoded/masked.

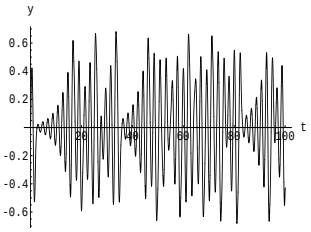


Figure 4.9: y signal containing the binary message in Figure 4.8.

3000 training data were taken for the affine LLR model construction. With $p_{\max} = 8$, we obtain a MSE of 3.3338×10^{-5} on the test data. The result is shown in Figure 4.10. The retrieved message appears as patterns on the error time series. The '0' and '1' of the message appear as pairs of 'blips' as in Figure 4.11 and Figure 4.12 respectively on the error time series. Therefore we can fairly easily detect 'blips' using our technique, but how well it can be used for distinguishing the difference between a '0' and a '1' signal, i.e. a recognition problem, requires further future investigation.

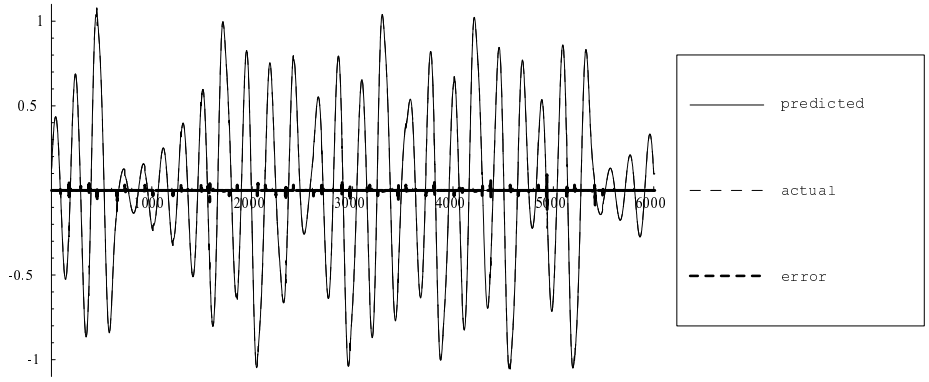


Figure 4.10: The result of the affine LLR model on the test data of the chaotic carrier with masked binary message of the y variable of the Chua circuit.

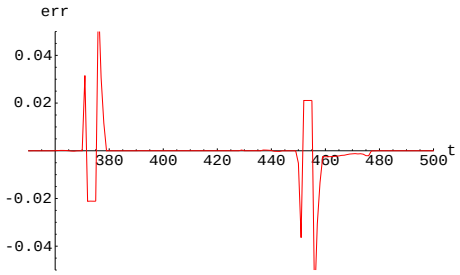


Figure 4.11: A '0' signal on the error time series.

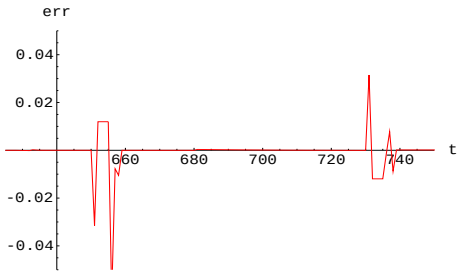


Figure 4.12: A '1' signal on the error time series.

Similarly, we tried the same problem but with the binary digit encoded as a single 'flash' blip signal (duration of 0.01 second for each digit) and successfully retrieved the masked message. Also, noise was added to the carrier to increase the difficulty of the modelling but we could again successfully retrieve the message up to certain amount of noise.

Modulated message

Next we tried the same technique on binary message modulated into one of the system parameter. The Lorenz system defined by

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = rx - y - xz \\ \dot{z} = -bz + xy \end{cases}, \tag{4.23}$$

where σ, r and b are constants, was used for the modulation scheme by using the parameter r as in [John and Amritkar 1994]. $\sigma = 10$ and $b = 8/3$. A '1' corresponds to a positive change in the parameter and if after a time interval $\delta t = 20$ no change occurs then a '0' is encoded. Each binary digit was encoded at evenly spaced time interval. We varied r between $r = 28.0$ and $r = 30.0$ for the encoding and we can decode the message from the carrier without much difficulty. Smaller change δr was tried but were not successful.

The binary message to be modulated, 10011010, is as Figure 4.13 and the message was modulated into r of the system. Figure 4.14 shows and Figure 4.15 shows the time series of y with and without the modulation with both starting at the same initial conditions. y from the modulated system clearly appears as another chaotic time series.

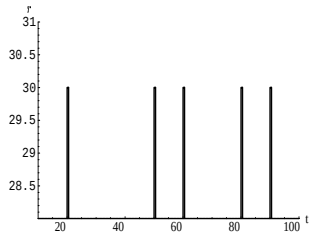


Figure 4.13: The binary message 10011010 is encoded as a variation of r of the Lorenz system.

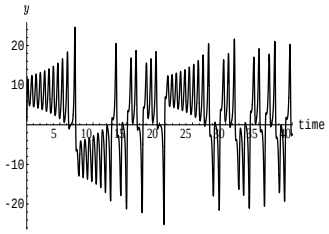


Figure 4.14: The original time series of y without modulation.

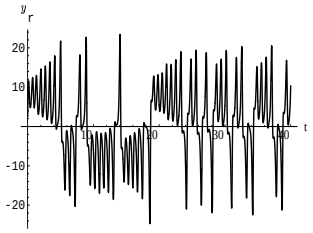


Figure 4.15: The time series of y starting at same initial conditions but with message modulated.

Using the same modelling procedure again, a embedding 1001110001 was found using 10000 data points, for the time series y of the system, i.e. the carrier, sampled at every 0.01 second. The corresponding Gamma value is $\bar{\Gamma} = 4.0067 \times 10^{-5}$ and $A = 0.21089$. The number of training data used is $M = 6000$ as estimated by the M-test. The model was created and tested and we obtained MSE of 4.5525×10^{-4} on the test data. This MSE value, relative to the other experiments, is not very small. One possible reason being that the Lorenz system is slightly harder to model. Nevertheless, the model result is shown in Figure 4.16 and the binary '1' can still be located by observing the pattern in Figure 4.17 on the errors.

This illustrates how simple and powerful this modelling strategy of combining the Gamma test and LLR really is. Further details on the application to eavesdropping a chaotic carrier are reported in [Oliveira *et al.* 1999].

4.2.3 Modelling a chaotic process by a neural network

Constructing a neural network from a chaotic map has been an interesting and significant area for the investigation of neural dynamics [Welstead 1991; Tsui and Jones 1997]. The typical method of constructing such a network is to choose a chaotic map and use a set of input and output data from

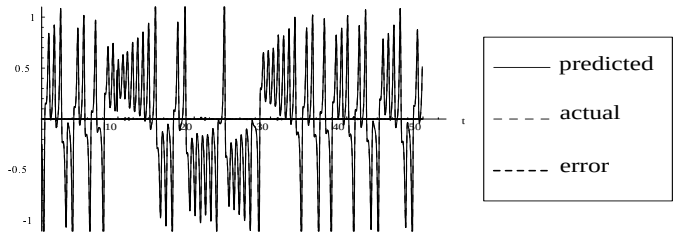


Figure 4.16: The predicted model result for the y time series.

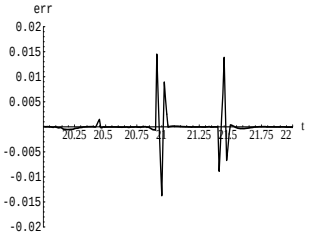


Figure 4.17: A '1' signal on the error time series.

this map to train a feedforward neural network. Then the outputs of the network are immediately fed back to the inputs to form a recurrent network. This is the approach used in our early experiment in studying chaos control in neural network as shown later and described in [Tsui and Jones 1997].

In practice, this is a rather difficult process due to the fact that the current state alone does not always contain the necessary information to predict the next state. In this section we apply the Gamma test modelling approach which can simplify the construction process, as well as introduce a new type of neural model which can capture the essential dynamical features of a given chaotic time series.

To capture the chaotic dynamics given by a time series of a system variable, we first use a sequence of Gamma tests to determine a best embedding (i.e. an embedding which minimises $|\bar{\Gamma}|$) so as to select the best set of inputs for the model dynamics $F : \mathbb{R}^d \to \mathbb{R}$. The dimension d is given accordingly by the best embedding.

As an example, we look at the Mackey-Glass equation defined by

$$\frac{d}{dt}x(t) = -0.1x(t) + \frac{0.2x(t-\tau)}{1-(x(t-\tau))^{10}}, \tag{4.24}$$

where $\tau = 30 > 17$ is the time delay. We then generate a time series of 800 points sampled at intervals of $\Delta t = 10$. Using a six dimensional embedding 111111, we reformatted the data into 794 data points which were then put to a sequence of Gamma tests to find the best embedding. The embedding 111100, which gave $\bar{\Gamma} = 0.00093817$ and $A = 0.30222$, was obtained. It is interesting to note that the full embedding search obtained the best model by omitting $x(t - 1 \cdot \Delta t)$ and $x(t - 2 \cdot \Delta t)$. Why is this? In the original time delay equation the value $x(t)$ depends on the value $x(t - 30)$. The values $x(t - 10)$ and $x(t - 20)$ are not needed at all, as the software discovered. This illustrates the utility of

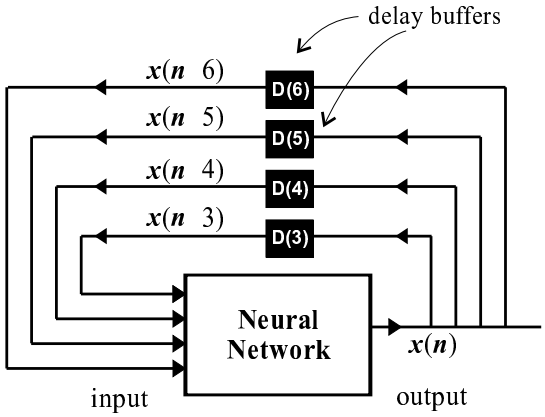


Figure 4.18: A recurrent neural net with delayed inputs, suggested by the embedding found by the Gamma test, for modelling a chaotic time series.

the Gamma test in finding the best embedding in a dynamical system with lags.

The M-test suggested that a minimum of around 500 data points were required for the model. In fact we chose to use 550 data points to train a feedforward neural network with the architecture 4-8-8-1 until the training MSE was about 0.000936. The feedforward neural network was trained using the BFGS algorithm, a quasi-Newton method (described in Appendix D), with an output function $f(x) = 1.5(2/(1+e^{-1.2x})-1)$, where x is the usual activation function, as used in the earlier sunspots experiment. The remaining 293 data points were tested, giving MSE of 0.001623. As suggested by the embedding 111100, we can feed the output of the system back into the input to construct an iterative network having delay feedback lines. An example of the architecture for this embedding is illustrated in Figure 4.18. Delay buffers are used on the feedback lines to give past values so that a map $F : \mathbb{R}^4 \to \mathbb{R}$ can be correctly represented.

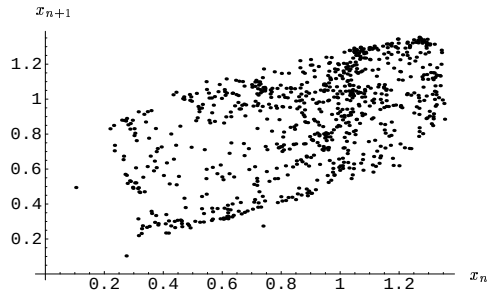


Figure 4.19: Mackey-Glass attractor from the sampled time series.

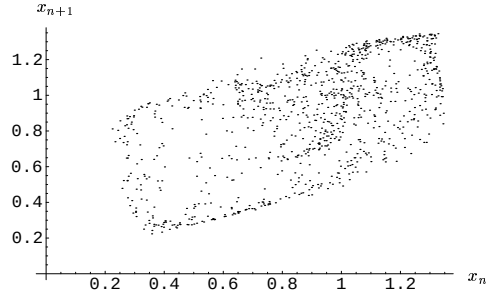


Figure 4.20: Chaotic attractor of the trained neural network.

The true attractor plotted from the sampled time series of this system and the trained neural net attractor are in Figure 4.19 and Figure 4.20 respectively, showing the similarity between them.

This idea of using suitably chosen time delayed feedback lines in an iterative neural network to obtain a feedforward neural model of a chaotic dynamic system specified by a single time series is an important step towards our ultimate goal. We shall return to this topic in Chapter 8. However, we note that this method is really an innovative application of Takens theorem to neural network modelling.

4.3 Discussion

We have presented several examples in which the Gamma test is used as a means of model identification to determine a 'good' embedding from which an iterative model of the time series can be constructed. Using *irregular* embeddings very often a better model than that provided by a literal interpretation of Takens theorem can be obtained. This is illustrated in our construction of a model for sunspot activity and by the detection of a message buried in a chaotic carrier.

The determination of the significant delayed components of the Mackey-Glass time series detected by the Gamma test also helped us in constructing a chaotic neural network which models this dynamics.

By incorporating several suitable delay buffers, based on a good embedding, and using these feedbacks to the inputs, a chaotic time series can easily be modelled. Previously training such an iterative neural network (to model a given chaotic system) was considerably harder and contained some element of 'hit-and-miss'. The techniques provided here have enabled us to construct a wealth of chaotic neural systems (only one of which is presented in this thesis) which can then be used as the basis for experiments in control and synchronisation [Oliveira 1999].

However, before we can construct our chaotic neural stimulus-response model, we need to study the nature of *chaos* and the techniques which have been used to control it.

Chapter 5

Chaotic Dynamics & Control of Chaos

This chapter attempts to give a concise description of the basic idea of dynamical systems and chaos, and introduces the terminologies used throughout the remainder of the thesis. Essential tools and techniques for studying chaotic dynamical systems are presented. Some further examples of chaotic neural dynamics are also given.

Finally we begin to explore the basic ideas required to effect *control* of a chaotic system. Several chaos control strategies will be described and illustrated with simple experiments.

5.1 Dynamical systems

The subject of *Dynamical systems* is a mathematical attempt to understand processes which evolve in time. A dynamical system may be defined as a deterministic mathematical prescription for evolving the state of a system forward in time. Time here either may be a continuous variable, or else it may be a discrete integer-valued variable. A typical continuous dynamical system is defined as:

$$\frac{dx(t)}{dt} = F\left[x(t)\right] \tag{5.1}$$

where $x(t)$ is an d-dimensional vector $(x_1, \ldots, x_d)$ representing a state of the system and it may be thought of as a point in a suitably defined space – which we shall call *phase space* or *state space*. For any initial state $x(0)$ of the system (5.1), we can in principle solve the equations to obtain the future system state $x(t)$ for $t > 0$. The path in state space followed by the system as it evolves with time is referred to as an *orbit* or *trajectory*. A trajectory therefore displays the history of the states of the system.

We describe here some terminology regarding dynamics of the trajectories. A *limit set* is a set of points in state space that a trajectory repeatedly visits, and it is defined only for discrete or continuous *autonomous*[1] systems. A *limit cycle* is a periodic solution of the system. The limit set is *stable* if all nearby trajectories remain nearby and it is *unstable* if no nearby trajectory, except those lying on the limit set, remain nearby. Sometimes the trajectory in state space will head for some final attracting region which might be a point, curve, area and so on. Such an attracting object is called the *attractor* of the system, since a number of distinct trajectories will be attracted to this set of points in the state space. The set of all initial conditions leading to trajectories that approach a given attractor is called the *basin of attraction* for that attractor.

[1] Time t is implicit to an *autonomous* system, i.e. $x(t) = f(t)$ for a continuous system. Otherwise if time t is explicit in the system, then such system is *non-autonomous*.

5.2 Chaos

Before we can give a formal definition of chaos, it is necessary to introduce some mathematical definitions:

Definition 5.2.1. *Let X be a metric space and let $Y \subset X$, then Y is* dense *in X if $\forall x \in X$, $\exists y \in Y$ arbitrary close to x.*

Definition 5.2.2. *A dynamical system is* transitive *if for any pair x, y and any $\epsilon > 0$, there exists a z within ϵ of x whose orbit comes within ϵ of y.*

Definition 5.2.3. *A dynamical system F* depends sensitively on initial conditions *if $\exists \beta > 0$ such that for any x and any $\epsilon > 0$, there is a y within ϵ of x and a t such that $d\left[F\left(x(t)\right),\ F\left(y(t)\right)\right] \geq \beta$ where d is a metric.*

Informally a chaotic dynamical system is a system which may superficially appear to behave randomly but when the system starts off at the same initial point, it always produces the same orbit. There does not seem to be an universally agreed definition of chaos between mathematicians. Here is one formal definition of chaotic dynamical system from Devaney [Devaney 1992].

Definition 5.2.4. *A dynamical system F is* chaotic *if*

- *The set of periodic points is dense,*
- *F is transitive,*
- *F depends sensitively on initial conditions.*

It is a characteristic of chaotic dynamics that the resulting attractors often have a much more intricate geometrical structure in the state space than those of *regularly behaving* dynamical systems. The dimension[2] of these attractors is not an integer. Such geometrical objects are fractals [Mandelbrot 1982]. When an attractor is *fractal*, it is called a *chaotic attractor* or a *strange attractor* [Ruelle and Takens 1971].

5.3 Essential tools

Some (but not all) essential mathematical tools for studying chaotic dynamical systems are introduced here. They are briefly explained and interested readers should be able to find them in any standard dynamical system books, e.g. [Devaney 1992; Hilborn 1994].

5.3.1 Bifurcation diagram

A non-linear dynamical system, say $F_r(x)$ where r is a system parameter, could change suddenly in terms of qualitative and quantitative behaviour as a result of a small change in some control parameter r, e.g. from order to chaos. *Bifurcation* is the word for describing such a sudden change in the nature of system as a control parameter is varied. To understand bifurcation behaviour, it is often helpful to look

[2]The standard definition of dimension is the *box-counting dimension* or the *capacity dimension*. We can imagine covering the space by a grid of N-dimensional cubes of edge length of ϵ. We then count the number of cubes $M(\epsilon)$ needed to cover the set. We do this for successively smaller ϵ values. Then the dimension is defined by $D_0 = \lim_{\epsilon \to 0} \ln M(\epsilon)/\ln(1/\epsilon)$.

at the *bifurcation diagram*. This is a picture in the r, x-plane of the relevant fixed and periodic points as functions of r. Therefore it is a plot of the periodic points for each parameter value r. An example of a bifurcation diagram is shown in Figure 5.4. The way to generate such diagrams is described in Algorithm 5.1.

Assume the system is $F_r(x)$ where r is a parameter $r_0 \leq r \leq r_s$ and x is a state of the system.

1. Set parameter $r = r_0$ initial parameter value.
2. Set $x_0 =$ system initial state.
3. Iterate the map F_r, say, 500 times (or more to remove transient states).
4. Iterate the map F_r another 1000 times (starting from x_{500}) and plot the resulting values of x.
5. Increase r by a small amount, $r \rightarrow r + \epsilon$ (The size of ϵ depends on the range of the parameter r), and *if* $r > r_s$ stopping parameter value *then* exit *else* return to step 2.

Algorithm 5.1: Generate bifurcation diagram of a dynamical system.

5.3.2 Poincaré section

A *Poincaré section* (or a Poincaré map) is a device invented by Henri Poincaré as a means of simplifying the analysis of a continuous dynamical system (or 'flow'), $d\boldsymbol{x}/dt = F(\boldsymbol{x}(t))$, $F : \mathbb{R}^d \rightarrow \mathbb{R}^d$, to a discrete map.

Considering d first-order autonomous ordinary differential equations, the Poincaré section represents a reduction of the d-dimensional flow to an $(d-1)$-dimensional map by choosing some appropriate $(d-1)$-dimensional surface Σ (a global cross section) in the d-dimensional phase space satisfying

- every orbit of F meets Σ for arbitrarily large positive and negative time and;
- if $\boldsymbol{x} \in \Sigma$ then the flow at $\boldsymbol{x}$ is not tangent to Σ.

Let $\boldsymbol{x}_0 = \boldsymbol{x}_0(t) \in \Sigma$ and define $\tau_F : \Sigma \rightarrow \mathbb{R}^+$ such that $\tau_F(\boldsymbol{x}_0) = \tau > 0$ is the least time for which $\boldsymbol{x}(t+\tau) \in \Sigma$. The Poincaré section of the flow through Σ is defined as

$$\mathcal{P} = \{\boldsymbol{x}(t + \tau_F(\boldsymbol{x}_0)) | \forall \boldsymbol{x}_0 \in \Sigma\} \tag{5.2}$$

i.e. each point on Σ is evolved forward in time until the trajectory intersects Σ. The set of all such re-intersections is the Poincaré section.

5.3.3 Lyapunov exponent

The *Lyapunov exponent* of a map may be used to obtain a measure of the sensitive dependence upon initial conditions that is characteristic of chaotic behaviour. For a one-dimensional iterative map, $x_{n+1} = f(x_n)$, the system is allowed to evolve from two slightly differing initial states, x and $x + \epsilon$, then after n iterations their divergence may be characterised approximately as

$$\epsilon(n) \approx \epsilon e^{n\mu} \tag{5.3}$$

where the Lyapunov exponent μ gives the average rate of divergence. The difference between two initially nearby states after the n^{th} step is written as

$$f^n(x + \epsilon) - f^n(x) \approx \epsilon e^{n\mu}. \tag{5.4}$$

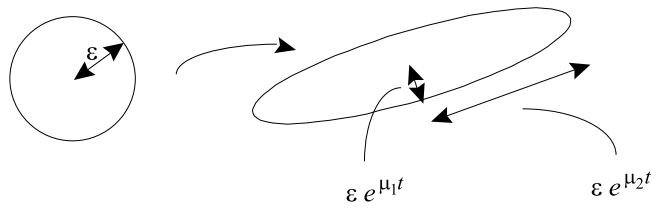


Figure 5.1: A two dimensional example of the calculation of Lyapunov exponents - the evolution of a sphere of initial points to an ellipsoid.

For small ϵ, using the chain rule for the derivative of the n^{th} iterate and taking the limit as n tends to infinity, we can derive

$$\mu = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \log_e |f'(x_i)|\,, \tag{5.5}$$

where f' is the first derivative of function f, and this illustrates the general idea behind the Lyapunov exponents.

For continuous time systems,

$$\frac{dx_i}{dt} = g_i(x_1, \ldots, x_d, p_i), \qquad (i = 1, \ldots, d), \tag{5.6}$$

there are two aspects of the time evolution which are of particular interest. The first aspect relates to the evolution of volume elements in state space. For a continuous time system described by a system of differential equations such as (5.6) an element of volume V will evolve over time according to the divergence equation

$$\frac{1}{V}\frac{dV}{dt} = \sum_{i=1}^{d} \frac{\partial g}{\partial x_i} \equiv \text{div} g \tag{5.7}$$

see for example [Hilborn 1994].

We first note that, if J can be written in diagonal form, $\text{div} g = \text{Trace}\, J$, where J is the Jacobian matrix of the system. Thus if the average over time of $\text{Trace}\, J < 0$, then the volume elements will contract and the system will be *dissipative*, whereas if the average over time of $\text{Trace}\, J = 0$ the system is '*conservative*' in the sense that it is measure preserving in phase space. Now

$$\text{Trace}\, J = \sum_{i=1}^{d} \lambda_i \tag{5.8}$$

where the λ_i are the eigenvalues of J. Thus the dissipative or preservative properties of a system in the phase space are determined by the average over time of the sum of the eigenvalues of J.

We are primarily interested in dissipative systems which are chaotic, so that the second aspect of time evolution which concerns us is whether nearby trajectories have a tendency to diverge exponentially on average.

For continuous systems, Lyapunov exponents provide a coordinate-independent measure of the asymptotic local stability of properties of a trajectory. The concept is very geometrical. Imagine a small infinitesimal ball of radius $\epsilon(0)$ centred on a point $x(0)$ in state space. Under the action of the dynamics the centre of the ball may move, and the ball becomes distorted, see Figure 5.1. Since the ball is infinitesimal, this distortion is governed by the linear part of the flow. The ball thus remains

an ellipsoid. Suppose the principal axes of the ellipsoid at time t are of length $\epsilon_i(t)$. The spectrum of Lyapunov exponents for the trajectory $x(t)$ is defined as

$$\mu_i = \lim_{t\to\infty} \lim_{\epsilon(0)\to 0} \left\{ \frac{1}{t} \log \left[\frac{\epsilon_i(t)}{\epsilon(0)} \right] \right\}, \qquad (1 \leq i \leq d). \tag{5.9}$$

Note the Lyapunov exponents depend on the trajectory $x(t)$. Their values are the same for any state on the same trajectory, but may be different for states on different trajectories. The trajectories of a d-dimensional state space have d Lyapunov exponents. This is often called the *Lyapunov spectrum*. It is conventional to order them according to size. The qualitative features of the asymptotic local stability properties can be summarised by the sign of each Lyapunov exponent; a positive Lyapunov exponent indicating an unstable direction, and a negative exponent indicating a stable direction. The motion will be dissipative if

$$\sum_{i=1}^{d} \mu_i < 0 \tag{5.10}$$

and chaotic if at least one $\mu_i > 0$.

Trajectories' divergence properties can also be expressed in terms of the eigenvalues of J, since the eigenvalues will determine the form of the solution to the locally linear differential equations which determine the trajectory at any particular point of the phase space. In general terms these locally linear solutions for the x_i will be of the form

$$A_1 e^{\lambda_1 t} + A_2 e^{\lambda_2 t} + \ldots + A_d e^{\lambda_d t}. \tag{5.11}$$

If for a particular trajectory we write the time average

$$\lim_{T\to\infty} \int_{t=0}^{T} \ln \left| e^{\lambda_i(t)} \right| dt, \qquad (1 \leq i \leq d), \tag{5.12}$$

[Otani and Jones 1997b] conjecture that this provides an alternative route to the Lyapunov exponents.

For an high-dimensional iterative map function $X^{(n)} = F(X^{(n-1)})$, where $F = (F_1, \ldots, F_d)$, with Jacobian

$$J = \left(\frac{\partial F_i}{\partial x_j} \right), \qquad (1 \leq i, j \leq d). \tag{5.13}$$

Volume elements will locally contract or diverge according as $|\det J|$ is less than or greater than 1, respectively. Thus in this case the condition for a dissipative system depends on the average of $|\det J|$, rather than $\mathrm{Trace}\, J$ as in the continuous case.

We can still speak of an average rate of divergence: if the system is allowed to evolve from two slightly differing initial states $X = (x_1, \ldots, x_d)$ and $X + \epsilon$ after n iterations the divergence of the two points may be characterised as

$$\epsilon(n) = (\epsilon(0)e^{n\mu_1}, \ldots, \epsilon(0)e^{n\mu_d}) \tag{5.14}$$

where the Lyapunov exponents μ_i give the average rate of divergence/convergence over a large number of iterations. For small ϵ we can express this as

$$\mu_i = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \ln \left| \frac{\partial F_i}{\partial x_i} \right|_{\boldsymbol{X} = \boldsymbol{X}^{(k)}}, \qquad (1 \leq i \leq d). \tag{5.15}$$

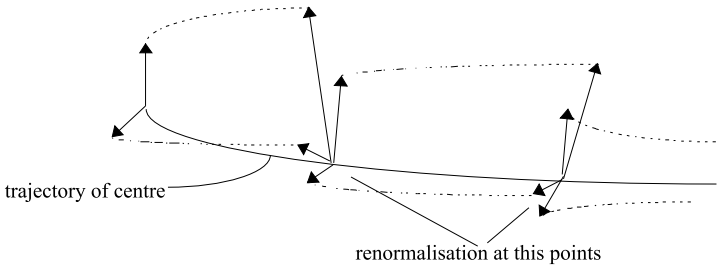


Figure 5.2: At t_0 an orthonormal set of vectors from the centre of the sphere evolves by stretching and contracting along the axes of the developing ellipsoid. At t_1 a new set of vectors generated such that one of the new vectors is parallel to the previous stretching direction.

which is analogous to (5.9) for a continuous system.

The Lyapunov exponents are essential for investigating chaos, convergence and divergence dynamics of any system, therefore a good numerical estimating technique is required. Two such algorithms are described here.

The first algorithm is based on the description from [Baker and Gollub 1990; Parker and Chua 1992] and is best used when the full mathematical description of the dynamics is available. The basic idea of the of the calculation of the Lyapunov exponents is same as the definition shown above. However, it is impractical to perform the actual calculation, because the initially close phase points would soon diverge from each other by distances approaching the size of the chaotic attractor, and the computation would then fail to capture the local contracting and diverging rates. Therefore, vectors connecting the surface of the ellipsoid to the centre must be reduced in size periodically or *renormalised*, to ensure that the size of the ellipsoid remains small and that the surface points correspond to trajectories near that of the centre point. The renormalisation is shown in Figure 5.2 and can be achieved by the linear algebra technique of Gram-Schmidt orthonormalisation. The Lyapunov exponents are taken to be the *averages* of those obtained over many segments of the central trajectory.

There are three main inputs - the numerical integration[3] time step, T, the maximum number of iterations of numerical integration, $k_{\max}$ and $x[]$, the current state of the n^{th}-order system. The pseudocode of this algorithm is shown in Algorithm 5.2.

These are some important notes for using this algorithm:

- The single square brackets [] indicate a vector and the double [][] indicates a matrix. Also, $[i][]$ means the i^{th} row of the matrix and $[][j]$ means the j^{th} column of the matrix.
- $u[][]$ is the orthonormalised perturbation matrix with initial value of I, the identity matrix.
- $\langle x, y\rangle$ denotes the inner product of the vectors x and y.
- $\phi_T(x[])$ is the solution of the differential equations, i.e. the current state of the system.
- $\Phi_T(x[])$ is the solution of the *variational equation* of a matrix-valued time-varying linear differential equation. It is the linearisation of the vector field along the trajectory ϕ_T, or in other word, it is the Jacobian at the current point on the trajectory. This can be solved numerically at the same time of solving ϕ_T with initial value $\Phi = I$ at t_0.

[3]Here we use fourth order Runge-Kutta method as a reasonable compromise between computer speed and accuracy of solution.

Procedure: Lyapunov exponents(T, $k_{\max}$, $x[]$)
{T size of time step for numerical integration.}
{$k_{\max}$ the maximum number of steps of numerical integration.}
{$x[]$ is the current state of the system.}
$u[][] = I$ {identity matrix}
for $i = 1$ to n **do**
  $\mu[i] = 0$
  $sum[i] = 0$
end for
$k = 0$
repeat
  $k = k + 1$
  if $k == k_{\max}$ **then**
    exit - no convergence
  end if
  {changes due to the local dynamics Φ_T at $x[]$}
  $\delta x[][] = \Phi_T(x[])u[][]$
  {next numerical integrated state with time step T}
  $x[] = \phi_T(x[])$
  for $i = 1$ to n **do**
    $v[][i] = \delta x[][i]$
    {renormalisation}
    for $j = 1$ to $(i-1)$ **do**
      $v[][i] = v[][i] - \langle v[][i], u[][i] \rangle u[][j]$
    end for
    $u[][i] = v[][i]/|v[][i]|$
    {accumulate the average divergent/convergent rates}
    $sum[i] = sum[i] + \ln |v[][i]|$
    $\mu[i] = sum[i]/kT$
  end for
until *convergence*
return$\mu[]$ the Lyapunov exponents

Algorithm 5.2: An algorithm for estimating Lyapunov exponents.

- A modified Gram-Schmidt orthonormalisation procedure is used here [Noble and Daniel 1979].

In practice, the choice of T in this algorithm plays an important role in the success of finding the Lyapunov exponents. Too small a value could result in excessive orthonormalisation and generally lead to inaccuracy of the Lyapunov exponents. Too large a value could lead to numerical overflow which happened quite easily in experiments.

In most cases for the experiments, neural systems are available but it would be difficult to obtain the precise mathematical description to obtain the solution of the variational equations, Φ_T or the Jacobian describing the local flow, which is essential for calculating the local divergence and convergence rates. Since the neural systems are available, close-by points are randomly generated near the point of trajectory concerned and iterated, so that a least squares fit can be performed to estimate the local Jacobian. Generally this worked very well but this brought in another problem of choosing the size of local region for estimating the local flow.

The second technique is very similar to the one just discussed, but the spectrum of Lyapunov exponents is estimated by calculation from the observed time series of a single scalar variable, x of the system [Sano and Sawada 1985]. For the single variable case, one can reconstruct the dynamics by the use of delay coordinates [Takens 1981], i.e.

$$\boldsymbol{x}_i = (x(i\tau), \ldots, x(i\tau + (d-1)t_D)) \tag{5.16}$$

where t_D is the delay time and d is the reconstructed dimension. However, as shown in a later experiment, if it is available the time series of system states may also be used for better accuracy without the 'hidden' problem from delay coordinates. The procedure is shown in Algorithm 5.3.

In this method, since a time series of state vectors, $\boldsymbol{x}_i$ ($i = 1, 2, \ldots, M$) measured at discrete time interval is available, we do not need to perform numerical integration as shown in the first method. The local flow, the Jacobian J, at each state $\boldsymbol{x}_j$ in the time series is estimated firstly by collecting points $\{\boldsymbol{x}_{ki}\}$ from $\{\boldsymbol{x}_i\}$ within a hypersphere centred at the point $\boldsymbol{x}_j$ with radius ϵ, i.e. forming the set

$$\{\boldsymbol{y}_i\} = \{\boldsymbol{x}_{ki} - \boldsymbol{x}_j \mid |\boldsymbol{x}_{ki} - \boldsymbol{x}_j| \leq \epsilon\}, \tag{5.17}$$

where $\boldsymbol{y}_i$ is the displacement vector between $\boldsymbol{x}_{ki}$ and $\boldsymbol{x}_j$. $|\cdot|$ is the usual Euclidean norm.

The displacement vectors $\boldsymbol{y}_i = \boldsymbol{x}_{ki} - \boldsymbol{x}_j$ is mapped to

$$\{\boldsymbol{z}_i\} = \{\boldsymbol{x}_{ki+1} - \boldsymbol{x}_{j+1} \mid |\boldsymbol{x}_{ki} - \boldsymbol{x}_j| \leq \epsilon\}. \tag{5.18}$$

If the radius ϵ is small enough for the displacement vectors $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$ to be regarded as good approximation of tangent vectors in the tangent space, then the evolution of $\boldsymbol{y}_i$ to $\boldsymbol{z}_i$ can be represented by some matrix M_j, as

$$\boldsymbol{z}_i = M_j \boldsymbol{y}_i. \tag{5.19}$$

In this case the M_j which minimises the average of the squared error norm between $\boldsymbol{z}_i$ and $\boldsymbol{y}_i$ with respect to all components of the matrix M_j, that is

$$\min_{M_j} \frac{1}{N} \sum_{i=1}^{N} |\boldsymbol{z}_i - M_j \boldsymbol{y}_i|^2, \tag{5.20}$$

is the optimal estimation of the linearised flow map from the data sets $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$ or the Jacobian J at $\boldsymbol{x}_j$. Finding the solution of this problem is basically a standard least squares fit problem and an

Lyapunov exponents(ts, T, ϵ, $maxN$)
{ ts list of time series of state vectors; T time step between each state vector.}
{ ϵ radius of the hypersphere for including local points to estimate local flow.}
{ $maxN$ maximum number of points to be included for estimating local flow.}
$n =$ dimension of the vector in ts; $u[][] = I$ {identity matrix}
for $i = 1$ to n **do**
  $\mu[i] = 0$; $sum[i] = 0$
end for
for $k = 1$ to (length(ts) - 1) **do**
  $x_k[] = k^{\text{th}}$ vector of ts; $x_{k+1}[] = (k+1)^{\text{th}}$ vector of ts
  $A = \{\}$; $B = \{\}$ { empty lists }
  for $j = 1$ to length(ts) **do**
    $x_j[] = j^{\text{th}}$ vector of ts
    if $\|x_k[] - x_j[]\| < \epsilon$ **then**
      { order of the vectors in the lists are important }
      append displacement vector ($x_k[] - x_j[]$) to A
      $x_{j+1}[] = (j+1)^{\text{th}}$ vector of ts
      append displacement vector ($x_{k+1}[] - x_{j+1}[]$) to B
    end if
  end for
  {if not enough points for estimating local flow, skip to the next state vector}
  if length(A) $> n$ **then**
    if length(A) $> maxN$ **then**
      $A =$ list of the first $maxN$ vectors from A
      $B =$ list of the first $maxN$ vectors from B
    end if
    form $A'[][]$ with each j^{th} column $A'[][j] = j^{\text{th}}$ vector from list A
    form $B'[][]$ with each j^{th} column $B'[][j] = j^{\text{th}}$ vector from list B
    $J[][] = B'[][]$ pseudoinverse($A'[][]$) {least squares fit on A and B}
    $\delta x[][] = J[][]u[][]$
    for $i = 1$ to n **do**
      $v[][i] = \delta x[][i]$
      { renormalisation }
      for $j = 1$ to $(i-1)$ **do**
        $v[][i] = v[][i] - \langle v[][i], u[][i] \rangle u[][j]$
      end for
      $u[][i] = v[][i] / \|v[][i]\|$
      { accumulate the average divergent/convergent rates }
      $sum[i] = sum[i] + \ln\langle v[][i] \rangle$
      $\mu[i] = sum[i]/kT$
    end for
  end if
end for
return $\mu[]$ the Lyapunov exponents

Algorithm 5.3: Lyapunov exponents for trajectories of a continuous systems estimate from a finite time series.

interesting way of solving this problem is shown in the next section. The remainder of this Lyapunov exponents estimation technique is same as in the previous technique.

This second method is possible only if the time series represents the dynamics of a chaotic attractor and thus has the *ergodic* property which ensures there are enough nearby points which can be collected for estimating local flow at each orbit point. The ergodic property relates the time average of a function to its average over phase space. This relationship, which is fundamental in statistical mechanics, was first conjectured by W. Gibbs prior to the invention of the Lebesgue integral. For Gibbs this was singularly unfortunate since without the Lebesgue integral it is impossible to express the idea precisely. Suppose that, except for a set of measure zero, the attractor $\mathcal{S}$ of a dynamical system is transformed into itself by an element $\boldsymbol{T}$ of a group of measure preserving transformations $\mathcal{T}$,[4] where without loss of generality we may suppose the measure $\mu_m(S) = 1$. Suppose f is a measurable function and sufficiently well behaved (e.g. $f \in L^1$), then Birkoff proved that for $\boldsymbol{T} \in \mathcal{T}$

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N} f(\boldsymbol{T}^n(x_0)) = \int f(x) d\mu_m(x), \tag{5.21}$$

except for a set of values of x_0 of zero measure. If $\mathcal{T}$ represents translations in time then this equation asserts that for almost all initial conditions the time average of f is equal to its measure-weighted phase average. An f with this property is called *ergodic* with respect to the transformation $\boldsymbol{T}$.

However, if sufficient samples of the time series are not available, or more generally such that the rank of the matrix A formed from these column vectors is less than the dimension of the local flow n, then a linear approximation to the local flow cannot be estimated; in which case the algorithm just ignores the current point and moves to the next point. Providing this skipping of points does not happen too often, this technique will still give very good estimation of the Lyapunov exponents. In practice, one would like to avoid computing the rank of A at each step. To this end we use the number of samples of the time series as an approximate guide. If this number is greater than n we perform the estimate and use the pseudoinverse (discussed in Section 3.4.3) rather than the inverse to cover the eventuality that the matrix may be possibly be singular. Whilst not ideal, this comprise seems to result in a faster algorithm without a significant loss of accuracy. Also, in Algorithm 5.3, there is an upper limit, $maxN$ of the number of local points used in estimating the Jacobian in order to reduce the computation time. Results of using these techniques will be shown later in later experiments.

5.4 Neural networks as dynamical systems

The evolution of the state of a neural network can be considered from a rigorous mathematical point of view as a dynamical system. Many dynamical behaviours, such as attracting or repelling fixed points and limit cycles, can be observed in non-linear artificial neural networks [Babcock and Westervelt 1986; Hirsch 1989; Marcus and Westervelt 1989]. Chaotic dynamics have also been observed in many artificial neural systems, either in continuous-time systems [Kürten and Clark 1986] or discrete-time systems [Wang 1991]. In this section, some simple chaotic neural models are briefly described. There are many chaotic biological and artificial neural models waiting to be discovered.

[4] We can think of $\mathcal{T}$ as the translation group on $\mathbb{R}$

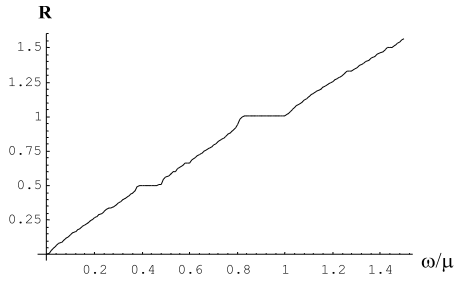


Figure 5.3: The rotation number R against ω/μ.

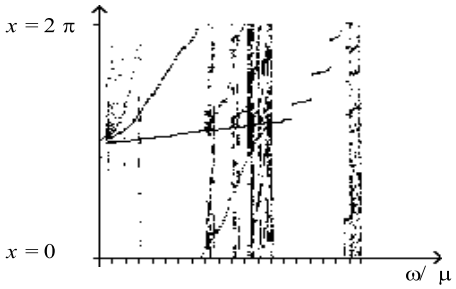


Figure 5.4: A bifurcation diagram of a VCON.

5.4.1 Chaos at the neuronal level

Science has long been modelling the biological neuron using mathematical descriptions. Here we examine the *voltage-controlled oscillator neuron* or VCON [Hoppensteadt 1989]. This model, in contrast to all-or-none neuron models, generates voltage spikes that phase-lock to oscillatory stimulation, similar to the phase-locking of action potentials to oscillatory voltage stimulation observed in Hodgkin-Huxley preparations of squid axons [Hodgkin and Huxley 1952].

The VCON model of a single neuron (cell body potential $\cos x$ with phase x) stimulated through a synapse on the cell body (presynaptic potential $\cos y$ with phase y) is

$$\frac{dx}{dt} = \omega + C\cos_+(x(t))\cos_+(y(t)), \tag{5.22}$$

where the constant C describes the polarity of the synapse ($+$ for excitatory, $-$ for inhibitory) and its strength $|C|$. V_+ denotes the super-threshold part of a voltage V so $V_+ = V$ if $V \geq 0$, but it is 0 otherwise. Thus, $\cos_+ x$ describes action potentials generated by the VCON. Finally, ω is the mean firing rate in the absence of interaction. If y has fixed frequency μ, then the model becomes

$$\frac{dx}{dt} = \omega + C\cos_+(x)\cos_+(\mu t). \tag{5.23}$$

One interesting aspect of this simple model is that it is chaotic. To illustrate the chaotic nature, we can look at the frequency encoding and processing which can be partly described in terms of the output/input phase ratio

$$R = \lim_{t\to\infty} \frac{x(t)}{y(t)}. \tag{5.24}$$

We can calculate $x(100\pi)$ and plot $x(100\pi)/(100\pi\mu)$ against ω/μ. The results appear in Figure 5.3 which is similar to the *devil's staircase* for the circle map[5]. Each plateau in this plot indicates an interval of phase locking. Irregular firing is observed for certain applied frequencies. We can describe these chaotic dynamics by using the bifurcation diagram[6] shown in Figure 5.4 with ω/μ as the varying parameter value. Parameter values for which iterates are widely scattered are ones for which there is a high periodic orbit or an ergodic solution. We can refer to these responses as being chaotic.

However, this chaotic aspect of a network of VCONs still has not been studied in detail or exploited in a practical way.

[5] See any standard text book on dynamical system and chaos, e.g. [Ott 1993], for further information.

[6] An initial point $x = \xi$ is selected and iterated 1000 times under $\boldsymbol{P} : x(0) = \xi \to x(2\pi/\mu) \pmod{2\pi}$, Poincaré's mapping. Then the interval $[0, 2\pi)$ is partitioned into 200 equal subintervals and a pixel is plotted if its support cell is hit during the iteration. (Ignore the first 10 iterates to suppress transients.)

5.4.2 Chaos at the network level

The dynamics of a collection of neurons can also shift from a orderly behaviour into chaos by a simple system parameter change. A very small network consisting of only two neurons can possess a chaotic attractor and a particular simple chaotic network has already been studied and proven to be chaotic by Wang [Wang 1991]. The details of this are discussed in Section 6.1.1.

Another network model that was examined is the Tsuda net [Tsuda 1992]. This network provides a model of dynamic link memory in terms of a self-organised chaotic transition in non-equilibrium neural networks. The network itself consists of a symmetrically coupled network which is defined in relation to a memory storage and an asymmetrically coupled network, which has no relation to a memory but causes the overall system be in a non-equilibrium state. The memories are stored on the transition states of the dynamics of the network. The chaotic transition blocks pin on *false* memories and thereby allow a successive retrieval of true memory. This combination of symmetric and asymmetric couplings give rise to a special kind of chaotic dynamics which allows neural networks to be temporarily unstable, keeping *stability due to convergent dynamics*. Tsuda suggests that the cortical chaos may serve for dynamically linking true memory as well as a memory search. The original paper gives a thorough explanation of this special kind of network and its dynamics.

A simple feedforward neural network can, in fact, learn the behaviour of a chaotic attractor/chaotic dynamics of a known chaotic system so that the network can behave chaotically [Welstead 1991]. That this can be done is due to the fact that multilayer feedforward neural networks are *universal approximators* [Hornik *et al.* 1989].

5.5 Controlling Chaos - Strategies

Chaos was historically considered unreliable, uncontrollable and unusable. For these reasons, engineers typically avoided it. However, in recent years, scientists have demonstrated that chaos is manageable, exploitable and many even consider it to be valuable [Ditto and Pencora 1993]. This progress in using chaotic systems is principally due to a control technique developed in 1990 by Ott, Grebogi and Yorke (OGY) [Ott *et al.* 1990]. Since the original paper a number of variations of the OGY control method and other chaotic control techniques have been published. The details of applying the OGY method are introduced to illustrate the basic idea of chaotic control. This chapter tries to give a 'snapshot' of some of the ideas of *bringing order into chaos* and therefore, a selection of methods to control chaotic dynamics are introduced. There are still many techniques waiting to be discovered, implemented and investigated.

The key idea behind most control methods takes advantage of the behaviour of the underlying chaotic (or strange) attractors. A chaotic attractor can be viewed as a dense set of unstable periodic orbits [Grebogi *et al.* 1988] and the principle on which the OGY control method is based is to exploit the already existing (unstable) periodic orbits. The word periodic here is used very loosely. We say $\boldsymbol{x}$ is a point on a (k, ϵ)-periodic orbit of a discrete system F if $|F^{(n+k)}(\boldsymbol{x}) - \boldsymbol{x}| \leq \epsilon$ for all $n \geq N$ and some $\epsilon > 0$, where $|\cdot|$ denotes the Euclidean norm. The periodic orbits of interest in this context do not satisfy this definition because they are unstable, the periodic behaviour is displayed intermittently, and we shall return to this discussion if and when a formal definition is required for our particular purposes.

In general, the control strategies can be divided into two main groups - controlling via *parameter perturbation*, e.g. the OGY method, and controlling via *system variable perturbation* such as continu-

ous delayed feedback control [Pyragas 1992].

The aim of any parameter perturbation control method such as the OGY method is to obtain desired performance, i.e. a desired attracting time-periodic motion by making only *small* time-dependent perturbations in an *accessible* system parameter. The typical approach is briefly described as follows:

- Determine some of the low-period unstable periodic orbits embedded in the chaotic attractor.
- Examine these orbits and choose one which yields desired system performance.
- Construct a rule for suitably small parameter perturbations which stabilises this already existing unstable periodic orbit.

The variable perturbation control technique has an almost identical approach except a control signal is added to the state variables of the system so that the system dynamics can be perturbed onto some periodic orbits embedded in the chaotic attractor.

5.6 Controlling via system parameter perturbation

In this section, the OGY method is presented with a simple experiment to reinforce the idea of controlling chaos via system parameter variation. Other direct variations of the OGY method are also introduced. Some parameter variation control methods are also demonstrated.

5.6.1 The original OGY control law

Assume the dynamical equations describing the system are not known, but that an experimental time series of some scalar dependent variable $z(t)$ is available. We define an *embedding* of the system using time delay coordinates [Packard *et al.* 1980] by

$$\boldsymbol{\xi}(y) = (z(t), z(t+\tau), \dots, z(t+(d-1)\tau)) \tag{5.25}$$

and we can then get a surface of section or a Poincaré section. As a result a continuous-time-periodic orbit appears as a discrete-time orbit cycling through a finite set of points.[7]

For $i \geq 1$, let

$$\delta p = p - p_0 \qquad \text{and} \qquad \delta\boldsymbol{\xi}_{i+1}(p_i) = \boldsymbol{\xi}_{i+1}(p_i) - \boldsymbol{\xi}_F(p_0) \tag{5.26}$$

where $\boldsymbol{\xi}_F$ is an unstable fixed point of the attractor. Suitable fixed points, which become candidate targets for control, are extracted from experimental data using relatively simple numerical search techniques (See [Otani and Jones 1997b]).

Suppose the iteration on this section is described by

$$\boldsymbol{\xi}_{i+1} = F(\boldsymbol{\xi}_i, p) \tag{5.27}$$

where $p \in (p_0 - \delta p_{\max}, p_0 + \delta p_{\max})$ is a control parameter, with maximum perturbation $\delta p_{\max}$, and p_0 is the control parameter nominal value for which the dynamics generates a chaotic attractor. In the vicinity of the fixed point $\boldsymbol{\xi}_F$, the behaviour of F can be described by the first order approximation

$$\delta\boldsymbol{\xi}_{i+1}(p_i) \approx J\delta\boldsymbol{\xi}_i(p_{i-1}) + \boldsymbol{u}\delta p_i \tag{5.28}$$

[7] **Comment**: Choosing the various parameters which enable a good reconstruction of the high-dimensional dynamics is a non-trivial issue. Efficient techniques for accomplishing this are discussed in [Otani and Jones 1997b].

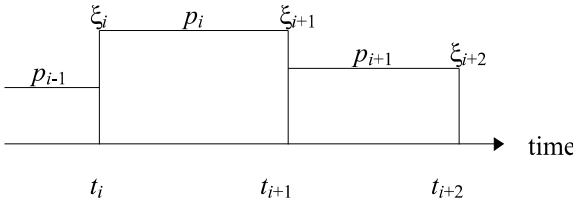


Figure 5.5: Intervals for which the variables are defined.

where J is the $d \times d$ Jacobian matrix:

$$J = \left[D_{\boldsymbol{\xi}} F(\boldsymbol{\xi}_p)\right]_{\boldsymbol{\xi}=\boldsymbol{\xi}_F, p=p_0} \tag{5.29}$$

and $\boldsymbol{u}$ is an d-dimensional column vector

$$\boldsymbol{u} = \frac{\partial F}{\partial p_i}(\boldsymbol{\xi}_F, p_0). \tag{5.30}$$

This $\boldsymbol{u}$ is also called the *sensitivity vector* of the OGY method. Since $\boldsymbol{\xi}_F$ is embedded in a chaotic attractor, the linearisation J is composed of stable eigenvectors $\boldsymbol{e}_s$ and unstable eigenvectors $\boldsymbol{e}_u$ with their corresponding stable and unstable eigenvalues λ_s and λ_u respectively, so that $|\lambda_s| < 1$ and $|\lambda_u| > 1$. One estimates J, $\boldsymbol{e}_s$, $\boldsymbol{e}_u$, λ_s, λ_u using linear regression based on observational data.

With this information about the local map F, one can derive the OGY control law. However, we first observe the following lemma which has been proved in [Otani and Jones 1997b].

Lemma 5.6.1. *Suppose the $d \times d$ matrix J has d linearly independent eigenvectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d$ with real eigenvalues $\lambda_1, \ldots, \lambda_d$. Thus we assume the eigenvectors form a basis in $\mathbb{R}^d$. Construct the dual basis $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_d$ defined by*

$$\boldsymbol{e}_i \cdot \boldsymbol{f}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} . \tag{5.31}$$

Then for any $\boldsymbol{x} \in \mathbb{R}^d$

$$\boldsymbol{f}_u \cdot J\boldsymbol{x} = \lambda_u \boldsymbol{f}_u \cdot \boldsymbol{x}. \tag{5.32}$$

The control law seeks to ensure that $\boldsymbol{\xi}_{i+1}$ falls on the local stable manifold of the fixed point, so that on the next iteration $\boldsymbol{\xi}_{i+1}$ will move closer to $\boldsymbol{\xi}_F(p_0)$. This can be formulated as

$$\boldsymbol{f}_u \cdot \delta\boldsymbol{\xi}_{i+1} = 0 \tag{5.33}$$

which together with (5.32),yields the *control formula* for the new value of the control parameter $p_i = p_0 + \delta p_i$,

$$\delta p_i = -\lambda_u \frac{\boldsymbol{f}_u \cdot \delta\boldsymbol{\xi}_i(p_{i-1})}{\boldsymbol{f}_u \cdot \boldsymbol{u}}. \tag{5.34}$$

The version of this control law seems different from the one in the original paper [Ott *et al.* 1990] stated as follows:

$$\delta p_i = \frac{\lambda_u}{\lambda_u - 1} \frac{\boldsymbol{f}_u \cdot \delta\boldsymbol{\xi}_i(p)}{\boldsymbol{f}_u \cdot \boldsymbol{g}}. \tag{5.35}$$

This is because the sensitivity vector $\boldsymbol{g}$ in the original OGY paper is defined in terms of the shift of the fixed point $\boldsymbol{\xi}_F$ with respect to the change in p, whereas in Dressler and Nitsche version [Dressler

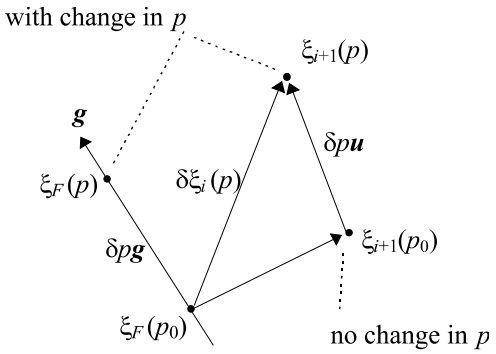


Figure 5.6: Comparison of the sensitivity vectors $\boldsymbol{g}$ and $\boldsymbol{u}$.

and Nitsche 1992], $\boldsymbol{u}$ is defined as the shift in $\boldsymbol{\xi}_{i+1}$ with respect to the change in p, see Figure 5.6. Whereas $\boldsymbol{g}$ is defined as

$$\boldsymbol{g} = \left[\frac{\partial \boldsymbol{\xi}_F}{\partial p}\right]_{p_0} = \lim_{p \to p_0} \frac{\boldsymbol{\xi}_F(p) - \boldsymbol{\xi}_F(p_0)}{p - p_0}. \tag{5.36}$$

However, this difference is explained by the relationship

$$\boldsymbol{u} = (I - J)\boldsymbol{g} \tag{5.37}$$

demonstrated in [Otani and Jones 1997b], where I is the $d \times d$ identity matrix (see also Figure 5.6). To illustrate this relationship, first consider that in the original OGY method, the Jacobian matrix J is defined to be the changes in $\boldsymbol{\xi}_i$ relative to the *shifted* fixed point, i.e.

$$\boldsymbol{\xi}_{i+1}(p) - \boldsymbol{\xi}_F(p) \approx J\left(\boldsymbol{\xi}_i(p) - \boldsymbol{\xi}_F(p)\right) \tag{5.38}$$

where from (5.36)

$$\boldsymbol{\xi}_F \approx \boldsymbol{\xi}_F(p_0) + (p - p_0)\boldsymbol{g}. \tag{5.39}$$

Then from these last two equations we have in the OGY notation

$$\boldsymbol{\xi}_{i+1}(p) - \boldsymbol{\xi}_F(p_0) \approx J\left(\boldsymbol{\xi}_i(p) - \boldsymbol{\xi}_F(p_0)\right) + (p - p_0)(I - J)\boldsymbol{g}. \tag{5.40}$$

By direct comparison with (5.40) and the corresponding equation in the Dressler and Nitsche notation,

$$\boldsymbol{\xi}_{i+1}(p) - \boldsymbol{\xi}_F(p_0) \approx J\left(\boldsymbol{\xi}_i(p) - \boldsymbol{\xi}_F(p_0)\right) + (p - p_0)\boldsymbol{u} \tag{5.41}$$

(which are both first order identities in p) yields the relationship in (5.37).

In fact using (5.34) makes more sense: it is much easier to measure $\boldsymbol{u}$ from observations than to measure $\boldsymbol{g}$.

5.6.2 OGY with the use of delay coordinates

It has been shown by Dressler and Nitsche [Dressler and Nitsche 1992] that with the use of delay coordinates from experimental data it can be beneficial to modify the original OGY method. They argue that in the case of activated control (i.e. switching the parameter from p_{i-1} to p_i at time t_i) the experimental surface of section map F depends not only the new actual value p_i but also on the preceding value p_{i-1}, i.e.

$$\boldsymbol{\xi}_{i+1} = F(\boldsymbol{\xi}_i, p_{i-1}, p_i). \tag{5.42}$$

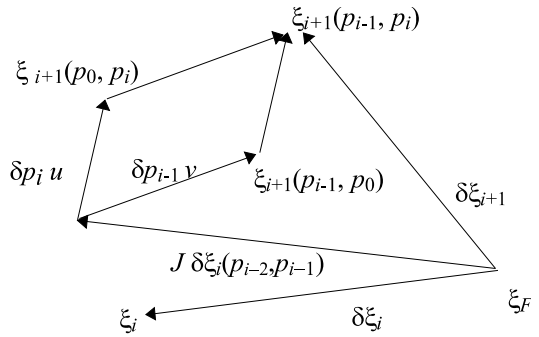


Figure 5.7: The combined effect of J in combination with δp_{i-1} and δp_i.

Using the previous argument for the derivation of the original OGY control law, we can then replace (5.26) by

$$\delta p = p - p_0 \qquad \text{and} \qquad \delta\boldsymbol{\xi}_{i+1}(p_{i-1}, p_i) = \boldsymbol{\xi}_{i+1}(p_{i-1}, p_i) - \boldsymbol{\xi}_F(p_0, p_0). \tag{5.43}$$

The linearisation which one has to consider now is given by

$$\delta\boldsymbol{\xi}_{i+1}(p_{i-1}, p_i) \approx J\delta\boldsymbol{\xi}_i(p_{i-2}, p_{i-1}) + \boldsymbol{v}\delta p_{i-1} + \boldsymbol{u}\delta p_i \tag{5.44}$$

where

$$\boldsymbol{v} = \left[\frac{\partial\boldsymbol{\xi}_{i+1}}{\partial p_{i-1}}\right]_{(p_0,p_0)} \qquad \text{and} \qquad \boldsymbol{u} = \left[\frac{\partial\boldsymbol{\xi}_{i+1}}{\partial p_i}\right]_{(p_0,p_0)}. \tag{5.45}$$

The combined effect of J, $\boldsymbol{v}$ and $\boldsymbol{u}$ is shown in Figure 5.7.

With the consideration of the requirement for placing $\boldsymbol{\xi}_{i+2}$ onto a stable manifold, i.e.

$$\boldsymbol{f}_u \cdot \delta\boldsymbol{\xi}_{i+2} = 0 \tag{5.46}$$

and the constraint to prevent δp from becoming large, i.e.

$$\delta p_{i+1} = 0 \tag{5.47}$$

and using the linearisation (5.44) we have the new first order control law

$$\delta p_i = \frac{-\lambda_u^2}{\lambda_u \boldsymbol{f}_u \cdot \boldsymbol{u} + \boldsymbol{f}_u \cdot \boldsymbol{v}} \boldsymbol{f}_u \cdot \delta\boldsymbol{\xi}_i(p_{i-2}, p_{i-1}) - \frac{\lambda_u \boldsymbol{f}_u \cdot \boldsymbol{v}}{\lambda_u \boldsymbol{f}_u \cdot \boldsymbol{u} + \boldsymbol{f}_u \cdot \boldsymbol{v}} \delta p_{i-1}. \tag{5.48}$$

The proof of this can be found in [Otani and Jones 1997b].

5.6.3 Applying OGY and OGY-derived variation

The OGY method was first implemented for an experiment by [Ditto *et al.* 1990]. The set-up requires a magnetostrictive metallic ribbon, whose stiffness can be changed by applying a magnetic field. The bottom end of the ribbon is clamped to a base; the top flops over to the left or right. When the ribbon is exposed to a field whose strength is varied periodically at a rate around one cycle per second, the ribbon buckles chaotically. A second magnetic field of small field strength served as the control parameter.

Following the original report, a sudden surge of analyses and experimental results were published using the OGY method. The OGY method has been applied to control chaos in an electronic circuit [Hunt 1991], a chaotic multimode laser [Roy *et al.* 1992], and even biological systems - cardiac arrhythmias in rabbit ventricle [Garfinkel *et al.* 1992] and rat brain [Moss 1994], etc. Variations of the OGY method have been used for synchronisation of chaos [Carroll and Pecora 1993;

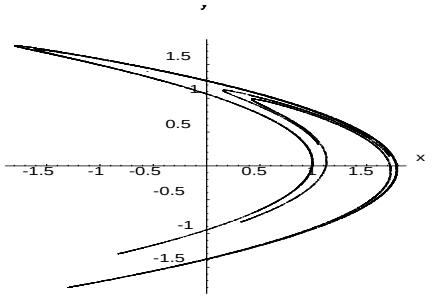


Figure 5.8: A chaotic attractor of the Hnon map with $a = 1.4$ and $b = 0.3$.

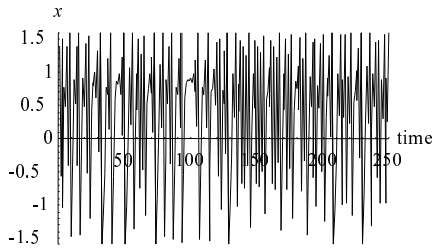


Figure 5.9: Chaotic signal x of the Hnon map with $a = 1.4$ and $b = 0.3$.

Roy and Thornbur 1994; Lai and Grebogi 1994] which allow the exploitation of chaos in communication.

There are also many improvements to the original OGY method, including control of higher periodic orbits [Hunt 1991; Auerbach *et al.* 1992], control of Hamiltonian chaotic systems [Lai *et al.* 1993], use of the past values of the control parameter [Dressler and Nitsche 1992], creation of non-existing periodic orbits [Hunt 1991] and tracking of unstable orbits [Schwartz and Triandaf 1994].

One noticeable problem with the OGY method is that a large amount of time may be wasted as the control system waits for the dynamical system to approach the desired orbit in the chaotic attractor [Ditto and Pencora 1993] in order to switch on the control. Shinbrot [Shinbrot *et al.* 1992] provided a technique that rapidly moves the chaotic states to the desired orbit of an attractor from an arbitrary initial state.

5.6.4 A simple experiment using OGY control

The OGY method has been being studied and investigated using simulation techniques in the software Mathematica^{TM}. It has been applied to a simple chaotic system, the Hénon map [Hénon 1976]. The aim of our early experiments was to implement the OGY method and study how to stabilise the system onto a 'fixed' point.

The two-dimensional Hénon map [Hénon 1976] was the main test bed for applying the OGY method. The reason for using the Hénon map is that this map is discrete and simple and the theoretical details, such as the Jacobian, eigenvalues and eigenvectors, etc., can be easily calculated for comparison with the experimental approximations. This dynamical system is defined as follows:

$$\begin{cases} x_{n+1} = a + by_n - x_n^2 \\ y_{n+1} = x_n \end{cases} \tag{5.49}$$

where a and b are non-zero constants. For different values of a and b, this map can produce all types of dynamic regular and irregular behaviours including different types of limit cycles and chaotic attractors. With the values $a = 1.4$, $b = 0.3$, this map produces a chaotic attractor as shown in Figure 5.8. By looking at the chaotic signal x of the Hénon map, shown in Figure 5.9, one can see that this system is very chaotic.

Due to the fact that system is known and it is discrete, the phase portrait of this system may be treated as the 'return map' and the OGY method can be applied directly. The parameter a, e.g. $p_0 = a = 1.4$, was chosen as the control parameter. A fixed point $\boldsymbol{\xi}_F = (x_F, y_F)$ at (0.883397, 0.876596) was located by looking at 20000 successive iterations of the system with radius distance of 0.01 and this point was then chosen as the control point.

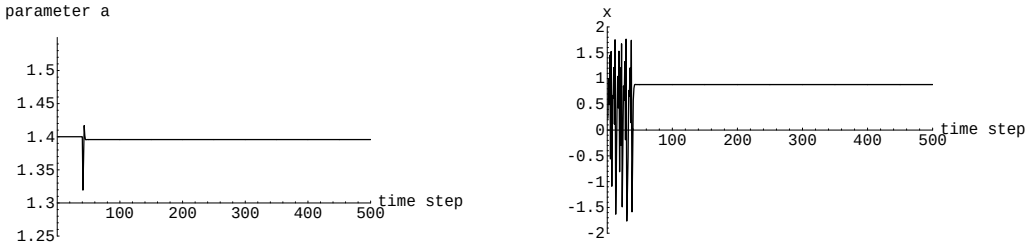


Figure 5.10: Controlled Hénon map with signal x stabilised onto the 'fixed' point under 50 time steps and controlled parameter a with values slightly less than 1.4, the initial parameter value.

The local dynamics of the control point were approximated by the estimated Jacobian matrix. This was done by collecting 200 points within radius distance of 0.045 from the fixed point and their next iteration and then performing standard least squares fit on these data to obtain the matrix:

$$J \approx \begin{bmatrix} -1.772920 & 0.302421 \\ 1.007215 & 0.000780 \end{bmatrix} \tag{5.50}$$

which was very close approximation to the theoretical Jacobian matrix

$$J = \begin{bmatrix} -2x_F & b \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -1.766 & 0.3 \\ 1 & 0 \end{bmatrix} \tag{5.51}$$

obtained from (5.49). The theoretical unstable and stable eigenvectors are $\boldsymbol{e}_u = (-0.887191, 0.461402)$ and $\boldsymbol{e}_s = (-0.154156, -0.988046)$ respectively with corresponding theoretical unstable and stable eigenvalues $\lambda_u = -1.92282$, $\lambda_s = 0.156021$ respectively. From this approximate Jacobian matrix the unstable and stable eigenvectors were found to be $\boldsymbol{e}_u = (-0.886675, 0.462394)$ and $\boldsymbol{e}_s = (-0.154695, -0.987962)$ respectively. The approximate unstable and stable eigenvalues were $\lambda_u = -1.93063$, $\lambda_s = 0.15849$ respectively. Therefore, the approximate values were very close to the theoretical ones.

The sensitivity vector was estimated by starting the system sufficiently close to the fixed point and then the control parameter, p, was changed from p_0 to some random value within the allowed range $1.25 \leq p \leq 1.55$. The vector was then estimated to be the difference between the starting point and the next data point. This is done several times and then an average was taken to obtain the sensitivity vector $\boldsymbol{u} = (0.926197, 0.152898)$

The control result is shown in Figure 5.10. Other control parameters and fixed points were tested with the same procedure just described, and the OGY method seems to be very successful in the simulation. However, during this experiment several critical issues emerged:

- The sensitivity vector is the key value in the control, and it can be very hard to obtain a good approximation.
- A poor approximation of the Jacobian matrix or the local linear map may not reflect the true nature of the dynamics near the point to be stabilised. In some cases poor approximation will incorrectly give us two stable or two unstable directions so that the OGY control method cannot be applied.
- Sometimes the system takes quite a long time to fall within the control region around the selected fixed point to be able to apply the control.

More examples, experimental results and discussion on using the OGY method to control chaotic neural systems are demonstrated in Chapter 6.

5.6.5 A brute force control law – Otani-Jones control

This Otani-Jones control, or OJ control for short, is based on an idea proposed in [Otani and Jones 1997b] and it has been successfully applied in many examples [Oliveira and Jones 1997; Oliveira *et al.* 1997; Otani and Jones 1997a]. This technique is an attempt to overcome some of the possible problems in the application of the OGY control method. This control method is based on the assumption that an effective short term (fast) predicting function $\boldsymbol{\xi}_{i+1} = P(\boldsymbol{\xi}_i)$, where $\boldsymbol{\xi}_i$ is a state of a d-dimensional iterative map $F : \boldsymbol{\xi}_i \to \boldsymbol{\xi}_{i+1}$ or a state on a Poincaré section of a continuous system, is available for the system and is accurate over the large part of the state space. This does not cause any difficulty if we were seeking to control an iterated feedforward neural network, e.g. Section 6.1.2 and Tsui's [Tsui and Jones 1997], where outputs are fed back to inputs, and which exhibits dynamical chaos. This is because the neural network can be iterated once without applying control to give an exact prediction of the next system state very rapidly. Therefore, the network is effectively its own Jacobian at every point in the state space. As demonstrated in [Dracopoulos and Jones 1993; Welstead 1991] for a dynamical system, a feedforward neural network trained on a single trajectory of the system can form an accurate short term predictor capable of generalising to other trajectories of the system.

The immediate benefit of the OJ control is that it does not require the computation of either $\boldsymbol{f}_u$ or λ_u because a short term predictor function P is available, although it is still necessary to perform sensitivity analysis for the variations of the control parameters. The method first assumes there is a short term predictor function $\boldsymbol{\xi}_{i+1} = P(\boldsymbol{\xi}_i)$ is available. Suppose that control parameter(s) $p = (p_1, \ldots, p_n)$ are available, with nominal value $p_0 = (p_{10}, \ldots, p_{n0})$ and that it is required to control the system about a fixed point $\boldsymbol{\xi}_F$. The situation can be described as follows,

$$\delta\boldsymbol{\xi}_{n+1} = \boldsymbol{\xi}_{n+1}(\boldsymbol{p}) - \boldsymbol{\xi}_F(\boldsymbol{p}_0) = P\left(\boldsymbol{\xi}_n(\boldsymbol{p}_0)\right) - \boldsymbol{\xi}_F(\boldsymbol{p}_0) + \delta p_1 \boldsymbol{s}_1 + \cdots + \delta p_l \boldsymbol{s}_l \tag{5.52}$$

where $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_l$ are the sensitivity vectors with respect to each control parameter, i.e.

$$\boldsymbol{s}_i = \left(\frac{\partial F}{\partial p_i}\right)_{\boldsymbol{\xi}=\boldsymbol{\xi}_F, p=p_i} \quad (1 \le i \le l). \tag{5.53}$$

As with the OGY control, we first estimate $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_l$ by collecting statistics from observations of the system state near $\boldsymbol{\xi}_F$ under small parameter variations. Since P is known, if sufficient observations are available, $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_l$ can be estimated via (5.52) using a least squares fit method, or equivalently a fast pseudoinverse algorithm. We assume that the choice of control parameters is such that $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_l$ are linearly independent, since there would seem to be no advantage in having a linearly dependent set of sensitivity vectors.

The essence of the OJ control law for any point $\boldsymbol{\xi}_n$ near $\boldsymbol{\xi}_F$ is to choose $\boldsymbol{p} = (p_1, \ldots, p_l)$ so as to minimise the squared Euclidean distance

$$\left|\boldsymbol{\xi}_{i+1}(\boldsymbol{p}) - \boldsymbol{\xi}_F(\boldsymbol{p})\right|^2 \tag{5.54}$$

with the known $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_l$, i.e. we choose $\boldsymbol{p}$ so as to minimise

$$\left|P\left(\boldsymbol{\xi}_i(\boldsymbol{p}_0)\right) - \boldsymbol{\xi}_F(\boldsymbol{p}_0) + \delta p_1 \boldsymbol{s}_1 + \cdots + \delta p_l \boldsymbol{s}_l\right|^2. \tag{5.55}$$

Let S be the matrix with column vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_l$, then the solution of this minimisation problem is given by

$$\boldsymbol{p} - \boldsymbol{p}_0 = -S^{-1} \left[P\left(\boldsymbol{\xi}_n(\boldsymbol{p}_0)\right) - \boldsymbol{\xi}_F(\boldsymbol{p}_0) \right] \tag{5.56}$$

where S^{-1} is the inverse of the matrix S if $l = d$ and the pseudoinverse of S otherwise. If $\boldsymbol{p}$ is outside its maximum allowed range of perturbation, then $\boldsymbol{p} = \boldsymbol{p}_0$, i.e. without any perturbation applied to the system, alternatively we could set $\boldsymbol{p} = \boldsymbol{p}_{\max}$ or $\boldsymbol{p}_{\min}$ appropriately.

In fact, a similar technique has already been implemented by Reyl [Reyl *et al.* 1993] who calls this the minimal expected deviation method. The OJ control method is basically a practical extension for cases with more than one available control parameters. In contrast to the OGY method, the OJ control method is brutally direct and seeks only to minimise the distance of the next iteration from the target unstable fixed point. Therefore, we might expect that the control perturbation needs to be applied at every step. The OJ control method has been successfully applied to synchronisation on chaotic systems - the Hénon map [Oliveira and Jones 1997], and has been demonstrated to be relatively robust in noisy systems. Experimental use of this technique is demonstrated in Chapters 6 and 7.

5.7 Controlling via system variable perturbation

The most typical technique for variable perturbation is to have some kind of feedback connection to the 'inputs' as the *delayed feedback control* [Pyragas 1992]. However, another technique - the *GM control* - uses a fixed amount of perturbation [Matías and Güémez 1994]. In most cases, systematic analysis such as local dynamics estimation, sensitivity analysis, etc. associated with parameter perturbation techniques is not required. Here the two types of system variable perturbation techniques just mentioned are described and some initial experiments will be reported.

5.7.1 Continuous delayed feedback control

Pyragas' continuous-time control technique [Pyragas 1992] deals with a chaotic system which can be represented by a set of ordinary differential equations

$$\frac{dy}{dt} = P(y, \boldsymbol{x}) + F(t), \quad \frac{d\boldsymbol{x}}{dt} = \boldsymbol{Q}(y, \boldsymbol{x}). \tag{5.57}$$

Here y is the observed variable and the vector $\boldsymbol{x}$ describes the remaining state variables of the dynamic system which are not available. The control signal $F(t)$ disturbs only the first equation, corresponding to the observed variable. We suppose that without a control signal the system being considered has a chaotic attractor.

The idea behind this method is to construct this perturbation $F(t)$ in such a way that it vanishes, or at least becomes very small, when the system moves along the desired unstable periodic orbit. One approach suggested by Pyragas [Pyragas 1992] is to use

$$F(t) = -k \left[y(t) - y(t - \tau) \right] = -kD(t), \tag{5.58}$$

where $k > 0$ (see Figure 5.11). Here τ is a delay time and $y(t - \tau)$ is the delayed value of the observed variable. Therefore the magnitude of the control signal is proportional to the difference $D(t) = y(t) - y(t - \tau)$. If this time τ coincides with the period of the unstable periodic orbit (i.e. $\tau = T$) then the control perturbation becomes small for the solution of the system (5.57), i.e.

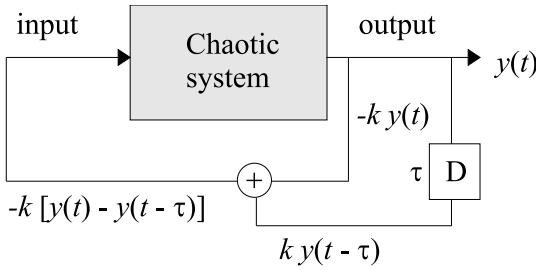


Figure 5.11: Delayed feedback control.

$y(t) = y(t - T)$. To ensure small values of the control perturbation at all times and to avoid multi-stability of the same control as a consequence of a large control signal, the control signal can be restricted in the following manner,

$$F(t) = \begin{cases} -F_0, & -kD(t) \leq -F_0 \\ -kD(t), & -F_0 < -kD(t) < F_0 \\ F_0, & -kD(t) \geq F_0 \end{cases} \tag{5.59}$$

where F_0 is a saturating value of the control.

Stabilisation of the system can be achieved by choosing an appropriate weight k so that a negative feedback is achieved. Though Qu [Qu *et al.* 1993] argued that in some cases, a positive feedback is needed. Therefore there are two variables, k and τ that can be adjusted in the experiment. The delay τ is expected to be the period of the stabilised orbit from the controlled chaotic system if the system eventually stabilises. Some experimental results can be found in [Pyragas 1993; Pyragas and Tamaševičius 1993; Celka 1994; Cooper and Schöll 1995].

The following experiment demonstrates this control technique. The Rössler attractor [Rössler 1976] was chosen for this experiment, defined as

$$\begin{aligned} \dot{x} &= -z - y \\ \dot{y} &= x + ay \\ \dot{z} &= b + z(x - c) \end{aligned} \tag{5.60}$$

where $a = 0.2$, $b = 0.2$ and $c = 5.7$ in the experiment. Without control being applied, this system is chaotic (see Figure 5.12).

A delayed feedback was applied to y with $k = 0.2$ and $\tau = 17.5$ as in

$$\begin{aligned} \dot{x} &= -z - y \\ \dot{y} &= x + ay - 0.2(y - y(t - 17.5)) \\ \dot{z} &= b + z(x - c). \end{aligned} \tag{5.61}$$

With the same starting conditions as in the unperturbed system, the system eventually stabilised into an orbit as shown in Figure 5.13. Here no restriction on the size of the perturbation was used.

The immediate conclusion is that this method does work, but is not always successful. If it is successful, the chaotic system can be stabilised very quickly. The drawback of this method is that although the choice of k and τ is important there are no specific guidelines for choosing these parameters. Moreover, unlike the OGY method which does have a systematic derivation, there is no adequate theoretical explanation of the mechanism of this control method. Qu [Qu *et al.* 1993] tried

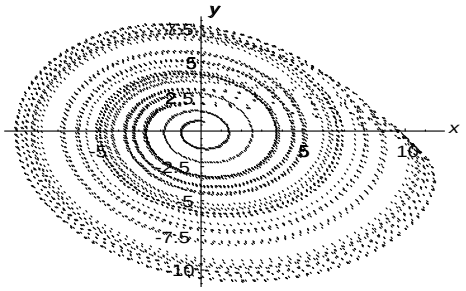


Figure 5.12: Chaotic Rössler attractor with $a = b = 0.2$ and $c = 5.7$.

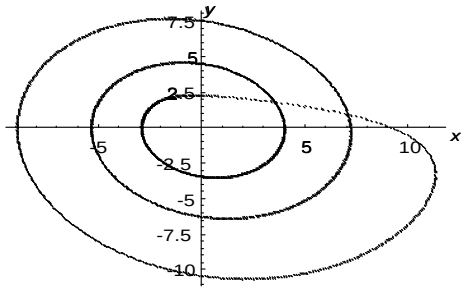


Figure 5.13: Stabilised orbit from the chaotic Rössler attractor.

to demonstrate the technique numerically and analytically by applying such feedback to a two dimensional artificial dynamical system model. They showed that the stability of the system is sensitive to the choice of k. The success of control depends on the choice of k, which in turn depends on the other system parameters. In fact, as shown in [Pyragas 1992], varying k changes the size of the largest Lyapunov exponent of the controlled system. By reducing this Lyapunov exponent below zero, the system will then be stabilised.

In fact, by careful examination of the controlled examples we quickly discover that the progression towards the fixed point, or unstable periodic orbit, is by no means monotone. At particular points of the phase space the local eigenvalues of the Jacobian of the controlled system may have modulus much larger than one, and so the system is often certainly *not stable* in the classical sense. However, provided k and τ are chosen suitably, indisputably the technique of delayed feedback does indeed work in the many examples we have examined. Why is this so?

In fact it is not necessary for the effect of delayed feedback to be contractive towards the fixed point, or unstable periodic orbit, at *every step*. It is only necessary that the effect be contractive *on average*. In [Oliveira and Jones 1998] this idea of *probabilistic local stability* is studied for an example of *synchronisation* of both the iterative Hénon map and the chaotic Ikeda iterative neural network introduced in Section 6.1.2. In the next section we give a similar empirical analysis for delayed feedback control applied to the control of *continuous* Rössler system.

Analysis of stabilised Rössler attractor

Using suitable parameters of k and τ for the delayed feedback on the right choice of system variable, the Rössler attractor can be stabilised onto a periodic orbit. Consider the setup for controlling the Rössler system

$$\begin{aligned} \dot{x} &= -z - y \\ \dot{y} &= x + 0.2y + k(y(t-\tau) - y(t)) \\ \dot{z} &= 0.2 + z(x - 5.7) \end{aligned} \tag{5.62}$$

with $k = 0.2$ and $\tau = 5.9$ and without restriction on the size of perturbation. The system stabilises onto a periodic orbit with period length about 5.9, allowing for intrinsic error caused by numerical integration, which is (as expected) equal to the value τ.

We generate random initial starting points near the Rössler attractor , and numerically integrate the controlled system using these points as the initial conditions. Next we examine the distance between each trajectory at time t and the closest point on the periodic orbit, i.e. the minimum distance between

a trajectory at time t and the periodic orbit, denoted as ν_t. Thereafter, we can define the ratio

$$\rho_t = \frac{\nu_t}{\nu_0} \tag{5.63}$$

where ν_0 is the minimum distance from the periodic orbit to the initial starting state of the trajectory. The quantity ρ_t provides a measure of the system contraction towards the periodic orbit. We then plot histograms of all the ρ_t for different times t. The result is shown in Figure 5.14. As time t increases, the probability that ρ_t is less than 1 becomes large. In fact it appears that ρ_t tends to zero in probability, i.e. $\forall \epsilon > 0$

$$P[\rho_t < \epsilon] \to 1 \tag{5.64}$$

as $t \to \infty$. Thus in this case the controlled system is probabilistically locally stable, although it is also clear from the histograms that the system is not stable in the classical sense.

Indeed careful examination shows that the control feedback signal does not necessarily remain at 0 once the system has been stabilised, it fluctuates in small quantity so as to keep the system stabilised onto the unstable periodic orbit.

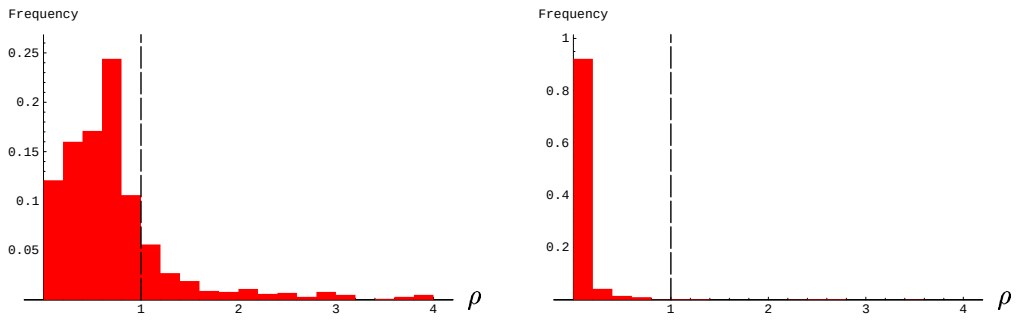


Figure 5.14: Histograms of ρ_t for Rössler attractor at $t = 11.8$ and 53.1.

A similar justification of delayed feedback control is also used later in Section 8.3. Note that in order to have confidence in the minimum distance estimate of the current state from the target unstable periodic orbit the sampling of points on the orbit must be sufficiently fine-grained. For complex target orbits it might be necessary to build a kd-tree for the sample points in order to facilitate rapid calculation of the minimum distance.

5.7.2 Periodic perturbation control (GM)

This periodic perturbation control technique is proposed by Güémez and Matías [Güémez and Matías 1993; Matías and Güémez 1994; 1996]. The application of this technique (GM) is very simple and it works by applying instantaneous periodic kicks to the system variables, that amount to changes that are proportional to their current values, and that take the form

$$x_i = x_i \left(1 + \gamma_i \delta(t - j\tau)\right), \tag{5.65}$$

where x_i represents the i^{th} variable of the system at a given instant of time, γ_i regulates the intensity of the perturbation applied to the i^{th} variable, δ is Dirac's δ function, and j runs over natural numbers, implying that the perturbations are applied at intervals that are uniformly spaced by τ. The proportional perturbations can be applied to all or only to some of the system variables.

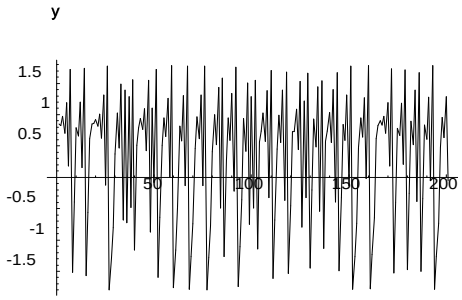


Figure 5.15: Chaotic Hénon time series of variable y.

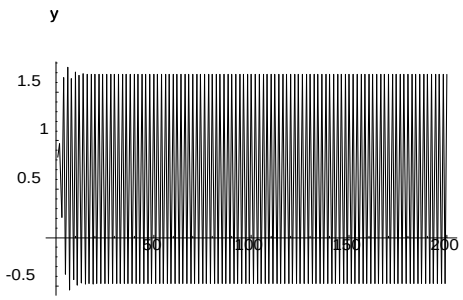


Figure 5.16: Controlled y with the same starting state.

As originally described the GM method is suitable for system with discrete variables. However, it can be applied to a continuous system if a Poincaré section is used, as in the example of controlling Rössler system described in [Matías and Güémez 1996] (More demonstrations can be found in the same paper). The following is a simple experiment on using this method on the Hénon map as is defined in (5.49).

In the experiment, the variable y was perturbed with $\gamma = -0.7$ and $\tau = 2$. The controlled result is shown in Figure 5.16. Figure 5.15 shows the system with the same initial conditions but without control. The system can be seen to quickly stabilise into a 2-cycle, $\{(1.682359, -0.471538) \rightarrow (-0.471538, 1.682359)\}$.

Similarly to the delayed feedback control, there are two parameters γ and τ associated with this control law. The correct choice of these parameters governs the success of the application of control. Again there does not seem to be any theoretical proof to explain the validity of the control technique. From an engineering point of view, this is a very quick and simple control technique if correct choices of the parameters are made. Further discussion of this can be found in later chapters after more experiments.

5.8 Discussion

This chapter has introduced some of the basic ideas of dynamical systems and chaotic dynamics, and of the control techniques used to bring chaotic motion into some type of orderly behaviour. Several chaos control methods have been described. They each have their advantages and disadvantages but are all capable of controlling low dimensional chaotic systems. However, further investigation and experiments are needed to study their effect on high-dimensional chaotic systems, as most neural systems are likely to be high dimensional.

Chapter 6

Controlling chaotic neural networks

Chaotic dynamics within the biological system seems to aid neural information processing as observed by Freeman (see Chapter 1). In order to take advantage of this idea in practical applications, it is necessary to study how chaotic neural systems can be encouraged to follow particular unstable periodic orbits. In this chapter, we demonstrate the feasibility of controlling a standard-model neuron, recurrent, artificial neural network; whose dynamical behaviour displays chaos, by using the control methods reviewed earlier.

6.1 Control with the OGY method

6.1.1 Controlling a simple chaotic neural network

The simple neural network that is used in this section for studying the OGY method applied to a chaotic neural net consists of only two neurons. This network was first studied by Wang who *proved* that there exists period-doubling to chaos and chaotic attractors in the network using a *homeomorphism*[1] from the network to a known dynamical system having these properties [Wang 1991].

The architecture of this simple network is shown in Figure 6.1. It consists of only two neurons with thresholds set to zero. The weight matrix is:

$$W = \begin{bmatrix} a & ka \\ b & kb \end{bmatrix} \tag{6.1}$$

for some non-zero numbers $a, b, k \in \mathbb{R}$. The states of the two neurons are denoted as x and y respectively, whose values range in the interval $\boldsymbol{I} = [0, 1]$, and a state of the network is denoted as a vector (x, y) in the state space $\boldsymbol{I}^2$. We consider that the neural network updates its state in discrete time $t = 0, 1, 2, \ldots$, according to the following dynamics:

$$(x(t+1), y(t+1)) = F_T\left(x(t), y(t)\right) \tag{6.2}$$

where

$$F_T\left(x(t), y(t)\right) = (\sigma_T(ax + kay), \sigma_T(bx + kby)) \tag{6.3}$$

[1] Two maps $F : X \to X$ and $G : Y \to Y$ are said to be *topologically conjugate* if there exists a *homeomorphism* (i.e., a one-to-one and continuous map with a continuous inverse) $H : X \to Y$ such that $G = H \circ F \circ H^{-1}$. The homeomorphism H is called a topological conjugacy of F and G. It is known that if F and G are topologically conjugate, then they have the same dynamical behaviour, i.e. the same orbit structure and stability [Devaney 1992].

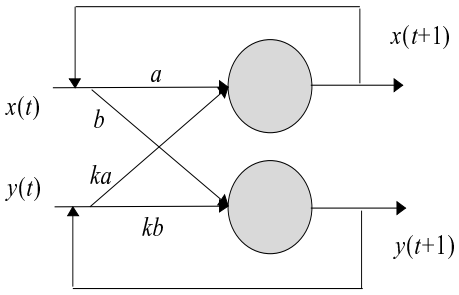


Figure 6.1: A simple network with outputs being fed back into inputs as a discrete dynamical system.

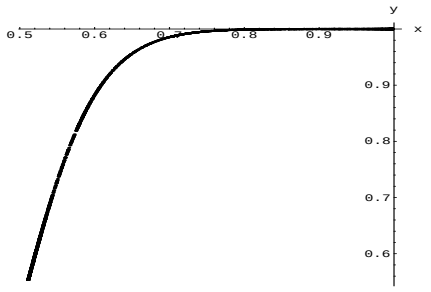


Figure 6.2: The attractor in network F_T with parameters $a = -5$, $b = -25$, $k = -1$ and $T = 1/4$.

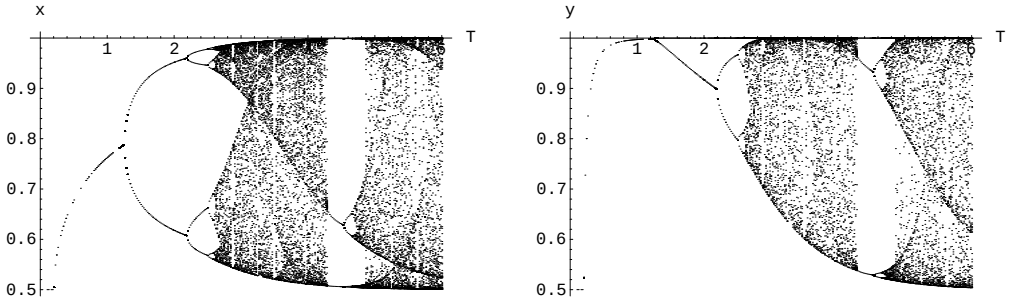


Figure 6.3: Bifurcation diagrams in x and y for the network with parameters $a = -5$, $b = -25$, $k = -1$.

and

$$\sigma_T(z) = \frac{1}{1 + e^{-z/T}} \tag{6.4}$$

The neuron activation function $\sigma_T(z)$ is sigmoidal with a parameter $T > 0$.

With the parameters $a = -5$,$b = -25$, $k = -1$ and $T = 1/4$, this system possesses a chaotic attractor shown in Figure 6.2. The *proof* that this network is chaotic derived from the bifurcation diagrams in x and y for the network in Figure 6.3.

Using the same OGY control mechanism as in the experiment on controlling the Hénon map (see Chapter 5), this system was stabilised onto the fixed point (0.896853, 0.999980), using T as the controlling parameter with initial value of $T = 1/4$. The local linear map near this fixed point was approximated by the Jacobian matrix

$$J = \begin{bmatrix} -1.96322 & 2.08867 \\ -0.00755664 & 0.00893465 \end{bmatrix} \tag{6.5}$$

where the unstable and stable eigenvectors were found to be $\boldsymbol{e}_u = (-0.999993, -0.00384731)$ and $\boldsymbol{e}_s = (-0.728493, -0.685053)$ respectively. The unstable and stable eigenvalues were $\lambda_u = -1.95519$, $\lambda_s = 0.000898838$ respectively. The approximate sensitivity vector was found to be (0.0516806, 0.00197867). The control result is shown in Figure 6.4 and Figure 6.5.

During this experiment, it was very difficult to approximate the sensitivity vector to obtain the desired control result. The difficulty was due to the unusual *shape* of the chaotic attractor which is thin and narrow. It might also be due to the fact that the stable eigenvalue of the local linear map was very small. This problem is reflected in the fact that this Jacobian has very small determinant. Different adjustments were made in order to achieve the control by:

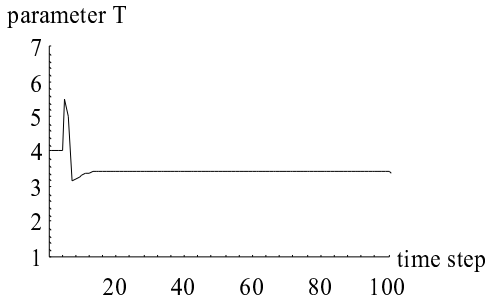


Figure 6.4: Changes of the control parameter T during the OGY control on the simple chaotic network.

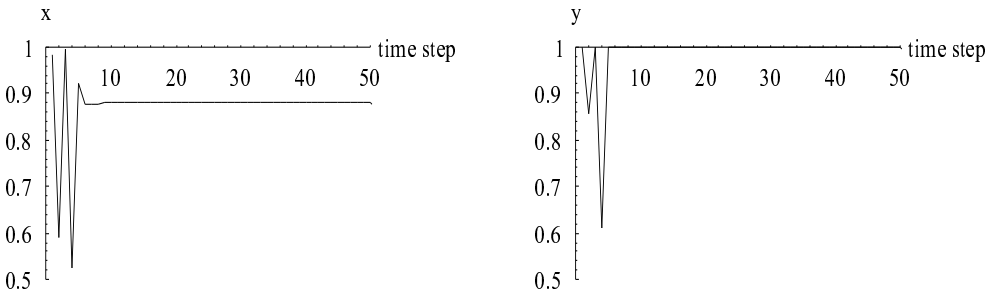


Figure 6.5: The simple chaotic network is stabilised onto the fixed point in under 10 time steps.

- increasing the allowed range of the control parameter;
- including more points near the fixed point for approximating the Jacobian matrix;
- resizing the local region near the fixed point to get a better estimate of the local linear map.

This experiment illustrates some potential problems which may arise in applying the OGY control method. This control technique is very sensitive to the quality of the required approximation.

6.1.2 Controlling a trained chaotic neural network

In the next experiment we trained a feedforward network on the Ikeda map [Hammel *et al.* 1985] and then by feeding the outputs back into the inputs empirically produced a neural network with chaotic attractor [Welstead 1991; Dracopoulos and Jones 1993] as shown in Figure 6.6. The training was done by the modified training software from Master's[2][Masters 1993].

The Ikeda map is defined by

$$g(z) = \gamma + Rz \exp\left[i\left(\kappa - \frac{\alpha}{1+|z|^2}\right)\right] \tag{6.6}$$

where z is a *complex* variable, of the form $x + iy$, and $i^2 = -1$. We can identify $x + iy$ with the point (x, y) on the complex plane so that g can also thought of as a mapping of $\mathbb{R}^2 \to \mathbb{R}^2$. The dynamical system is then defined by $z_{n+1} = g(z_n)$. For parameter values $\alpha = 5.5$, $\gamma = 0.85$, $\kappa = 0.4$ and $R = 0.9$, this mapping has a chaotic attractor illustrated in Figure 6.7. With only 4000 training

[2]The software uses the conjugate gradient method training algorithm (see Appendix D) which is very efficient. The disadvantage is that it requires to read all the training data into memory. This causes a problem in running the software under MS-DOS(tm) when training with a large set of data. It was necessary to modify the software for running under an environment without 'memory restriction'.

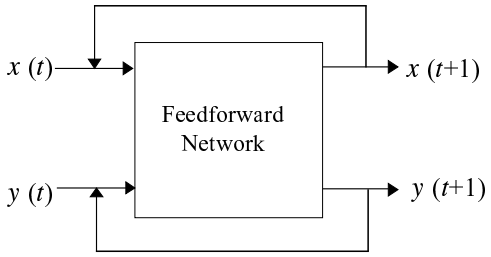


Figure 6.6: Feedforward network as a dynamical system.

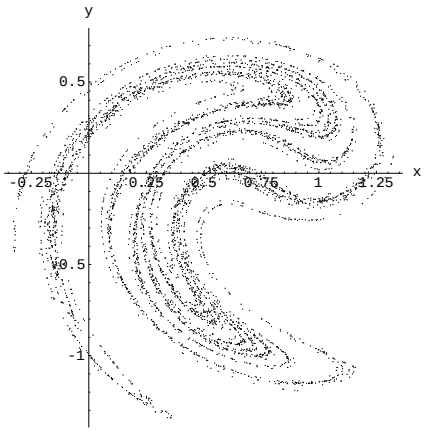


Figure 6.7: The Ikeda chaotic attractor, for parameter values $\alpha = 5.5$, $\gamma = 0.85$, $\kappa = 0.4$ and $R = 0.9$.

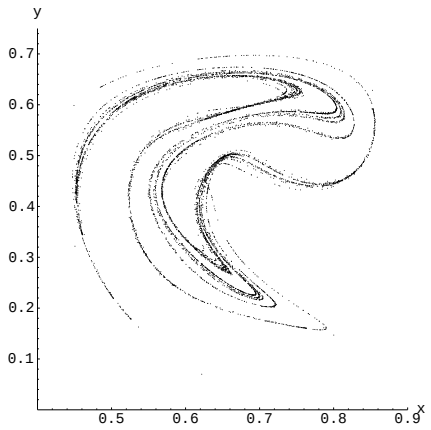


Figure 6.8: Attractor for chaotic network with architecture 2-10-10-2 (with inputs and outputs rescaled to [0,1]).

pairs (re-scaled into the range [0,1]) and training MSE error of about 9.9×10^{-5}, the network already produces an attractor shown in Figure 6.8 with features similar to the original Ikeda map chaotic attractor.

Using the first Lyapunov exponent estimation algorithm with 10000 iterations the Lyapunov exponents of this neural system are estimated to be $\{$0.368973, -0.769616$\}$. For comparison, the second algorithm was performed on a time series of 10000 data points from this network and the estimated Lyapunov exponents were $\{$0.367997, -0.660926$\}$. Both techniques give a positive Lyapunov exponent and a negative Lyapunov exponent which indicates that this network dynamics is definitely chaotic.

We use this network as the basis for the initial control experiments, the objective being to determine which parameters or system variables are most effective in stabilising the system onto an unstable periodic attractor.

The OGY control method was applied to control the chaotic neural network described above. An unstable fixed point $\xi_F =$ (0.626870, 0.553256) was located by examining successive iterations of the system and was used as the unstable periodic point to be stabilised. The Jacobian at this point was

$$J = \begin{bmatrix} -1.26617 & -1.03629 \\ -0.564996 & -1.06779 \end{bmatrix} \tag{6.7}$$

with eigenvalues $\lambda_s = -0.395399$ and $\lambda_u = -1.93857$, and stable eigenvector $\boldsymbol{e}_s =$ (0.7656, -0.643317) and unstable eigenvector $\boldsymbol{e}_u =$ (-0.838887, -0.544306).

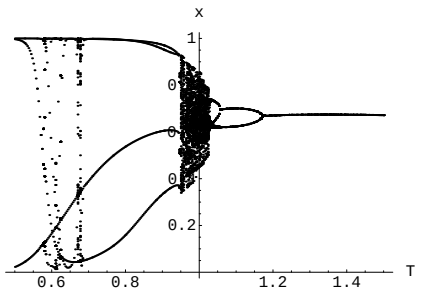


Figure 6.9: Bifurcation diagram for output x obtained by varying parameter T simultaneously in all nodes.

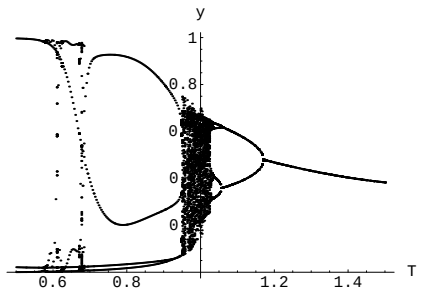


Figure 6.10: Bifurcation diagram for output y obtained by varying parameter T simultaneously in all nodes.

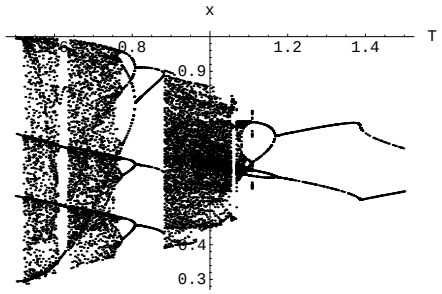


Figure 6.11: Bifurcation diagram x obtained by varying T in the output layer only.

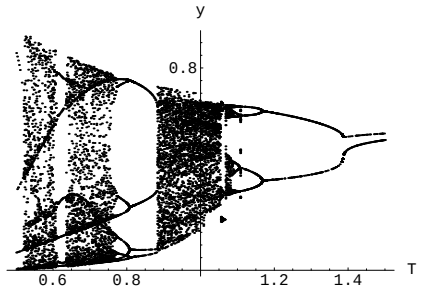


Figure 6.12: Bifurcation diagram y obtained by varying T in the output layer only.

Using T as a control parameter

The first attempt used T (i.e. effectively the slope of the sigmoidal in (6.4)) as a control parameter with $T = 1$ being the nominal value, as this was the value used in training. In these initial experiments T was varied in all nodes of the network simultaneously. These attempts to effect control were unsuccessful.

By examining the bifurcation diagrams Figure 6.9 and Figure 6.10 we conclude that the possible explanation is that the chaotic region around $T = 1$ is *small*. Slight changes of T will result in changes in dynamics from chaos to stability. Therefore the system looses the original 'not-perturbed' dynamics rapidly due to high sensitivity to this parameter change (i.e. even small variations of T change the nature of the attractor).

The next attempts were made by varying T of nodes in a particular layer of the network and here the OGY control method worked better. It seems that by varying T in only one particular layer the chaotic regions of the bifurcation diagrams become broader (see Figure 6.11 and Figure 6.12) and so control becomes easier with small variations of T. The variations of T and the controlled result are illustrated in Figure 6.13.

Using variation of the inputs

The results of using an external signal feeding into one of the inputs as a control parameter, whose nominal value is set to zero, were significantly more interesting. The bifurcation diagrams for $x(t)$ are given in Figure 6.14 and Figure 6.15. We use the same fixed point as before, so the Jacobian and associated eigenvectors and eigenvalues remain unchanged.

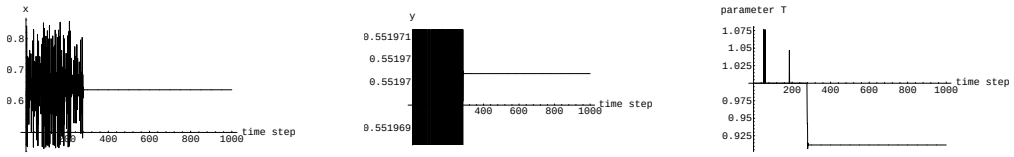


Figure 6.13: Control results of using T in the output layer as the control parameter.

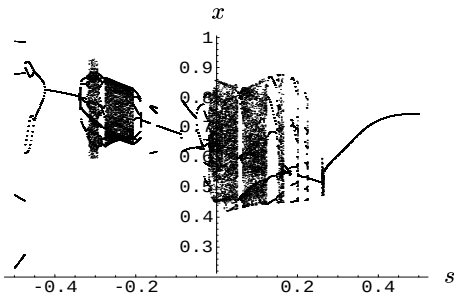


Figure 6.14: Bifurcation diagram for the output $x(t+1)$ using an external variable added to the input $x(t)$.

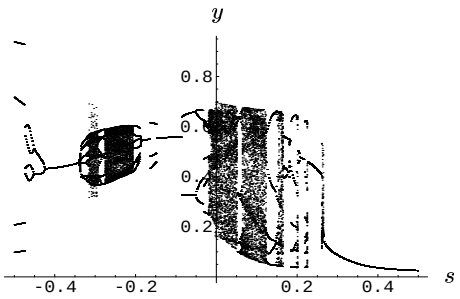


Figure 6.15: Bifurcation diagram for the output $y(t+1)$ using an external variable added to the input $x(t)$.

Using an external signal feeding into input x (as shown in Figure 6.6), the sensitivity vector $\boldsymbol{u}_x = (-1.076260,\ -0.675875)$ was approximated. After applying the OGY control for less than 25 time steps the system rapidly stabilised onto the unstable fixed point as illustrated in Figure 6.16.

The bifurcation diagrams for the outputs $x(t+1)$ and $y(t+1)$ using an external variable with nominal value zero added to the input $y(t)$ are given in Figure 6.17 and Figure 6.18. Similarly, an external signal feeding into input y (c.f. Figure 6.6) was used as the control parameter with sensitivity vector $\boldsymbol{u}_y = (-1.204806, -1.062638)$. The controlled result is shown in Figure 6.19.

In these experiments, an improved technique due to [Otani and Jones 1997b] was actually used to estimate the sensitivity vectors $\boldsymbol{u}$. The Jacobian is used to obtain a prediction of where the system would be at the next iteration if no control were applied. However, in the case of a neural network this is unnecessary since *the neural network is effectively its own Jacobian at every point*. We can therefore obtain an exact prediction of the next system state by simply iterating the network without control. This resulted in much more accurate estimations of the sensitivity vectors, which itself made control of the system using the OGY method much easier.

6.1.3 Experiment summary

The OGY method can be applied to the control of conventional feedforward networks whose behaviour under iterated feedback has been trained to be chaotic. Whilst the method is computationally expensive

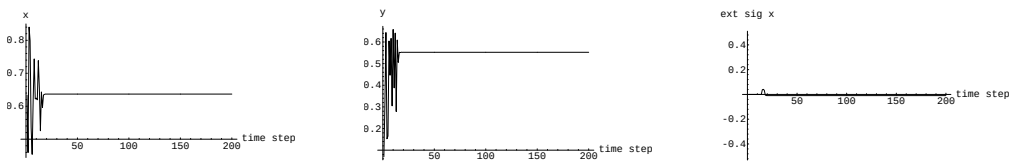


Figure 6.16: Controlled results of the network using external signal perturbation to input x.

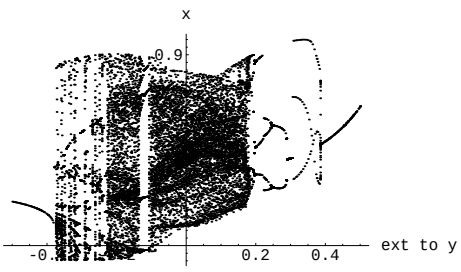


Figure 6.17: Bifurcation diagram for the output $x(t+1)$ using an external variable added to the input $y(t)$.

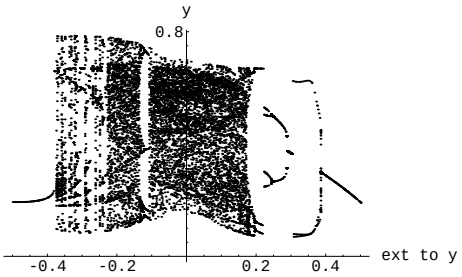


Figure 6.18: Bifurcation diagram for the output $y(t+1)$ using an external variable added to the input $y(t)$.

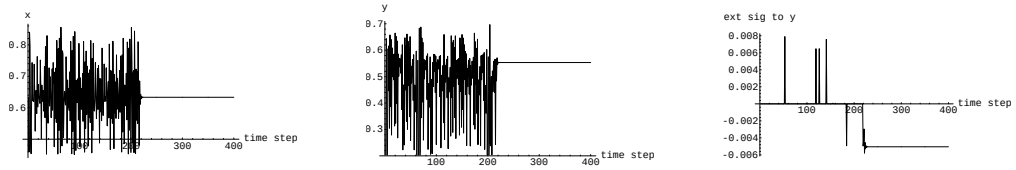


Figure 6.19: Controlled results of the network using external signal perturbation to input y.

and, in its original form subject to a number of limitations (for example inaccuracies in estimating the Jacobian or sensitivity vectors can make control difficult if not impossible), nevertheless we see that stabilisation of unstable fixed points is perfectly feasible. However, this relaxation onto a fixed point is achieved by a control external to the network itself rather than as an implicit consequence of network function.

It is interesting to observe that control by variation of a global slope parameter is not easy to achieve, but becomes easier when the control variations are applied to a single layer rather than to the whole network. It is notable that control becomes very much easier when the controlling parameter is a small signal applied to one of the inputs. This may be closer to being a biological analogy than control of behaviour through global or selective slope control.

Quite how easy it would be to extend such control to networks with many outputs being fed back to many inputs remains to be determined. It also remains to be determined whether it is practical to control high dimensional networks to follow unstable periodic orbits rather than fixed points. It is likely that more sophisticated variations of the OGY technique or some completely different control method would be required to accomplish this goal.

6.2 Otani-Jones control on a trained chaotic neural network

Similarly, we also tried the OJ method on the trained chaotic neural network described in the last section. The unstable fixed point (0.630579, 0.551984) was found for this experiment. First we tried using T of all the nodes in any particular layer of the network, but this was not successful. Instead we chose to use external signal perturbation to the two inputs x and y as in the earlier experiment. We estimated the sensitivity vectors $\boldsymbol{u}_x$ =(-1.37770, -0.602572) and $\boldsymbol{u}_y$ =(-1.08691, -1.06530), again for the external signal to x and external signal to y respectively. The predictor used in this case is the feedforward network itself because the neural network can be iterated without control to give an exact prediction of the next system state very rapidly.

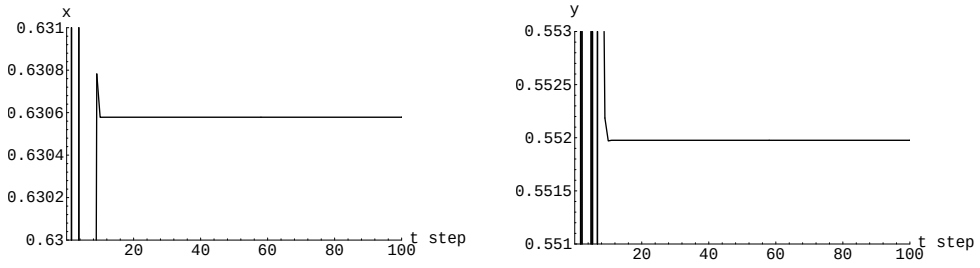


Figure 6.20: OJ controlled x and y of the chaotic neural network.

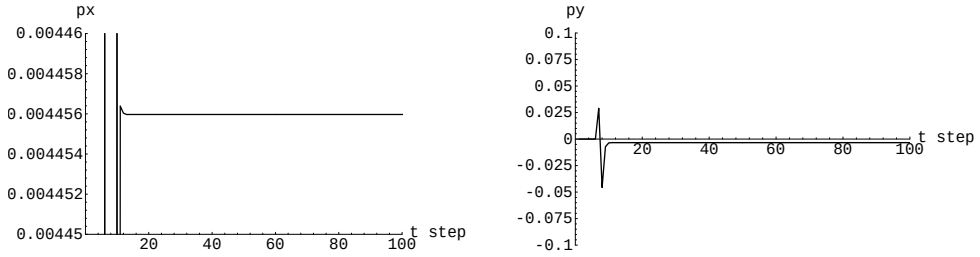


Figure 6.21: Control signals to x and y during the OJ control on the chaotic neural network.

The results of applying the OJ control are shown in Figure 6.20. The system was stabilised very rapidly by the control in less than 20 time steps. The external perturbation control signals to x and y are shown in Figure 6.21 and the perturbations were very small and were applied continuously during the control.

Certainly, this method seems to be very effective. However, the problems of this control method are

- unstable fixed points/orbits have to be located before applying the control;
- a predictor is required and
- it still requires sensitivity analysis on parameter changes.

Therefore, it does not seem to be biological plausible and it is very unlikely to be the control method required for constructing a chaotic neural memory system.

6.3 Proportional delayed feedback control on a chaotic neural network

The original delayed feedback control is a continuous control method for controlling continuous chaotic systems (see Section 5.7.1). However, we applied the same idea but modified the method to control the chaotic discrete neural network which was trained on the Ikeda map described in the last section.

We applied a delayed perturbation to the input y of the form

$$F(t) = -kD(t) = -k(y(t) - y(t-\tau)) \tag{6.8}$$

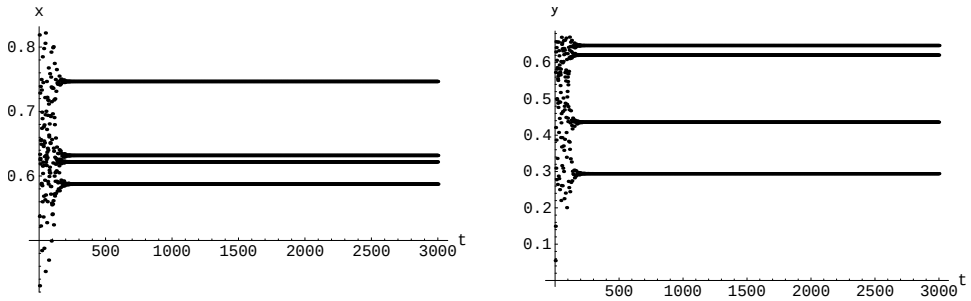


Figure 6.22: Controlled results of the chaotic neural net using the delayed feedback technique. The control signal was switched on at $t = 50$.

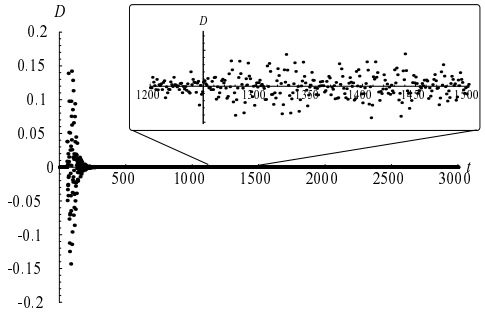


Figure 6.23: Perturbation signal during control. (Signal does not go to zero, as shown in zoomed view.)

which is added to the input y when the control was switched on, where $k = 0.687263$ and $\tau = 5$. These parameters were chosen by trial and error. There was no restriction on the size of the perturbation during the control. Note that y here is updated discretely. The initial state of this system was chosen at $x = 0.329222$ and $y = 0.996373$. The controlled results are shown in Figure 6.22.

After the control was switched on at $t = 50$, the system immediately started converging to a near-periodic-four cycle at $\{$(0.631890, 0.294159) $\rightarrow$ (0.587467, 0.647057) $\rightarrow$ (0.621878, 0.436043) $\rightarrow$ (0.746626, 0.620744)$\}$. This method required a very small perturbation signal as shown in Figure 6.23. Closer examination of the graph reveals that a small perturbation was continuously applied to the system to maintain this near periodic behaviour. This technique was tried for various different initial condition and different values of k and τ. In most cases, the system stabilised onto the *same* or different periodic orbits. It occasionally seemed to be the case that the initial starting point determined the basin of attraction.

However, choosing k and τ is a random process and therefore it is still a *black box* technique. Nevertheless, this method does not require any estimation and pre-calculations, as with the OGY method. The idea of this control method is also biologically sound: having a feedback in a neural system in order to stabilise the system dynamics. Babloyantz's group [Sepulchre and Babloyantz 1993; Lourenço and Babloyantz 1994; Babloyantz *et al.* 1995] have succeeded in controlling a network of oscillator neurons using this technique. This method might form the basic ingredient for constructing a chaotic memory system as also proposed by Hoff [Hoff 1994]. By having many delay lines within a neural network external stimulation can be fed into the system as the variation of signals of the values k and τ.

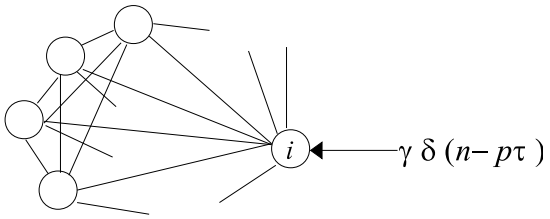


Figure 6.24: Applying GM perturbation to node i only of a fully connected network of m neurons.

6.4 Periodic perturbation control method on discrete neural networks

Here we also tried using the periodic perturbation control [Güémez and Matías 1993] (or GM for short) on discrete neural networks as in [Solé and de la Prida 1995]. The details of this method can be found in Section 5.7.2. In this experiment, we used an m-neuron fully connected network defined as

$$x_i(n+1) = \sigma_T \left(\sum_{j=1}^{m} w_{ij} x_j(n) \right), \tag{6.9}$$

with $i = 1, \ldots, m$, which is basically a m-dimensional map $\boldsymbol{X}_{n+1} = F_T(\boldsymbol{X}_n)$, where $\boldsymbol{X} \in \mathbb{R}^m$. Here $\sigma_T(z)$ is as defined in (6.4). (w_{ij}) is the connectivity matrix. With suitable connectivity, this system can generate deterministic chaos [Wang 1991].

In the experiment, we applied the GM method on a single neuron of the m-network (see Figure 6.24), therefore we have the system

$$x_i(n+1) = \sigma_T \left(\sum_{j=1}^{m} w_{ij} x_j(n) \right) (1 + \gamma\delta(n - p\tau)), \tag{6.10}$$

$$x_k(n+1) = \sigma_T \left(\sum_{j=1}^{m} w_{kj} x_j(n) \right), \qquad (1 \le k \le m, k \ne i), \tag{6.11}$$

as described in (5.65) and p is any natural number. A fixed perturbation γ is applied at intervals that are uniformly spaced by τ.

First we tried the control on the system $m = 3$ with $T = 1/4$ and the connectivity matrix [Wang 1991]

$$(w_{ij}) = \begin{pmatrix} -5 & 5 & -2 \\ -25 & 25 & -2 \\ -2 & 2 & -2 \end{pmatrix}. \tag{6.12}$$

The chaotic attractor of this system is shown in Figure 6.25.

Setting $\tau = 4$ and $\gamma = -0.1$, the neural system quickly stabilised into a period-4 orbit {(0.449371, 0.499302, 0.499301) → (0.0476165, 0.7308, 0.0267278) → (0.999999, 1, 0.994788) → (0.000349632, 0.000349673, 0.000349626)}, as shown in Figure 6.26. It is very interesting that by applying a fixed amount of periodic perturbation to one of the nodes the whole system stabilises.

Similarly we performed the same experiment on a system with $m = 4$, $T = 1/4$ and the connec-

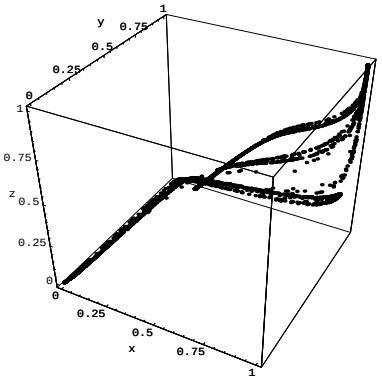


Figure 6.25: Chaotic attractor for the neural network with $m = 3$.

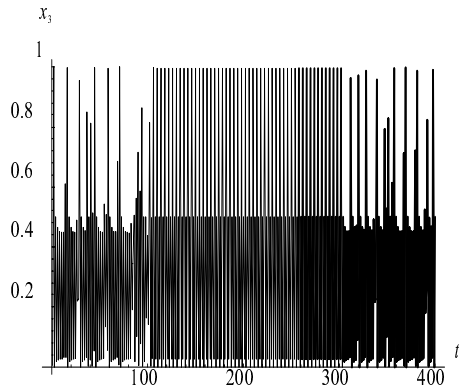


Figure 6.26: Controlled result of node 3 of the neural system. Control switched on at $t = 101$ and switched off at $t = 300$.

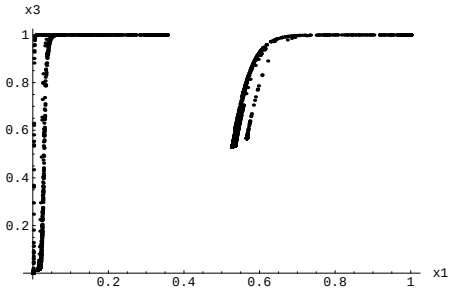


Figure 6.27: A projection of the chaotic attractor for the system with $m = 4$.

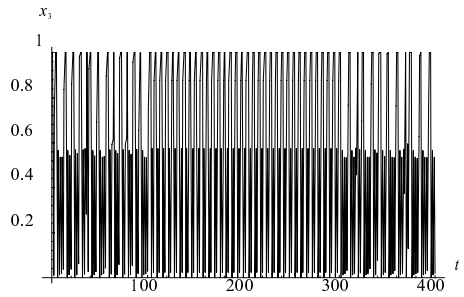


Figure 6.28: Signal from node 3 of the system with $m = 4$ with control switched on at $t = 101$ and switched off at $t = 300$.

tivity matrix

$$(w_{ij}) = \begin{pmatrix} -5 & 5 & -2 & 0.1 \\ -25 & 25 & -2 & 0.1 \\ -25 & 25 & -2 & 0.1 \\ -0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}. \tag{6.13}$$

A projection of this 4-dimensional chaotic attractor is shown in Figure 6.27. With the values $\tau = 2$ and $\gamma = -0.7$ for the control, this chaotic system stabilised onto a period-6 orbit, {(0.000135249, 0.000451377, 0.000451377, 0.667242) → (0.566997, 0.573195, 0.573195, 0.566406) → (0.0042818, 0.0232216, 0.0232216, 0.612616) → (0.607799, 0.875801, 0.875801, 0.565106) → (0.0583833, 1, 1, 0.664534) → (0.999985, 1, 1, 0.739324)} shown in Figure 6.28, similar to the results from [Solé and de la Prida 1995].

This method is similar to the delayed feedback control. It suffers from a similar drawback that the choice of the critical values τ and γ is important and there are no specific guidelines to choose these values. There is also no formal theoretical explanation to support the success of this control technique.

6.5 Discussion

Many control methods have been tried on some neural networks exhibiting chaotic dynamics. It has been shown that stabilisation of unstable fixed points is perfectly feasible. However, this relaxation onto a fixed point, with methods such as the OGY and the OJ techniques, is achieved by a control external to the network itself rather than as an implicit consequence of network function. Also it seems to be that the choice of the control parameter in such control techniques plays an important role in successfully controlling chaotic neural networks. The experiments suggest that a small external signal applying to the system inputs can control such a neural system fairly easily. This is also supported by the GM control method and the delayed feedback control, which can successfully control a chaotic neural system by employing perturbations to the system variables.

In fact, delayed feedback control can easily be imagined in a real biological neural network for controlling its chaotic dynamics. Having feedbacks in neural systems is already known to enrich the neural dynamics, by increasing the range of achievable periods in a network of oscillators as observed by Baldi and Atiya [Baldi and Atiya 1994]. However, Baldi and Atiya did not consider chaotic dynamics in their neural models. Therefore delayed feedback control might well be the only technique for realising a chaotic neural memory system with the behaviour observed by Freeman [Freeman 1991]. However, further understanding of this control technique is required.

Chapter 7

Higher Dimensional Chaos Control

There is now an extensive literature demonstrating experiments on controlling low (usually 2 or 3) dimensional chaotic physical systems using the original chaos control techniques, such as the OGY method or similar variants. Most of these control methods, as introduced in the last chapter, are designed for (or restricted to) low dimensional chaotic systems. We are interested in applying chaos control techniques to higher dimensional chaotic systems. Using simulations, we apply several higher dimensional chaos control techniques in an attempt to extract much more *regular motion* from a tumbling satellite. The chaotic behaviour of the satellite in our experiments uses linear feedback of the angular velocities based on the original suggestion in [Leipnik and Newton 1981]. However, if only this type of perturbation is used the attitude angles are essentially decoupled from the other equations. To ensure that the system really does illustrate higher dimensional chaos we have introduced additional non-linear perturbation terms dependent on the attitude angles.

First, a continuous delayed feedback control, is applied to the same chaotic satellite. Then we implement the OJ control technique as introduced earlier. Finally, we experiment with the higher dimensional control technique proposed in [Ding *et al.* 1996], which we shall refer to as the DYIDSG method. This method assumes that there is only a times series of a single scalar variable available, and the control strategy is similar to the original OGY method but extended for higher dimensional chaotic systems. We present experimental results on the chaotic attitude control problem and compare the difficulties and merits of each of these techniques.

The OJ method and the DYIDSG higher dimensional OGY method are both formulated for discretely updated systems. For the OJ method there are two possibilities: one can work with the full state description at discrete steps or one can work with a single scalar variable and perform a reconstruction of the dynamics using an embedding. The DYIDSG method, on the other hand, bases the control on the assumption that only a single time series variable is available.

In addition both these methods of control require specification of an unstable fixed point which will act as the target of the control method. In order to locate an unstable fixed point one first needs to select a suitable jump time. However, for the purpose of comparison of these techniques, we should try to apply the techniques to control onto the *same* stabilised behaviour.

For illustrative purposes we apply the OJ method using the first option, i.e. a discretely updated vector which consists of the full state description, and the DYIDSG method using a suitably constructed embedding vector.

First we apply the continuous feedback control to achieve a periodic motion. If the feedback control signal vanishes on the stabilised orbit then this stabilised periodic motion is known to be an embedded

unstable orbit of the original chaotic attractor [Pyragas 1993; Pyragas and Tamaševičius 1993] and we shall ensure that this is indeed the case. Having thus located an unstable orbit of the original system (which in practice by other techniques is not actually that easy) we shall then use the same orbit again in the other two control experiments, in order to compare the results. Based on this stabilised orbit, we can construct the corresponding fixed point in some form of discrete motion for the OJ method using a full state description, and also in an embedding of a single variable for the DYSDSG control. In this way we ensure that the same periodic motion is being controlled by all three control methods.

There are a number of available techniques for obtaining fixed points and their associated jump times, which we can broadly classify into two types. First, if the equations which govern the dynamics are known we can employ iterative approximation methods using the equations [Sparrow 1982; Sepulchre and Babloyantz 1993]. Alternatively we may not know the equations and therefore have to examine time series information and possibly construct an embedding. Given that the jump time and/or delay time have been determined from the time series the method described in [Schmelcher and Diakonos 1997] could be used to determine one or more fixed points. In fact in the case of the chaotic satellite we know the equations and can easily determine an approximate unstable fixed point by iterative refinement. However, the technique described in [Schmelcher and Diakonos 1997] is of particular interest because it would appear to be closely related to the control method based on delayed feedback [Pyragas 1992], and we shall discuss this relationship at the end of the experiments.

7.1 Satellite with chaotic dynamics

The dynamical system we seek to control is the rotation of a rigid body with external perturbing forces chosen so that the resulting system exhibits chaotic behaviour. A similar stylised version of a real satellite attitude control problem subjected to chaotic perturbation has been studied using a variety of adaptive control techniques, see for example [Dracopoulos and Jones 1997; Končar and Jones 1995]. We first briefly outline the dynamical equations which describe the system.

We imagine a satellite controlled by three pairs of thrusters on the mutually orthogonal principal axes. This system is described by the Euler equations with additional terms to account for the effects of the control torques, and we follow the notation of [Crouch 1984]. The system consists of *kinematic* equations relating the attitude angles with the angular velocities, and *dynamic* equations describing the evolution of the angular velocities [Crouch 1984; Meyer 1966].

The orientation of the satellite at a given point can be locally described in terms of three angles ϕ, θ and ψ, which are successive clockwise rotations about inertial axes $\boldsymbol{I}$, $\boldsymbol{J}$ and $\boldsymbol{K}$ respectively. The corresponding rotation matrices are

$$M_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}, \quad M_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}, \qquad (7.1)$$

$$M_z(\psi) = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

respectively. These successive rotations transform the inertially fixed set of orthonormal axes $\boldsymbol{I}$, $\boldsymbol{J}$ and $\boldsymbol{K}$ (regarded as initially instantaneously coincident with the body axes) into the axes $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ fixed in the body. The angular position (the combined effect of the three rotation matrices (7.1)) can

be described by a single orthogonal rotation matrix

$$A = M_z M_y M_x \quad \left(AA^T = I\right) \tag{7.2}$$

and for some purposes it is more convenient to work with this global representation. The evolution of A may be expressed as

$$\dot{A} = S(\omega)A \tag{7.3}$$

where $\omega = (\omega_x, \omega_y, \omega_z)$ are the angular velocities of the satellite and $S(\omega)$ is the matrix defined by

$$S(\omega) = \begin{pmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{pmatrix}. \tag{7.4}$$

Equation (7.3) is the *kinematic* equation of the satellite. Alternatively this can be represented [Crouch 1984] as

$$\begin{aligned} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} &= \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}\begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} \\ &\quad + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}\begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}\begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \end{aligned} \tag{7.5}$$

and on collecting terms and inverting we get the following form

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{pmatrix}\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{7.6}$$

which, provided one uses an adaptive integration algorithm that can deal with isolated singularities, is in some respects a more suitable form for solving by numerical integration, and this is the approach adopted here. In general, numerical integration algorithms, such as Runge-Kutta, applied directly to chaotic systems often lead to significant cumulative errors. Recent studies of conservation algorithms for the dynamics of Hamiltonian systems on Lie groups using the technique commonly called *sympletic integration* [De Vogelaere 1956] could be applied to the problem of an accurate integration of (7.3) subject to the constraint $AA^T = I$. The basic idea of sympletic integration algorithms is to design into the procedure the constraints on the system which one knows in advance must apply, e.g. energy-momentum conservation. If this is done carefully the resulting procedure will be much more accurate than a conventional numerical integrator, see for example [Lewis and Simo 1994].

The *dynamical* equations are

$$\begin{aligned} I_x\dot{\omega}_x &= (I_y - I_z)\,\omega_y\omega_z + G_x + H_x \\ I_y\dot{\omega}_y &= (I_z - I_x)\,\omega_z\omega_x + G_y + H_y \\ I_z\dot{\omega}_z &= (I_x - I_y)\,\omega_x\omega_y + G_z + H_z \end{aligned} \tag{7.7}$$

where I_x, I_y and I_z are the principal moments of inertia with respect to body axes; G_x, G_y and G_z are the three control torques produced by the thrusters; and H_x, H_y and H_z are the perturbing torques which can be chosen so as to force the uncontrolled satellite into chaotic motion.

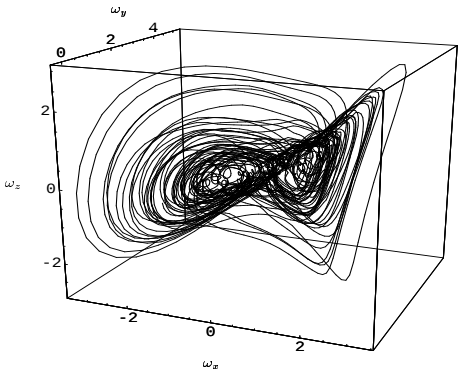


Figure 7.1: Chaotic attractor: phase portrait of the angular velocities.

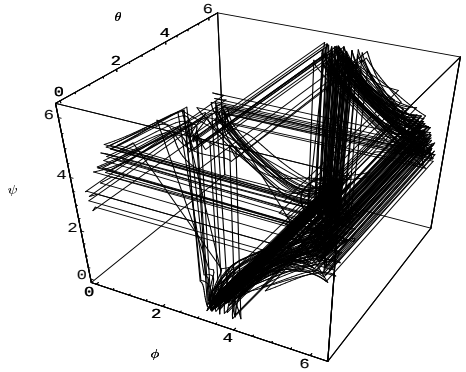


Figure 7.2: Chaotic attractor: phase portrait of the attitude angles.

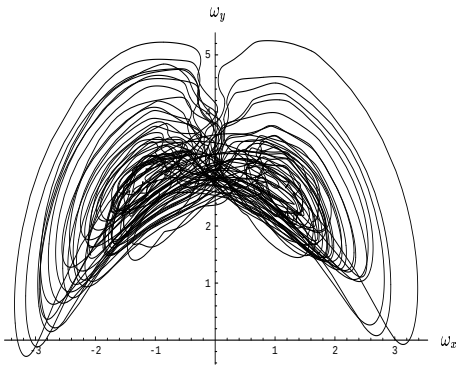


Figure 7.3: Chaotic attractor: angular velocities ω_y against ω_x.

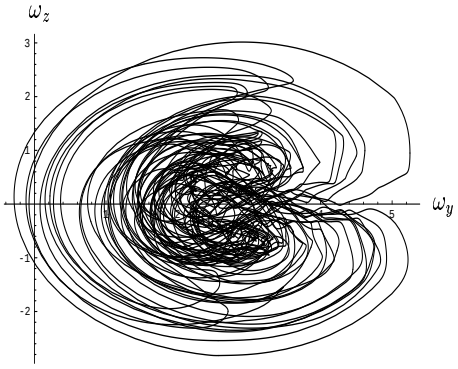


Figure 7.4: Chaotic attractor: angular velocities ω_z against ω_y

Earlier papers [Dracopoulos and Jones 1997; Končar and Jones 1995] have taken $I_x = 3$, $I_y = 2$ and $I_z = 1$ with the perturbing torques defined by

$$\begin{pmatrix} H_x \\ H_y \\ H_z \end{pmatrix} = \begin{pmatrix} -1.2 & 0 & \frac{\sqrt{6}}{2} \\ 0 & 0.35 & 0 \\ -\sqrt{6} & 0 & -0.4 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{7.8}$$

(a linear feedback matrix with suitable elements). These torques are chosen to be sufficiently large to induce chaotic motion and are comparable in magnitude with the available thruster torques. The dynamics of the satellite will then exhibit chaotic motion [Leipnik and Newton 1981].

However, by examining the dynamical equations in (7.7) and (7.8) one can see that these only involve the angular velocities. Therefore these equations can be integrated without reference to the attitude angles. To achieve a truly higher dimensional problem, and thereby obtain a more challenging control problem, we introduce extra terms involving the attitude angles ϕ, θ and ψ into the perturbation:

$$\begin{pmatrix} H_x \\ H_y \\ H_z \end{pmatrix} = \begin{pmatrix} -1.2 & 0 & \frac{\sqrt{6}}{2} \\ 0 & 0.35 & 0 \\ -\sqrt{6} & 0 & -0.4 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} + \begin{pmatrix} \cos\theta \sin\psi \\ \cos\phi \sin\theta \\ \cos\psi \sin\phi \end{pmatrix}. \tag{7.9}$$

The chaotic attractor of the system defined by (7.7) and (7.9) is shown in Figure 7.1 and Figure 7.2, which show the phase portrait of the angular velocities for time duration of $t = 500$, and the phase portrait of the attitude angles (modulus 2π) for time duration of $t = 200$, respectively. Figure 7.3 and Figure 7.4 show the cross sections of the attractor in Figure 7.1. We estimate the Lyapunov

exponents using the technique described in [Parker and Chua 1992 p.80], with initial conditions of $\omega_x = 2$, $\omega_y = 4.1$, $\omega_z = 3$, $\phi = \theta = \psi = 0$ and 50,000 integration time steps of size 0.01. In this way the Lyapunov exponents for the system $\boldsymbol{\xi} = (\omega_x, \omega_y, \omega_z, \phi, \theta, \psi)$ are estimated to be $\{0.3629, 0.1313, 0.0174, -0.0077, -0.0721, -0.7509\}$ to 4 decimal places. Having both positive and negative Lyapunov exponents indicates that the dynamical system is indeed chaotic.

7.2 Continuous delayed feedback control

The continuous delayed feedback control technique described in [Pyragas 1992] and also in Section 5.7.1 was tested on this satellite dynamical system. We chose the angular velocity ω_z as the feedback control variable and no restriction was applied to the magnitude of the control variable, i.e. $F_0 = \infty$, although the range of the resulting control torques was relatively small (see Figure 7.14). Under this control regime the dynamic equations (7.7) become

$$\begin{aligned} \dot{\omega}_x &= \frac{(I_y - I_z)\omega_y\omega_z}{I_x} + \frac{H_x}{I_x} + \frac{G_x}{I_x} \\ \dot{\omega}_y &= \frac{(I_z - I_x)\omega_z\omega_x}{I_y} + \frac{H_y}{I_y} + \frac{G_y}{I_y} \\ \dot{\omega}_z &= \frac{(I_x - I_y)\omega_x\omega_y}{I_z} + \frac{H_z}{I_z} + \frac{G_z}{I_z} - k\left(\omega_z - \omega_z(t-\tau)\right) \end{aligned} \tag{7.10}$$

where H_x, H_y and H_z are as defined in (7.8) and τ is a delay time. Since nominally $G_x = G_y = G_z = 0$, we can translate these delayed feedback perturbation equations into

$$\begin{aligned} \dot{\omega}_x &= \frac{(I_y - I_z)\omega_y\omega_z}{I_x} + \frac{H_x}{I_x} \\ \dot{\omega}_y &= \frac{(I_z - I_x)\omega_z\omega_x}{I_y} + \frac{H_y}{I_y} \\ \dot{\omega}_z &= \frac{(I_x - I_y)\omega_x\omega_y}{I_z} + \frac{H_z}{I_z} - k\left(\omega_z - \omega_z(t-\tau)\right). \end{aligned} \tag{7.11}$$

Hence the thruster G_z control is defined to be

$$G_z = -I_z k\left(\omega_z - \omega_z(t-\tau)\right). \tag{7.12}$$

In the experiment the control parameters are set to $k = 0.5$ and delay $\tau = 2.12$, which leads to the unstable periodic motion described in Figure 7.5 and Figure 7.6. In fact, many other values of k and τ could also achieve periodic, or nearly periodic, motions. However finding such parameter combinations is mainly a trial and error exercise. At present this is the principal weakness of this control method. We would stress that the delay τ chosen in the control does not necessarily always give a stabilised periodic orbit with time period τ (as suggested in the original paper [Pyragas 1992]) because sometimes the stabilised period can be a multiple of τ.

The results are shown in Figure 7.5 – Figure 7.15. Figure 7.7 shows the position of the controlled orbit in relation to the chaotic attractor in the (ω_y, ω_x) state space. In this case, the stabilised motion is ≈ 2.12 second, i.e. close to the original set-up value of $\tau = 2.12$. As we shall see, in comparison with the OJ and DYIDSG experiments these results are very impressive:

- The only information regarding the state of the system used in the control calculation is the angular velocity ω_z (i.e. five of the six possible state variables are ignored).

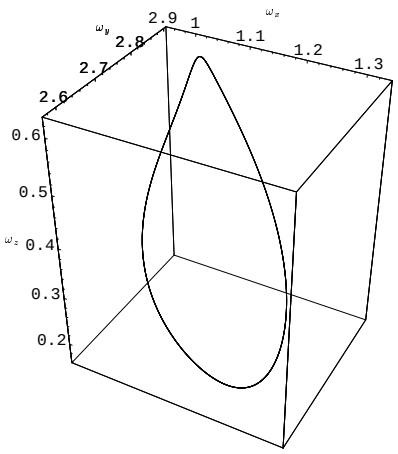


Figure 7.5: Phase portrait of angular velocities of delayed feedback controlled motion.

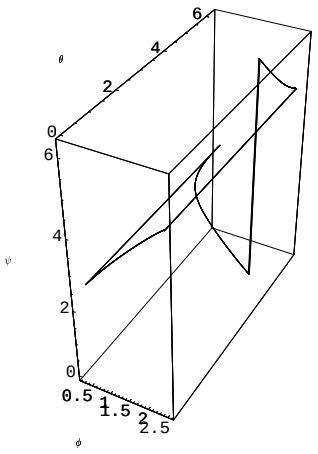


Figure 7.6: Phase portrait of attitude angles (modulo 2π) of delayed feedback controlled motion.

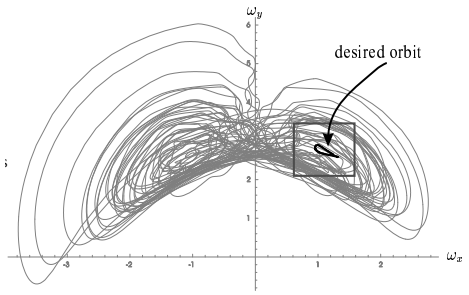


Figure 7.7: The desired periodic orbit in relationship with the chaotic attractor on the (ω_y, ω_x) state space.

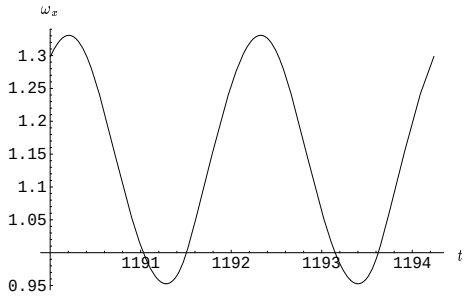


Figure 7.8: Angular velocity ω_x (delayed feedback control).

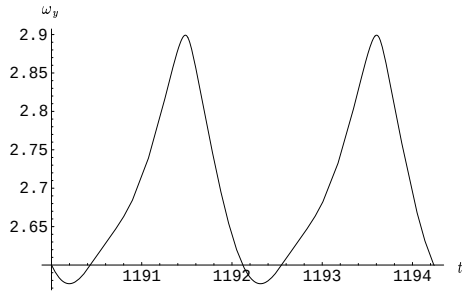


Figure 7.9: Angular velocity ω_y (delayed feedback control).

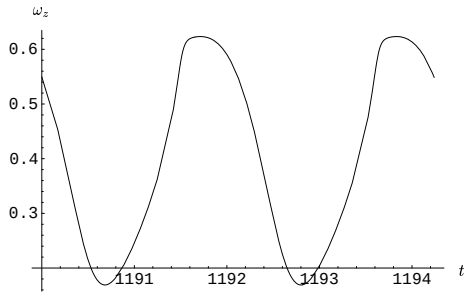


Figure 7.10: Angular velocity ω_z (delayed feedback control).

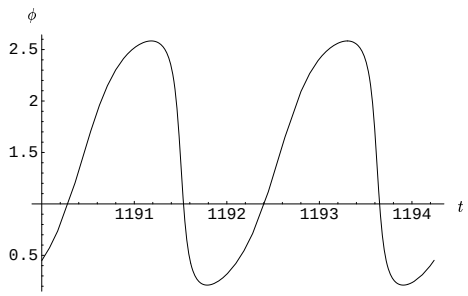


Figure 7.11: Stabilised attitude angle ϕ (delayed feedback control).

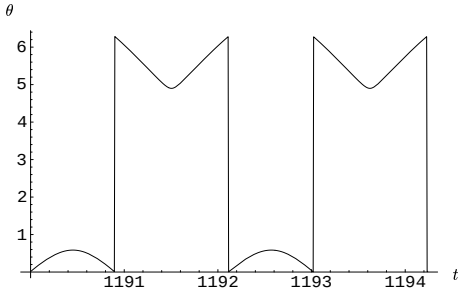


Figure 7.12: Stabilised attitude angle θ (delayed feedback control).

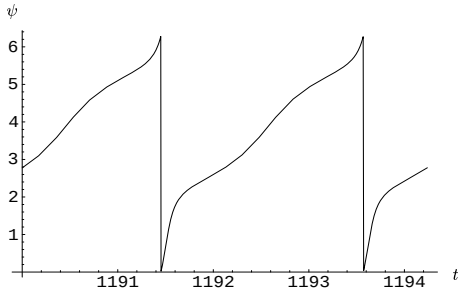


Figure 7.13: Stabilised attitude angle ψ (delayed feedback control).

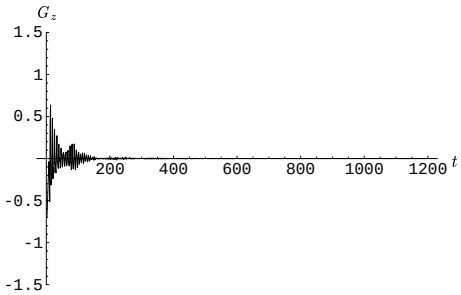


Figure 7.14: Change of control torque G_z.

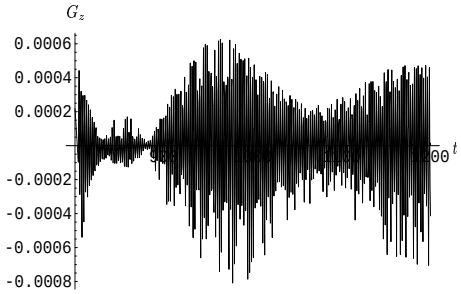


Figure 7.15: Control torque G_z when satellite is stabilised between $t = 800$ and 1200.

- Control is easily achieved using only one of the three thrusters.
- In contrast to the OJ and DYIDSG experiments no prior calculation and very little real-time calculation is required.
- The system is easily stabilised into a periodic motion for which the control thruster adjustments are very small, i.e. the energy cost of maintaining this behaviour is small as observed in Figure 7.14 – 7.15.

7.3 Direct Otani-Jones control

The *Otani-Jones* control method (OJ control) [Otani and Jones 1997a] appears to be a feasible control technique for controlling high-dimensional chaotic systems and in this section we present some results of applying the OJ method to the chaotic satellite system.

All six state variables, the angular velocities and the attitude angles, were used for the system state $\boldsymbol{\xi} = (\omega_x, \omega_y, \omega_z, \phi, \theta, \psi)$ and the control parameters were the thruster torques G_x, G_y and G_z.

The OJ method is designed for discretely updated systems, so the first step is to discretise the system in a suitable way to ensure that controlling the corresponding unstable fixed point/orbit of the discrete system is equivalent to controlling the original fixed point/orbit of the continuous system. Normally we could generate data for analysis by numerically integrating the dynamical equations and collecting observational data, say for 20,000 points in time steps of 0.1 second. However, in our case we just sample the stabilised periodic orbit achieved from the continuous delayed feedback control to obtain the target unstable periodic orbit of the discrete system. In our experience, it is easier to achieve successful control if we sample the system in a small time steps. At the start of control we find the sampled point of the target orbit which is closest to the current system state. At each step

we then choose the control solution which minimises the distance between the next system state and the corresponding next sampled state of the target orbit. The whole process is basically tracking an unstable orbit [Carroll *et al.* 1992; Gills *et al.* 1992].

We thus need to modify the original OJ method to accommodate this orbit tracking strategy. The control strategy in (5.55) can be re-written as minimise

$$\left|P\left(\boldsymbol{\xi}_{n-1}(\boldsymbol{p}_0)\right) - \boldsymbol{\xi}_n^{\text{target}}(\boldsymbol{p}_0) + \delta p_{1,n-1}\boldsymbol{s}_{1,n} + \cdots + \delta p_{l,n-1}\boldsymbol{s}_{l,n}\right|^2. \tag{7.13}$$

where $\boldsymbol{\xi}_n^{\text{target}}$ is the target next state on the unstable periodic orbit, and $\boldsymbol{s}_{i,n}$ is the sensitivity vector for the i^{th} parameter $p_{i,n-1}$ at time $n-1$.

Notice that now instead of using the original sensitivity vector, which normally is the change of the fixed point $\boldsymbol{\xi}_n^{\text{target}}$ due to the variation of the control parameter p_i at time n, we use the sensitivity vectors

$$\boldsymbol{s}_{i,n} = \left(\frac{\partial P\left(\boldsymbol{\xi}_{n-1}, p_{1,n-1}, \ldots, p_{l,n-1}\right)}{\partial p_{i,n-1}}\right) \ (1 \leq i \leq l), \tag{7.14}$$

This is the change of the predicted next state of $\boldsymbol{\xi}_{n-1}$ at time n, due to the variation of the parameters applied at time $n-1$, from the predicted state with all the parameters at their nominal values. Thus a final variation of the OJ method for tracking in this way is that the predictor function P is now dependent on both the current system state as well as the system control parameter values.

In situations where the iterated map F is unknown we could imagine constructing a fast predictive function P by training a neural network. However, in the present case, where we assume the equations are known and the objective is to demonstrate the technique, there is no virtue in training such a network and we shall therefore calculate P using F (i.e. by simply integrating the equations over the jump time).

The unstable orbit in this experiment has a periodicity of $T = 2.12$, which we divide into 120 target states. In this way we can ensure that at every control step relatively small parameter perturbations are required. Of course, using such small time steps, together with appropriate control variations, the satellite could be forced to any desired orbit, whether this orbit is an embedded unstable periodic orbit in the original chaotic attractor or not. But the point here is to achieve the particular target, which corresponds to an unstable periodic orbit of the original system, with *small* control perturbations.

In our experiment, the results indicated that using if the target periodic orbit is estimated inaccurately then even with small time steps, it was very difficult to control the system without using large parameter perturbations.

Because the sensitivity vectors in (7.14) are now dependent on the current system state, they have to be correctly estimated and recalculated at every time step. One way to achieve this for a real time application is to use a neural network as demonstrated in [Oliveira *et al.* 1997], where the trained neural network is used to calculate the sensitivity vectors for synchronisation of the chaotic systems using OJ control. To get the information necessary to train such a network we would in practice use the predictor P to analyse how the next (predicted) state changes with small variations in the control parameters. However, for this experiment we simply collect data by making small variations of the control parameters $\boldsymbol{p}$ at time $n-1$ and then integrate the system forward to time n. This is repeated about 50 times and then least squares fit is used on the data so collected to estimate the sensitivity vectors.

Finally the required control perturbation at time $n-1$ is calculated according to (7.13). However, the attitude angles ϕ, θ and ψ ($\mod 2\pi$) are not continuous and this would pose a problem for the least

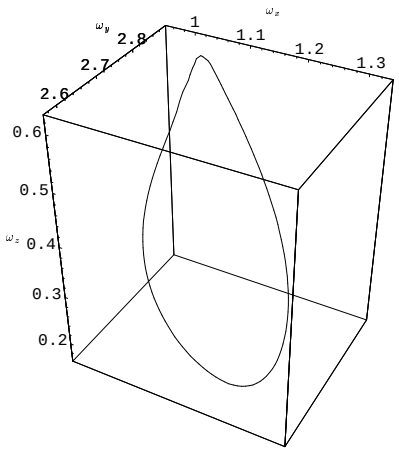


Figure 7.16: Phase portrait of angular velocities of OJ controlled motion.

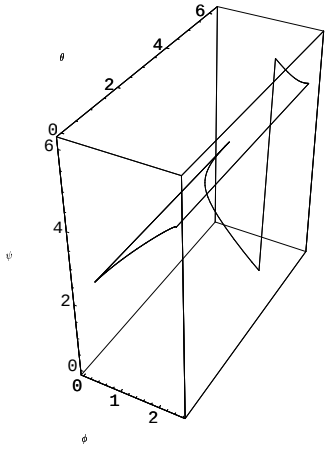


Figure 7.17: Phase portrait of attitude angles (modulo 2π) of OJ controlled motion.

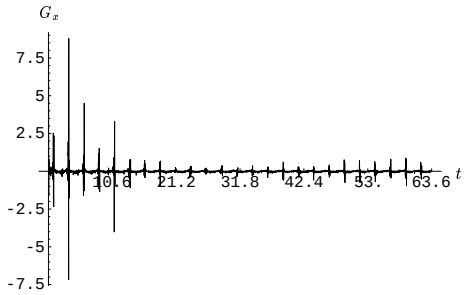


Figure 7.18: Changes of control torque G_x during OJ control.

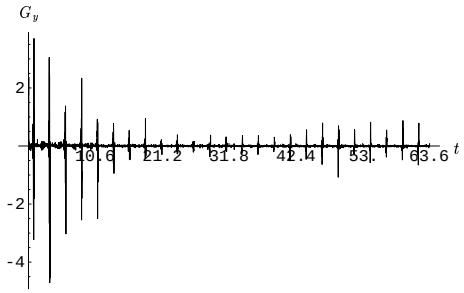


Figure 7.19: Changes of control torque G_y during OJ control.

squares fit step needed to calculate the required control perturbation. Therefore for the minimisation step of the OJ method we replace the state description in terms of angular velocities and attitude angles by a description using the orthogonal rotation matrix defined in (7.2). The state description then becomes

$$(\omega_x, \omega_y, \omega_z, a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33}) \tag{7.15}$$

where a_{ij} is an element of the rotation matrix A as in (7.2).

The results appear in Figure 7.16 – 7.17 showing that under control the satellite follows the desired unstable orbit. Note the close similarity to the results in Figure 7.5 – 7.6 for the delayed feedback control. Figure 7.18 – Figure 7.20 show the control thruster torques G_x, G_y and G_z and Figure 7.21 –

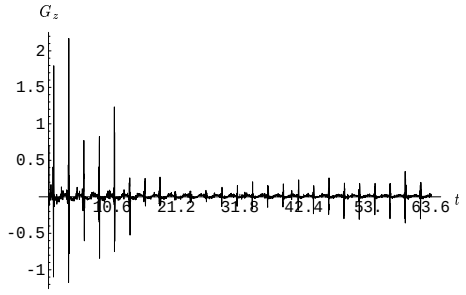


Figure 7.20: Changes of control torque G_z during OJ control.

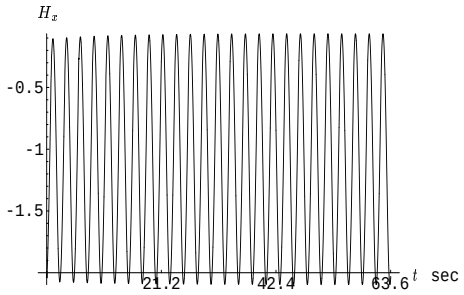


Figure 7.21: Changes of perturbing torque H_x during OJ control.

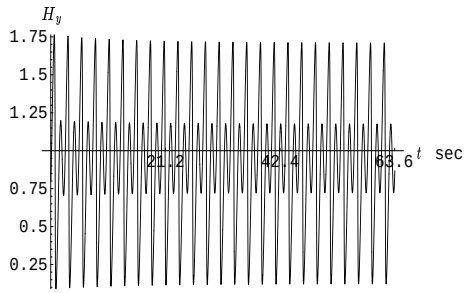


Figure 7.22: Changes of perturbing torque H_y during OJ control.

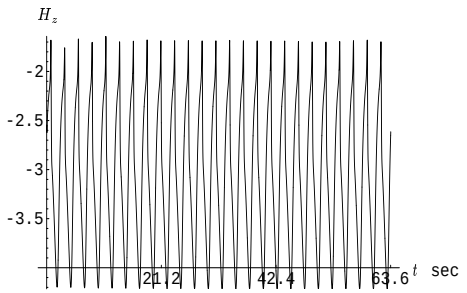


Figure 7.23: Changes of perturbing torque H_z during OJ control.

Figure 7.23 show the perturbing torques H_x, H_y and H_z acting on the satellite during the OJ control.

The control thruster torques are for the most part relatively small, compared to the perturbing torques, which is in line with the original objective. Occasionally, there is a 'burst' of larger control signals which we believe occurs because the 'natural period' of the controlled orbit does not quite align with the period $\tau = 2.12$ of the target, the 'burst' serving the function of bringing the two back into phase for a while.

If the target orbit is not a very close approximation to an embedded unstable periodic orbit of the original chaotic attractor, i.e. the target orbit has been estimated inaccurately, the control torques required to achieve stabilisation are much larger. In our earlier attempts these varied in the range from -10 to 10. However, in comparison with delayed feedback control, this method does require slightly higher torques to control the satellite with a higher computational cost, in terms of constructing the one-step predictor and the estimation of the sensitivity vectors.

7.4 DYIDSG control

In this section we first describe the DYIDSG method in its original form [Ding *et al.* 1996] and then report on our attempt to control the six-dimensional chaotic satellite system using only one thruster. This is designed to enable us to contrast the method with the continuous delayed feedback experiment, which also only used one thruster to achieve control.

In fact, despite all our efforts, the result of the DYIDSG experiment was not very successful and we shall discuss some possible reasons after presenting the details. Perhaps one should not be too surprised: in the original description of the DYIDSG method, only one control variable is used, but this control is applied in discrete steps and so is held constant for variable periods. It might be that this method could be effective if it were further extended to incorporate more control parameters.

The basic idea of the DYIDSG method [Ding *et al.* 1996] is to apply a sequence of small parameter variations so as to force the system at the next several iterates into the stable subspace associated with the unstable fixed point or unstable periodic orbit. It is therefore a natural extension of the classic OGY method. One essential ingredient of this method is to incorporate dependence of past parameter variations in the control scheme, an extension first described by Dressler and Nitsche [Dressler and Nitsche 1992] for the original OGY method. The derivation of the control law is rather complicated so we attempt only a summary below, see [Ding *et al.* 1996] for full explanation.

First assume that the original dynamical system can be described by a k-dimensional state variable X. In experimental studies of chaotic dynamical systems, especially high-dimensional ones, it is often the case that the only accessible information is a time series of some scalar function $x_n =$

$h(X(n))$. However, as shown by [Takens 1981], employing delay coordinates with a suitable delay time, the high-dimensional dynamics from the time series (x_n) can be reconstructed using the vector $\boldsymbol{z}_n$ assigned as

$$\boldsymbol{z}_n = \left(z_n^{(1)}, z_n^{(2)}, \ldots, z_n^{(m)}\right)^T \stackrel{\text{def}}{=} (x_{n-m+1}, x_{n-m+2}, \ldots, x_n)^T \tag{7.16}$$

where m is the dimension of the reconstructed state space. For suitably large m, $\boldsymbol{z}_n$ is generically a global one-to-one representation of the system variable $X(n)$.

Then the discrete map for $\boldsymbol{z}_n$ is

$$\boldsymbol{z}_{n+1} = R(\boldsymbol{z}_n, p_{n-m+1}, p_{n-m+2}, \ldots, p_n) \tag{7.17}$$

where R generally depends on all the parameter variations effective during the time interval $n-m+1 \leq t \leq n$ spanned by the delay vector $\boldsymbol{z}_t$ [Dressler and Nitsche 1992].

Assume there is an unstable fixed point $\overline{X}(\overline{p})$ in the original attractor for $p = \overline{p}$. [1] This is reflected in the delay coordinates by

$$\overline{\boldsymbol{z}}(\overline{p}) = R(\overline{\boldsymbol{z}}(\overline{p}), \overline{p}, \overline{p}, \ldots, \overline{p}) \tag{7.18}$$

where $\overline{\boldsymbol{z}}(\overline{p}) = [\overline{x}(\overline{p}), \overline{x}(\overline{p}), \ldots, \overline{x}(\overline{p})]^T$, T denoting the matrix transpose, and $\overline{x}(\overline{p}) = h(\overline{X}(\overline{p}))$.

The linear dynamics (without parameter changes) at the fixed point $\overline{\boldsymbol{z}}(\overline{p})$ can be described by the $m \times m$ Jacobian matrix

$$J = [D_{\boldsymbol{z}_n} R(\boldsymbol{z}_n, p_{n-m+1}, p_{n-m+2}, \ldots, p_n)]_{\boldsymbol{z}_n = \overline{\boldsymbol{z}}(\overline{p}), p_{n-m+1} = p_{n-m+2} = \cdots = p_n = \overline{p}} \tag{7.19}$$

where $D_{\boldsymbol{z}_n}$ denotes the Jacobian matrix operator of partial derivatives. We denote the partial derivatives due to the variations of the parameter (and the past values) by

$$\begin{aligned}
\boldsymbol{B}(m) &= \left[D_{p_{n-m+1}} R(\boldsymbol{z}_n, p_{n-m+1}, \ldots, p_n)\right]_{\boldsymbol{z}_n = \overline{\boldsymbol{z}}(\overline{p}), p_{n-m+1} = \cdots = p_n = \overline{p}}, \\
\boldsymbol{B}(m-1) &= \left[D_{p_{n-m+2}} R(\boldsymbol{z}_n, p_{n-m+1}, \ldots, p_n)\right]_{\boldsymbol{z}_n = \overline{\boldsymbol{z}}(\overline{p}), p_{n-m+1} = \cdots = p_n = \overline{p}}, \\
&\vdots \\
\boldsymbol{B}(1) &= [D_{p_n} R(\boldsymbol{z}_n, p_{n-m+1}, \ldots, p_n)]_{\boldsymbol{z}_n = \overline{\boldsymbol{z}}(\overline{p}), p_{n-m+1} = \cdots = p_n = \overline{p}}.
\end{aligned} \tag{7.20}$$

Therefore, the local linear flow near the unstable fixed point is described by

$$\begin{aligned}
\boldsymbol{z}_{n+1} - \overline{\boldsymbol{z}}(\overline{p}) &= J(\boldsymbol{z}_n - \overline{\boldsymbol{z}}(\overline{p})) + (p_{n-m+1} - \overline{p})\boldsymbol{B}(m) \\
&\quad + (p_{n-m+2} - \overline{p})\boldsymbol{B}(m-1) + \cdots + (p_n - \overline{p})\boldsymbol{B}(1).
\end{aligned} \tag{7.21}$$

Because we are using delay coordinates from (7.16) the next iterate $\boldsymbol{z}_{n+1} = (z_n^{(1)}, \ldots, z_n^{(m+1)})$ and we can therefore write (7.17) in component form as

$$\begin{aligned}
\boldsymbol{z}_{n+1} &= \left(z_{n+1}^{(1)}, z_{n+1}^{(2)}, \ldots, z_{n+1}^{(m-1)}, z_{n+1}^{(m)}\right)^T = \\
&\quad \left(z_n^{(2)}, z_n^{(3)}, \ldots, z_n^{(m)}, r(\boldsymbol{z}_n, p_{n-m+1}, p_{n-m+2}, \ldots, p_n)\right)^T
\end{aligned} \tag{7.22}$$

[1] Only a fixed point - 'period 1' orbit - is being discussed here. The technique can be extended and generalised for stabilising a period-N orbit [Ding *et al.* 1996].

where r is an appropriate function. We then see that most of the entries in the matrix J and the vectors $\boldsymbol{B}$ are zero. Explicitly,

$$J = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ a(m) & a(m-1) & a(m-2) & \cdots & a(1) \end{pmatrix}_{m\times m} \tag{7.23}$$

and

$$\boldsymbol{B}(i) = (0, \ldots, 0, b(i))^T_{1\times m} \quad (1 \le i \le m) \tag{7.24}$$

The estimation of $a(i)$ and $b(i)$ is discussed in the experiment.

Now assume that J in (7.23) has u unstable directions and s stable directions ($s + u = m$) with eigenvalues λ_i satisfying $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_u| > 1 > |\lambda_{u+1}| > |\lambda_{u+2}| > \cdots > |\lambda_m|$. Let $\boldsymbol{e}_i$ denote the corresponding eigenvectors. Then a possible control approach is to push the trajectory $\boldsymbol{z}_{n+1}$ into the stable subspace spanned by the stable directions $\boldsymbol{e}_i$, $(u + 1 \le i \le m)$, by suitable parameter variations according to (7.21). Instead the DYIDSG method expands the original state space as suggested in [So and Ott 1995], to a $(2m-1)$-dimensional space whose extended vectors are given by

$$\boldsymbol{Y}_n = \left(\boldsymbol{z}_n^T, p_{n-m+1}, p_{n-m+2}, \ldots, p_{n-1}\right)^T_{1\times(2m-1)} \tag{7.25}$$

which includes $\boldsymbol{z}_n$ and all the previous $m-1$ variations of the parameter p. The equivalent unstable fixed point in the extended system then becomes become

$$\overline{\boldsymbol{Y}} = \left(\overline{\boldsymbol{z}}(\overline{p})^T, \overline{p}, \overline{p}, \ldots, \overline{p}\right)^T_{1\times(2m-1)} \tag{7.26}$$

and the linear dynamics near the fixed point will be

$$\boldsymbol{Y}_{n+1} - \overline{\boldsymbol{Y}} = \tilde{J}(\boldsymbol{Y}_n - \overline{\boldsymbol{Y}}) + (p_n - \overline{p})\tilde{\boldsymbol{B}} \tag{7.27}$$

where

$$\tilde{J} = \begin{pmatrix} J & \boldsymbol{B}(m) & \boldsymbol{B}(m-1) & \boldsymbol{B}(m-2) & \cdots & \boldsymbol{B}(2) \\ \mathbf{0} & 0 & 1 & 0 & \cdots & 0 \\ \mathbf{0} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & 0 & 0 & 1 & \cdots & 1 \\ \mathbf{0} & 0 & 0 & 0 & \cdots & 0 \end{pmatrix}_{(2m-1)\times(2m-1)} \tag{7.28}$$

with $\mathbf{0}$ an m-dimensional row vector of 0's and

$$\tilde{\boldsymbol{B}} = \left(\boldsymbol{B}(1)^T, 0, \ldots, 0, 1\right)^T_{1\times(2m-1)}. \tag{7.29}$$

Now the eigenvalues of J are also eigenvalues of $\tilde{J}$ with corresponding eigenvectors

$$\boldsymbol{k}_i = \left(\boldsymbol{e}_i^T, 0, \ldots, 0, 0\right)^T \quad (1 \le i \le m) \tag{7.30}$$

in the $(2m-1)$-dimensional space. Suppose that the eigenvectors $\boldsymbol{e}_{u+1}, \ldots, \boldsymbol{e}_m$ of J corresponding to the stable subspace are linearly independent. Then we can extend the set of $s = m-u$ (stable) vectors $\boldsymbol{k}_{u+1}, \ldots, \boldsymbol{k}_m$ by adding vectors $\boldsymbol{k}_{m+1}, \ldots, \boldsymbol{k}_{2m-1}$ so as to construct a basis for the $(2m-1-u)$-dimensional stable subspace analogue $\boldsymbol{E}_s(\overline{\boldsymbol{Y}})$ of $\tilde{J}$. We note in passing that the m vectors of (7.30) are insufficient to span the $(2m-1)$-dimensional expanded state space but the additional $m-1$ linearly independent vectors required to span the full state space are fairly easily found, as shown in [Ding *et al.* 1996].

At this stage the idea of the DYIDSG control method becomes very similar to the original OGY method. Suppose that at time n the system trajectory falls in the neighbourhood of $\overline{\boldsymbol{Y}}$ called the control region. To stabilise the subsequent motion around this fixed point with u unstable directions, u successive small parameter perturbations $\delta p_n, \delta p_{n+1}, \ldots, \delta p_{n+(u-1)}$ in such a way that the deviation

$$\delta \boldsymbol{Y}_{n+u} = \boldsymbol{Y}_{n+u} - \overline{\boldsymbol{Y}} \tag{7.31}$$

lies entirely in the stable subspace $\boldsymbol{E}_s(\overline{\boldsymbol{Y}})$. For a short period the natural dynamics should then cause the orbit to relax onto the fixed point. The parameter can be set back to its nominal value $\overline{p}$ until further parameter adjustments are required.

Without going into details, we should note that both $\tilde{J}$ and $\tilde{J}^T$ have the same eigenvalue spectrum. In fact, the contravariant unstable eigenvectors $\boldsymbol{v}_i$ determined by

$$\tilde{J}^T \boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \tag{7.32}$$

for $1 \leq i \leq u$ have the property that they are orthogonal to the stable subspace $\boldsymbol{E}_s(\overline{\boldsymbol{Y}})$ of $\tilde{J}$, i.e. $\boldsymbol{v}_i^T \boldsymbol{k}_j = 0$ for $j = u+1, u+2, \ldots, m, m+1, \ldots, 2m-1$. Then the control perturbations required are simply obtained by solving

$$\begin{aligned} \boldsymbol{v}_1^T \delta \boldsymbol{Y}_{n+u} &= 0, \\ \boldsymbol{v}_2^T \delta \boldsymbol{Y}_{n+u} &= 0, \\ &\vdots \\ \boldsymbol{v}_u^T \delta \boldsymbol{Y}_{n+u} &= 0, \end{aligned} \tag{7.33}$$

for $p_n, p_{n+1}, \ldots, p_{n+(u-1)}$. Although the solution gives us the next u perturbation values together with p_n at time n, in practice, it is preferable to compute p_n at every iterate n to avoid the problem of system noise.

7.4.1 Experimental description and results

In our experiment using the DYIDSG technique the thruster G_z is chosen as the control parameter with the nominal value $G_z = \overline{G}_z = 0$. We adhere closely to the method described in the original paper.

First we need to choose a hyperplane to create a Poincaré section to reconstruct a discretised dynamics of this autonomous system. Unlike the original description of choosing a fixed point from the reconstructed dynamics on the Poincaré section for the control, we want to control the dynamics onto the same unstable periodic orbit used earlier in the continuous delayed feedback experiment. Therefore we have to choose a hyperplane which cuts the trajectory of this target orbit to obtain a corresponding unstable fixed point on this Poincaré section. The hyperplane $\omega_z = 0.3$ was chosen and the original unstable periodic orbit is approximately at $\boldsymbol{\xi} =$ (1.30484, 2.59193, 0.30000, 1.23980, 0.57523, 3.63385) on this hyperplane.

First we generated about 75000 data on this Poincaré section whenever the trajectory cut this hyperplane. These data were then used for our dynamic reconstruction and data analysis for the control. Two different strategies for the reconstruction of the dynamics were tried with this experiment: the *interspike intervals* reconstruction as introduced in [Ding *et al.* 1996] and the simple method of using a single system variable on the Poincaré section to reconstruct the dynamics.

The *interspike* interval is the time interval I_n, required for the trajectory to return to a point on the Poincaré section entering from the opposite halfspace to that from which it left. The interspike interval is then used as the basis for an embedding for the purpose of reconstructing the system dynamics. [Ding *et al.* 1996] demonstrated that I_n, the time between the $(n-1)$th and the nth crossings of the section, is uniquely determined by the original system dynamics and corresponds to a Poincaré map. With interspike sampling the target orbit then becomes the fixed point $(I_F, \ldots, I_F)$, where $I_F = 2.12$ is the estimated period of the orbit stabilised by the delayed feedback control.

We reconstructed the dynamics with an embedding of dimension 8. The first problem noticed was that estimating the local linear dynamics was difficult. For example, slightly increasing the local region, or equivalently including a few more 'close' data point for linear approximation, caused the resulting Jacobian to vary significantly, for example to have a different number of unstable eigenvectors.

We followed the DYIDSG method to control the satellite using initial conditions close to the unstable fixed point. The sensitivity vectors were estimated as described in the original paper. The result is shown in Figure 7.24, which shows the variation of the interspike interval (if control had been successful this should be approximately constant), and Figure 7.25, which shows the control signal G_z. The control signal rapidly becomes zero, but this is because after initial control is lost the system does not make a close return to the target state in the interspike interval space within the time period observed. For a close return in interspike embedding space to occur the trajectory in the original state space would have to make 8 successive close returns, which seems relatively unlikely. Unfortunately, a smaller embedding space does not seem to capture the original dynamics very well.

Figure 7.26 shows the evolution of the angular velocity ω_y against ω_x from the time control was initiated for approximately 10 interspike intervals. If control were successful this graph should be that of a simple closed curve.

As we can see, this experiment was not very successful although at the first 10 steps or so (See Figure 7.26), the dynamics was under control. Later, once control was lost, the system came back only in an occasional fashion, with control switched on for just a few steps. By reducing the maximum allowed perturbation, the control could only produce a 'trapped' periodic behaviour which was not the desired orbit. Having a larger allowed perturbation can lead to a 'bifurcation change' on the attractor and in many cases, the trajectory does not then come back to the Poincaré section for several thousand seconds. Many alternative settings were tried, such as changing the reconstruction dimension of the embedding, refining the approximation technique for the estimation of the Jacobian and the sensitivity vectors, and incorporating the fixed point tracking adaptive technique as reported in [Gluckman *et al.* 1997; Ding *et al.* 1997], but despite these efforts we were unable to achieve satisfactory control of the system.

We next tried to reconstruct the dynamics by observing the system variable ω_x on the Poincaré section. The value $\omega_x = 1.3048$ was then used as our fixed point value for constructing a delay coordinate with an embedding, corresponding to the original unstable periodic orbit. We could not achieve any successful control and similar problems arose. In comparison with the interspike interval technique, this approach seemed to be performing less well - but since both attempts were essentially

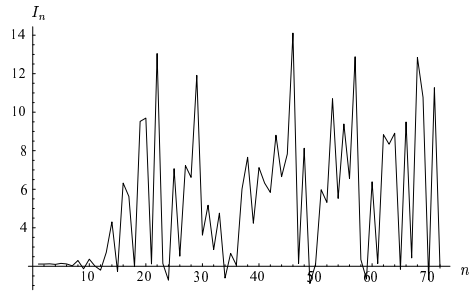


Figure 7.24: Interspike time interval I_n against n during DYIDSG control.

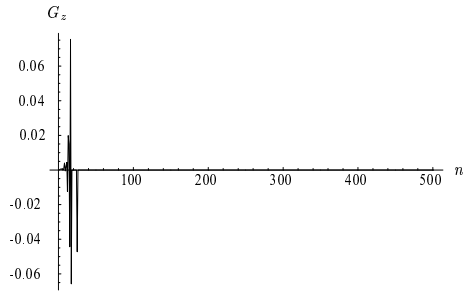


Figure 7.25: Changes G_z against n during DYIDSG control.

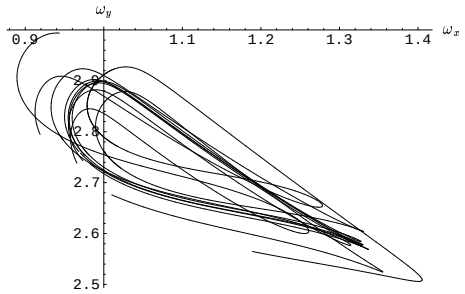


Figure 7.26: The dynamics gradually moves away from the desired orbit being under control perturbations.

unsuccessful this does not say a great deal.

Even without successful control, we have learnt that there are many problems which seem to affect this control method when applied to higher dimensional chaotic systems. The first problem is that for higher dimensional systems the local linear dynamics is not necessarily easy to estimate even with a reasonably large observed data set. Similarly, performing an accurate sensitivity analysis is difficult, due in part to the fact that it is not obvious how to determine the size of local region which defines which embedding vectors (from the observed data set) to include as 'close' points.

By examining the cross sections of the angular velocities in Figure 7.3 – 7.4 and Figure 7.7, especially in the region where our desired periodic orbit is situated, we can see how difficult it is to estimate the local dynamics (with suitable linearisation) using a finite set of data (the calculation of the sensitivity vectors was extremely time consuming). Examining Figure 7.7 closely shows the dynamics, corresponding to the next 10 iterates on the Poincaré section has discontinuities. This graph is plotted based on the information of the actual state on the Poincaré section and the return time, then the dynamics is numerically integrated from each initial condition for a period corresponding to the 'return time'. The gaps clearly indicate the problem of tracking the point at which the trajectory hits the hyperplane. Therefore, a combination of inaccurate estimates for the eigenvectors and sensitivity vectors, and the cumulative small inaccuracies of numerical integrations due to floating point errors for the chaotic dynamics, contributed to the difficulty of this experiment. In other words, the DYIDSG method seems to be highly sensitive to the accuracy of the eigenvectors and the sensitivity vectors. Intuitively, we could reasonably ask: how can we expect a single control parameter, fixed for the (variable) period of each control step, to effectively perturb a (say) 10-dimensional chaotic system onto a stable behaviour?

The DYIDSG method may be effective in controlling systems with dimension higher than two or three (which is the limit of the original OGY) by using a slightly higher dimensional embedding. How-

ever, this does not mean that the current form of the method can be used successfully on problems with a much higher dimension which, very often in practice, may require several parameter perturbations. In our example more thrusters could be used, but without a reformulation of the method there is no effective way to incorporate this fact into the control.

7.5 Summary of experiment results

Both the OJ and DYIDSG methods require considerable observation and calculations prior to implementing the control method. In practice, both methods require the location of an unstable fixed point from observational data. For higher dimensional systems (even without embedding) we are likely to need a very long sequence of observations in order to derive suitable delay and jump times and extract an unstable fixed point. If an embedding technique is used on a sequence of observations of a single system variable then we need to employ efficient techniques for the choice of delay and jump times. For both the OJ and DYIDSG methods we also need to perform a sensitivity analysis from observations of the system under small control parameter variations. All this analysis is required before control can be turned on. The real-time computational requirements of the OJ method are a fast pseudoinverse calculation, whereas the overhead of the DYIDSG is comparable to the OGY calculation, i.e. relatively low.

However, we have seen that DYIDSG control seems to be ineffective against our high-dimensional problem. This is due in part to the fact that only one control parameter perturbation is allowed, as opposed to using all three thrusters for the OJ method. Also in DYIDSG control the single thruster produces a fixed torque for a much longer period of time, where each time length depends on the return time for the trajectory to the Poincaré section. In contrast, the successful result of the Pyragas' delayed feedback method relies on the small, continuous variation of a single control thruster.

Numerous experiments on low dimensional systems have been reported using Pyragas' delayed feedback method [Cooper and Schöll 1995; Namajūnas *et al.* 1995; Qu *et al.* 1993] or its discrete equivalent, i.e. applied on a Poincaré section rather than in continuous time, see [Oliveira and Jones 1998; Tsui and Jones 1999b]. In the second case if the system is described by a map $\boldsymbol{\xi}_{n+1} = F(\boldsymbol{\xi}_n)$ and the controlled dynamics by

$$\boldsymbol{\xi}_{n+1} = \boldsymbol{\xi}_n + \Lambda \left(F(\boldsymbol{\xi}_n) - \boldsymbol{\xi}_n \right) \tag{7.34}$$

where Λ is a matrix defining the feedback constant, e.g. in the case of our satellite experiment if we just consider the angular velocities

$$\Lambda = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & k \end{pmatrix}, \tag{7.35}$$

then if the method is successful the control perturbation(s) approach zero. In this case the system dynamics stabilises onto an unstable fixed point of the original (uncontrolled) system. Thus one could also consider the technique as a method of finding unstable fixed points, provided one has already determined a suitable jump time. In essence this is the technique described in [Schmelcher and Diakonos 1997] for determining unstable fixed points.

In [Schmelcher and Diakonos 1997] the matrix Λ is required to be invertible with sufficiently small components. If d is the dimension of the system then the $d \times d$ matrix Λ is chosen so that $\Lambda = \lambda C$,

with $1 \gg \lambda > 0$, where in practice C is chosen from a set of $d!2^d$ matrices which correspond to special reflections in space and have components in $\{0, \pm 1\}$. Each matrix C stabilises a whole class of unstable periodic orbits, rather than a single fixed point.

Thus the method described in [Schmelcher and Diakonos 1997] to locate unstable periodic orbits can be viewed as a discrete equivalent of Pyragas' control method, but with much stronger restrictions on the matrices Λ. Since we know that Pyragas' method works in many cases without such restrictions it seems likely that what is important here is that the effect of Λ is contractive, or at least contractive on average. When the method of Pyragas stabilises the original system it might be because the Lyapunov exponents along the trajectory of the extended system are all negative, but in practice it is often the case that the trajectory converges rapidly to the fixed point and remains there.

Thus the important theoretical issue becomes for which systems and under what constraints on Λ can trajectories of the extended system be proved to either have negative Lyapunov exponents or stabilise to a fixed point. A satisfactory answer to this question would provide both a theoretical basis for a very effective control method and simultaneously offer an elegant method of locating unstable fixed points or periodic orbits, thereby extending our understanding of [Schmelcher and Diakonos 1997]. A possible approach is indicated in [Oliveira and Jones 1998; Tsui and Jones 1999b] as well as in the next chapter, which discusses the relevance of *probabilistic* local stability.

In this chapter we have compared three methods of controlling a six dimensional chaotic system. Both the OJ and the DYIDSG method require prior observations and computation, in particular the location of a suitable unstable fixed point, and both require a detailed sensitivity analysis. In each case the real-time computational overhead is reasonable but significant. The results for the OJ stabilisation were more satisfactory than those for the DYIDSG method (with which we were not able to achieve effective control).

We have also illustrated the Pyragas' method in its original continuous-time form using a single delayed feedback variable applied to the same system. The advantage of Pyragas' method is that no prior calculations of any kind are required and the real-time computational overhead is trivial. It seems remarkable that:

- The only information regarding the state of the system used in the control calculation is the angular velocity ω_z (i.e. five of the six possible state variables are ignored).
- Control is achieved using only one of the three thrusters.

7.6 Discussion

The results of the control of the chaotic satellite provide a better understanding of the problems of control of a higher dimensional chaotic system. The conclusion of these experiments have already been discussed in Section 7.5. In summary, these experiments suggest that a rigorous analysis of Pyragas's method is long overdue.

We have tried to provide an analysis of the delayed feedback control technique in Section 5.7.1 on a simple system. (We also attempted to provided a similar analysis on the satellite but the result is not conclusive and unsatisfactory, due to the fact that stabilised motion is complicated. The basin of attraction of this particular stabilised orbit analysed also is small and therefore, it was difficult to study how arbitrary orbits stabilise onto this orbit with random initial starting states.) Although preliminary, this has enabled us to better understand this control method and leads us into the idea

that the delayed feedback control technique may be the essential building ingredient for constructing a stimulus-response neural system based on chaos control. Having such delayed feedback connections within a biological neural system is not hard to imagine.

Chapter 8

An Artificial Chaotic Neural Stimulus-Response System

On the basis of studies of the olfactory bulb of a rabbit [Freeman 1991] Freeman has suggested an interesting model of recognition in biological neural systems via stabilisation of neural chaotic dynamics as discussed in Chapter 1. In this chapter we propose a chaotic iterative neural system which does produce stimulus-response behaviour similar to that observed by Freeman in a biological system. Our proposed system is based on the delayed feedback control idea which has proven very valuable in chaos control applications.

8.1 Construction strategy - an introduction

To construct such a neural system, we require some form of chaos control. There is now an extensive literature demonstrating experiments on controlling chaotic physical systems using the original chaos control techniques, such as the OGY method [Ott *et al.* 1990] or its similar variants as also described and investigated in earlier sections of this work. Many such methods require careful and systematic analysis of the chaotic dynamical behaviour, such as the OGY method, OJ method and the high-dimensional DYIDSG control, which is usually difficult and computationally expensive, before successful control can be achieved. Moreover, such control techniques are *external* to the system being controlled, whereas for a neural system to behave as described by Freeman the control should be *intrinsic* to the neural dynamics.

Therefore for constructing an iterative neural model, we implement a much simpler delayed feedback control, similar to Pyragas' original continuous delayed feedback [Pyragas 1992]. One of the attractions of this method is that it has a very low computational overhead, shown in Section 7.2 on the control of a chaotic satellite for the continuous case, and so is extremely easy to implement in hardware. It would also be very easy to implement in biological neural circuitry and so offers one plausible mechanism whereby such stabilisation might occur.

We use the chaotic neural network described in Section 6.1.2 for out chaotic iterative neural network. Delayed feedback is then introduced into the model and this provides a mechanism for stabilisation onto unstable periodic behaviours. The particular unstable periodic orbit which is stabilised depends quite strongly on the precise character of the applied stimulus. Thus the system can act as an associative memory in which the act of recognition corresponds to stabilising onto an unstable peri-

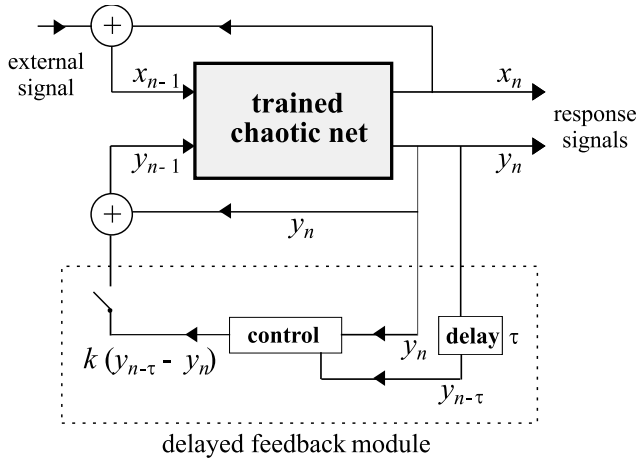


Figure 8.1: Delayed feedback on chaotic neural net.

odic orbit which is characteristic of the applied stimulus. The entire artificial system therefore will then exhibit an overall behaviour and response to stimulus which precisely parallels the biological neural behaviour observed by Freeman.

8.2 Delayed feedback applied to the chaotic neural net

A simple delayed feedback, similar to the Pyragas' delayed feedback, can be added to the chaotic neural net to control the chaotic behaviour with a careful choice of the parameters k and τ. The basic control setup of the neural model is shown in Figure 8.1. Here the trained chaotic feedforward neural net described earlier in Section 6.1.2 is now equipped with extra delayed feedback control circuitry, which is activated on presentation of an external stimulus. The delayed feedback is added to the state variable y_n to effect the control. External stimulation is performed by feeding signals into input line x_n of the network. Let FF be the feedforward network mapping such that $FF[(x_n, y_n)] = (x_{n+1}, y_{n+1})$ then the controlled system with external stimulation s_n at time n is described by

$$(x_{n+1}, y_{n+1}) = FF[(x_n + s_n, y_n + p_n)] \tag{8.1}$$

where $p_n = k(y_{n-\tau} - y_n)$ is the delayed feedback control signal.

After some initial investigation we fixed $k = 0.5$ and $\tau = 6$ for the experiments. These values stabilised the system with control switched on but with no external stimulus present. Other values of k and τ can also stabilise the system (x_n, y_n) successfully.

We imagine that the presence of an external stimulus excites (activates) the control circuitry which is otherwise inhibited. Thus to achieve a stabilised dynamical regime in response to a stimulus the control is switched on at the same time as the external signal is fed into the input line x_n. By varying the external signal in small steps and holding the new setting fixed long enough for the system to stabilise we can observe the response of the network to small changes in stimulus.

In Figure 8.2 the system is iterated for 100 cycles to eliminate any initial transients. Next an external constant stimulus $s_n = s$ is applied for 400 steps. In Figure 8.2 the stimulus is varied in steps of 0.025 over the interval [0, 1] every 400 network iterations. We can see that the system exhibits a fairly 'smooth' transition of stabilised behaviour from one stimulus to the next. For the most part in this case the response is a 1-period behaviour but a 2-period behaviour is also exhibited after the

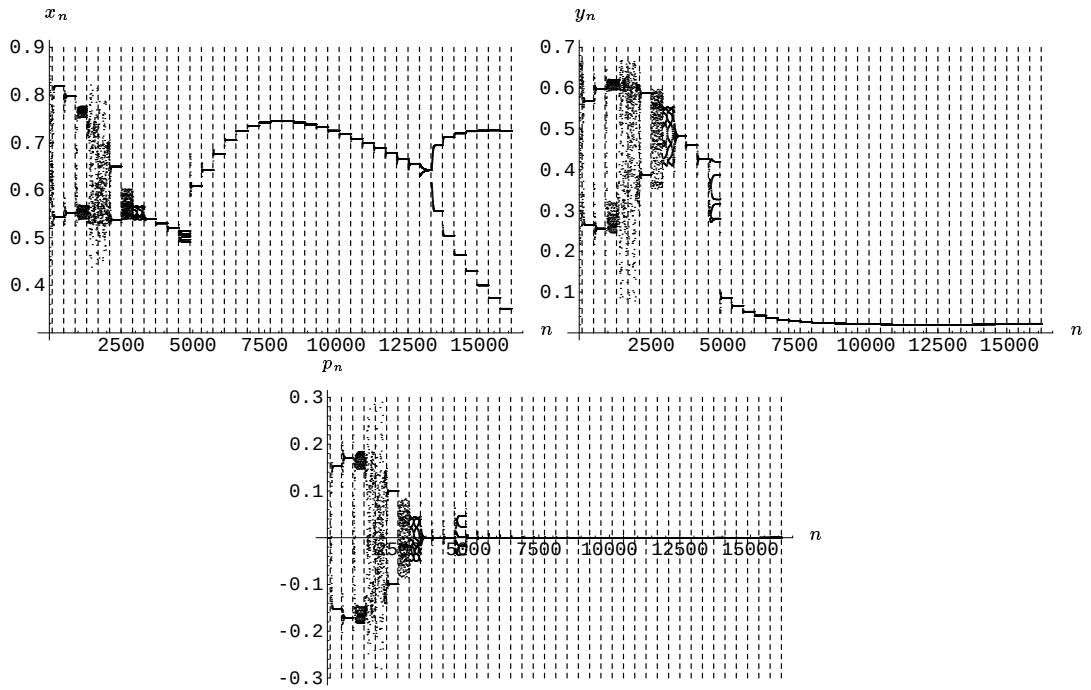


Figure 8.2: Responses of x_n and y_n and the size of delayed feedback control signal p_n due to external constant stimulation of $[0, 1]$ varying in steps of 0.025. The stimulus changes at 400 iteration steps after an initial 100 iterations to eliminate transients. The control parameters were $k = 0.5$ and $\tau = 6$.

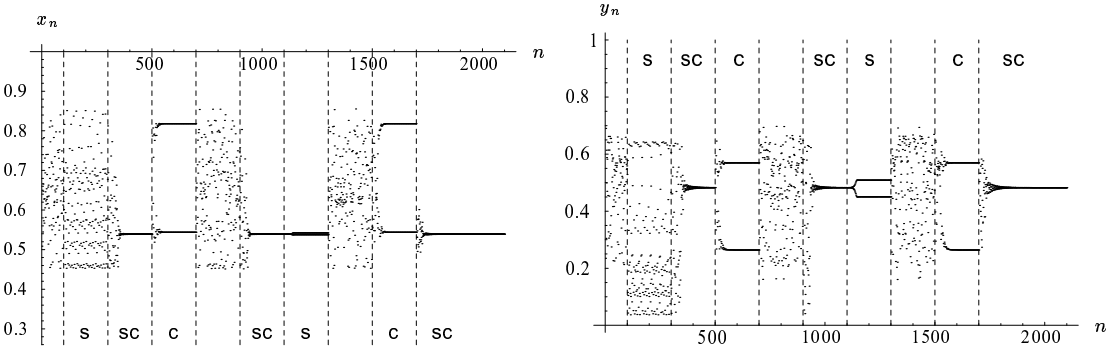


Figure 8.3: Responses of x_n and y_n to presentation and removal of stimulus 0.2 with and without control. Intervals labelled 's' indicate the presence of the stimulus, intervals labelled 'c' indicate control is switched on, a label 'sc' indicates both, and no label indicates no stimulus and no control. The particular regime is changed every 200 iterations after 100 iterations have been allowed for transient removal. $k = 0.5$ and $\tau = 6$.

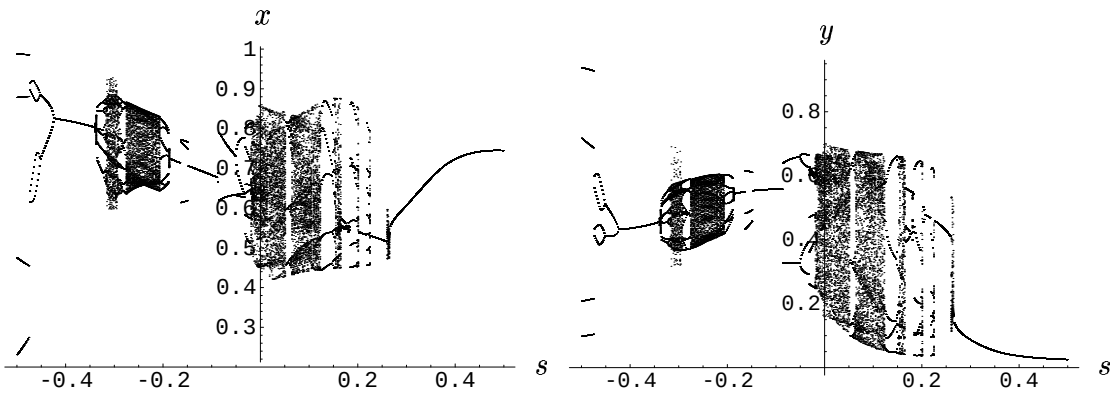


Figure 8.4: Bifurcation diagrams for the outputs x_{n+1} and y_{n+1} using an external variable s added to the input x_n.

strength of the external signal crosses a threshold at around 0.8 which is therefore a *bifurcation point*. For a stimulus $s_n = s$ with $s > 0.2$ the delayed feedback control signal quickly becomes small, which indicates that the system has stabilised onto one of its own unstable periodic behaviours. However, for a stimulus $s_n = s$ with $s < 0.2$ a *large* feedback control signal p_n often seems to create some new periodic behaviour.

We can study the response of the system as the stimulus 0.2 is applied and removed and as control is turned on and off. This is shown in Figure 8.3. In general terms the system stabilises after about 50 iterations. If the stimulus is applied without control the dynamical regime seems not to correspond to an unstable periodic behaviour of the original network, but with control switched on the dynamics quickly stabilises to a 1-period corresponding to an unstable periodic behaviour of the iterated network.

Note that in the transition sc $\rightarrow$ s of Figure 8.3, in which control is removed but the stimulus remains, surprisingly the system shifts from a 1-period to a 2-period, rather than reverting to the more chaotic regime illustrated in the first 400 step interval, where the same stimulus without control proved unable to stabilise the system.

In some cases, the external stimulation signal is enough to stabilise the system without switching on the control module. The explanation of this might be that when such an external signal is strong enough, or it is a particular kind of signal, it may shift the underlying dynamics from a chaotic region into a periodic region in the bifurcation diagrams, as shown in Figure 8.4. This figure originally appeared in [Tsui and Jones 1997] which studied the same feedforward neural network.

Apart from a constant external stimulation signal applied to one of the inputs, other forms of s_n can also be used. Low period square waves can also result in stabilised periodic responses as shown in Figure 8.5.

A completely different way of applying a stimulus was suggested in [Hoff 1994]. The stimulus can be applied directly to the control variable k. In this way different behaviours can be achieved by using the external signal s_n to directly modify k. Some results of this type of control applied to our system are illustrated in Figure 8.6.

These experiments are merely illustrative and many variations are possible. For example, delayed feedback control could equally be applied to several (or all) of the network outputs. With the same τ and multiple feedbacks it should be easier to achieve stabilisation compared to the case where feedback is applied to just one variable. However, if delayed feedback on different network outputs also had differing τ then the outcome is less predictable. There remain many possibilities for exploring this

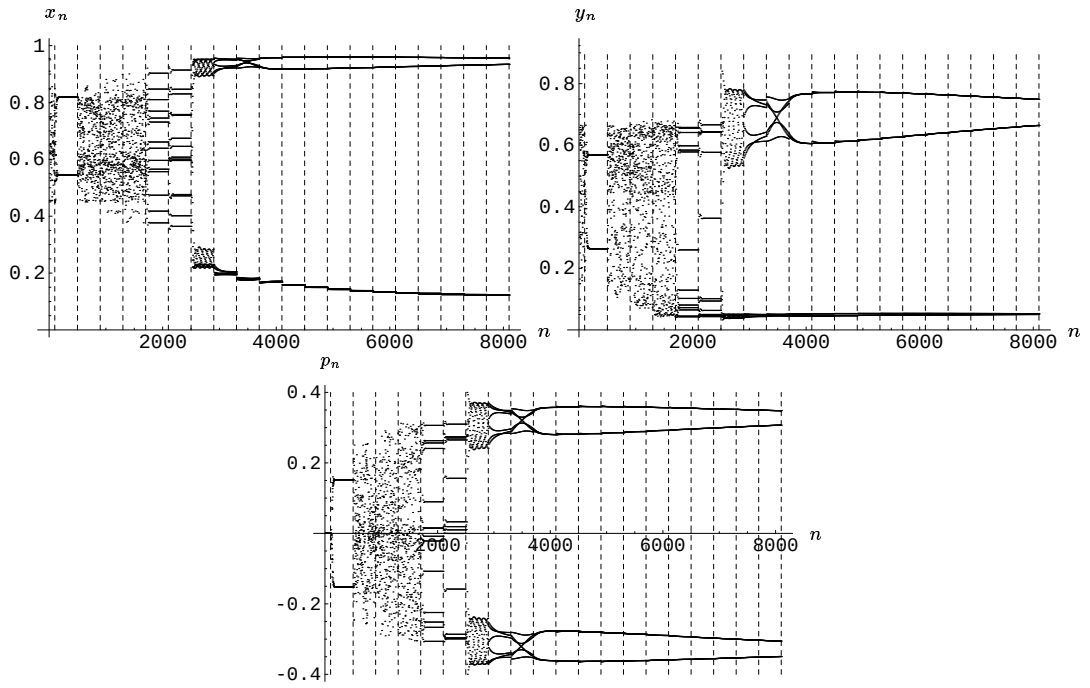


Figure 8.5: Responses of x_n and y_n and the size of delayed feedback control signal p_n due to the periodic stimulation $s_n = \{j, 0, j, 0, \dots\}$ of strength j from 0 to 1 in steps of 0.05. The stimulus changes at 400 iteration steps after an initial 100 iterations to eliminate transients. The control parameters were $k = 0.5$ and $\tau = 6$.

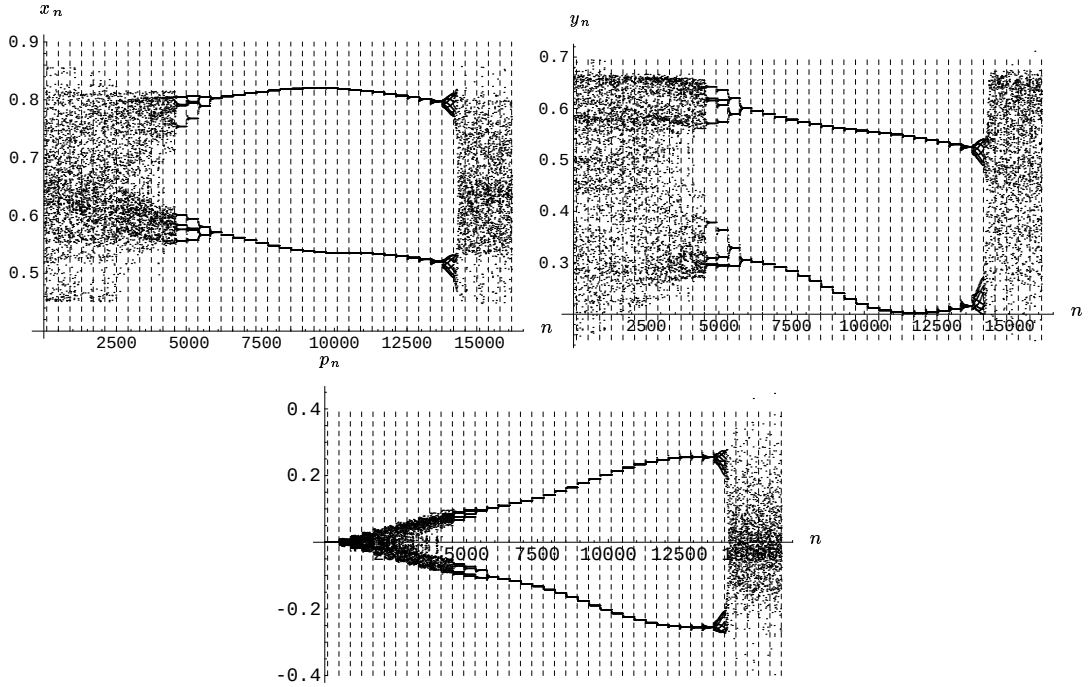


Figure 8.6: Responses of x_n and y_n and the size of delayed feedback control signal p_n due to the external signal s_n added to the value k from -0.5 to 0.5 in steps of 0.025. The stimulus changes every 400 network iterations (after 100 initial iterations with no stimulus and no control. $k = 0.5$ and $\tau = 6$).

type of neural model.

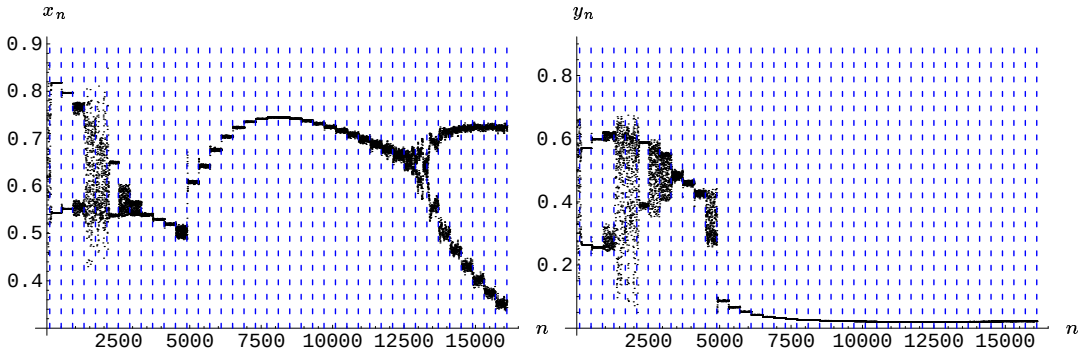


Figure 8.7: The response of the system to noise. The stimulus s_n is replaced by $s_n r$ at each iteration step, where r is Gaussian noise with mean 1 and variance Var $(r) = 0.005$. The stimulus changes at 400 iteration steps after an initial 100 iterations to eliminate transients. The control parameters were $k = 0.5$ and $\tau = 6$.

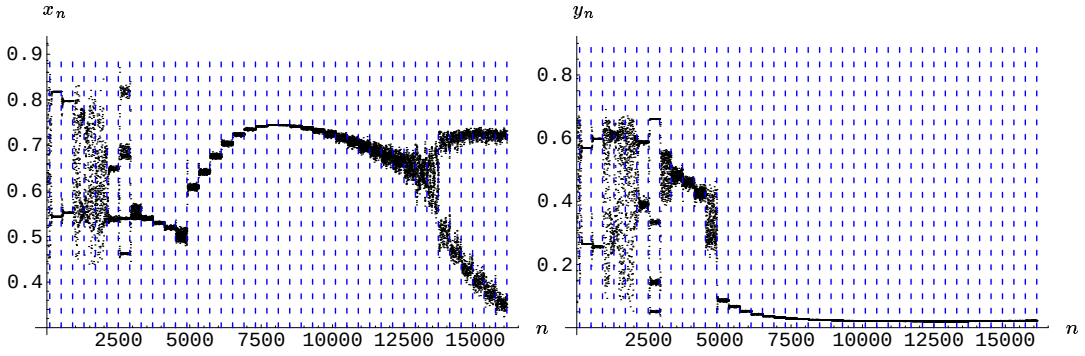


Figure 8.8: The response of the system to noise. The stimulus s_n is replaced by $s_n r$ at each iteration step, where r is Gaussian noise with mean 1 and variance Var $(r) = 0.01$. The stimulus changes at 400 iteration steps after an initial 100 iterations to eliminate transients. The control parameters were $k = 0.5$ and $\tau = 6$.

We also investigated the response of the system when sensory input was perturbed by stochastic noise. The stimulus was perturbed at each iteration step by multiplying it by Gaussian noise with a mean of 1 and a variance σ, where σ varied from $\sigma = 0$ to $\sigma = 0.1$. The response was surprisingly robust as illustrated in Figures 8.7 – 8.9. These results should be compared with the non-noisy case of Figure 8.2. The noisy dynamics remain essentially unchanged, although as one might expect the attractor becomes progressively 'blurred' as the noise level increases.

8.3 Local stability analysis

As we have seen earlier, little theoretical analysis is available for the Pyragas method of continuous delayed feedback control, let alone for the discrete form of the method used here. However, a discrete version of a variation of Pyragas' method has already successfully been applied to the synchronisation of two identical iterative chaotic maps in [Oliveira and Jones 1998]. The version used there for *synchronisation* is similar to but not identical to the method used here for *stabilisation* and that paper contained a suggestive account of the local stability properties. We gave a similar analysis for a continuous system in Section 5.7.1.

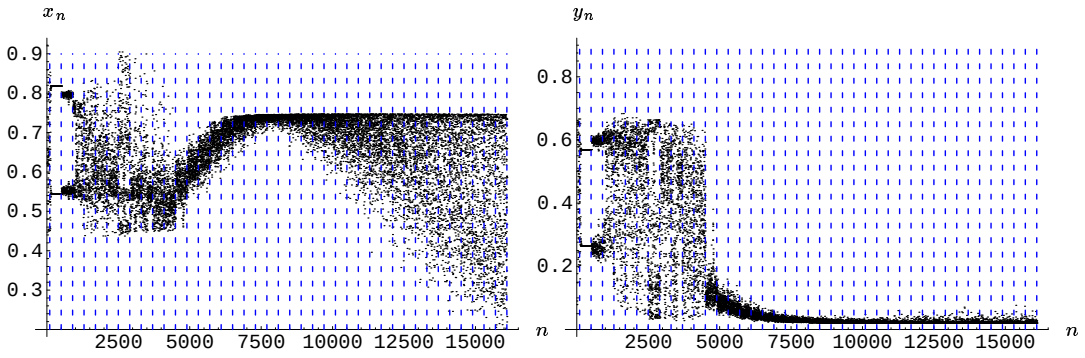


Figure 8.9: The response of the system to noise. The stimulus s_n is replaced by $s_n r$ at each iteration step, where r is Gaussian noise with mean 1 and variance Var $(r) = 0.1$. The stimulus changes at 400 iteration steps after an initial 100 iterations to eliminate transients. The control parameters were $k = 0.5$ and $\tau = 6$.

We next try to provide a similar empirical analysis for the method of stabilisation proposed here in the case where no external stimulus is present. First, we note the stabilised state when control is switched on with $k = 0.5$, $\tau = 6$ and no external stimulus is applied. This gives a 2-period controlled behaviour $\{\xi_{F1}, \xi_{F2}\} = \{$(0.81808, 0.569261), (0.543838, 0.264166)$\}$.

If we again define a measure of contraction

$$\mu_n = \frac{\min\left(|\xi_{n+1} - \xi_{F1}|, |\xi_{n+1} - \xi_{F2}|\right)}{\min\left(|\xi_n - \xi_{F1}|, |\xi_n - \xi_{F2}|\right)} \tag{8.2}$$

towards $\{\xi_{F1}, \xi_{F2}\}$ from step n to step $n+1$ then μ_n depends on the eigenvalues of the Jacobian of the associated four dimensional system $\{\xi_n, \xi_{n+1}\}$ in the vicinity of $\{\xi_{F1}, \xi_{F2}\}$ and these (although bounded) can be much larger than 1. Thus it is simply not true that with this control method the system will monotonically approach the unstable periodic behaviour. However, if we examine the effects of control after several iterations we find that the *probability* that the cumulative net contraction becomes small is very large.

To establish this we generate a random initial point ξ_0 and iterate the controlled system. At the n^{th} iteration we define

$$\rho_n = \frac{\min\left(|\xi_n - \xi_{F1}|, |\xi_n - \xi_{F2}|\right)}{\min\left(|\xi_0 - \xi_{F1}|, |\xi_0 - \xi_{F2}|\right)}. \tag{8.3}$$

The quantity ρ_n gives us an measure of the extent to which after n iterations with control the system has *contracted* towards the unstable 2-period.

By showing that ρ_n becomes small with high probability, i.e. that $\rho_n \to 0$ as $n \to \infty$, where the convergence is in probability, we can establish that the method is *probabilistically locally stable*.

We repeated the calculation of ρ_n for 1000 different initial starting points and $n \leq 80$ and created histograms showing the frequency of ρ_n against the value. These results are shown in Figure 8.10. These histograms suggest that $\forall \epsilon > 0$,

$$P[\rho_n < \epsilon] \to 1 \tag{8.4}$$

as $n \to \infty$. Thus the system without stimulus is probabilistically locally stable.

The application of an external stimulus basically modifies the system dynamics by *shifting* the dynamic behaviour along the bifurcation diagrams as mentioned earlier. Many new chaotic and non-chaotic behaviours are produced by the neural system which are different from its initial built-in dynamics without stimulation. Thus the delayed feedback control seems to act as a supporting tool for

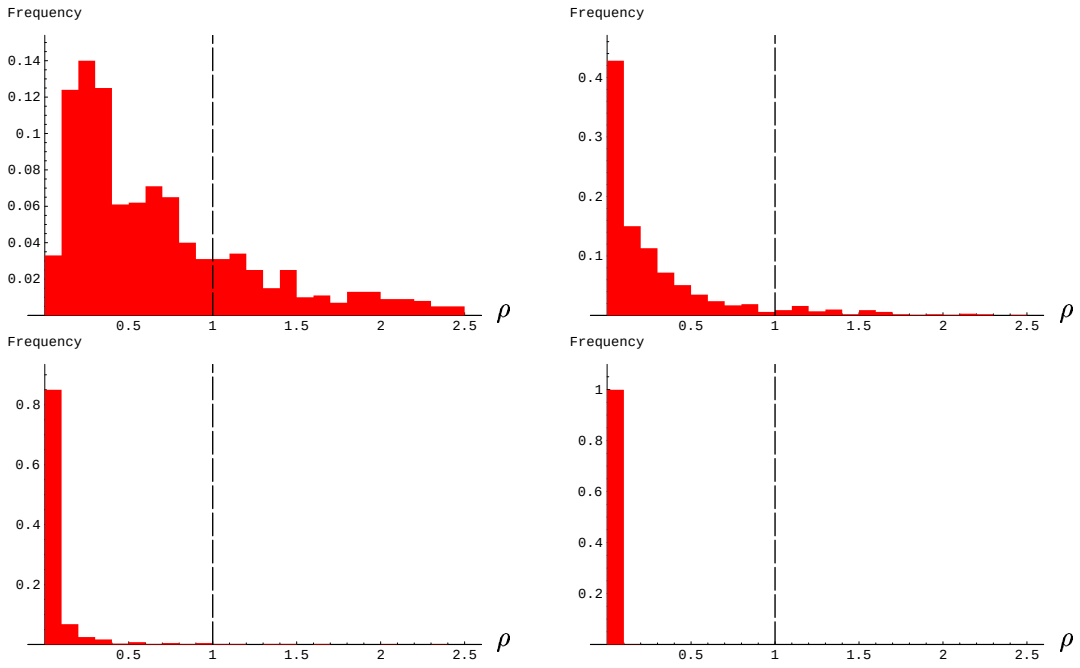


Figure 8.10: Histograms of ρ_n at $n = 20$ (top left), 40 (top right), 60 (bottom left), 80 (bottom right) of 1000 random initial starting points for control $k = 0.5$ and $\tau = 6$.

stabilising the system into periodic states. Although there is insufficient theoretical explanation for the dynamical behaviour of our neural system, the above heuristic analysis seems to fit very well with the observed simulation results.

8.4 Generic stimulus-response neural model

In fact, we can generalise the model shown in Figure 8.1. The chaotic feedforward network can be trained and modelled on a known chaotic time series, using our *irregular* embedding technique with the embedding found by using the Gamma test technique shown in Section 4.2.3.

A generic scheme of such stimulus-response recurrent network is shown in Figure 8.11. The single output of the network feeds back into inputs using delay buffers accordingly to a *suitable* embedding – i.e. should contain enough information for predicting the next system state. A multiple of delayed feedbacks can be used for each input of this recurrent neural network as control lines (based on the idea from Pyragas' delayed feedback control). The control module shown in Figure 8.11 is similar to the one as shown in Figure 8.1 and the control perturbation for the i^{th} input at each iterate n is

$$k_i(x_i(n-i) - x_i(n-i-\tau)) \tag{8.5}$$

where k_i and τ are the usual parameters as in the Pyragas delayed feedback control. Each control perturbation signal should be switched on and off in the control module as in the earlier example. In the diagram, τ is the same for each control perturbation but of course, we could set τ to be different on each control line. External stimulus to the network can be applied to the controlled inputs as shown in the diagram. The control module should switch on automatically and simultaneously whenever there is an external simulation. Variations of stimulation, such as on the control delayed feedback lines, as shown earlier may also be used.

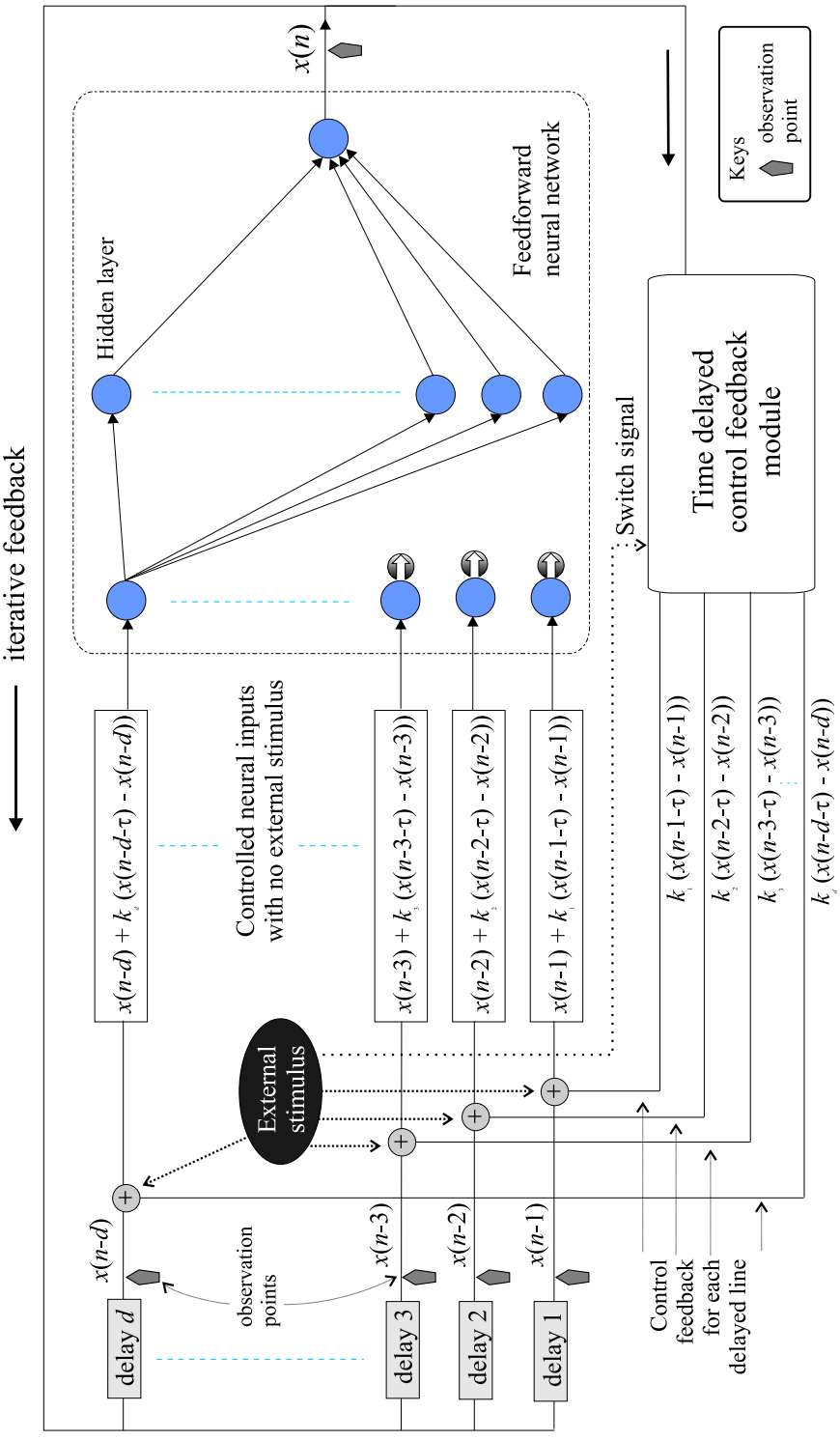


Figure 8.11: A general scheme for constructing a stimulus-response chaotic recurrent neural network: the chaotic "delayed" network is trained on suitable input-output data constructed from a chaotic time series; a delayed feedback control is applied to each input line; entry points for external stimulus are suggested, with a switch signal to activate the control module during external stimulation; signals on the delay lines or output can be observed at the "observation points".

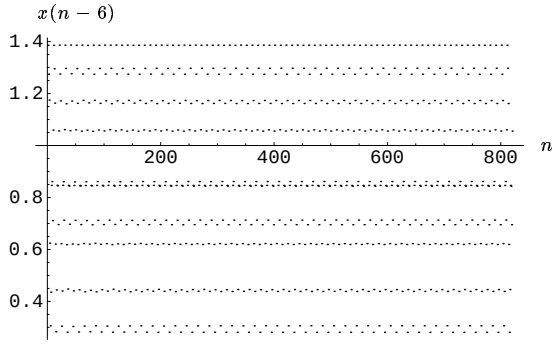


Figure 8.12: Response signal on $x(n-6)$ with control signal activated on $x(n-6)$ using $k = 5$ and $\tau = 0.414144$ and without external stimulation.

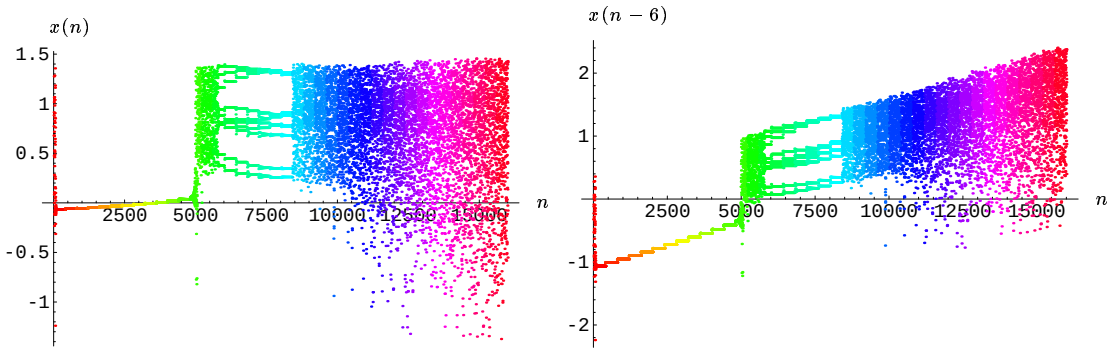


Figure 8.13: Response signals on network output $x(n)$ and on observation point (on delay line) $x(n-6)$, with control signal activated on $x(n-6)$ using $k = 5$ and $\tau = 0.414144$ and with constant external stimulation s_n added to $x(n-5)$, where s_n varies from -1 to 1 in steps of 0.05 at each 400 iterative steps (indicated by the change of Hue of the plot points) after initial 20 transient steps.

8.4.1 Examples

Using the neural network trained on the Mackey-Glass time series as in the example in Section 4.2.3, we can construct a stimulus-response neural system based on the generic model described. There follows a gallery of different responses of the system using different settings of controls and external stimulation. The response signals of the system can be observed at the output $x(n)$ of the feedforward neural network module or the "observation points" on the delay lines $x(n-1), \ldots, x(n-d)$, as indicated in Figure 8.11. Due to the increased complexity of this neural system, of course, not all possible settings are tried and presented.

We use $k = 5$ and $\tau = 0.414144$ for our control parameters on all the possible feedback control lines. The control is applied to the delayed feedback line $x(n-6)$. Without any external stimulation and using only a single control delayed feedback, the network quickly produces a periodic response as shown in Figure 8.12.

Figure 8.13 shows the signals on the output $x(n)$ of the feedforward neural network module and $x(n-6)$ (observed at the observation point on the delay line $x(n-6)$) with the control signal on $x(n-6)$ using $k = 5$ and $\tau = 0.414144$ and with external stimulation s_n added to $x(n-5)$. This simple combination using a single control line plus a stimulation on the delay line already produces a variety of dynamical behaviours, but when the external stimulus is high, the system appears to be chaotic.

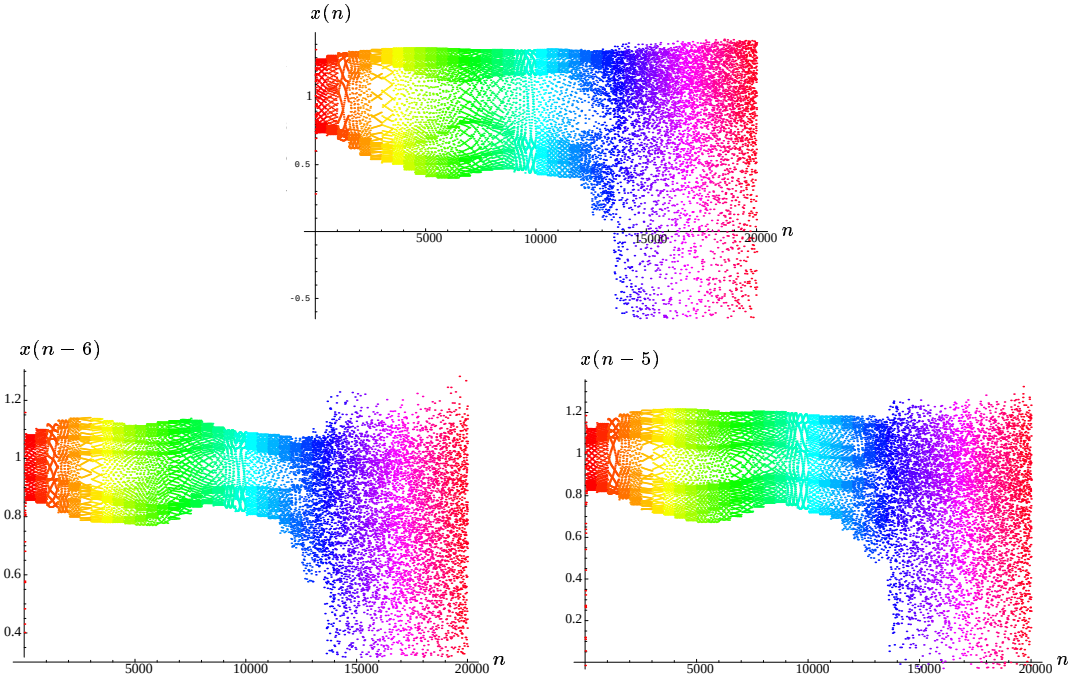


Figure 8.14: Response signals at network output $x(n)$ and at the observation points on $x(n-6)$ and $x(n-5)$ delay lines with control signal activated on all delay lines using $k = 5$ and $\tau = 0.414144$ with external stimulation s_n added to $x(n-6)$, where s_n varies from -1 to 1 in steps of 0.05 changing at every 500 iterative steps (indicated by the change of Hue of the plot points) after initial 20 transient steps.

The precise results depend on which delayed feedback control lines are activated. Using the same multiple control settings for all delay lines, the system can be stimulated on the delay line $x(n-6)$ (just after the delay buffer) by a constant external signal s_n, where s_n varies from -1 to 1 in steps of 0.05 at every 500 iterations after the first 20 steps of transient. The result of the signals on $x(n)$ and at the observation points on the $x(n-6)$ and $x(n-5)$ delay lines are shown in Figure 8.14 and exhibit highly periodic stabilised behaviour for some stimuli. In some cases, some response signals seem to be quasi-periodic. Figure 8.15 illustrates another example using two different external stimulation signals at $x(n-5)$ and $x(n-6)$ and achieving a wide variety of periodic responses.

Even without external stimulation, we see quite significant modifications of the dynamics when different configurations of delayed feedback control lines are activated, using $k = 5$ and $\tau = 0.414144$ for each control lines. Figure 8.16 illustrates, after first 20 transient iterations without any control, the response signals of the network due to a sequence of different delayed feedback settings which change at every 1200 iterative steps. Only particular ranges of multiple delayed feedbacks can stabilise the chaotic system into a high periodic response.

In general, using the generic model we can produce different types of network exhibiting different types of chaotic attractors and reproduce a rich variety of stabilised dynamical behaviours using only suitable delayed feedback control and external stimulation of the network. The resulting behaviour is comparable to the behaviour observed by Freeman as noted several times previously. This section has provided only a glimpse of the possibilities inherent in using these models.

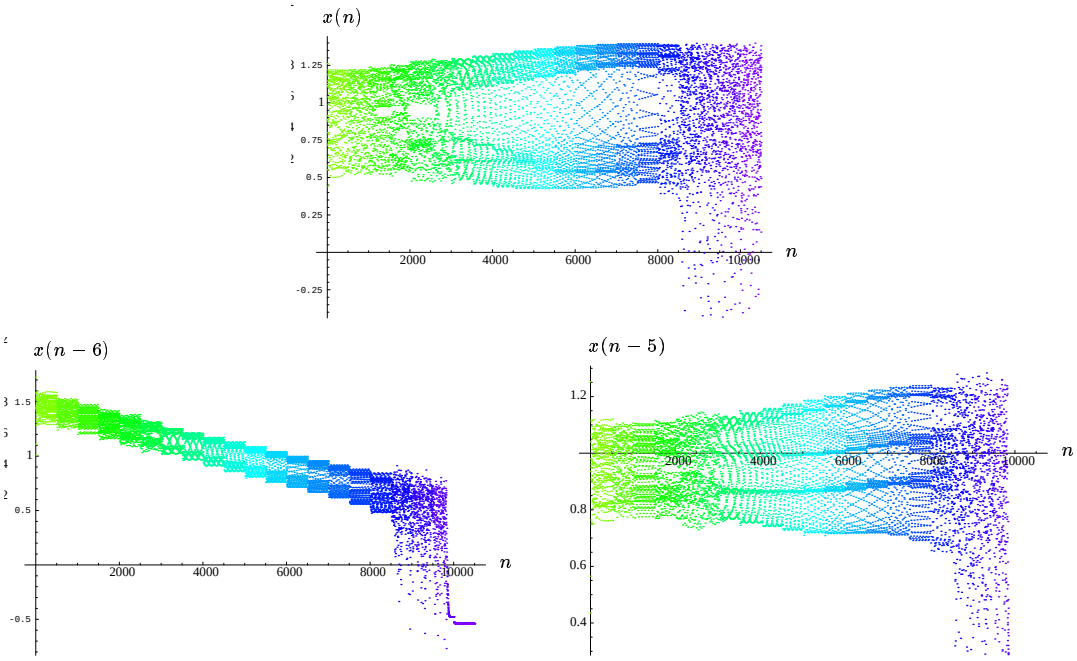


Figure 8.15: Response signals on the network output $x(n)$ and at the observation points on $x(n-6)$ and $x(n-5)$ delay lines with control signal activated on all delay lines using $k = 5$ and $\tau = 0.414144$ and with external stimulation, $s_n^{(1)}$ added to $x(n-6)$, where s_n varies from -0.5 to 0.5 in increasing steps of 0.05, and $s_n^{(2)}$ added to $x(n-5)$, where s_n varies from 0.5 to -0.5 in decreasing steps of 0.05, changing at every 500 iterative steps (indicated by the change of Hue of the plot points) after initial 20 transient steps.

8.5 Discussion

We have shown how a conventional artificial feedforward neural network equipped with delayed feedback control can simulate the type of rest behaviour and response to stimuli observed by Freeman in the olfactory bulb of the rabbit. The system is in effect an associative memory in which the act of recognition corresponds to the stabilisation of the system onto an unstable periodic orbit characteristic of the applied stimulus.

If the dynamics are chaotic then unstable periodic orbits are dense on the chaotic attractor and there are infinitely many of them. Thus such an associative memory for which the computations are performed to an *arbitrary* precision could in principle accommodate infinitely many memories; at any rate such a system is not subject to the conventional Hopfield upper bound of $0.15n$, where n is the number of neurons [Amit *et al.* 1987]. Of course, for the Hopfield net the situation is rather different. In the Hopfield model memories are associated with specified (preferably uncorrelated) point attractors, whereas in the present model memories are associated with unstable *periodic* behaviours which cannot be specified *ab initio*. This introduces the possibility of responding to stimuli over varying *time scales*.

The experiments here were based on high precision digital simulations. In a low arithmetical precision analog implementation it is possible that much of the rich variety of dynamical behaviour would be lost.

Nevertheless, the model has a certain compelling simplicity which is suggestive. The responses described are *intrinsic* to the network model and control is not artificially applied from outside the network itself. The method of delayed feedback control is simple to apply in hardware and feasible in

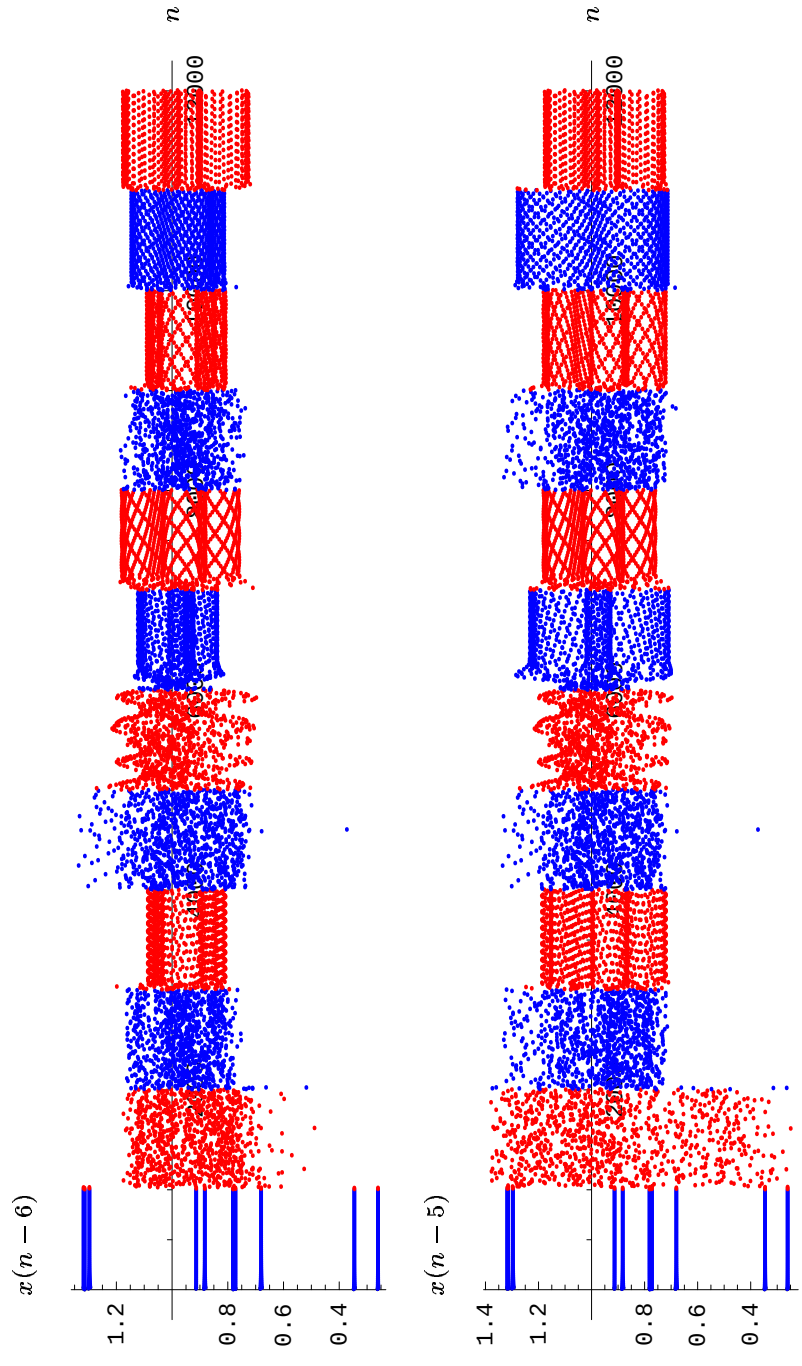


Figure 8.16: After the first 20 transient iterations without any control, this graphs show the signals at the observation points on the delayed lines $x(n-6)$ and $x(n-5)$ as delayed feedback is activated on the following sets of control lines: $\{\{x(n-6)\} \rightarrow \{x(n-6), x(n-5)\} \rightarrow \{x(n-6), x(n-5), x(n-4)\} \rightarrow \{x(n-6), x(n-5), x(n-4), x(n-3)\} \rightarrow \{x(n-6), x(n-4)\} \rightarrow \{x(n-6), x(n-3)\} \rightarrow \{x(n-5), x(n-3)\} \rightarrow \{x(n-4), x(n-3)\} \rightarrow \{x(n-5), x(n-4)\} \rightarrow \{x(n-5), x(n-4), x(n-3)\} \rightarrow \{x(n-6), x(n-5), x(n-3)\} \rightarrow \{x(n-6), x(n-4), x(n-3)\}\}$, which change at every 1200 iterative steps as indicated by the change of plot colour.

biological neural circuitry.

As with the many applications of the method of Pyragas to control more conventional chaotic dynamics our approach lacks a full formal analysis. However, we have investigated the local stability properties of the method applied to the particular model described here and have concluded that although control is not stable in the conventional sense it is nevertheless *probabilistically locally stable*.

The experiments described raise several interesting issues. An investigation of essentially the same model could be performed with delayed differential equations using a more biologically accurate description of the neurons. As in [Tsui and Jones 1999a] and Chapter 7, we describe delayed feedback control applied to the stabilisation of a six dimensional *smooth* dynamical system and this illustrates that the ideas described here could quite probably be applied successfully to a similar model based on differential equations.

Another question which naturally arises is whether 'the basin of attraction' of a particular unstable periodic orbit, which has emerged as the response to a specific stimulus, could be 'widened' by repeated presentations using some form of weight adjustment based on Hebbian learning. The critical aspect to investigate here would be whether this could be done without destroying the essential underlying chaotic dynamics or other conditioned responses.

The periodic responses exhibited are common in coupled oscillator models (e.g. [Stewart 1992]) which are very different from the model described here. It is therefore interesting to note that, by incorporating delayed feedback, periodic neural responses can be achieved with an essentially conventional feedforward neural network model without the introduction of an oscillator neuron.

Chapter 9

Conclusions

During this work, many diverse and interesting topics have been investigated:

- Smooth data modelling implementing techniques based on various paradigms, including the study of feedforward artificial neural network (FANN) modelling approaches and other aids for improvement in modelling, e.g. the Gamma test and embeddings;
- A study of chaos and the applications of chaos control on a variety of chaotic systems, including controlling simple chaotic neural systems and the high dimensional satellite attitude control problem;
- An experimental chaotic artificial neural system under external stimulation together with the effects of delayed feedback control.

All of these ideas have led us to the accomplishment of the original goal, of constructing a chaotic artificial neural network capturing the stimulus-response behaviour observed by Freeman in the biological network of neurons in the olfactory bulb of a rabbit.

This chapter, beside giving a concise summary of the work achieved so far, also revisits some of the essential ideas and techniques discovered during the whole investigation. However, due to the diversity of the topics studied, many unverified methods and thought-provoking concepts suggested by this research are also highlighted for possible future investigation.

9.1 Achievements

To construct our chaotic stimulus-response model we have had to examine a number of diverse ideas.

Smooth data modelling

We first studied the ability of a feedforward neural network (FANN) to model an arbitrary smooth function from inputs to outputs. In principle such networks can play the role of universal approximators but in practice finding the architecture, weights and thresholds, and the number of training data required, so that the network will model a given function to a given degree of precision, is not such a simple process.

With this in mind we examined the construction of FANNs using Lapedes' graphical approach. This graphical approach, originally used simply to explain *why* neural networks could act as universal

approximators, suggested a technique for constructing neural networks to approximate a surface by combining a series of neural modules (generating 'sigmoidal surfaces').

We then investigated the *Gamma test*, which emerged as an invaluable tool for data modelling in this research. Essentially, the use of the Gamma test together with the Lapedes' recipe for surface construction enables us to estimate the complexity of the neural architecture required directly from the data. We have exploited the possibility of using the slope value returned from the Gamma test applied to the the training data, to estimate the minimal architecture necessary for modelling the input-output surface and to estimate the number of training data required to give a suitable model.

In fact, this idea (joint investigation) eventually led to N. Končar's Metabackpropagation algorithm [Končar 1997].

Next we examined various modelling methods using 'local' information derived from the training data in the vicinity of the query point. We first discussed the virtue of the kd-tree data structure, which allows fast query times for locating the near neighbours (in input space) of any point. In this way prediction can be simplified by modelling using a small subset of local data, as opposed to a global modelling technique.

Borrowing ideas from computational geometry we examined the possibility of using Delaunay triangulation as an aid to local reconstruction of a surface in the vicinity of a query point. This led to the prediction method we refer to as LDT. The Delaunay triangulation of the data can be calculated via a convex hull construction technique. Our implementation is based on the convex hull calculation method called Qhull. This technique brought many new and interesting ideas from computational geometry into this research, although ultimately the approach was discarded in favour of local linear regression.

We next devised the Gamma-minimum-predictor (GMP) based on the Gamma test. This approach was based on the idea that given an unseen query point $\boldsymbol{x}$ the associated output value y should be chosen so as to satisfy the condition: when ($\boldsymbol{x}$, y) is added to the data set the resulting $|\bar{\Gamma}|$ value should be minimised. This criterion can be used to analytically determine the required value of y and it emerges that this value can be computed reasonably quickly.

We then examined a simple but effective prediction technique we called *local linear regression* (LLR). This is done by performing least squares fit on the local data near the query point. A series of experiments were performed comparing the performance of various modelling techniques. In many situations LLR emerged as the ideal choice in terms of accuracy and computation time. It is rather difficult to understand why neural network modellers have not used this technique as a baseline for comparison with their neural networks.

Using a series of experiments, e.g. modelling sunspot activity and detecting a binary message embedded in a chaotic carrier, we further demonstrated the practical virtue of the Gamma test. The Gamma test facilitates the determination of a best embedding (using our irregular embedding) for constructing a very good model from a time series. The Gamma test-embedding technique seems to be comparable with the other standard model identification tools, such as principal component analysis and dimension reduction techniques.

From the present perspective our most important result is that we can model any chaotic time series using a recurrent neural network with suitable delay lines based on the 'best' embedding suggested by the Gamma test. This forms the essential component for the construction of our chaotic stimulus-response neural model.

Stabilisation via chaos control

Having seen how to produce neural chaos in iterative versions of conventional feedforward neural networks we next explored the possibility of stabilising chaos using chaos control methods. The key idea behind most control methods takes advantage of the local properties of the underlying chaotic attractor, using small (usually minimal) and 'suitable' perturbation to stabilise the system dynamics onto the already existing (unstable) periodic orbits.

The OGY method was first studied because it was the classical chaos control technique. This method uses a small variation of a system parameter to perturb the system dynamics onto the stable manifold of the selected unstable fixed point in order to achieve the stabilisation. We then investigated other similar methods such as the OJ (Otani-Jones) method, which tries to directly minimise the 'distance' between the next system state and the desired stabilised state (an unstable fixed point).

However, these control methods have many problems in applications to real systems. The OGY method especially was originally designed with a low dimensional system in mind, and it is only effective in 2-dimensional discrete systems or 3-dimensional continuous systems if a Poincaré section is used. Both the OGY and OJ methods require working in a discrete space constructed from the original continuous system. As a result, an embedding for the construction of delay coordinates is usually required and this procedure simply complicates the problem of finding a good reconstruction/representation of the original dynamics. The steps for these control techniques are:

- locating an unstable fixed point or a periodic orbit for stabilisation;
- local stability analysis at the chosen control point, e.g. estimation of stable and unstable eigenvectors and eigenvalues;
- sensitivity analysis to calculate the sensitivity vectors for estimating how the system state varies with respect to a small variation of the control parameter.

For the OJ method we also require a good one-step predictor. Poor estimation of the Jacobian and sensitivity vector(s) may hinder the success of the control.

In contrast, we also studied Pyragas' delayed feedback and briefly demonstrated the GM periodic feedback method. These methods are based on a very simple control mechanisms, using an appropriate feedback signal to directly perturb the system dynamics. The feedback signal is usually determined by the current and some past system states and suitably chosen parameter values. However, such methods suffer the disadvantage of inadequate theoretical justification. Moreover, there is the practical problem of choosing appropriate parameter values. From a conventional chaos control theory perspective it might also be considered a disadvantage that one cannot specify *ab initio* the desired unstable periodic orbit of the original uncontrolled dynamics. However, given the goals of the present work this hardly seems relevant.

Using these control methods, we demonstrated the possibility of controlling chaotic neural systems with a series of experiments on simple artificial neural networks. The results indicate that whilst most of the methods described can be used effectively to externally control low dimensional neural chaos they are unlikely to be effective on high dimensional systems and moreover lack any serious biological plausibility. Only methods based on some type of feedback control seem to offer both the prospect of being capable of dealing with high dimensional systems and at the same time some degree of biological plausibility.

In order to put this conclusion to a practical test we further investigated various techniques, which might perhaps be suitable for high-dimensional control, by applying them to the chaotic satellite attitude control problem. We induced chaotic dynamics into the attitude control problem by using nonlinear feedback perturbation. In this way we could be sure that the system was truly representative of six dimensional chaos.

We applied the delayed feedback method, the OJ method (with modification for tracking an orbit) and an extended method based on the OGY method for high-dimensional spaces – the DYIDSG method – to control this system. Although we could not achieve the desired stabilised behaviour from the DYIDSG method (despite many attempts made), we did obtain successful control results using delayed feedback and the OJ method. This enabled us to highlight several important aspects and to compare the relative merits of the techniques.

Rather remarkably delayed feedback using knowledge based only one system variable easily stabilised the satellite onto a periodic orbit using only a single thruster. Whereas, for the OJ method, it was necessary to modify the control into tracking the same unstable periodic orbit which proved to be a rather difficult process. The unsuccessful application of DYIDSG provided a list of difficulties similar to the application of the OGY method. This also emphasises the problems of using a high-dimensional embedding to reconstruct the dynamics and led to poor accuracy of the estimation of the local stability analysis and sensitivity analysis.

This feasibility study enabled us confidently to choose the delayed feedback technique as the main control component for the chaotic stimulus-response neural system.

Stimulus-response neural model

To realise such an artificial neural model, we first demonstrated the possibility of chaos control via a simple delayed feedback on a chaotic recurrent neural network. Using this system, we suggested ways to produce varying responses for the system dynamics using an external stimulus together with delayed feedback.

At the same time, we have from time-to-time attempted to explain why such stabilisation, via simple delayed feedback, is effective using a new idea of *probabilistic local stability*. Examples can be found in Sections 5.7.1 and 8.3.

Although there is insufficient theoretical explanation for the dynamical behaviour of our neural system, our simple heuristic analysis seems to fit well with the observed simulation results.

We have further demonstrated that one can construct a chaotic iterative neural network by training the network on a chaotic time series using suitable feedback with delay lines having connections according to a *best* embedding. This embedding can easily be determined using the Gamma test. Using multiple delayed feedback controls on this system many more stimulus-response behaviours can be achieved. A *generic model* is suggested in Figure 8.11. Although this model does not claim to be an explanation for the chaotic stimulus-response behaviour observed in biological systems, having simple delay lines to elicit chaotic behaviour, and having delayed control lines to stabilise chaos, seems perfectly feasible in biological neural circuitry. Certainly, this generic scheme of a chaotic iterative neural system does produce stimulus-response behaviours similar to those described by Freeman in a biological system. This model also offers the possibility of stimulus-response systems capable of integrating stimulus events happening on differing time scales, which offers a rich new area for further research.

Therefore, the original goal of this piece of research, to a large extent, has been achieved. More-

over, we would venture to suggest that we should be surprised if it eventually transpires that such biological neural behaviour is based on some entirely different principle.

9.2 Future work

This series of investigations has suggested a generic artificial neural model which appears to have a similar behaviour to Freeman's observed neural behaviour. Although we do not claim that our generic model precisely reflects the realities of biological neural dynamics, it has provided us with an interesting investigation and at the same time perhaps made a useful step towards a full implementation in an artificial neural system in which neural chaos is exploited for recognition.

Starting from the early work on the Gamma test and surface modelling, there is still room for improving the Metabackpropagation algorithm, e.g. using clever placement of 'hills' for constructing an initial output surface for neural network training. In fact, alternative training algorithms such as BFGS could be used to improve the efficiency of Metabackpropagation. In addition with regard to the Gamma test there is much work to be done in exploiting it for practical applications and providing a detailed theoretical analysis and justification.

With regard to modelling techniques. The LDT method has surely left us a series of investigations in computational geometry. Many questions are left to be answered, such as how correct is a Delaunay triangulation in a high dimensional space if the currently available algorithms are used, and what is the best possible bound in terms of running speed of such algorithms. Many alternative algorithms for computing Delaunay triangulations are yet to be implemented and studied. In fact, this is still an actively researched area in computational geometry. Improvements in this area can surely improve our LDT prediction. Also further work is required to handle the outside query problem.

The GMP technique may now be simply viewed as a linear regression of "distances" and it may not be worth further investigation, but the LLR can still be improved by directly incorporating other preprocessing techniques, because the presentation here is still in its simplest form. There is also the possibility of developing an effective adaptive algorithm for the choice of the number of near neighbours used to construct the LLR model.

Embedding techniques have played an important role in this work, especially the use of irregular embeddings. However, many further extension, such as using an embedding containing multiple variables from several time series, may also be significant in terms of providing a good model and prediction. Recently, [Judd and Mees 1998] has also suggested using a local variable embedding to improve local modelling. This has yet to be further studied.

Controlling chaos has proven to be important in many applications. The chaotic satellite control problem in Chapter 7, illustrates several pitfalls that prevent many conventional techniques (at least in their present form) from being extended to high dimensional systems.

Better techniques for estimating the location of unstable fixed points, and estimating local dynamics and sensitivity vectors, are required for effective control of high-dimensional systems using the conventional techniques. To fully exploit the DYIDSG method, it is necessary to modify the technique so as to incorporate many more control and/or system variables, rather than using a single variable embedding in an attempt to reconstruct the dynamics of a high-dimensional chaotic attractor.

9.3 Final conclusions

The generic stimulus-response model has provided a basic framework for future investigations of this type.

A primary goal of such research is a better understanding and analysis of delayed feedback control applied to chaotic systems.

Whilst we now have some hint of the guiding principles for this type of chaotic neural stimulus-response system, we have left untouched the vexing problem of how *desirable* responses could be learnt or encouraged by some type of Hebbian learning.

As things stand progressive modification of the weights of the system might easily cause a radical modification of the geometry of the attractor, thereby possibly eliminating chaos altogether, or at least altering the attractor to such an extent that all other stimulus-response pairs are radically disrupted (progressive disruption is not so much of a conceptual problem). The problem of how a system such as we have described might *learn* is an area which we are content to leave to a future date.

Appendix A

Solving Pseudoinverse via Singular Value Decomposition

The least square fit (LSF) problem can be solved by calculating the pseudoinverse of a matrix. It is important to demonstrate the uniqueness of the pseudoinverse of a matrix.

A.1 Some theoretical background

Theorem A.1.1. *Every matrix possesses a unique pseudoinverse.*

Proof. First we just assume the existence of pseudoinverse (which can be established via the Singular Value Decomposition as shown later) and try to show uniqueness. Let $A \in \mathbb{R}^{m\times n}$ be given and suppose that $X, Y \in \mathbb{R}^{n\times m}$ are pseudoinverse of A. Then

$$\begin{aligned}
X &= XAX && \text{(by pseudoinverse condition 1)}\\
&= X(AX)^T && \text{(by pseudoinverse condition 3)}\\
&= XX^TA^T = XX^TA^TY^TA^T && \text{(by transpose of pseudoinverse condition 1)}\\
&= XX^TA^TAY && \text{(by pseudoinverse condition 3)}\\
&= XAXAY && \text{(by pseudoinverse condition 3)}\\
&= XAY && \text{(by pseudoinverse condition 2)}\\
&= XAYAY && \text{(by pseudoinverse condition 1)}\\
&= XAA^TY^TY && \text{(by pseudoinverse condition 4)}\\
&= A^TX^TA^TY^TY && \text{(by pseudoinverse condition 4)}\\
&= A^TY^TY && \text{(by pseudoinverse condition 1)}\\
&= YAY && \text{(by pseudoinverse condition 4)}\\
&= Y && \text{(by pseudoinverse condition 2).}
\end{aligned} \tag{A.1}$$

Therefore, $X = Y$ and hence, the pseudoinverse of A is unique. ∎

To demonstrate the existence of the pseudoinverse of a matrix, we need to have a discussion of the *Singular Value Decomposition* or SVD of any matrix. The SVD is based on a generalisation of the result in linear algebra that any symmetric matrix can be diagonalised via an orthogonal transformation.

Theorem A.1.2 (Singular Value Decomposition). *For any given non-zero matrix $A \in \mathbb{R}^{m\times n}$, there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ and positive real numbers $w_1 \geq w_2 \geq \cdots \geq w_r > 0$, where $r = \operatorname{rank} A$, such that*

$$A = UDV^T \tag{A.2}$$

where $D \in \mathbb{R}^{m\times n}$ has entries $D_{ii} = w_i$ $(1 \leq i \leq r)$ and all other entries are zero.

Proof. Suppose $m \geq n$, then $A^TA \in \mathbb{R}^{n\times n}$ and $A^TA \geq \mathbf{0}$ (meaning the entries are greater or equal to zero). We first show there is an orthogonal $n \times n$ matrix V such that

$$A^TA = V\Sigma V^T \tag{A.3}$$

where $\Sigma \in \mathbb{R}^{n\times n}$ is given by

$$\Sigma = \begin{bmatrix} \mu_1 & & 0 \\ & \ddots & \\ 0 & & \mu_n \end{bmatrix} \tag{A.4}$$

where $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_n$ are the eigenvalues of A^TA, counted according to multiplicity. If $A \neq \mathbf{0}$, then $A^TA \neq \mathbf{0}$ and so has at least one non-zero eigenvalue. Thus there is an r $(0 < r \leq n)$ such that $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_r > \mu_{r+1} = \cdots = \mu_n = 0$. Write $\Sigma = \begin{bmatrix} W & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$, where

$$W = \begin{bmatrix} w_1 & & 0 \\ & \ddots & \\ 0 & & w_r \end{bmatrix}, \tag{A.5}$$

with $w_1^2 = \mu_1, \ldots, w_r^2 = \mu_r$. Partition V as $V = [V_1, V_2]$ where $V_1 \in \mathbb{R}^{n\times r}$ and $V_2 \in \mathbb{R}^{n\times(n-r)}$. Since V is orthogonal, its columns form pairwise orthogonal vectors, and so $V_1^TV_2 = \mathbf{0}$. We have

$$\begin{aligned} A^TA &= V\Sigma V^T \\ &= [V_1, V_2] \begin{bmatrix} W^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} V^T \\ &= [V_1W^2, \mathbf{0}] \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \\ &= V_1W^2V_1^T. \end{aligned} \tag{A.6}$$

Hence

$$V_2^TA^TAV_2 = \underbrace{V_2^TV_1}_{=(V_1^TV_2)^T=\mathbf{0}} W^2 \underbrace{V_1^TV_2}_{=\mathbf{0}}, \tag{A.7}$$

so that $V_2^TA^TAV_2 = (AV_2)^TAV_2 = \mathbf{0}$ and hence $AV_2 = \mathbf{0}$.

Now the equality $A^TA = V_1W^2V_1^T$ suggests at first sight that we might hope that $A = WV_1^T$. However, this cannot be correct in general, since $A \in \mathbb{R}^{m\times n}$. whereas $WV_1^T \in \mathbb{R}^{r\times n}$, and so the dimensions are incorrect. However, if $U \in \mathbb{R}^{k\times r}$ satisfies $U^TU = I_r$, then $V_1W^2V_1^T = V_1WU^TUWV_1^T$ and we might hope that $A = UWV_1^T$. We use this idea to *define* a suitable U. Accordingly, we define

$$U_1 = AV_1W^{-1} \in \mathbb{R}^{m\times r}, \tag{A.8}$$

so that $A = U_1 W V_1^T$, as discussed above. We compute

$$U_1^T U_1 = W^{-1} \underbrace{V_1^T A^T A V_1 W^{-1}}_{W^2} = I_r. \tag{A.9}$$

This means that the r columns of U_1 are an orthonormal set of vectors in $\mathbb{R}^m$. Let $U_2 \in \mathbb{R}^{m\times(m-r)}$ be such that $U = [U_1, U_2]$ is orthogonal in $\mathbb{R}^{m\times m}$ – thus the columns of U_2 are made up of $(m-r)$ orthonormal vectors such that these, together with those of U_1, form an orthonormal set of m vectors. Thus $U_2^T U_1 = \mathbf{0} \in \mathbb{R}^{(m-r)\times r}$ and $U_1^T U_2 = \mathbf{0} \in \mathbb{R}^{r\times(m-r)}$. Hence we have

$$\begin{aligned} U^T A V &= \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} A[V_1, V_2] \\ &= \begin{bmatrix} U_1^T A \\ U_2^T A \end{bmatrix} [V_1, V_2] \\ &= \begin{bmatrix} U_1^T A V_1 & U_1^T A V_2 \\ U_2^T A V_1 & U_2^T A V_2 \end{bmatrix} \\ &= \begin{bmatrix} U_1^T A V_1 & \mathbf{0} \\ U_2^T A V_1 & \mathbf{0} \end{bmatrix} \qquad (\text{since } AV_2 = \mathbf{0}) \\ &= \begin{bmatrix} W & \mathbf{0} \\ U_2^T U_1 W & \mathbf{0} \end{bmatrix} \end{aligned} \tag{A.10}$$

using $U_1 = A V_1 W^{-1}$ and $U_1^T U_1 = I_r$, so that $W = U_1^T A V_1$,

$$U^T A V = \begin{bmatrix} W & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \qquad (\text{using } U_2^T U_1 = \mathbf{0}). \tag{A.11}$$

Hence,

$$A = U \begin{bmatrix} W & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} V^T, \tag{A.12}$$

as claimed. Note that the condition $m \geq n$ means that $m \geq n \geq r$, and so the dimensions of the various matrices are all valid.

If $m < n$ in the other case, consider $B = A^T$ instead. Then by the same argument as above, we get

$$A^T = B = U' \begin{bmatrix} W' & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} V'^T, \tag{A.13}$$

for orthogonal matrices $U' \in \mathbb{R}^{n\times n}$, $V' \in \mathbb{R}^{m\times m}$ and where W'^2 holds the positive eigenvalues of AA^T. Taking the transpose, we have

$$A = V' \begin{bmatrix} W' & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} U'^T. \tag{A.14}$$

Finally, we observe that from the given form of A, it is clear that $\operatorname{rank} A = r$. ∎

Using the above existence theorem, we can decompose any matrix as above and implicitly show the existence of the pseudoinverse of any matrix. The calculation of the pseudoinverse is based on the following theorem.

Theorem A.1.3. *Let $A \in \mathbb{R}^{m\times n}$ and let $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$, $W \in \mathbb{R}^{r\times r}$ be as given above via the singular value decomposition of A, so that $A = UDV^T$ where $D = \begin{bmatrix} W & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{m\times n}$. Then the pseudoinverse of A is given by*

$$A^{\#} = V \underbrace{\begin{bmatrix} W^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{n\times m} U^T. \tag{A.15}$$

Proof. To prove this is just a matter of checking that $A^{\#}$ of (A.15) satisfies the defining conditions of the pseudoinverse. We will verify two of these conditions as a simple illustration. Let $X = VHU^T$, where $H = \begin{bmatrix} W^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{n\times m}$. Then

$$\begin{aligned} AXA &= UDV^TVHU^TUDV^T \\ &= U \underbrace{\begin{bmatrix} I_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{m\times m} \underbrace{\begin{bmatrix} W & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{m\times n} V^T = A. \end{aligned} \tag{A.16}$$

Similarly, one finds that $XAX = X$. Next we consider

$$\begin{aligned} XA &= VHU^TUDV^T \\ &= V \underbrace{\begin{bmatrix} W^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{n\times m} \underbrace{U^TU}_{I_m} \underbrace{\begin{bmatrix} W & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{m\times n} V^T \\ &= V \underbrace{\begin{bmatrix} I_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{n\times n} V^T \end{aligned} \tag{A.17}$$

which is clearly symmetric. Similarly, one can verify that $AX = (AX)^T$, and the proof is complete. ■

A.2 Computation of SVD

The actual computation of SVD of a matrix $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ is described in detail in [Golub and Van Loan 1996], which is based on the method described in [Golub and Kahan 1965]. Therefore, only a concise description of the required steps which is based on [Golub and Van Loan 1996] is given here. The method depends mainly on two main matrix decomposition operations, the Householder transformations and Givens rotations.

A.2.1 Householder transformation

Let $\boldsymbol{v} \in \mathbb{R}^n$ be nonzero, then an $n \times n$ matrix P of the form

$$P = I - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}} \tag{A.18}$$

is called a *Householder transformation*, and very often being referred as *Householder matrix* or *Householder reflection*. Such a vector $\boldsymbol{v}$ is called a *Householder vector*. If a vector $\boldsymbol{x}$ is multiplied by P, then

it is reflected in the hyperplane $\mathrm{span}\{\boldsymbol{v}\}^{\perp}$. Note that P is symmetric and orthogonal. The Householder transformation can be used to zero selected components of a vector.

Suppose we are given nonzero vector $\boldsymbol{x} \in \mathbb{R}^n$ and want $P\boldsymbol{x}$ to be a multiple of $\boldsymbol{e}_1$, where $\|\boldsymbol{e}_i\|^2 = 1$ and all components of $\boldsymbol{e}_i$ are zeroes except at i^{th} component which is 1. Now

$$P\boldsymbol{x} = \left(I - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}\right)\boldsymbol{x} = \boldsymbol{x} - \frac{2\boldsymbol{v}^T\boldsymbol{x}}{\boldsymbol{v}^T\boldsymbol{v}}\boldsymbol{v} \tag{A.19}$$

and $P\boldsymbol{x} \in \mathrm{span}\{\boldsymbol{e}_1\}$ imply $\boldsymbol{v} \in \mathrm{span}\{\boldsymbol{x}, \boldsymbol{e}_1\}$. Setting $\boldsymbol{v} = \boldsymbol{x} + \alpha\boldsymbol{e}_1$ gives

$$\boldsymbol{v}^T\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{x} + \alpha\boldsymbol{x}_1 \tag{A.20}$$

and

$$\boldsymbol{v}^T\boldsymbol{v} = \boldsymbol{x}^T\boldsymbol{x} + 2\alpha\boldsymbol{x}_1 + \alpha^2 \tag{A.21}$$

and therefore

$$P\boldsymbol{x} = \left(1 - 2\frac{\boldsymbol{x}^T\boldsymbol{x} + \alpha\boldsymbol{x}_1}{\boldsymbol{x}^T\boldsymbol{x} + 2\alpha\boldsymbol{x}_1 + \alpha^2}\right)\boldsymbol{x} - 2\alpha\frac{\boldsymbol{v}^T\boldsymbol{x}}{\boldsymbol{v}^T\boldsymbol{v}}\boldsymbol{e}_1. \tag{A.22}$$

In order to zero the coefficient of $\boldsymbol{x}$, we set $\alpha = \pm\|\boldsymbol{x}\|$ for then

$$\boldsymbol{v} = \boldsymbol{x} \pm \|\boldsymbol{x}\|\boldsymbol{e}_1 \Rightarrow P\boldsymbol{x} = \left(I - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}\right)\boldsymbol{x} = \mp\|\boldsymbol{x}\|\boldsymbol{e}_1. \tag{A.23}$$

Example A.2.1
Suppose $\boldsymbol{x} = [1, 3, 1, 5]^T$ and $\boldsymbol{v} = [7, 3, 1, 5]^T$, then we have

$$P = \begin{bmatrix} -1/6 & -1/2 & -1/6 & -5/6 \\ -1/2 & 11/14 & -1/14 & -5/14 \\ -1/6 & -1/14 & 41/42 & -5/42 \\ -5/6 & -5/14 & -5/42 & 17/42 \end{bmatrix}, \tag{A.24}$$

which gives $P\boldsymbol{x} = [-6, 0, 0, 0]^T$ as in the calculation shown above. $*$

It is essential to know how a Householder reflection is applied to a matrix. Let the notation $A_{(r_i:r_j, c_i:c_j)}$ denote the submatrix of A defined by row r_i to row r_j and column c_i to column c_j. Also let $[\boldsymbol{v}, \beta] = \mathbf{house}(\boldsymbol{x})$ define the Householder transformation on $\boldsymbol{x}$ where $\beta = 2/(\boldsymbol{v}^T\boldsymbol{v})$.

Suppose we have $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$, we want to obtain $B = Q^TA$ where Q is an orthogonal matrix chosen so that $B_{(j+1:m,j)} = \mathbf{0}$ for some j that satisfies $1 \leq j \leq n$. Then we just first calculate $[\boldsymbol{v}, \beta] = \mathbf{house}(A_{(j:m,j)})$ to obtain the required Householder matrix $P = I_{m-j+1} - \beta\boldsymbol{v}\boldsymbol{v}^T$ and the required

$$Q = \begin{bmatrix} I_{j-1} & \mathbf{0} \\ \mathbf{0} & P \end{bmatrix} = I_m - \beta\tilde{\boldsymbol{v}}\tilde{\boldsymbol{v}}^T, \qquad \tilde{\boldsymbol{v}} = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{v} \end{bmatrix}. \tag{A.25}$$

A.2.2 Givens Rotations

Householder transformations are useful for introducing zeroes on a large scale by annihilating all but the first component of a vector. However, *Givens rotations* are the choice in calculations where it

is necessary to zero elements more selectively. The Givens rotations are rank-two corrections to the identity of the form

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ i \\ \\ j \\ \\ \\ \end{matrix} \qquad \text{(A.26)}$$
$$\begin{matrix} & & i & & j & & \end{matrix}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$ for some θ. Givens rotations are orthogonal and by pre-multiplication by $G(i,j,\theta)^T$ amount to a counterclockwise rotation of θ radians in the (i,j) coordinate plane.

The basic purpose of Givens rotations is to zero an element. Givens scalar values a, b, we want to compute $c = \cos(\theta)$ and $s = \sin(\theta)$ so that

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \qquad \text{(A.27)}$$

where r is some scalar value.

A.2.3 Bidiagonalisation

Bidiagonalisation is an essential first step for solving SVD. This basically involves several Householder transformations. Suppose $A \in \mathbb{R}^{m\times n}$ and $m \geq n$. We next demonstrate how to compute orthogonal U_B $(m \times m)$ and V_B $(n \times n)$ such that

$$U_B^T A V_B = \left[\begin{array}{ccccc} d_1 & f_1 & 0 & \cdots & 0 \\ 0 & d_2 & f_2 & & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & & d_{n-1} & f_{n-1} \\ 0 & \cdots & & 0 & d_n \\ \hline & & \mathbf{0} & & \end{array}\right] \qquad \text{(A.28)}$$

a bidiagonal matrix.

Basically, $U_B = U_1 \cdots U_n$ and $V_B = V_1 \cdots V_{n-2}$ can each be determined as a product of Householder matrices as follows:

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \xrightarrow{U_1} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix} \xrightarrow{V_1} \begin{bmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix} \xrightarrow{U_2}$$

$$\begin{bmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{V_2} \begin{bmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{U_3} \begin{bmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{U_4} \begin{bmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore, each U_k introduces zeroes into the k^{th} column, while V_k zeroes the appropriate entries in row k. The whole process is summarised in Algorithm A.1.

Given $A \in \mathbb{R}^{m\times n}$, $m \geq n$, this will calculate $B = U_B^T A V_B$ which is upper bidiagonal and $U_B = U_1 \cdots U_n$ and $V_B = V_1 \cdots V_{n-2}$.

Procedure Householder Bidiagonalisation (A)
for $j = 1$ to n **do**
$[\boldsymbol{v}, \beta] = \mathbf{house}\,(A_{(j:m,j)})$, $U_j^T = \begin{bmatrix} I_{j-1} & \mathbf{0} \\ \mathbf{0} & I_{m-j+1} - \beta \boldsymbol{v}\boldsymbol{v}^T \end{bmatrix}$
$A = U_j^T A$
if $j \leq n-2$ **then**
$[\boldsymbol{v}, \beta] = \mathbf{house}\,(A^T_{(j,j+1:n)})$, $V_j = \begin{bmatrix} I_j & \mathbf{0} \\ \mathbf{0} & I_{n-j} - \beta \boldsymbol{v}\boldsymbol{v}^T \end{bmatrix}$
$A = AV_j$
end if
end for
$B = A$, $U_B = U_1 \cdots U_n$, $V_B = V_1 \cdots V_{n-2}$
return (U_B, V_B, B)

Algorithm A.1: Householder Bidiagonalisation (an illustrative version without optimisation of speed and storage)

A.2.4 The SVD algorithm

Given $A \in \mathbb{R}^{m\times n}$, $m \geq n$, we can calculate the SVD of A first by reducing A to upper bidiagonal form as described above in Algorithm A.1 so to obtain

$$U_B^T A V_B = \begin{bmatrix} B \\ \mathbf{0} \end{bmatrix} \qquad B = \begin{bmatrix} d_1 & f_1 & & \cdots & 0 \\ 0 & d_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & f_{n-1} \\ 0 & \cdots & & 0 & d_n \end{bmatrix} \in \mathbb{R}^{n\times n}. \tag{A.29}$$

Then the remaining problem is to compute the SVD of B. The immediate next step is to try to diagonalise B by reducing each f_i to zero. The steps are:

- Compute the eigenvalue λ of

$$T_{(n-1:n,n-1:n)} = \begin{bmatrix} d_{n-1}^2 + f_{n-2}^2 & d_{n-1}f_{n-1} \\ d_{n-1}f_{n-1} & d_n^2 + f_{n-1}^2 \end{bmatrix} \tag{A.30}$$

  that is closer to $d_n^2 + f_{n-1}^2$.

- Compute $c_1 = \cos(\theta_1)$ and $s_1 = \sin(\theta_1)$ such that

$$\begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix} \begin{bmatrix} d_1^2 - \lambda \\ d_1 f_1 \end{bmatrix} = \begin{bmatrix} \times \\ 0 \end{bmatrix} \tag{A.31}$$

  and set $G_1 = G(1, 2, \theta_1)$, a Givens rotation.

Then we apply the Givens rotation G_1 above to B directly. For illustration, we assume $n = 6$ and this gives

$$B \longleftarrow BG_1 = \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ + & \times & \times & 0 & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}. \tag{A.32}$$

We then can determine Givens rotations $U_1, V_2, U_2, \ldots, V_{n-1}$ and U_{n-1} to chase the unwanted nonzero element down the bidiagonal as follows:

$$\begin{aligned} B \longleftarrow U_1^T B &= \begin{bmatrix} \times & \times & + & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \\ B \longleftarrow BV_2 &= \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & + & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \\ B \longleftarrow U_2^T B &= \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & + & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \end{aligned} \tag{A.33}$$

and so on. Eventually we obtain a new bidiagonal $\overline{B}$,

$$\overline{B} = (U_{n-1}^T \cdots U_1^T) B (G_1 V_2 \cdots V_{n-1}) = \overline{U}^T B \overline{V}. \tag{A.34}$$

However, the above procedure can only applied if f_k and d_k are non-zeros. If $f_k = 0$ for some k, then B can be split into

$$\begin{array}{cc} \begin{bmatrix} B_1 & \mathbf{0} \\ \mathbf{0} & B_2 \end{bmatrix} & \begin{array}{c} k \\ n-k \end{array} \\ \begin{array}{cc} k & n-k \end{array} & \end{array} \tag{A.35}$$

two matrices B_1 and B_2 and the original SVD problem therefore decouples into two smaller problems. If $d_k = 0$ for $k < n$, then by pre-multiplying a sequence of Givens transformations can zero f_k. For example, if $n = 6$ and $k = 3$, then by rotating in row planes (3,4), (3,5), and (3,6) we can zero the

entire third row as follows:

$$B = \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{(3,4)} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & 0 & + & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}$$

$$\xrightarrow{(3,5)} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & + \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{(3,6)} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \tag{A.36}$$

If $d_n = 0$, then the last column can be zeroed with a series of column rotations in planes $(n-1, n), (n-2, n), \ldots, (1, n)$. In summary, we can decouple if $f_1 \cdots f_{n-1} = 0$ or $d_1 \ldots d_n = 0$. The precise background idea is presented in [Golub and Kahan 1965; Golub and Van Loan 1996] on which this description is based. The whole process can be summarised in Algorithm A.2.

Given a bidiagonal matrix $B \in \mathbb{R}^{m \times n}$ having no zeroes on its diagonal or super diagonal, the algorithm will return $\overline{B} = \overline{U}^T B \overline{V}$, orthogonal matrix $\overline{U}$ and orthogonal matrix $\overline{V}$.

Procedure Golub-Kahan SVD Step (B)
Let μ be the eigenvalue of the trailing 2×2 submatrix of $T = B^T B$ that is closer to t_{nn}.
$y = t_{11} - \mu$
$z = t_{12}$
for $k = 1$ to $n - 1$ **do**
  Determine $c = \cos(\theta)$ and $s = \sin(\theta)$ such that $\begin{bmatrix} y & z \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} \star & 0 \end{bmatrix}$
  $V_k = G(k, k+1, \theta)$ where G is a Givens rotation.
  $B = BV_k$
  $y = b_{kk}$; $z = b_{k+1,k}$
  Determine $c = \cos(\theta)$ and $s = \sin(\theta)$ such that $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} \star \\ 0 \end{bmatrix}$
  $U_k = G(k, k+1, \theta)$ where G is a Givens rotation.
  $B = U_k^T B$
  if $k < n - 1$ **then**
    $y = b_{k,k+1}$; $z = b_{k,k+2}$
  end if
end for
$\overline{U} = U_1 U_2 \cdots U_{n-1}$; $\overline{V} = V_1 V_2 \cdots V_{n-1}$
$\overline{B} = B$
return $(\overline{B}, \overline{U}, \overline{V})$

Algorithm A.2: Golub-Kahan SVD step.

Typically, after a few of the Golub-Kahan SVD step in Algorithm A.2, the super diagonal entry f_{n-1} becomes negligible. Some criteria for smallness within B's band can then be used to zero such

negligible values. Typically, the criteria is of the form

$$\begin{aligned} |f_i| &\leq \epsilon(|d_i| + |d_{i+1}|) \\ |d_i| &\leq \epsilon\|B\| \end{aligned} \tag{A.37}$$

where ϵ is a small multiple of the unit roundoff and $\|\cdot\|$ is some form of norm.

Combining all of these ideas, we can then obtain the full SVD algorithm in Algorithm A.3. This is only a crude description demonstrating the steps necessary to calculate the SVD. Many details in terms of computational implementation have been deliberately omitted since [Golub and Van Loan 1996] has provided a thorough discussion on the derivation and the possible implementation. Depending on how much information is needed from SVD, the computational time is in order of $O(mn^2) + O(n^3)$ (see also [Golub and Van Loan 1996]). A SVD algorithm in terms of C in source code is readily available from [Press *et al.* 1992].

Given $A \in \mathbb{R}^{m\times n}$ ($m \geq n$) and ϵ, a small multiple of the unit roundoff, the algorithm returns orthogonal $U \in \mathbb{R}^{m\times n}$, orthogonal $V \in \mathbb{R}^{n\times n}$ and diagonal matrix $\Sigma \in \mathbb{R}^{m\times n}$ such that $U^T AV = \Sigma + E$ where E is an error matrix.

Procedure SVD (A**)**

Use Algorithm A.1 to compute the bidiagonalisation:

$$\begin{bmatrix} B \\ \mathbf{0} \end{bmatrix} \leftarrow (U_1 \cdots U_n)^T A (V_1 \cdots V_{n-2})$$

$U = U_1 \cdots U_n$; $V = V_1 \cdots V_{n-2}$

while $q \neq n$ **do**

for $i = 1$ to $n-1$ **do**

if $|b_{i,i+1}| \leq \epsilon(|b_{ii}| + |b_{i+1,i+1}|)$ **then**

Set $b_{i,i+1}$ to zero

end if

end for

Find the largest q and the smallest p such that if B can be represented by

$$\begin{array}{ccc} \begin{bmatrix} B_{11} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & B_{22} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & B_{33} \end{bmatrix} & \begin{matrix} p \\ n-p-q \\ q \end{matrix} \\ \begin{matrix} p & n-p-q & q \end{matrix} & \end{array}$$

then B_{33} is diagonal and B_{22} has nonzero super diagonal.

if $q < n$ **then**

if any diagonal entry in B_{22} is zero **then**

Zero the super diagonal entry in the same row. Update B.

Update U and V accordingly to the orthogonal transforms used.

else

Apply Algorithm A.2 to B_{22} to get $\overline{U}$ and $\overline{V}$.

$B = \text{diag}\,(I_p, \overline{U}, I_{q+m-m})^T B \text{diag}\,(I_p, \overline{V}, I_q)$

$U = U\,\text{diag}\,(I_p, \overline{U}, I_{q+m-m})$; $V = V\,\text{diag}\,(I_p, \overline{V}, I_q)$

end if

end if

end while

$$\Sigma = \begin{bmatrix} B \\ \mathbf{0} \end{bmatrix}$$

return (U,Σ,V)

Algorithm A.3: The SVD algorithm.

Appendix B

Normal vector

Theorem B.0.1. *Given a set of linearly independent d-dimensional vectors $\boldsymbol{a}_j$, $(1 \leq j \leq d-1)$, the vector orthogonal to the span $[\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_{d-1}]$ is given by expressing the determinant*

$$\begin{vmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \cdots & \boldsymbol{e}_d \\ a_{11} & a_{12} & \cdots & a_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(d-1)1} & a_{(d-1)2} & \cdots & a_{(d-1)d} \end{vmatrix} \tag{B.1}$$

in terms of $\boldsymbol{e}_k$, the first row elements of the determinant, where $\boldsymbol{e}_k$ are d-dimensional vectors with all elements equal to zeroes except at the k^{th} position the element is 1, i.e. they are forming a basis for the d-dimensional space.

Proof. Let $\boldsymbol{e}_k$ be vectors as defined in the theorem, and let $\boldsymbol{b} = (b_1, \ldots, b_d)$ and $\boldsymbol{a}_j = (a_{j1}, \ldots, a_{jd})$ $(1 \leq j \leq d-1)$ be d-dimensional vectors. Also let

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(d-1)1} & a_{(d-1)2} & \cdots & a_{(d-1)d} \end{bmatrix} \tag{B.2}$$

be a $(d-1) \times d$ matrix formed by the row vectors $\boldsymbol{a}_j$.

Consider the following determinant

$$\det(A, \boldsymbol{b}) = \begin{vmatrix} b_1 & b_2 & \cdots & b_d \\ a_{11} & a_{12} & \cdots & a_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(d-1)1} & a_{(d-1)2} & \cdots & a_{(d-1)d} \end{vmatrix} \tag{B.3}$$

which can be expanded by the first row into

$$\begin{aligned}\det(A,\boldsymbol{b}) &= b_1\begin{vmatrix} a_{12} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{(d-1)2} & \cdots & a_{(d-1)d} \end{vmatrix} - b_2\begin{vmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{(d-1)1} & \cdots & a_{(d-1)d} \end{vmatrix} \\ &\quad + \cdots + (-1)^{d+1}b_d\begin{vmatrix} a_{11} & \cdots & a_{1(d-1)} \\ \vdots & \ddots & \vdots \\ a_{(d-1)1} & \cdots & a_{(d-1)(d-1)} \end{vmatrix} \\ &= \sum_{k=1}^{d} b_k\boldsymbol{e}_k \cdot v_k\boldsymbol{e}_k = \boldsymbol{b}\cdot\boldsymbol{v},\end{aligned} \tag{B.4}$$

expressing in terms of $\boldsymbol{v} = (v_1, \ldots, v_d)$ where

$$v_k = (-1)^{k+1}\det(A_k) \quad (1 \le k \le d) \tag{B.5}$$

and A_k is the submatrix of A with the k^{th} column removed. But $\det(A,\boldsymbol{b}) = 0$ if and only if either $\boldsymbol{b}$ is orthogonal to $\boldsymbol{v}$ or $\boldsymbol{b}$ is linearly dependent to the span $\left[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{(d-1)}\right]$ or the set of vectors $\boldsymbol{a}_k$ are linearly dependent, i.e. the elements of $\boldsymbol{v}$ are all zeroes.

Provided the vectors $\boldsymbol{a}_j$ are linearly independent or in other words $\boldsymbol{v}$ is a non-zero vector, we can always choose $\boldsymbol{b}$ such that $\det(A,\boldsymbol{b}) = 0$. This implies that we can choose any $\boldsymbol{b} \in$ span $\left[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{(d-1)}\right]$ such that $\boldsymbol{v}$ is orthogonal to $\boldsymbol{b}$. Hence,

$$\boldsymbol{v} = \begin{vmatrix} \boldsymbol{i}_1 & \boldsymbol{i}_2 & \cdots & \boldsymbol{i}_d \\ a_{11} & a_{12} & \cdots & a_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(d-1)1} & a_{(d-1)2} & \cdots & a_{(d-1)d} \end{vmatrix} \tag{B.6}$$

is always orthogonal to span $\left[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{(d-1)}\right]$. ∎

Appendix C

Point location in Delaunay cell via LP

This is an alternative method [Fukuda 1998] for efficiently determining the nearest point set associated with the Delaunay cell containing a given point $\boldsymbol{c} \in \mathbb{R}^d$. This is done by translating the problem into a standard linear programming (LP) problem, which can easily be solved by using the simplex method or other similar techniques. This may provide a more efficient method for simplex location to improve the geometrical method, LDT, on data modelling. The idea is first to realise that the Delaunay triangulation can be represented by the convex hull of appropriately lifted points in $\mathbb{R}^{d+1}$, and the projected *lower* facets of the convex hull coincide with the Delaunay triangulation once they are projected to the original space $\mathbb{R}^{d+1}$, as discussed in Section 3.2.2. The technique to find the Delaunay cell containing the point $\boldsymbol{c}$ can be simplified to locate the first facet of a polyhedron "hit" by a ray.

C.1 Reformulation into LP

As described earlier we first lift the point into $\mathbb{R}^{d+1}$. Let

$$f(\boldsymbol{x}) = x_1^2 + x_2^2 + \cdots + x_d^2 \tag{C.1}$$

and let

$$\tilde{\boldsymbol{p}} = (\boldsymbol{p}, f(\boldsymbol{x})) \in \mathbb{R}^{d+1} \tag{C.2}$$

for $\boldsymbol{p} \in S$, i.e. in the given set P of points and the position of the point $\boldsymbol{p}$ is indicated by $\boldsymbol{x}$.

Then the lower convex hull P of the lifted points, $\tilde{S} = \{\tilde{\boldsymbol{p}} : \boldsymbol{p} \in S\}$, represents the Delaunay complex. For any fixed vector $\tilde{\boldsymbol{y}} \in \mathbb{R}^{d+1}$ and $y_0 \in \mathbb{R}$, let $\tilde{\boldsymbol{y}} \cdot \boldsymbol{x} \geq -y_0$ denote a general inequality of a vector $\boldsymbol{x} \in \mathbb{R}^{d+1}$. For such an inequality to represent a facet of P, it must be satisfied by all points in $\tilde{S}$,

$$\tilde{\boldsymbol{y}} \cdot \tilde{\boldsymbol{p}} \geq -y_0, \quad \forall \tilde{\boldsymbol{p}} \in \tilde{S}, \tag{C.3}$$

and by any points shifted vertically upwards, that is point with the last component $\tilde{y}_{d+1} \geq 0$.

Furthermore any non-vertical facet can be represented by such an inequality with $\tilde{y}_{d+1} = 1$. For a given point $\boldsymbol{c}$, let $\tilde{\boldsymbol{c}} = (\boldsymbol{c}, 0)$ and let $L(\lambda) = \tilde{\boldsymbol{c}} + \lambda \boldsymbol{e}_{d+1}$, $\lambda \geq 1$, where $\boldsymbol{e}_{d+1}$ is the unit vector in $\mathbb{R}^{d+1}$ whose last component is 1.

Determining the Delaunay cell containing $\boldsymbol{c}$ is equivalent to finding the first facet hit by the halfline L. Therefore, we need to find a non-vertical facet inequality such that the intersection point of the corresponding hyperplane $\{\boldsymbol{x} : \boldsymbol{y} \cdot \boldsymbol{x} = -y_0\}$ and the halfline $L(\lambda)$, $\lambda \geq 0$, is highest possible.

By substituting $L(\lambda)$ for $\boldsymbol{x}$ in $\boldsymbol{y} \cdot \boldsymbol{x} = -y_0$ with $\tilde{y}_{d+1} = 1$, we obtain

$$\lambda = -y_0 - \boldsymbol{y} \cdot \boldsymbol{c}, \tag{C.4}$$

where $\boldsymbol{y}$ denotes the vector $\tilde{\boldsymbol{y}}$ without the last coordinate $\tilde{y}_{d+1}$. The LP formulation is therefore

$$\begin{array}{ll} \text{minimise} & z = y_0 + \boldsymbol{y} \cdot \boldsymbol{c} \\ \text{subject to} & f(\boldsymbol{p}) + y_0 + \boldsymbol{y} \cdot \boldsymbol{p} \geq 0 \quad \text{for all } \boldsymbol{p} \in S. \end{array} \tag{C.5}$$

Although an optimal solution (y_0,$\boldsymbol{y}$) to this LP problem does not directly determine any facet in general, the simplex method can return an optimal basic solution which can determine a facet inequality in this case. The Delaunay cell containing $\boldsymbol{c}$ is the one determined by the set of points in S whose corresponding inequalities are satisfied with equality at an optimal solution.

In some cases, the above LP might be unbounded. This corresponds to the case in which $\boldsymbol{c}$ is not in any Delaunay cell, or in other words, not in the convex hull of S.

Appendix D

Function optimisation for neural network training

The aim of learning in a typical feedforward artificial neural network (FANN) or multilayer perceptron (MLP) is to minimise the instantaneous squared error of the output signal E, by modifying the total number of n synaptic weights $\boldsymbol{w} = (w_1, w_2, \dots, w_n)$ (simply labelled by single subscript for the ease of explanation) of the whole network, on the given set of input-output data pairs. Therefore, we want to minimise the scalar cost function $E(\boldsymbol{w})$ subject to $\boldsymbol{w} \in \mathbb{R}^n$. Without going into details, the usual technique is to train the network, so that the global minimum of $E(\boldsymbol{w})$ is attained. This is done via *backpropagation* using a gradient steepest descent approach to update the weights. At each i^{th} training cycle, the weights are changed by the following heuristic rule,

$$\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} - \mu^{(\mathrm{i})} \nabla_{\boldsymbol{w}} E(\boldsymbol{w}^{(i)}) \tag{D.1}$$

where $\mu^{(\mathrm{i})} > 0$ is the learning rate and it usually has a fixed value and $\nabla_{\boldsymbol{w}}$ is the gradient operator with respect to $\boldsymbol{w}$, i.e

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \nabla E(\boldsymbol{w}) = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_n} \right]^T. \tag{D.2}$$

However, alternative techniques for choosing $\boldsymbol{w}$ to minimise $E(\boldsymbol{w})$, based on known function optimisation techniques, are proven to have a much faster convergence rate in many practical applications.

D.1 Error function optimisation strategies

There following are several optimisation strategies which are suitable for neural network training. The ideas are introduced with minimal but sufficient information for implementation. Much theoretical background is omitted.

D.1.1 The Newton's method

Newton's method is a well known technique for simple function optimisation but is presented here for the higher dimensional approach necessary for neural network training. The function $E(\boldsymbol{w})$ near the point $\boldsymbol{w}^{(i)}$ can be approximated by the first few terms of the Taylor series expansion

$$E(\boldsymbol{w}) \approx E(\boldsymbol{w}^{(i)}) + (\boldsymbol{w} - \boldsymbol{w}^{(i)})^T \nabla E(\boldsymbol{w}^{(i)}) + \frac{1}{2} (\boldsymbol{w} - \boldsymbol{w}^{(i)})^T \nabla^2 E(\boldsymbol{w}^{(i)}) (\boldsymbol{w} - \boldsymbol{w}^{(i)}), \tag{D.3}$$

where

$$\nabla^2 E(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_n} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_n \partial w_1} & \frac{\partial^2 E}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_n^2} \end{bmatrix}, \tag{D.4}$$

the Hessian matrix of the scalar function $E(\boldsymbol{w})$. To minimise this series the next point $\boldsymbol{w}^{(i+1)}$ must satisfy

$$\nabla E(\boldsymbol{w}^{(i)}) + \nabla^2 E(\boldsymbol{w}^{(i)})(\boldsymbol{w}^{(i+1)} - \boldsymbol{w}^{(i)}) = \boldsymbol{0} \tag{D.5}$$

because, at a minimum point, $\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$. If the inverse matrix of the Hessian matrix exists, the above (D.5) can be rewritten as

$$\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} - [\nabla^2 E(\boldsymbol{w}^{(i)})]^{-1} \nabla E(\boldsymbol{w}^{(i)}) \tag{D.6}$$

This is the basic updating rule which can be used for the weights training in a MLP. The main disadvantages are to require the calculation of the first and second order derivatives and the calculation of the inverse of the Hessian matrix $[\nabla^2 E(\boldsymbol{w})]^{-1}$, with the possible problems of computational difficulties and singularity. If the starting point $\boldsymbol{w}^{(0)}$ is far away from a minimum, the algorithm may diverge. This happens when the Hessian matrix is not *positive definite* – a symmetric matrix $A \in \mathbb{R}^{m \times m}$ is said to be *positive definite* if the quadratic form $\boldsymbol{x}^T A \boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq \boldsymbol{0}$, $\boldsymbol{x} \in \mathbb{R}^m$.

D.1.2 The quasi-Newton method

The *quasi-Newton method*, also called the *variable metric method* , is designed to overcome the problem of computing the Hessian matrix in Newton's method. It is performed by iteratively using successively improved *approximations* to the inverse Hessian instead of the true inverse. The improved approximation are obtained from the information generated during the gradient descent optimisation process. The sequential quasi-Newton method employs the differences of two successive iteration points and the difference of the corresponding gradients to approximate the inverse Hessian matrix.

One implementation of this powerful and sophisticated quasi-Newton method is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [Fletcher 1987]. It can be formulated by the following equations,

$$\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} + \mu^{(i)} \boldsymbol{d}_i, \tag{D.7}$$

where

$$\boldsymbol{d}_i \approx \boldsymbol{w}^{(i+1)} - \boldsymbol{w}^{(i)} = -\boldsymbol{H}_i \nabla E(\boldsymbol{w}^{(i)}). \tag{D.8}$$

Be defining

$$\boldsymbol{y}_i = \nabla E(\boldsymbol{w}^{(i+1)}) - \nabla E(\boldsymbol{w}^{(i)}), \tag{D.9}$$

we can update the matrix $\boldsymbol{H}$ by

$$\boldsymbol{H}_{i+1} = \left(I - \frac{\boldsymbol{d}_i \boldsymbol{y}_i^T}{\boldsymbol{d}_i^T \boldsymbol{y}_i} \right) \boldsymbol{H}_i \left(I - \frac{\boldsymbol{y}_i \boldsymbol{d}_i^T}{\boldsymbol{d}_i^T \boldsymbol{y}_i} \right) + \frac{\boldsymbol{d}_i \boldsymbol{d}_i^T}{\boldsymbol{d}_i^T \boldsymbol{y}_i}. \tag{D.10}$$

The learning rate $\mu^{(i)} \geq 0$ is determined from the one-dimensional line search

$$\mu^{(i)} = \arg\min_{\mu \geq 0} E\left[\boldsymbol{w}^{(i)} - \mu \boldsymbol{H}_i \nabla E(\boldsymbol{w}^{(i)})\right]. \tag{D.11}$$

The matrix $\boldsymbol{H}_i$ denotes the current approximation to $[\nabla^2 E(\boldsymbol{w})]^{-1}$. The iterative procedure starts at an arbitrary point $\boldsymbol{w}^{(0)}$, preferably close to the true minimum point, with an initial approximation $\boldsymbol{H}_0$ usually taken to be the identity matrix I. This type of variable metric method has eliminated the need of deriving the second-order derivatives. A C-implementation of this method can be found in [Press *et al.* 1992].

D.1.3 The conjugate gradient method

The *conjugate gradient method* is an alternative function optimisation technique which is often used in neural network training. This unconstrained minimisation is derived in such a way that it will work well, or even exactly, if applied to a quadratic function (usually with positive definite Hessian $\boldsymbol{H}$). This method is said to be derived from a *quadratic model*. Also this method is derived with the property of *quadratic termination* which means that the method will locate the minimising point $\boldsymbol{w}^*$ of a quadratic function in a known finite number of iterations, yet can be well applied iteratively to minimise non-quadratic functions. In this case, the non-quadratic function is the feedforward neural network.

A particular way of obtaining a quadratic termination is to invoke the concept of the *conjugacy* of a set of non-zero vectors $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(n)}$ to a given positive definite matrix $\boldsymbol{H}$ that is

$$\boldsymbol{v}^{(i)^T} \boldsymbol{H} \boldsymbol{v}^{(j)} = 0 \qquad \forall i \neq j. \tag{D.12}$$

A *conjugate direction method* is one which generates such directions when applied to a quadratic function with Hessian $\boldsymbol{H}$.

The conjugate gradient method is a technique, of the combination of the conjugate direction information and steepest descent method, often enable us to improve the convergence speed of the optimisation. A simple form of this algorithm is formulated by the following equations,

$$\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} + \mu^{(i)} \boldsymbol{d}_i, \tag{D.13}$$

$$\boldsymbol{d}_i = \beta_i \boldsymbol{d}_{i-1} - \nabla E(\boldsymbol{w}^{(i)}), \tag{D.14}$$

where

$$\beta_i = \frac{|\nabla E(\boldsymbol{w}^{(i)})|^2}{|\nabla E(\boldsymbol{w}^{(i-1)})|^2} \tag{D.15}$$

and

$$\mu^{(i)} = \arg\min_{\mu \geq 0} E\left[\boldsymbol{w}^{(i)} + \mu \boldsymbol{d}_i\right]. \tag{D.16}$$

Therefore, the conjugate gradient algorithm uses information about the direction search $\boldsymbol{d}_{i-1}$ from the previous iteration in order to accelerate the convergence, and each search direction is conjugate if the objective function is quadratic.

Theoretically, the algorithm will minimise a quadratic function in n or fewer iterations but in practice, it is usually necessary to restart the optimisation process periodically due to numerical inaccuracy

of the results in a search direction and/or due to the non-quadratic nature of the problem. The conjugate gradient method can be regarded as lying in between the method of steepest descent and the quasi-Newton methods in terms of the convergence properties and the complexity. The advantage of the conjugate gradient algorithm is its simplicity for estimation of optimal values of the parameters $\mu^{(i)}$ and β_i and no Hessian matrix need to be generated. However, in practice this gradient method does not seem to be as effective as BFGS (quasi-Newton) method. Details of the theoretical background can be found in [Fletcher 1987].

Bibliography

[Amit *et al.* 1987] D.J. Amit, H. Gutfreund and H. Sompolinsky. Information storage in neural networks with low levels of activity. *Physical Review A*, **35**:2239–2303, 1987.

[Auerbach *et al.* 1992] D. Auerbach, C. Grebogi, E. Ott and J.A. Yorke. Controlling chaos in high dimensional systems. *Physical Review Letters*, **69**(24):3479–3482, 1992.

[Babcock and Westervelt 1986] K.L. Babcock and R.M. Westervelt. Complex dynamics in simple neural circuits. In *Neural Networks for Computing, AIP Conference Proceedings*, volume 151, pages 23–28. Snowbird, UT, 1986.

[Babloyantz *et al.* 1995] A. Babloyantz, C. Loureno and J.A. Sepulchre. Control of chaos in delay differential equations, in a network of oscillators and in model cortex. *Physica D*, **86**:274–283, 1995.

[Baker and Gollub 1990] G.L. Baker and J.P. Gollub. *Chaotic dynamics an introduction*. Cambridge University Press, 1990.

[Baldi and Atiya 1994] Pierre Baldi and Amir F. Atiya. How delays affect neural dynamics and learning. *IEEE Transactions on Neural Networks*, **5**(4):612–621, 1994.

[Barber *et al.* 1996] C. Bradford Barber, David P. Dobkin and Hannu Huhdanpaa. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, **22**(4):469–483, 1996. WEB: http://www.geom.umn.edu/software/qhull/ and personal communication via email.

[Bentley 1975] J.L. Bentley. Multidimensional binary search trees used for associative search. *Comm. ACM*, **18**:309–517, 1975.

[Bentley 1979] J.L. Bentley. Decomposable searching problems. *Information Processing Letters*, **8**:244–251, 1979.

[Blum and Li 1991] E. Blum and L. Li. Approximation theory and feedforward networks. *Neural Networks*, **4**:511–515, 1991.

[Bombieri *et al.* 1969] E. Bombieri, E. de Giorgi and E. Giusti. Minimal cones and the Bernstein problem. *Inventiones Mathematicae*, **7**:243–268, 1969.

[Brown 1979] D. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, **9**:223–228, 1979.

[Carroll and Pecora 1993] Thomas L. Carroll and Louis M. Pecora. Using chaos to keep period-multiplied systems in phase. *Physics Review E*, **48**(4):2426–2436, 1993.

[Carroll *et al.* 1992] Thomas L. Carroll, Ioana Triandaf, Ira Schwartz and Lou Pecora. Tracking unstable orbits in an experiment. *Physical Review A*, **46**(10):6189–6192, 1992.

[Celka 1994] P. Celka. Experimental verification of pyragas's chaos control method applied to Chua's circuit. *International Journal of Bifurcation and Chaos*, **4**(6):1703–1706, 1994.

[Clarkson 1987] K.L. Clarkson. New applications of random sampling in computational geometry. *Discrete Computational Geometry*, **2**:195–222, 1987.

[Clarkson *et al.* 1993] K.L. Clarkson, K. Mehlhorn and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry: Theory and Applications*, **3**:185–211, 1993.

[Clarkson and Shor 1989] K.L. Clarkson and P.W. Shor. Applications of random sampling in computational geometry II. *Discrete Computational Geometry*, **4**:387–421, 1989.

[Cooper and Schöll 1995] D.P. Cooper and E. Schöll. Tunable real space transfer oscillator by delayed feedback control of chaos. *Zeitschrift für Naturforschung, Section A – A Journal of Physical Sciences*, **50a**:117–124, 1995.

[Crouch 1984] P.E. Crouch. Spacecraft attitude control and stabilization: Application of geometric control theory to rigid body models. *IEEE Transactions on Automation Control*, **AC-29**(4), 1984.

[Cuomo and Oppenheim 1993] K. Cuomo and A. Oppenheim. Circuit implementation of synchronized chaos with applications to communications. *Physical Review Letters*, **71**(1):65–68, 1993.

[Cybenko 1989] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signal and Systems*, **2**:303–314, 1989.

[de Berg and Schwarzkopf 1995] M. de Berg and O. Schwarzkopf. Cuttings and applications. *International Journal of Computational Geometry and Applications*, **5**:343–355, 1995.

[de Berg *et al.* 1995] M. de Berg, M. van Kreveld and J. Snoeyink. Two- and three- dimensional point location in rectangular subdivisions. *Journal of Algorithms*, **18**:256–277, 1995.

[De Vogelaere 1956] R. De Vogelaere. Methods which preserve the contact transformation properties of Hamiltonian equations. Report 4, Department of Mathematics, University of Notre Dame, 1956.

[Delaunay 1934] B. Delaunay. Sur la sphere vide. *Bull. Acad. Sci. USSR(VII), Classe Sci. Mat. Nat.*, pages 793–800, 1934.

[Devaney 1992] Robert L. Devaney. *A First Course in Chaotic Dynamical Systems - Theory and Experiment*. Addison-Wesley, 1992.

[Ding *et al.* 1997] Mingzhou Ding, E-Jiang Ding, William L. Ditto, Bruce Gluckman, Visarath In, Jian-Hua Peng, Mark L. Spano and Weiming Yang. Control and synchronization of chaos in high dimensional systems: Review of some recent results. *Chaos*, **7**(4):644–652, 1997.

[Ding *et al.* 1996] Mingzhou Ding, Weiming Yang, Visarath In, William L. Ditto, Mark L. Spano and Bruce Gluckman. Controlling chaos in high dimensions: Theory and experiment. *Physical Review E*, **53**(5):4334–4344, 1996.

[Ditto and Pencora 1993] W. Ditto and M. Pencora. Mastering Chaos. *Scientific American*, pages 62–68, 1993.

[Ditto *et al.* 1990] W. Ditto, S.N. Rauseo and M.L. Spano. Experimental control of chaos. *Physical Review Letters*, **65**:3211, 1990.

[Dracopoulos and Jones 1993] D. C. Dracopoulos and Antonia J. Jones. Neuromodels of analytic Dynamic Systems. *Neural Computing & Applications*, **1**(4):268–279, 1993.

[Dracopoulos and Jones 1997] D.C. Dracopoulos and Antonia J. Jones. Adaptive neuro-genetic control of chaos applied to the attitude control problem. *Neural Computing & Applications*, **6**(2):102–115, 1997.

[Dressler and Nitsche 1992] U. Dressler and G. Nitsche. Controlling chaos using time delay coordinates. *Physics Review Letters*, **68**(1):1–4, 1992.

[Dwyer 1987] R.A. Dwyer. A faster divide-and-conquer algorithm for constructing Delaunay triangulations. *Algorithmica*, **2**:137–151, 1987.

[Dwyer 1991] R.A. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. *Discrete Computational Geometry*, **6**:343–367, 1991.

[Edelsbrunner 1987] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[Edelsbrunner *et al.* 1986] H. Edelsbrunner, G. Haring and D. Hilbert. Rectangular point location in d dimensions with applications. *Comput. J.*, **15**:317–340, 1986.

[Ekeland 1998] Ivar Ekeland. How to melt if you must. *Nature*, **392**:654–657, 1998.

[Finkel and Bentley 1974] R.A. Finkel and J.L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, **4**:1–9, 1974.

[Finkel *et al.* 1977] R.A. Finkel, J.H. Friedman and J.L. Bently. An algorithm for finding best matches in logarithmic expected time. *ACM Transaction on Mathematical Software*, **3**(3):200–226, 1977.

[Fletcher 1987] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, 2nd edition, 1987.

[Fortune 1987] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, **2**:153–174, 1987.

[Freeman 1991] W.J. Freeman. The physiology of perception. *Scientific American*, pages 34–41, 1991.

[Fukuda 1998] Komei Fukuda. Frequently asked questions in polyhedral computation. Internet homepage, 1998. `http://www.ifor.math.ethz.ch/ifor/staff/fukuda/polyfaq/polyfaq.html`.

[Garfinkel *et al.* 1992] A. Garfinkel, M. Spano, W. Ditto and J. Weiss. Controlling cardiac chaos. *Science*, **257**:1230, 1992.

[Gerbrands 1981] Jan J. Gerbrands. On the relationships between SVD, KLT and PCA. *Pattern Recognition*, **14**(1–6):375–381, 1981.

[Gills *et al.* 1992] Zelda Gills, Christian Iwata and Rajarshi Roy. Tracking unstable steady states: Extending the stability regime of a multimode laser system. *Physical Review Letters*, **69**(22):3169–3172, 1992.

[Gluckman *et al.* 1997] Bruce J. Gluckman, Mark L. Spano, Weiming Yang, Mingzhou Ding, Visarath In and William L. Ditto. Tracking unstable periodic orbits in nonstationary high-dimensional chaotic systems: Method and experiment. *Physical Review E*, **55**(5):4935–4942, 1997.

[Golub and Van Loan 1996] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[Golub and Kahan 1965] G.H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Num. Anal.*, **2**:205–224, 1965.

[Grebogi *et al.* 1988] C. Grebogi, E. Ott and J.A. Yorke. Unstable periodic orbits and the dimensions of multifractal chaotic attractors. *Physical Review A*, **37**:1711–1723, 1988.

[Grünbaum 1961] B. Grünbaum. Measure of symmetry for convex sets. In *Proceedings of the 7th Symposium in Pure Mathematics of the American Mathematical Society, Symposium on convexity*, pages 233–270. AMS, Providence, R.I., 1961.

[Güémez and Matías 1993] J. Güémez and M.A. Matías. Control of chaos in unidimensional maps. *Physics Letters A*, **181**:29–32, 1993.

[Guibas *et al.* 1992] L. Guibas, D. Knuth and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, **7**:381–413, 1992.

[Guibas and Stolfi 1985] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transaction of Graphics*, **4**(2):75–172, 1985.

[Hammel *et al.* 1985] S. Hammel, C. Jones and J. Moloney. Global dynamical behaviour of the optical field in a ring cavity. *J. Opt. Soc. Am. B*, **2**(4):552–564, 1985.

[Hastings-James and Sage 1969] R. Hastings-James and M.W. Sage. Recursive generalised-least-squares procedure for online identification of process parameters. *Proceedings The Institution of Electrical Engineers, Control & Science*, **116**(12):2057–2062, 1969.

[Haykin 1994] Simon Haykin. *Neural Networks A Comprehensive Foundation*. Macmillan College Publishing Company, Inc., 1994.

[Hénon 1976] M. Hénon. A two dimensional mapping with a strange attractor. *Comm. Math. Phys.*, **50**:69, 1976.

[Hilborn 1994] Robert C. Hilborn. *Chaos and Nonlinear Dynamics, an introduction for scientists and engineers*. Oxford University Press, 1994.

[Hirsch 1989] Morris W. Hirsch. Convergent activation dynamics in continuous time networks. *Neural Networks*, **2**:331–349, 1989.

[Hodgkin and Huxley 1952] A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology (London)*, **117**:500–544, 1952.

[Hoff 1994] Axel A. Hoff. Chaos Control and Neural Classification. *Zeitschrift für Naturforschung, A, Physik Physikalische Chemie Komophysik*, **49**(a):589–593, 1994.

[Hoppensteadt 1989] F.C. Hoppensteadt. Intermittent chaos, self-organization, and learning from synchronous synaptic activity in model neuron networks. *Proceedings of the Natational Academy of Sciences of the United States of America*, **86**:2991–2995, 1989.

[Hornik *et al.* 1989] Kurt Hornik, Maxwell Stinchcombe and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**:356–366, 1989.

[Hunt 1991] E.R. Hunt. Stabilizing high-period orbits in a chaotic system: The diode resonator. *Physical Review Letters*, **67**:1953–1955, 1991.

[John and Amritkar 1994] J. K. John and R. E. Amritkar. Synchronization of unstable orbits using adaptive control. *Physical review E*, **49**(6):4843–4848, 1994.

[Judd and Mees 1998] Kevin Judd and Alistair Mees. Embedding as a modeling problem. *Physica D*, **120**:273–286, 1998.

[Kerr 1985] Thomas H. Kerr. The proper computation of the matrix pseudoinverse and its impact in MVRO filtering. *IEEE Transactions on Aerospace and Electronic Systems*, **21**(5):711–724, 1985.

[Klee 1966] V. Klee. Convex polytopes and linear programming. In *Proceedings of the IBM Scientific Computing Symposium: Combinatorial Problems*, pages 123–158. IBM, Armonk, N.Y., 1966.

[Končar 1997] N. Končar. *Optimisation methodologies for direct inverse neurocontrol*. Ph.D. thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, UK, 1997.

[Končar and Jones 1995] N. Končar and Antonia J. Jones. Adaptive real-time neural network attitude control of chaotic satellite motion. In *SPIE International Symposium: Applications and Science of Artificial Neural Networks*, volume 2492, pages 112–123, 1995. ISBN 0-8194-1845-5.

[Kürten and Clark 1986] K.E. Kürten and J.W. Clark. Chaos in neural systems. *Physics Letter A*, **114**:413–418, 1986.

[Lai *et al.* 1993] Ying-Cheng Lai, Mingzhou Ding and Celso Grebogi. Controlling Hamiltonian chaos. *Physics Review E*, **47**(1):86–92, 1993.

[Lai and Grebogi 1994] Ying-Cheng Lai and Celso Grebogi. Synchronization of spatiotemporal chaotic systems by feedback control. *Physics Review E*, **50**(3):1894–1899, 1994.

[Lapedes and Farber 1988] A. Lapedes and R. Farber. How neural nets work. In D. Anderson, editor, *Neural Information Processing Systems*. American Institute of Physics, New York, 1988.

[Lee and Wong 1980] D.T. Lee and C.K. Wong. Quintary trees: a file structure for multidimensional database system. *ACM Transactions on Database Systems*, **5**:339–353, 1980.

[Leipnik and Newton 1981] R.B. Leipnik and T.A. Newton. Double strange attractors in rigid body motion with linear feedback control. *Physics Letters A*, **86**:63–67, 1981.

[Lewis and Simo 1994] D. Lewis and J.C. Simo. Conserving algorithms for the dynamics of Hamiltonian systems on Lie groups. *J. Nonlinearity Sci*, **4**:253–299, 1994.

[Lourenço and Babloyantz 1994] C. Lourenço and A. Babloyantz. Control of chaos in networks with delay: A model for synchronization of cortical tissue. *Neural Computation*, **6**:1141–1154, 1994.

[Lueker 1978] G.S. Lueker. A data structure for orthogonal range queries. In *Proceedings of 19th Annual IEEE Symposium of Foundation of Computer Science*, pages 28–34, 1978.

[Maeda and Mutata 1984] Junji Maeda and Kazumi Mutata. Adaptive regularizations in an iterative regularized pseudoinverse method. *Optics Communications*, **51**(6):382–385, 1984.

[Mandelbrot 1982] B.B. Mandelbrot. *The Fractal Geometry of Nature*. Freeman, San Francisco, 1982.

[Marcus and Westervelt 1989] C.M. Marcus and R.M. Westervelt. Dynamics of iterated-map neural networks. *Physical Review A*, **40**:501–504, 1989.

[Margetts 1996] Steve Margetts. Nearest neighbours using kd-trees. Technical report, University of Wales, Cardiff, 1996. Available with source code from `http://www.mathsource.com/`, document number 0208-471.

[Masters 1993] Timothy Masters. *Practical Neural Network Recipes in C++*. Academic Press, 1993.

[Matías and Güémez 1994] M.A. Matías and J. Güémez. Stabilization of chaos by proportional pulses in the system variables. *Physical Review Letters*, **72**(10):1455–1458, 1994.

[Matías and Güémez 1996] M.A. Matías and J. Güémez. Chaos suppression in flows using proportional pulses in the system variables. *Physical Review E*, **54**(1):198–209, 1996.

[Matoušek and Schwarzkopf 1993] J. Matoušek and O. Schwarzkopf. On ray shooting convex polytopes. *Discrete Computational Geometry*, **10**:215–232, 1993.

[Maus 1984] A. Maus. Delaunay triangulation and convex hull of n points in expected linear time. *BIT*, **24**, 1984.

[Melzak 1979] Z.A. Melzak. Multi-indexing and multiple clustering. *Mathematical Proceedings of the Cambridge Philosophical Society*, **86**:313–337, 1979.

[Meyer 1966] G. Meyer. On the use of Euler's theorem on rotations for the synthesis of attitude control systems. Technical Report TN D-3643, NASA, Ames Res. Cen., Moffet Field, CS., 1966.

[Moss 1994] Frank Moss. Chaos under control. *NATURE*, **370**:596–597, 1994.

[Namajūnas *et al.* 1995] A. Namajūnas, K. Pyragas and A. Tamaševičius. Stabilization of an unstable steady state in a Mackey-Glass system. *Physics Letters A*, **204**:255–262, 1995.

[Noble and Daniel 1979] B. Noble and J.W. Daniel. *Applied Linear Algebra*. Prentice-Hall, Englewood Cliffs, N.J., 1979.

[Oja 1983] E. Oja. *Subspace Methods of Pattern Recognition*. Letchworth, UK: Research Studies Press, 1983.

[Oketani and Ushio 1996] N. Oketani and T. Ushio. Chaotic synchronization of globally coupled maps with an application in communication. *International Journal of Bifurcation and Chaos*, **6**(11):2145–2152, 1996.

[Oliveira 1999] Ana Guedes de Oliveira. *Synchronization of Chaos and Applications to Secure Communications*. Ph.D. thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, U.K., 1999.

[Oliveira and Jones 1997] Ana Guedes de Oliveira and Antonia J. Jones. Synchronisatoin of chaotic trajectories using parameter control. In F.L. Chernousko and A.L. Fradkov, editors, *Proceedings of the First International Conference on Control of Oscillations and Chaos (COC'97)*, volume 1, pages 46–49. St. Peterburg, Russia, 1997. ISBN 0-7803-4247-X (softbound), ISBN 0-7803-4248-8 (microfiche), IEEE Catalog Number 97TH8329.

[Oliveira and Jones 1998] Ana Guedes de Oliveira and Antonia J. Jones. Synchronisation of chaotic maps by feedback control and application to secure communications using neural networks. *International Journal of Bifurcation and Chaos*, **8**(11):2225–2237, 1998.

[Oliveira *et al.* 1997] Ana Guedes de Oliveira, Alban Pui Man Tsui and Antonia J. Jones. Using a neural network to calculate the sensitivity vectors in synchronisation of chaotic maps. In *Proceedings 1997 International Symposium on Nonlinear Theory and its Applications (NOLTA'97)*, volume 1, pages 105–108. Research Society of Nonlinear Theory and its Applications, IEICE, Honolulu, U.S.A., 1997.

[Oliveira *et al.* 1999] Ana Guedes de Oliveira, Alban Pui Man Tsui and Antonia J. Jones. A simple method to decode binary messages masked by chaos., 1999. To be published.

[Otani and Jones 1997a] Masayuki Otani and Antonia J. Jones. Automated embedding and the creep phenomenon in chaotic time series, 1997a. Submitted to Physica D.

[Otani and Jones 1997b] Masayuki Otani and Antonia J. Jones. Guiding chaotic orbits. Research report, Dept. Computer Science, University of Wales, Cardiff, Dept. Computer Science, Univ. of Wales, PO Box 916, Cardiff CF2 3XF, UK, 1997b.

[Ott *et al.* 1990] E. Ott, C. Grebogi and J.A. Yorke. Controlling chaos. *Physical Review Letters*, **64**(11):1196–1199, 1990.

[Ott 1993] Edward Ott. *Chaos in Dynamical Systems*. Cambridge University Press, 1993.

[Packard *et al.* 1980] N.H. Packard, J.P. Crutchfield, J.D. Farmer and R.S. Shaw. Geometry from a time series. *Physics Review Letters*, **45**:712, 1980.

[Parker and Chua 1992] T.S. Parker and L.O. Chua. *Practical Numerical Algorithms for Chaotic Systems*. Springer-Verlag, New York, 1992.

[Parlitz *et al.* 1992] U. Parlitz, L.O. Chua, Lj. Kocarev, K.S. Halle and A. Shang. Transmission of digital signals by chaotic synchronization. *International Journal of Bifurcation and Chaos*, **2**(4):973–977, 1992.

[Pecora *et al.* 1997] L. Pecora, T. Carroll, G. Johnson and D. Mar. Fundamentals of synchronization in chaotic systems, concepts and applications. *Chaos*, **7**:520–543, 1997.

[Penrose 1955] R. Penrose. A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, **51**:406–413, 1955.

[Penrose 1956] R. Penrose. On best approximate solution of linear matrix equations. *Proceedings of the Cambridge Philosophical Society*, **52**:17–19, 1956.

[Press *et al*. 1992] William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[Pyragas 1992] K. Pyragas. Continuous control of chaos by self-controlling feedback. *Physics Letters A*, **170**:421–428, 1992.

[Pyragas 1993] K. Pyragas. Predictable chaos in slightly perturbed unpredictable chaotic systems. *Physics Letters A*, **181**:203–210, 1993.

[Pyragas and Tamaševičius 1993] K. Pyragas and A. Tamaševičius. Experimental control of chaos by delayed self-controlling feedback. *Physics Letters A*, **180**:99–102, 1993.

[Qu *et al.* 1993] Zhilin Qu, Gang Hu and Benkun Ma. Controlling chaos via continuous feedback. *Physics Letters A*, **178**:265–270, 1993.

[Rawlings 1988] John O. Rawlings. *Applied Regression Analysis, A Research Tool*. The Wadsworth & Brooks/Cole Statistics/Probability Series. Wadsworth & Brooks/Cole Advanced Books & Software, A Division of Wadsworth, Inc., 1988.

[Reyl *et al*. 1993] C. Reyl, L. Flepp, R. Badii and E. Brun. Control of NMR-laser chaos in high-dimensional embedding space. *Physical Review E*, **47**(1):267–272, 1993.

[Rosenstein *et al.* 1994] M.T. Rosenstein, J.J. Colins and C.J. de Luca. Reconstruction expansion as a geometry-based framework for choosing proper delay times. *Physica D*, **73**:82–98, 1994.

[Rössler 1976] O.E. Rössler. An equation for continuous chaos. *Physics Letters A*, **57**:397–398, 1976.

[Roy *et al.* 1992] R. Roy, T.W. Murphy, T.D. Maiez, Z. Gills and E.R. Hunt. Dynamical control of a chaotic laser: Experimental stabilization of a globally coupled system. *Physical Review Letters*, **68**(9):1259–1262, 1992.

[Roy and Thornbur 1994] Rajarshi Roy and K. Scott Thornbur, Jr. Experimental synchronization of chaotic lasers. *Physical Review Letters*, **72**(13):2009–2012, 1994.

[Ruelle and Takens 1971] D. Ruelle and F. Takens. On the nature of turbulence. *Communication of Mathematical Physics*, **20**:17, 1971.

[Sano and Sawada 1985] M. Sano and Y. Sawada. Measurement of the Lyapunov spectrum from a chaotic time series. *Physical Review Letters*, **55**(10):1082–1085, 1985.

[Schmelcher and Diakonos 1997] P. Schmelcher and F.K. Diakonos. Detecting periodic orbits of chaotic dynamical systems. *Physical Review Letters*, **78**(25):4733–4736, 1997.

[Schwartz and Triandaf 1994] I.B. Schwartz and I. Triandaf. Controlling unstable states in reaction-diffusion systems modelled by time series. *Physical Review E*, **50**(4):2548–2552, 1994.

[Seidel 1991] R. Seidel. Exact upper bounds for the number of faces in d-dimensional Voronoi diagram. In P. Gritzmann and B. Sturmfels, editors, *Applied Geometry and Discrete Mathematics – The Victor Klee Festschrift*, DIMACS Series in Discrete Mathematics an Theoretical Computer Science, pages 517–529. American Mathematics Society, Providence, RI, 1991.

[Sepulchre and Babloyantz 1993] J.A. Sepulchre and A. Babloyantz. Controlling chaos in a network of oscillators. *Physical Review E*, **48**(2):945–950, 1993.

[Shinbrot *et al.* 1992] T. Shinbrot, C. Grebogi, E. Ott and J.A. Yorke. Using chaos to target stationary states of flows. *Physics Letters A*, **169**:349–354, 1992.

[Short 1994] K. M. Short. Steps toward unmasking secure communications. *International Journal of Bifurcation and Chaos*, **4**(4):959–977, 1994.

[Short 1997] K. M. Short. Signal extraction from chaotic communications. *International Journal of Bifurcation and Chaos*, **7**(7):1579–1597, 1997.

[So and Ott 1995] Paul So and Edward Ott. Controlling chaos using time delay coordinates via stabilization of periodic orbits. *Physical Review E*, **51**(4):2955–2962, 1995.

[Solé and de la Prida 1995] Ricard V. Solé and Liset Menéndez de la Prida. Controlling chaos in discrete neural networks. *Physics Letters A*, **199**:65–69, 1995.

[Sparrow 1982] Colin Sparrow. *The Lorenz Equations: Bifurcations, Chaos and Strange Attractors*. Springer-Verlag New York, 1982.

[Stefánsson *et al.* 1997] Aðalbjörn Stefánsson, N. Končar and Antonia J. Jones. A note on the Gamma test. *Neural Computing & Applications*, **5**:131–133, 1997.

[Stewart 1992] Ian Stewart. Introduction to Nonlinear Oscillator. In J.G. Taylor and C.L.T. Mannion, editors, *Coupled Oscillating Neurons*, Perspectives in Neural Computing, pages 1–20. Springer-Verlag, 1992.

[Su 1994] Peter Su. *Efficient parallel Algorithms for Closest Point Problems*. Ph.D. thesis, Department of Mathematics and Computer Science, Dartmouth College, Hanonver, New Hampshire, 1994. Technical Report PCS-TR94-238.

[Su and Drysdale 1997] Peter Su and Robert L. Scot Drysdale. A comparison of sequential Delaunay triangulation algorithms. *Computational Geometry, Theory and Applications*, **7**:361–385, 1997. Personal communication via email (psu@jprc.com).

[Takens 1981] F. Takens. Detecting strange attractors in turbulence. In D.A. Rand and L.S. Young, editors, *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 366–381. Springer-Verlag, 1981.

[Telfer and Casasent 1989] Brian Telfer and David Casasent. Updating optical pseudoinverse associative memories. *Applied Optics*, **28**(13):2518–2528, 1989.

[Tsuda 1992] Ichiro Tsuda. Dynamic link of memory - chaotic memory map in nonequilibrium neural networks. *Neural Networks*, **5**:313–326, 1992.

[Tsui and Jones 1997] Alban P.M. Tsui and Antonia J. Jones. Parameter choices for control of a chaotic neural network. In F.C. Morabito, editor, *Advances in Intelligent Systems*, volume 41 of *Frontiers in Artificial Intelligence and Applications*, pages 118–123. AMSE, SIGEF and BUFSA, IOS Press, University of Reggio Calabria, Italy, 1997.

[Tsui and Jones 1999a] Alban P.M. Tsui and Antonia J. Jones. The control of higher dimensional chaos: comparative results for the chaotic satellite attitude control problem, 1999a. Forthcoming publication in Physica D.

[Tsui and Jones 1999b] Alban P.M. Tsui and Antonia J. Jones. Periodic response to external stimulation of a chaotic neural network with delayed feedback. *International Journal of Bifurcation and Chaos*, **9**(4):713–722, 1999b.

[Wang 1991] Xin Wang. Period-doubling to chaos in a simple neural network. In *Proceedings of IIJCNN-91-Seattle*, volume II, pages 333–339, 1991.

[Weigend *et al.* 1990] Andreas S. Weigend, Bernardo A. Huberman and David E. Rumelhart. Predicting the future: a connectionist approach. *International Journal of Neural Systems*, **1**:193–209, 1990.

[Welstead 1991] S. Welstead. Multilayer feedforward networks can learn strange attractors. In *Proceedings of IIJCNN-91-Seattle*, volume II, pages 139–144, 1991.

Index