Real Time Recursive Block Parameter Estimation of Second Order Systems

Christopher K. Goodwin B.Sc, M.Phil

July 1997

This thesis is submitted to The Nottingham Trent University in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

"Truth is our currency." — *Martin Bell*

Abstract

Many real world dynamic systems can be approximated well using second order systems. It is often required, therefore, in engineering and other situations to deterimine the characterizing parameters of observed data, with the assumption that the data represents a second order system.

This study investigates the parameter estimation problem encompassing a wide range of techniques and algorithms. Conventional approaches are tested and in some cases combined to produce hybrid algorithms. Two novel methods are also applied, and compared with the other techniques. These novel methods are neural networks and genetic algorithms.

Further, a new algorithm is proposed which is applied to all techniques tested. This new algorithm adaptively adjusts the sampling frequency at which observed data is read, based on previous estimates of the parameters. It is shown that this improves the accuracy of the parameter estimation process.

A complete simulation environment is devised enabling parameter estimation to be tested under a range of situations. Firstly, when the system parameters are constant with time. Then secondly, when the parameters vary through the time period of the observed data. The simulation enables the parameters to be estimated in blocks of data. Further enhancement of the algorithms enable them to perform recursively, taking account of previous block's estimates. Finally, all algorithms are tested on their tolerance to two types of noise. The complete simulation allows recursive block parameter estimation which adaptively varies the sampling frequency to increase the

accuracy of the estimation, under a range of noise conditions.

Acknowledgements

When I started my studies towards this thesis, I had only recently arrived in the East Midlands. I would like to thank my supervisor, Dr. David Al-Dabass, for giving me the opportunity to take the student bursary position; I hope he has no cause to regret it. The award gave me a stable background for making my way in a new and wonderful part of the country.

Further thanks must go to Dr. David Al-Dabass for his contribution, both in ideas and experience, throughout the period of the study. I would also like to thank several of the other lecturers in the Computing Department. Dr. Dominic Palmer-Brown has always been enthusiastic and inspiring as well as an important source of ideas. Thanks also go to Pete Halstead, Pete Thomas and others for their lucid explanations at times of my own consternation.

The support staff in the department, both technical and administrative, deserve particular thanks. Without the technical support which enabled both the local and global computer networking I would be significantly less well-informed. Notable names include Tim Gittins, Pat Hamilton, Chris Noble and John Haslam. They have always done their best to help, even when the backlog of work had no end in sight. The secretaries too have always done their utmost to help and make my time as straightforward as possible.

One of the most important group of people to thank are my peers — those who started their journey in post-graduate research in the department at the same time as myself, plus those who were already present when I joined and those who started

after. I would like to thank Andy "Norm" Cheetham for showing me the ropes and for valuable discussions. Thanks to Simon Treavis and Nick Burton for the musical education I received. I now know what I don't like! Thanks also to Stu Barker — your music was better — plus I hope we were able to sort out some problems together. Thanks also to Jonny "radiation leak" Tepper — I hope my grammars okay! Finally, to Chris Roadknight, thanks for the supply of doughnuts, cyclomania and unerring help in uncountable ways in producing this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A Stereo Camera System

1.1.1 Introduction and Applications

To see the world around us, we have two eyes. This allows us to pinpoint the position of something we can see. Two eyes are needed to eliminate any ambiguity about the object's position. So it is too with a vision system based on a pair of cameras, or a stereo camera pair.

With such a stereo camera system it becomes theoretically possible for a machine to locate itself within a three dimensional world. There are many machines and robots which use vision based on a single camera, but these systems always dedicate their attention to one plane, and in this manner allow accurate location tasks to be performed. A stereo camera vision system is not limited to one plane, and is capable of shifting attention from one point to any other.

Machines and robots with a stereo vision system would be able to locate objects' positions relative to themselves, and hence locate itself in the world. If the machine or robot were able to process the image information it received it would become an autonomous robot, navigating itself unaided. This is the aim of developing a stereo camera vision system.

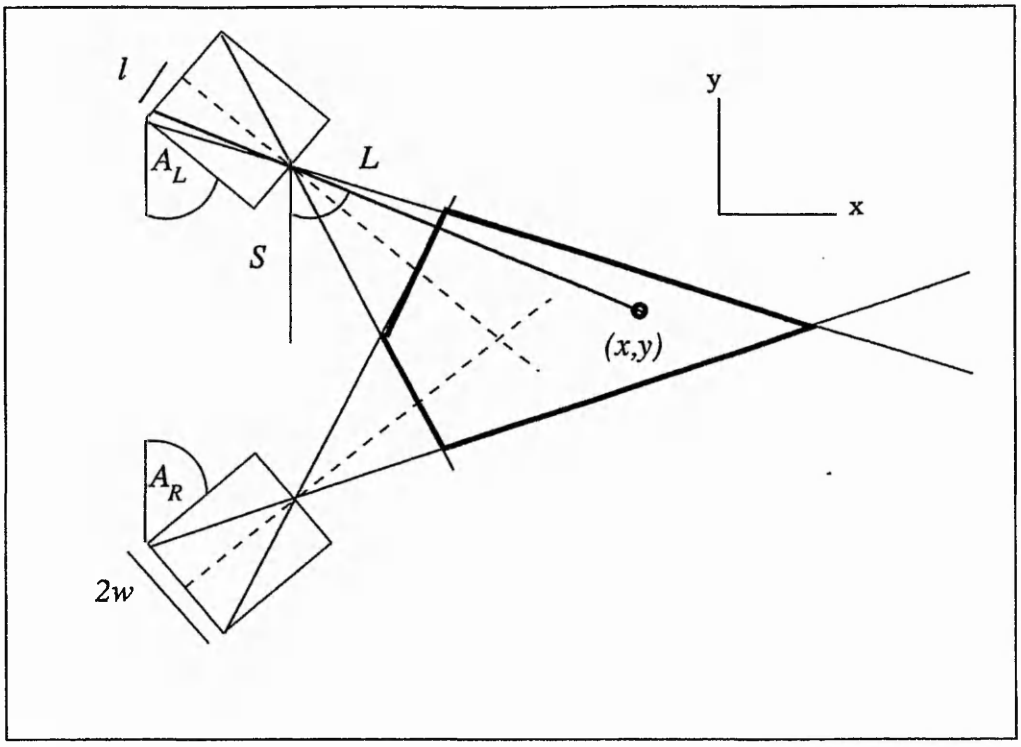


Figure 1.1: *The stereo camera system, viewing an object in the xy plane.*

1.1.2 Geometry of A Stereo Camera System

Notation used in this section follows that of [1]. Consider the two camera system depicted in Figure 1.1. The figure is in the xy-plane with the origin and axes shown. Each camera is assumed to be simply a box, with a single lens at the entrance opening onto a screen on which the image is focused. The lens of the left and right cameras are placed at $+S$ and $-S$ on the y-axis, respectively. The left camera is at an angle A_L and the right camera at an angle A_R to the y-axis. Each camera's focal length is f, and the screen width is $2w$. The straight-through view for each camera will pass through the centre of the lens and end at the centre of the screen.

The field of view of each camera depends on the length f and width w. In most adjustments these overlap, creating an area in the xy-plane common to each camera's field of view. It is points within this area which may be pinpointed in the camera reference frame.

Given a point within the field of view of both cameras it is the aim to determine its co-ordinates (x, y). Each camera will produce an image of the scene on its screens. The point on the left camera's screen will be displaced from the centre of the screen by a distance l, and the point's image on the right camera's screen by a distance r.

To determine (x, y) it is necessary to know each camera's position, orientation and dimensions, plus the displacements of the point on each screen.

1.1.3 Point Object Location

For the left hand camera, the line joining the point object and the centre of the lens makes an angle L with the y-axis. Thus,

$$\frac{x}{S-y} = \tan L. \tag{1.1}$$

Similarly, for the right hand camera, the line joining the point object and the camera lens makes an angle R with the y-axis,

$$\frac{x}{S+y} = \tan R \tag{1.2}$$

Solving these equations simultaneously for x and y,

$$x = 2S\left(\frac{\tan L \tan R}{\tan L + \tan R}\right) \tag{1.3}$$

$$y = S\left(\frac{\tan L - \tan R}{\tan L + \tan R}\right) \tag{1.4}$$

Each of these lines will make an angle with the central axis of each camera. For the left hand camera this is θ_L, and for the right hand camera it is θ_R. Obviously,

$$\tan L = \tan(A_L - \theta_L) \tag{1.5}$$

$$\tan R = \tan(A_R - \theta_R) \tag{1.6}$$

But θ_L and θ_R can also be expressed in other terms. From the projections on the back of the cameras, $\tan\theta_L = l/f$ and $\tan\theta_R = r/f$. Inserting these into equations 1.5 and 1.6 and using the identity $\tan(\alpha \pm \beta) = (\tan\alpha \pm \tan\beta)/(1 \mp \tan\alpha\tan\beta)$,

$$\tan L = \frac{f\tan A_L - l}{f + l\tan A_L} \tag{1.7}$$

$$\tan R = \frac{f\tan A_R - r}{f + r\tan A_R}. \tag{1.8}$$

By inserting these equations into equations 1.1 and 1.2, co-ordinates for the position of the point object in the xy plane can be calculated. This calculation is based on knowledge of the camera separation, the camera's angles, the focal length of the camera lenses and the displacements of the point object images on the camera screens.

It is important to define a sign convention here for the angles used. Angles measured from the central axis of the left camera in an *anti-clockwise* direction are *positive*, and those measured *clockwise* are *negative*. For the right hand camera this rule is reversed. It is natural for the displacements l and r to have the same sign as their corresponding angle θ_L and θ_R. This means that for the example of Figure 1.1, the angles A_L, A_R, θ_L and θ_R are all positive. The displacements l and r in this figure are thus also positive.

To calculate the position of the object in the z-direction once its position in the xy plane is known is not difficult. Either camera can be used to do this. The image of the point object in say, the left camera, will be a vertical displacement d_l above the central axis. giving co-ordinates (l, d_l). These can be used to calculate the angle θ_z between the line passing from the object to its image with the horizontal plane. Then z may be obtained from $\tan\theta_z = z/\sqrt{x^2 + (y+S)^2}$. This is presented for completeness and discussion in this work is limited to the xy plane.

One major assumption has been made so far — a method exists for extracting the location of point object on each camera screen. This task is not a simple one. In this work, however, the image processing algorithms needed for this problem are not investigated.

1.1.4 Point Object Tracking and Noise

With a stereo camera vision system as described above, it is possible to accurately determine the position of a point object which is in the field of view of both cameras. An image from each camera can be collected at discrete time intervals, and delivered to an image processing algorithm which extracts the point object displacements. These are then used together with the knowledge of the camera system geometry to calculate the co-ordinates of the point object in the camera reference frame. This process can be performed repeatedly and a series of values for the object's position recorded at discrete time points. The object's trajectory can hence be tracked.

In the real world it is not always possible to obtain completely accurate trajectories. Noise can be introduced at a number of points in the process. Camera vibration, limited pixel resolution and image processing limitations can all lead to noise.

Reducing the corruption due to noise of object trajectories can be addressed with a number of digital signal processing techniques. It is reducing such noise with novel methods that forms part of the work for this thesis.

Camera Vibration

There are many applications of stereoscopic camera systems where noise may be introduced due to camera vibration. Applications most at risk are those where the cameras are mounted on a moving platform or platforms. For example, cameras mounted on a moving vehicle which view the lane boundary markers as the vehicle proceeds along the road will experience vibration from the road surface, engine vibration and buffeting from the air.

If each camera is mounted on a separate platform, vibration of each platform will affect the camera separation, corrupting the geometry information.

A further problem arises with the relatively slow sampling frequency of the cameras. Video cameras operate typically at twenty-five frames per second. Anything moving

within the view of such a camera with oscillatory motion greater than half of this sampling frequency will then not be accurately analysed as aliasing effects will prevail. In this work it is assumed that noise is limited to within the half sampling rate limit.

Limited Pixel Resolution

One source of noise stems from the finite number of pixels present on the camera screen. Each pixel projects a solid angle out into the real world. The image of a point object within this solid angle will fall upon a single pixel. As the point object moves, its image will move across the finite area of a single pixel until it moves enough to pass across onto an adjacent pixel. The object can therefore move by some amount and still fire the same pixel.

The extent of this problem depends on the camera pixel size, and the distance of the object from the camera. The magnitude of the problem can vary, and an example is given here to illustrate its extent.

Consider a camera pair where each camera has 640 pixels across its screen. The focal length of the camera is 50mm and its width is 20mm. Each pixel is then 20mm/640 = 0.03125mm wide. A pixel on the central axis will cover an angle of $\tan^{-1}(0.03125/50) = 0.0358°$ whilst a pixel at the edge of the screen will project an angle 0.0344°.

An object 1m from the camera lens would then need to move 0.64mm off the vision axis to ensure changing pixel, and 0.57mm at the extremity of vision. These values scale up proportionally to the distance of the object from the camera, so an object on the vision axis 1km away has to move 0.64m to ensure changing pixel.

The limited pixel resolution problem causes only minor noise problems. Whether these distances are significant is dependent on the application of the stereoscopic cameras.

1.2 Second Order System

1.2.1 Formulation

The general second order equation for a variable, y, moving through time, t, is,

$$\frac{d^2y}{dt^2} + 2\xi\omega_n\frac{dy}{dt} + \omega_n^2 y = U \tag{1.9}$$

where ξ relates to the damping of the system, w_n relates to the natural frequency with $\omega_n = 2\pi f_n$ and U is the external force on the system. Each parameter is assumed to be independent of time.

A solution of this equation is,

$$y_t = \left(c_1 + \frac{U}{r_1(r_1 - r_2)}\right)e^{r_1 t)} + \left(c_2 - \frac{U}{r_2(r_1 - r_2)}\right)e^{r_2 t} + \frac{U}{r_1 r_2} \tag{1.10}$$

where,

$$r_1 = \frac{-\xi\omega_n + \sqrt{\xi^2\omega_n^2 - 4\omega_n^2}}{2} \tag{1.11}$$

$$r_2 = \frac{-\xi\omega_n - \sqrt{\xi^2\omega_n^2 - 4\omega_n^2}}{2} \tag{1.12}$$

$$c_1 = \frac{-x_{2o} + r_2 x_{1o}}{r_2 - r_1} \tag{1.13}$$

$$c_2 = \frac{x_{2o} - r_1 x_{1o}}{r_2 - r_1} \tag{1.14}$$

and x_{1o} is the initial starting value, and x_{2o} is the initial value of the first derivative of the variable y. These are also known as the state variables of the system. The derivation of this solution is given in Appendix A.

A common example of a second order system is a pendulum. The weight at the end of a line will move with Simple Harmonic Motion (SHM) moving back and forth about a central equilibrium position. In the case of no damping, the angular frequency of the pendulum will equate to the natural frequency w_n of Equation 1.9. When the damping, ξ, has a positive and non-zero value then the amplitude of the pendulum's

swing will decrease with time. The actual frequency, ω_a, would also decrease according to,

$$\omega_a = \sqrt{(1 - \xi^2)}\omega_n \tag{1.15}$$

If the damping has a negative value then the pendulum would actually increase its amplitude. The external force variable, U, is related to the equilibrium position of the pendulum and indicates an offset.

1.2.2 Generality of the Second Order System

The second order equation is important because it can be used in a wide range of situations as an approximation to the actual process. This is especially true for short periods of time. Despite the fact that all of the parameters of the second order system are time independent, it is often applied in situations where one or more of the parameters is *not* time invariant. The approximation is only valid here when the time dependency of the variable over the time sample is small.

1.2.3 Parameter Estimation

Given values for the parameters of a second order equation, the trajectory of the variable y can be calculated for any time t. This trajectory is defined by the values of the parameters used to create it.

The process of parameter estimation aims to perform the opposite of this scenario. Given a trajectory of y through time, what are the values of the parameters that generated this trajectory?

This is far from a trivial problem. Each point on the trajectory requires the solution of the nonlinear Equation 1.10. The values of the parameters which satisfy all the solutions must match. Since there are five unknowns, three parameters and two initial values, it is necessary to use at least five points of the trajectory. Since the equation

is nonlinear, however, well-known solution guaranteed methods are unlikely to work.

There are several "standard" methods for solving this problem. None can guarantee finding a solution, however, and they vary in their speed and accuracy. The entire problem also becomes much more difficult to solve when noise is present on the trajectory, as it can aggravate the algorithms attempts to solve the problem.

1.3 Main Aims

This study works in the context of observing and tracking an object using a stereo camera system. It will examine the problem of determining the characterizing parameters of the object's motion, in the assumption that it is a second order system.

Several methods for performing parameter estimation will be investigated and compared. Novel methods will be tested against conventional methods. Merit will be based on robustness, accuracy and computational load.

An important part of this study will examine the performance and robustness of the parameter estimation algorithms when the incoming trajectories are corrupted with noise. White noise and impulse noise will be used in varying magnitudes, and with impulse noise, different probabilities.

Chapter 2

Review

2.1 Parameter Estimation

The general problem of parameter estimation, or system identification as it is also known, is not new. It has been a fundamental element of engineering and many other fields for many decades, if not hundreds of years.

Given an observed system, parameter estimation attempts to determine the values of parameters which characterize the system. This topic is well covered in many standard text books such as the [17]. Such works cover a wide range of system identification problems.

In this work, the general second order system is taken as the target. A second order system is controlled by three parameters: damping, natural frequency and external input to the system. There are two conventional ways that parameter estimation is tackled in this situation, both of which are tested for comparison in this study.

Firstly, a power series approach is used. Here, a polynomial is fitted to the signal data which is assumed to be of a second order system. Time derivatives of this polynomial can then be calculated. These time derivatives can be inserted into three equations which are derived from the general form of the second order equation, the results of which are values for the characterizing parameters. This method depends on

the polynomial fit being a sufficiently accurate estimate of the time derivatives.

The second conventional method is based on frequency analysis using a Fourier transform. This method returns the dominant frequency of the signal which can be approximated as the natural frequency of the second order system. Some extra manipulation is required to obtain estimates for the other parameters, and this is covered in full detail in Chapter 3.

There are several other conventional methods which can be used for the problem of system identification. These include a simple random guess policy algorithm, a binary search iterative algorithm, the Newton-Raphson root finding algorithm [31, 32] and the Downhill Simplex algorithm [32]. All of these algorithms are iterative and work on a similar principle. Each algorithm in some manner generates an estimate of the parameters. From these it generates a signal based on the second order system, and compares this internally generated signal with the input signal. A measurement of the difference between the two is then formulated and fed back into the algorithm. With this information it can adjust the estimate values in such a way as to reduce the measure of difference between the generated and actual signal. Once the difference falls below a predefined level, the algorithm terminates.

In this work, the Downhill Simplex is compared with other methods and is described in full in Chapter 3. The Genetic Algorithm also used in this work follows a similar basis as just described and is also described fully in the next chapter and reviewed below.

2.2 Vision Systems

The basis of a 3D vision system as described in Chapter 1 is elementary. To implement a working system in practice requires a number of significant other problems to be overcome. These range from suitability of cameras for a given application, through image processing and extraction of image elements, to control and actions to be taken

based on image data. This study examines the feasibility of such a system from the position of processing the collected information.

There are many applications of stereo camera vision systems, not least of which is that for a vehicle control system. This application has been receiving increasingly wider investigation over the last few decades. With the evolution of faster and more powerful processing abilities the image processing algorithms have become more complex and powerful.

Baluja *et al* [5] developed their ALVINN vehicle in the late 80's which used a feedforward neural network (FNN) to process the images obtained from a digital camera mounted on the vehicle. This system was limited, however, since the FNN could not tolerate the presence of noise, or distractors, such as other vehicles, pedestrians and confusing road markings. They developed the use of a simple recurrent neural network which fed the input with past outputs from both the context units and the output units [5]. This proved capable of handling many types of distractor.

A modular system has been developed by Foresti *et al* [18] where each module acts at a certain level of the processing of the image. Each module gives information to those above and below it in the chain. Each module is a knowledge based system requiring *a priori* information to obtain fast results.

Matteucci *et al* in a similar way used a model of the motion to aid in analysing the image sequences of road scenes, and in addition use Kalman filtering to remove noise [27].

A study of the effects of the camera angle and tilt on the noise were carried out by Sohn and Kehtarnavaz [39]. They determined the optimal orientation of the noise to minimize the noise. They also defined a region of the image which was constantly within view. As the camera vibrated the exact zone it viewed would alter, but would remain viewing a similar region. There was a safe region which was always viewed regardless of the vibration. This is an important area of the image since it is only

objects within it that can be constantly tracked.

Noise is becoming a major problem now that the computers monitoring a vehicle's motion by camera are capable of processing several frames a second, meaning the vehicle can travel at higher speeds. The original ALVINN was able to steer the buggy at speeds of only centimetres per second. When vehicles travel at 30mph the vibration would cause a camera pair to be constantly shaken, altering their alignment, and affecting the calculation of the position of objects in view. Zhuang [46] describes how the extrinsic parameters of a stereo camera pair can be automatically calculated when an external object of known dimensions is viewed, such as road markings or telegraph poles.

2.3 Signal Processing and Neural Networks

2.3.1 Digital Signal Processing

Digital signal processing (DSP) has been around for many decades. Many of its techniques are borrowed or adapted from its analogue partner. Many tasks which people used to be able to do with analogue electronics want to be accomplished in the digital domain eg. when designing digital IIR filters, their properties are based on a number of possible analogue equivalents.

One of the great interests in modern DSP is adaptive filtering. Whereas previously filters were passive, not changing their characteristics throughout their use, adaptive filters are dynamic, altering the way they treat signals during use. A target signal is necessary for the filter to be trained with, and in such cases as channel equalization, training can continue when the filter is in use.

Many neural networks can be viewed as nonlinear adaptive filters, although their development did not stem from the DSP sector. Therefore, many of the applications and problems which DSP developers have known about with linear filtering, are also present with neural networks. It has only been in the last few years that the relevance

of neural networks in DSP has been understood by the majority and a huge amount of work published in this area.

Many good texts on DSP exist [22, 30] which detail many of the important aspects of the field, including design of filters, adaptive linear filters and frequency domain processing. Shynk undergoes a tutorial paper [38] in which he covers many IIR adaptive algorithms. He notes the reduction in computational load the use of an IIR filter, rather than an FIR filter, represents. There are several problems with their use, however. Quantification of an adaptive filter's convergence, stability, and susceptibility to local minima and saddle points is discussed. The current understanding of IIR filters means that convergence rates and stability can only be estimated. Use of the Least Mean Square (LMS) algorithm for IIR filtering is noted to be unsatisfactory, but this is when uncorrupted signals are used for training; when noisy data is used, the biasing problem previously encountered lessens significantly. Puskorius and Feldkamp [33] explain why IIR filters, including fully recurrent neural networks, are harder to train, and the algorithms that do exist are more complex than the FIR algorithms. With the arrival of the Extended Kalman Filter (EKF) algorithm, however, they feel that this powerful algorithm should confirm recurrent networks as important dynamical system controllers, and describe their successful simulations with the pole-balancing and bioreactor problems. Wieland [43] describes his use of an algorithm less complex to understand, a genetic algorithm, which he uses to find the weights of recurrent networks used to control a cart balancing a pole. The power of the architecture is demonstrated by reducing the number of inputs from the normal 4, right down to 1 (with zero inputs tried unsuccessfully!). Success was achieved even when a hinged pole was used.

2.3.2 Adaptive Filter Techniques

Widrow, the co-inventor of the LMS algorithm [42], discusses many of the possible applications of adaptive filters in signal processing and control problems — time series prediction, system modelling, inverse modelling, channel equalization, echo cancelling, noise cancelling and inverse control among them [41]. The applications described can use any type of adaptive filter, both linear and nonlinear, and Jundi reviews some uses for neural network adaptive filters [23].

Conell and Xydeas [12] give an excellent paper on their attempts to use adaptive methods to reduce the background noise of traffic and general city noise in the telephone kiosk situation. A reference microphone is used above the kiosk to take in the outside noise. The signal from the speaker's mouthpiece is used as the signal that must be cleaned up. Taking direct noise cancellation use spectral subtraction as the worst method of filtering, they use many other methods to achieve their goal. Noise cancellation is poor due to the poor correlation between the reference and primary signal. Use of the LMS algorithm in both the time and frequency domain achieve good results. Then filtering only the real part of the frequency domain signal improves upon this. Their final system uses an auto-regressive moving average (ARMA) model on each of the LMS weights. They did try neural networks, but networks of a size which could be implemented fast could not achieve sufficient noise reduction.

Rahman *et al* [34] also filtered signals in both the frequency and time domain. They used phase Shift Key (PSK) and Frequency Shift Key (FSK) signals, and found that good results were obtained, with fast convergence of the FNN. As Conell and Xydeas point out, however, there is no real advantage in performing a simple transition to the frequency domain since this process is a linear one and no advantage is thus made. Anderson and Montgomery [2] found that a FNN out-performed an Optimal filter for a chaotic signal, but performed as well as the Optimal filter for two sine signals corrupted with noise. In confirmation with Conell and Xydeas, they noted that neural networks

are not yet implemented in readily available hardware, with the computer simulations being too slow. Other filtering methods which perform just as well can be obtained in dedicated hardware. They state the main advantage of the neural network is in its ability to act with little or no prior knowledge of the signals.

2.3.3 Tapped Delay Lines

Medvedev and Toivonen [29] demonstrate the advantage of using a tapped delay line on each weight in an adaptive filter, in a similar way to Conell and Xydeas found with the ARMA model. Known as Time Delay neural networks (TDNN's) when tapped delay lines are used on each weight in a neural network, these have been used in speech recognition systems.

Fechner [16] demonstrates an advantage of neural networks over Optimal filtering. Sinusoidal signals were corrupted with Gaussian noise, and non-Gaussian noise such as impulse spikes. The neural net and Optimal filters achieved similar results with the Gaussian noise corrupted signals, but the Optimal filter acted badly with the impulse corrupted signals, whilst the neural network was able to cope.

The use of fully recurrent networks, and the simpler Elman network (SRN) [15] and the powerful EKF training algorithm in other areas such as keyword recognition and channel equalization is common [6, 26]. There is a decision to be made by anyone needing the use of adaptive filters — whether the difficulties involved with the use of IIR filters is outweighed by the benefits in performance and compactness.

Genetic algorithms (GA), and the related topics of Evolutionary Programming (EP) or Genetic Programming (GP), have been around for many years. Their usefulness lies in their ability to search large areas of the search space, and for their ability to escape local minima. Critics of GA's claim they are slow and have limited ability to converge to a precise solution, although this latter problem has been addressed successfully in recent works. They also suffer from epistasis, where a change in the chromosome must

occur in two different places simultaneously to be beneficial, and is thus unlikely. There is also vibrant discussion on the best values for such parameters as the population size, which type of cross-over to use and the choice of several other parameters. Beasely *et al* give an excellent overview and introduction to genetic algorithms [7], and also has a further look in more detail at some of the difficulties [8].

2.4 Genetic Algorithms

Recent research has combined the two fields of neural networks and GA's, with their combined attractiveness of both being inspired by the natural world. A review of evolutionary neural networks is given by Yao [45].

Whitley *et al* [40] was possibly the first to use a GA to find the weights of a neural network for several different benchmark problems. In [40] the GA tagged GENITOR is described, which encodes a population of weight matrices in binary form. A similar algorithm was later used by Wieland [43] in his work with recurrent network controllers. Whitley made the conclusion that the accuracy and convergence rate of the GA were respectively proportional and inversely proportional to the population size. Whitley advocated the use of large population sizes, with 6000 chromosomes being typical. This allowed several solutions to the problem to co-exist within the population. Koza [25], however, preferred the use of small populations, between 50 and 100 chromosomes, as well as the use of real number representations. The small population size allowed only one solution to the problem to exist, and rapid convergence when possible was obtained.

Koza, along with Rice [25], went on to develop a GP method that evolved LISP code that found both the weights and the connectivity of a neural network. Whitley also performed this, and both found there were strong improvements in network performance with a tailored connectivity.

Hugo de Garis demonstrated in [14] the versatility of GA's to find weight values for

a variety of network problems. He considered networks which had time independent input and output, and time dependent input and output, as well as the other two combinations. His GenNETS were able to find good solutions for the networks, resulting in a network that controlled a simulated spider that could hunt, eat and flee.

McDonnell and Waagen [28] devised a GA that created recurrent neural networks for time series prediction. Their GA was able to converge very accurately on good solutions, since they used a real number representation, and the operators of cross-over and mutation varied the value of each weight by a decreasing amount as time proceeded. They produced recurrent networks that predicted the Mackey Glass equation [28], and the sun spot data of the last few centuries.

Angeline *et al* [3] devised a GP technique they called GNARLY which constructed neural networks of any configuration and with any number of hidden units. The only limits were set by the number of required input and output units. It used the concept of a temperature to vary the probabilities of the parameters of the GA. Although this is a very general algorithm, there is no reason for it to scale better than any other GA.

Convergence rates of neural networks "trained" by GA's is not as slow as is often expected. Along with their ability to search large areas of the search space, they are a sensible option for many neural net users.

Neural networks is a vast subject. They are used for finding relations between highly nonlinear input and output sequences. Their advantage lies in the lack of *a priori* knowledge about the system, and model systems purely from examples of input and output. For complex applications they can often be more computationally efficient and simpler than alternative methods. Some applications involve speech and word recognition, machine fault diagnosis, moving target classification, financial investment planning, non-rigid body analysis and in medicine [4, 19, 35, 9, 11].

Chapter 3

Adaptive Block Recursive Parameter Estimation Algorithm

3.1 Introduction

In real-time systems data is collected using devices taking measurements from the real world. The data is fed to an analysis module which produces some sort of classification of the input. This can be fed in turn to a controller module which can make a decision based upon the analysis module's output. Often, there is an element of feedback, and the action of the controller affects the outside world. In a real-time system, the modules perform their tasks at a speed which keeps them up to date with the incoming data.

A continuously running parameter estimation algorithm collects data in the same fashion, receiving a data point at known, discrete times. An analysis module then estimates the parameters of this input signal and outputs the estimates. This output may then be passed to a control module which may take specific action depending on the parameter estimates.

In this work, a continuously running recursive block algorithm is proposed. A block of data is read into the input buuffer, and this is then passed in one step to the analysis module. This module outputs estimates for the signal's three characterizing parameters

— damping coefficient, natural frequency and external input. The estimate of the natural frequency is then used to control the sampling frequency at which incoming data is read.

Such an algorithm would need to be able to cope with a large range of parameter values, and would ideally be scale and position invariant. It would need to adapt itself to changing parameter values to maintain optimal estimation accuracy.

Any such system will need to comprise two main phases. The first phase will require an initial estimate of the parameters to be made with no previous knowledge. This is to start the system. This phase then seeds the second phase of the system, which is the continuously running algorithm which uses previous estimates to aid the current estimation.

Each time the algorithm makes an estimate, it will assume the parameters remain constant during the block duration. This will not necessarily be true, and the algorithm will adapt itself to cope with changing parameters.

This chapter describes a scheme for realizing such a system. A hybrid approach is proposed. Section 3.2 gives a detailed breakdown of the proposed algorithm's steps. Individual modules of the algorithm are then detailed in later sections. Section 3.3 gives a description of the adaptive sampling frequency algorithm which allows later processes to work with higher accuracy.

Subsequent sections then give details of the estimation algorithms explored, which are also categorized on whether they can be used for initial or continuously running parameter estimation. All algorithms can be used for initialization, but not all can be used with an element of feedback in a continuously running mode.

Some of the algorithms determine the parameters by first estimating the time derivatives of the input signal. These include polynomial least squares fitting, difference equations and neural networks. These methods may also be augmented with digital filtering. The remaining methods estimate the parameters directly; these include a

high level approach using signal processing methods, the Downhill Simplex algorithm, neural networks and genetic algorithms.

3.2 Real Time Parameter Estimation

The real-time recursive block parameter estimation system proposed here has several desirable features. Firstly, it adapts the sampling frequency at which data is collected, allowing the parameter estimation methods to operate at high efficiency and high accuracy. This also allows a large range of parameters to be estimated accurately. Secondly, it incorporates recursive estimation algorithms which allow current estimates to consider previous estimates. Since the parameters vary relatively slowly with time, the previous estimates provide useful information.

The steps of the complete algorithm are enumerated below, and shown pictorially in Figure 3.1.

1. Collect the input sequence of length N at the highest sampling frequency f_s allowed by the equipment, which must be at least double that of the highest signal frequency expected since the actual frequency is unknown.

2. Use a non-recursive parameter estimation method to produce an initial estimate of the parameters. Pass these estimates on to the recursive estimation algorithm to initialize it.

3. Based on the frequency estimate, adjust the sampling frequency by a factor $f_d n_c / f_s$. f_d is the dominant frequency and n_c is the desired number of data points per cycle.

4. Collect N data points from the input stream at the new sampling frequency.

5. Use a recursive estimation method to estimate the current parameter values.

6. Return to step 3.

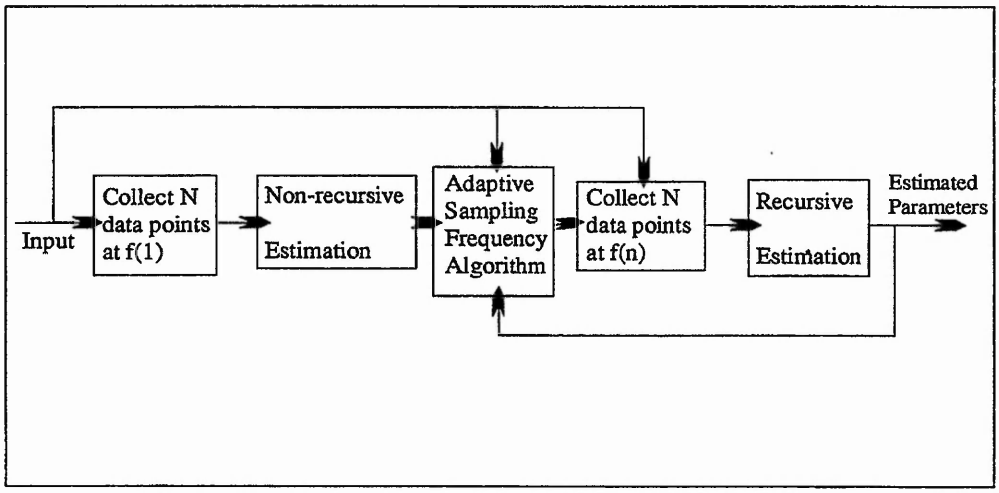


Figure 3.1: *The recursive parameter estimation algorithm.*

At the very beginning of the algorithm, a sampling frequency must be selected which will be able to cope with the incoming signal. To satisfy the Nyquist condition and eliminate aliasing problems, this must be at least double the highest frequency to be detected.

The first use of an estimation module allows an initial estimate of the parameters to be made. The algorithm must be a non-recursive one, since there are no preceding estimates. This will produce only an approximate estimate of the parameters. The frequency estimate, however, is used by the adaptive sampling frequency algorithm to adjust the sampling frequency so that a specified number of data points per cycle of the input is collected during each block. This will allow subsequent estimation algorithms in the recursive blocks to improve their performance. At this stage, a recursive algorithm is created by starting the estimation process using a non-recursive algorithm.

A new block of input data is received by the estimation algorithm from the input stream at the new sampling frequency. This is used to generate estimates using a

recursive algorithm. The first time the recursive estimator is used, it will also use the estimates from the non-recursive estimator, otherwise it uses its own past estimates. The adaptive sampling frequency algorithm can use this algorithms frequency estimate to update the sampling frequency again before the next block of data is received.

3.3 Adaptive Sampling Frequency Algorithm

3.3.1 Introduction

When digital data is received by an input device, the time between data points is accurately known, $\Delta t = 1/f_s$. If the input signal is oscillatory, and its frequency is low compared to that of the sampling frequency, then there will be many data points during each cycle of the signal. Conversely, a high frequency input signal, relative to the sampling frequency, will have only a few data points describing each cycle. A signal of frequency $f_s/4$ for example will have only four data points within each cycle.

At each extreme of low and high, there is either an excess of data points or a shortage, respectively, for efficient accurate analysis. Too many data points per cycle will lead to excessive computation for parameter estimation, although the accuracy can be high. With a small number of data points the speed of analysis will be high, but estimation accuracy will deteriorate.

It is therefore desirable to adjust the sampling frequency to optimize the number of data points collected per cycle to produce a good compromise between excessive computation and accuracy. This is the aim of the adaptive sampling frequency approach.

3.3.2 Algorithm Details

The algorithm will attempt to adjust the sampling frequency, f_s, of the data collection device so that the signal's dominant frequency and the sampling frequency maintain a constant ratio. This can also be viewed as keeping the number of data points per cycle

of the dominant frequency constant, n_c. It will collect a sequence of the input data with N data points. An estimation algorithm is then used to determine the dominant frequency in the signal, and adjust the sampling frequency accordingly. The next N length input sample is then collected.

The following steps are taken:

1. Fill the current input buffer, length N, with the latest input.

2. Make an estimate of the input signal's dominant frequency, f_d.

3. Calculate a scaling factor, λ, such that,

$$\lambda = \frac{f_d n_c}{f_s}$$

   where n_c is the number of points per cycle of the input signal that is desired.

4. Update the sampling frequency by a factor of λ and return to step 1.

When the sampling frequency is at the correct frequency, λ will be unity. For the condition $\lambda = 1$ to be met, f_s/n_c must equal the dominant frequency.

The sampling frequency is adjusted after each sequence of N data points is collected. Other processes which estimate the parameters of the signal will be applied to the same data. These methods will work optimally at a given value of n_c and when λ is near unity. It may take several iterations of this algorithm for the most accurate parameter estimates to be obtained.

3.4 Parameter Estimation Algorithms

Algorithms used to perform parameter estimation of second order systems can be broadly classified as direct or indirect via time derivative estimates, as well as recursive or non-recursive.

3.4.1 Time Derivative Methods

In his paper [1] Al-Dabass describes a method whereby the time derivatives of a signal may be used to determine the parameters of a second order system. This approach is described here.

The general second order system can be written using state variable notation. If $x_1 = x$, $x_2 = dx/dt$, $x_3 = d^2x/dt^2$ and so on, and $a = 2\omega_n\xi$ and $b = \omega_n^2$, then,

$$x_3 + a.x_2 + b.x_1 = U \tag{3.1}$$

is the general second order system. By differentiating this twice and re-arranging, the following equations give values for a and b,

$$a = -\frac{x_5.x_2 - x_4.x_3}{x_4.x_2 - x_3^2} \tag{3.2}$$

$$b = -\frac{x_5.x_3 - x_4^2}{x_3^2 - x_2.x_4} \tag{3.3}$$

and ξ and ω_n are determined from a and b. Equations 3.2 and 3.3 are used in Equation 3.1 to estimate U.

It is therefore possible to obtain parameter estimates via time derivative estimation. This is achieved by several methods described below.

3.4.2 Difference Equations

If no noise is present, and the recorded signal is exactly correct, then derivative estimation poses little problem since difference equations may be used. x_1 is simply the current position, x_n. x_2 is the first derivative or velocity, $(x_n - x_{n-1})/\Delta t$. The second derivative, acceleration, is $(x_{2n} - x_{2n-1})/\Delta t$. And so on (fig. 3.2). Here n is the sample number, and Δt is the time between samples, or $1/f_s$.

Introduction of noise on the signal when using difference equations can be expected to affect the accuracy of estimates substantially. Even small amounts of noise will give

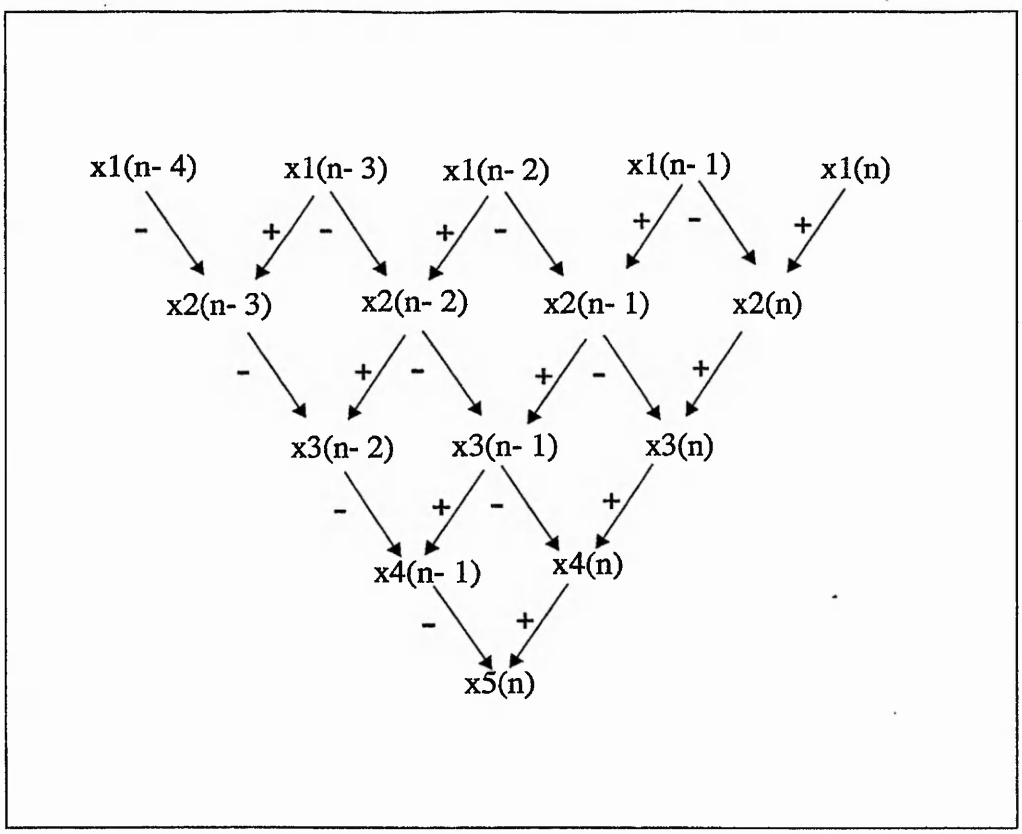


Figure 3.2: *The construction of the time derivatives from difference equations.*

significant errors in the derivative values.

3.4.3 Polynomial Least Squares Fitting

The state variables x_1 through to x_5 can be estimated using a polynomial fitted to the input signal with the least squares criterion. The LS algorithm returns the coefficients of a polynomial of the specified order which fits the target data with the lowest least squares error.

$$x = \sum_{i=0}^{i=O} a_i t^i. \tag{3.4}$$

Differentiating this equation n times gives the $d^n y/dt^n$ time derivative. As long as the order, O, is greater than four, then the state variable x_5 can be evaluated. Further, the derivatives can be estimated at any time by simply choosing the relevant value for t.

Polynomial LS fitting can also act as a smoother when noise is present, performing

lowpass filtering. This can be expected to work particularly well when the adaptive sampling frequency is used since the signal will be low relative to the sampling frequency.

3.4.4 Digital Filtering

Filtering the signal can be used to reduce the noise and allow for more accurate derivative estimation. An ideal filter would be a bandpass filter centred on the natural frequency of the signal. This could be determined by use of a Fourier transform. This ideal situation may, however, not be totally necessary.

Application of the adaptive sampling frequency algorithm always means that the signal's frequency is low relative to the sampling frequency. In this case, lowpass filters can be used with good effect. Filters can include the simple averaging filter, or the digital Butterworth filter.

3.4.5 Neural Networks for Time Derivative Estimation

Neural networks also provide a method for dealing with noise. Feedforward neural networks are capable of performing any nonlinear mapping between their input and output, given sufficient units in their hidden layer. They also possess "generalization" properties which also helps to give accurate answers despite noisy input.

Two configurations are possible: a single neural network is trained to output all of the time derivatives in its output layer, or alternatively, five separate networks can each be trained to output just one derivative each.

Figure 3.3 shows a neural network architecture to estimate only one derivative. A three layered network has an input layer, a hidden layer and an output layer. The input layer consists of a tapped delay line with a one time step delay between units. This layer is the same length as the input window, N. There is also a bias unit for this layer. The input layer is fully connected to the units in the hidden layer, whose output

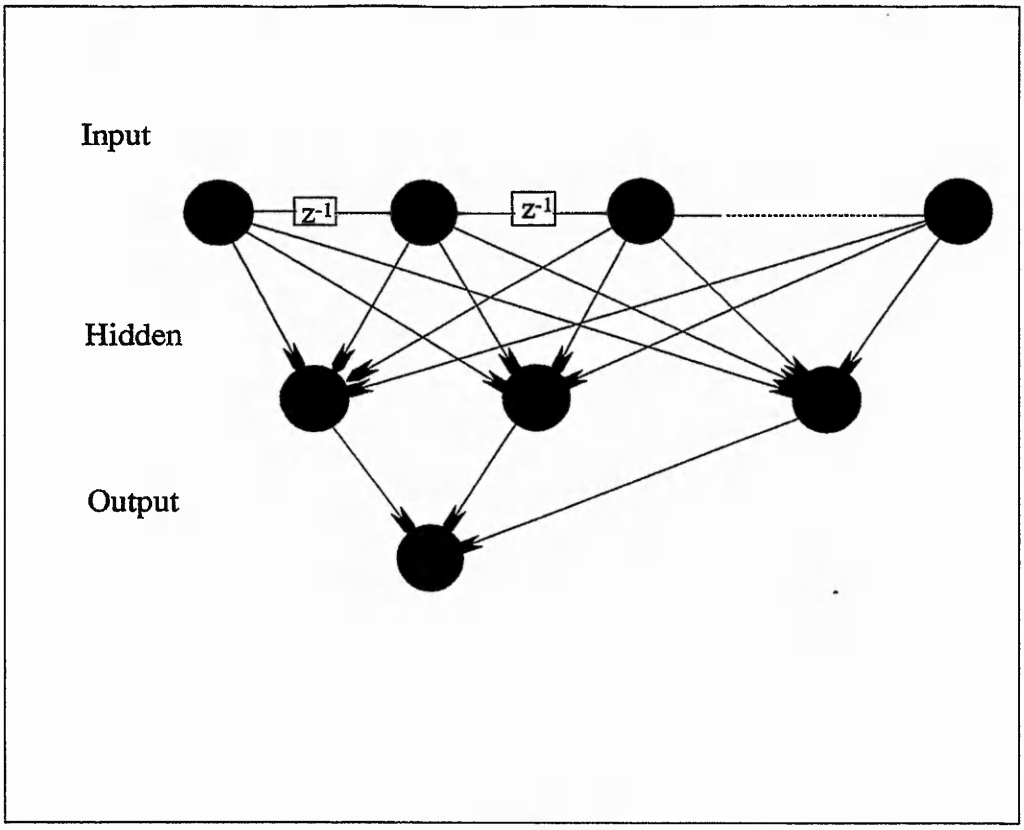


Figure 3.3: *A neural network for time derivative estimation. The input is fed via a tapped delay line. Each output unit corresponds to a desired output estimate.*

is connected to the unit in the output layer. The output unit is trained to output a time derivative. The architecture for a network that estimates all five derivatives will have an additional four output units, each fully connected to the hidden layer above.

The full mode of operation of a neural network is described in detail in Appendix B. Briefly, however, each unit in the input layer attains the value of the input signal. This is then propagated down to the hidden layer via the connections. Each connection has an associated weight value. Each hidden unit sums the input layer via the weighted connections. Each unit then transforms this sum with either a sigmoid or a hyperbolic tangent function, or alternatively, leaves it unchanged. In this latter case, it is termed a "linear" unit. This process is then repeated from the hidden layer to the output layer. The unit(s) of the output layer then describe the estimated derivative(s). This is called a forward pass of the network.

Of course, the result depends upon the weights in the network and it is first neces-

sary to obtain a weight matrix that will produce the desired results for a large range of input patterns. This is done by "training" the network by repeatedly presenting example inputs and slightly adjusting the weights each time to move the output towards a desired value. After many presentations and ensuring that the network is performing well with the training patterns, training can stop and the network used with unchanging weights to estimate derivatives on patterns previously unseen. Network performance will depend on many factors, but primarily on the representative quality of the training data.

The algorithm that actually changes the weights in the training of the network is the backpropagation algorithm. This is described in full in Appendix B.

3.4.6 High Level Parameter Estimation

There are a number of processes that can be performed to obtain the parameters of the second order solution. Recall that the parameters are damping (ξ), natural angular frequency (ω_n) and the external input (U). These are assumed to be constant over the N length sample. Described below are methods for estimating these parameters by extracting features of the signal.

The core of these methods is the Fourier Transform from which a Power Spectral Density (PSD) can be obtained. This describes the magnitude of the $N/2+1$ frequency bins ranging from zero to $f_s/2$Hz.

Damping (ξ)

The observed damping, let it be called Ξ, is a function of both ξ and ω_n A. Especially for low frequency signals, the overall damping will equal $2\xi \times \omega_n$. Therefore, to calculate an estimate for ξ, the observed damping must first be estimated and then divided by the frequency estimate.

Figre 3.4 shows a typical observed signal, along with some of its characterizing

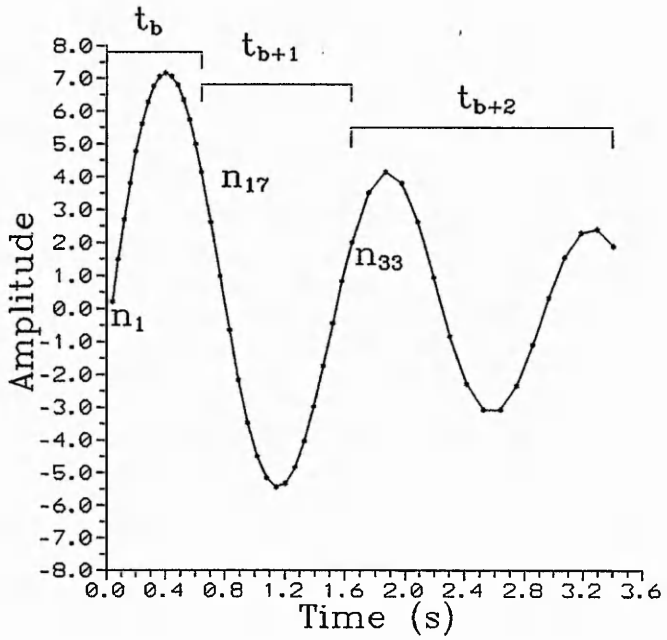


Figure 3.4: *A typical observed input wave, seen over three blocks. Due to the ASFA the time period for each block increases as the sampling frequency is reduced to improve the estimates.*

features. The amplitude in each block is shown, as well as the range of each block. A value for ξ is obtained with successive estimates of the magnitude of the dominant frequency from one block of input signal to the next. If A_b and A_{b+1} are the magnitudes of the dominant frequency in blocks b and $b+1$, ξ is approximated by,

$$\xi = \frac{1}{\hat{\omega_n}(t_b + t_{b+1})/2} \ln \frac{A_b}{A_{b+1}} \tag{3.5}$$

where $\hat{\omega_n}$ is the estimated frequency, and t_b is the duration time of incoming blocks of data. It is necessary to have more than one block to make an estimate for ξ, over which time the sampling frequency, and hence t_b, may change. In this case, the average of the period of each block is used. Note that the better the estimate for frequency, the more accurate the ξ estimate will be.

It is possible to perform a similar operation with a single block, by considering a

block as two equally sized blocks and performing a Fourier transform on each of these. It is intended, however, that the block lengths be kept to a minimum for this study, and as a result, the length will already be short. Any further reduction would increase the error in the frequency estimate above the current level.

Natural Frequency (ω_n)

A Fourier Transform is used in the form of an FFT to obtain the PSD of the input signal. The highest peak in the PSD is taken as the dominant frequency, or $\hat{\omega_n}$. It is important to disregard the first point of the PSD, as this relates to the DC offset of the input signal. Equivalently, the mean of the signal can be subtracted from it before the FFT is applied. This method is limited in its accuracy due to the quantizing effect of the frequency bins. Each frequency bin is of size $f_s/2N$. Increasing the sample length will therefore increase accuracy. Use of the adaptive sampling frequency algorithm will also aid the estimation by adjusting the sampling frequency to keep it in a fixed ra... with the target frequency.

External Input (U)

The external input to the signal manifests itself as an offset to the signal. It also determines the value x will oscillate about and, if damping is present, eventually settle down to. This can be seen by putting x'' and x' to zero in the second order system (where x'' and x' denote the second and first time derivatives of x). The result is,

$$x = \frac{U}{w_n^2}$$

The external input is estimated by first evaluating the mean of the input signal, μ, and also using the estimated frequency so that,

$$U = \mu\omega_n^2.$$

This estimate should work better the longer the input signal length, so that μ can be evaluated more accurately, or when the number of cycles is near an integer value

which is attained with the adaptive sampling frequency algorithm. Further, the greater the sample length, the greater the accuracy of the frequency estimate. If the actual frequency is not constant, as in the time varying case, the frequency estimate will be less accurate, reducing the accuracy for the external input.

3.4.7 Downhill Simplex

This algorithm is an iterative method that progressively reduces an error function starting from an initial guess. It was first devised by Nelder and Mead in 1965 [32], and the description below is based on the Numerical Recipes publication [32]. It is a general method for searching an error space for nonlinear equations.

The Downhill Simplex method requires an initial guess $\boldsymbol{P_o}$, of the parameter vector to be made to seed it. Based on this initial guess, a further set of parameter vectors $\boldsymbol{P_i}$, are generated using Equation 3.6,

$$\boldsymbol{P_i} = \boldsymbol{P_o} + \lambda \boldsymbol{\delta_i} \tag{3.6}$$

where the $M \times M$ size matrix $\boldsymbol{\delta_i}$ has all its elements set to zero except for the ith element which is unity. This equation can be enhanced by changing λ also into an M length vector where each element reflects the expected range of the parameter.

The result is $n+1$, n length vectors, where each vector represents a location in parameter space.A method for visualizing the mechanics of this algorithm uses the geometrical form of the simplex, from the which the algorithm gets its name. A simplex is a form consisting of $n+1$ points plotted in n-dimensional space. A line is then joined from each point of the simplex to every other point, enclosing a volume of the n-dimensional space. In its simplest form, a 2-dimensional space will support a 3-point simplex, or as it is more commonly called, a triangle.

It is the task of the search algorithm to move the simplex through parameter space until the solution is found. This is performed by evaluating an error function at each

point of the simplex. The point is then moved through the opposite face of the simplex. If this does not result in a lower error, then it is moved back by half the distance. If this doesn't improve the error, it is moved away from the centre of the simplex by a factor of two. If still this doesn't work then the entire simplex is reduced in size. By this method, the simplex moves around in parameter space, changing shape and size according to the terrain of the error space, all the time searching for a lower error. It is therefore an error gradient descent method.

Termination of the algorithm can arise from three situations. Firstly, if it has exceeded the maximum number of allowable error function evaluations set by the user. Secondly, if the error of the lowest point falls below a pre-set tolerance level, indicating a successful find. Thirdly, the simplex may collapse upon a local minima, and in so doing, fall below a critical user set size..

Performance of this algorithm is strongly dependent upon the initial conditions since it is a gradient descent algorithm. If the starting point is not close to the global minima, the simplex may descend down a local minima giving a non-optimal solution.

This study attempts to find the values for the three characterizing parameters of a second order system, namely, damping, frequency and external input. These three alone, however, do not define the initial conditions. There needs to be an error function to measure how close the estimated parameters are. It is achieved by generating a signal from the estimated parameters and comparing this with the observed profile. As a result, the two initial conditions, x_{1o} and x_{2o}, must also be estimated by the simplex algorithm to define a unique profile to compare against the observed profile. This adds two dimensions to the optimization task; the alternative is to estimate only the three parameters and sweep through a range of possible values for x_{1o} and x_{2o}. This is a great increase in the computational load.

Comparison between the target and estimated signals is achieved with one of two functions. The first is the standard RMS difference. The second combines the correla-

tion between the two profiles and the difference between their means. Details of this are given in Section 4.6.

3.4.8 Recursive Downhill Simplex

This algorithm can be expected to benefit from introducing a recursive element, allowing it to run usefully in a continuously running mode of operation. It introduces a negligible amount of computational load and can be expected to increase the robustness of the algorithm.

Since performance depends strongly on the accuracy of the initial guess, initializing a run with the estimated parameters from the previous data block will provide a good place to start the search. Although the parameters may be varying in time, and so be different from one block to the next, the variation is slow, and estimates from one block will be relevant to estimates for the next block.

It is also necessary to update the estimates for x_{1_o} and x_{2_o}. This requires that the values for x_1 and x_2 at the end of the estimated signal be calculated and passed on.

3.4.9 Artificial Neural Networks for Parameter Estimation

Artificial Neural networks (ANNs) are a powerful method for performing nonlinear mappings between input and output vectors. It has been shown that they can map any function to an arbitrary accuracy given sufficient units in their hidden layer(s [40].

ANNs can be used for this work by training a network to output an estimate of a parameter given the input signal as the network input. The network is then performing the nonlinear transformation from the input signal trajectory to parameter value i.e. damping coefficient, natural frequency or offset.

Figure 3.3 shows a neural network known as a feedforward neural network (FFNN) which can be used for this task. A tapped delay line of N input units are fully connected via weights to units in a hidden layer. These hidden units are in turn connected via

weights to the unit(s) in the output layer. In addition, bias units which have values fixed at unity feed into the hidden and output layers allowing for a permanent offset for units in the subsequent layer.

As each block of N data points is received, it is fed down the tapped delay line filling the input layer. The units adopt these values as their activations. The activations of the input units are passed down the weighted connections to the units in the hidden layer. Each unit in the hidden layer sums its input, and passes the total through a nonlinear transfer function. This is now the activation of the hidden unit. Once this has been performed for each unit in the hidden layer, the output layer starts receiving input. These units also sum and perform a transfer function. The output unit's activation is taken as the output of the network. This process is described in the equations included in Appendix B.

With an appropriate weight matrix such a network can perform the nonlinear mapping between input signal and, say frequency estimation. Obtaining a weight matrix to perform the desired task is done by "training" the network. The backpropagation algorithm is used to adjust the weights when the network is given examples of the input and output mappings. Training often involves showing the network many examples of the mapping, called a set. Showing the set once to the network is known as an "epoch". Many epochs are often needed before the network generates outputs close to the desired output.

Once training is complete the network can be tested by showing a test set. The networks performance can then be demonstrated with input patterns it has not yet seen. If the performance is good, the network has been trained successfully and can be used with real data. Otherwise, further training for more epochs is necessary.

3.4.10 Recursive Neural Networks

A FFNN only uses current and past inputs to generate its output. Its architecture is non-recursive. This can be used as an initial parameter estimation method, like the high level parameter estimation or time derivative methods, but the architecture can be upgraded to a recursive one by adding some extra connections. This would allow NN's to be used for the block recursive mode.

Figure 3.5 shows a recursive architecture. Connections lead from the output unit(s) of the network to units in the input layer. These connections have fixed weight values of one. Input to the network consists of the tapped delay line input, plus a copy of the last output estimate of the network. This arrangement can be further extended by creating a tapped delay line on the past outputs. Since the parameter values vary relatively slowly over time, the past output estimates, and a possible tapped delay line of past inputs, provide additional useful information for the network.

Recursion can be taken to another level. Connections can be made from the hidden units to input nodes. This arrangement (fig. 3.6) was first used by Elman [15] and allows the input to have knowledge of the previous state of the hidden units. This feedback can give time-context information and it is for reason that Elman used this architecture. Maintaining an input from the previous output of the network gives the network considerable temporal information.

The concept of recursive connections can be taken to its extreme in the form of a fully recurrent neural network (fig. 3.7). Here, a layer of input units feeds into a cluster of fully recurrent units. Each recurrent unit is connected to every other recurrent unit, including itself. One or more of the units is selected to be an output unit. Such a highly recursive network can be compact, but training is more difficult and computationally more expensive than the FFNN.

In both types of recursive network, the important principle is the same. Past information is used to estimate the current time step's estimate. Since the continuously

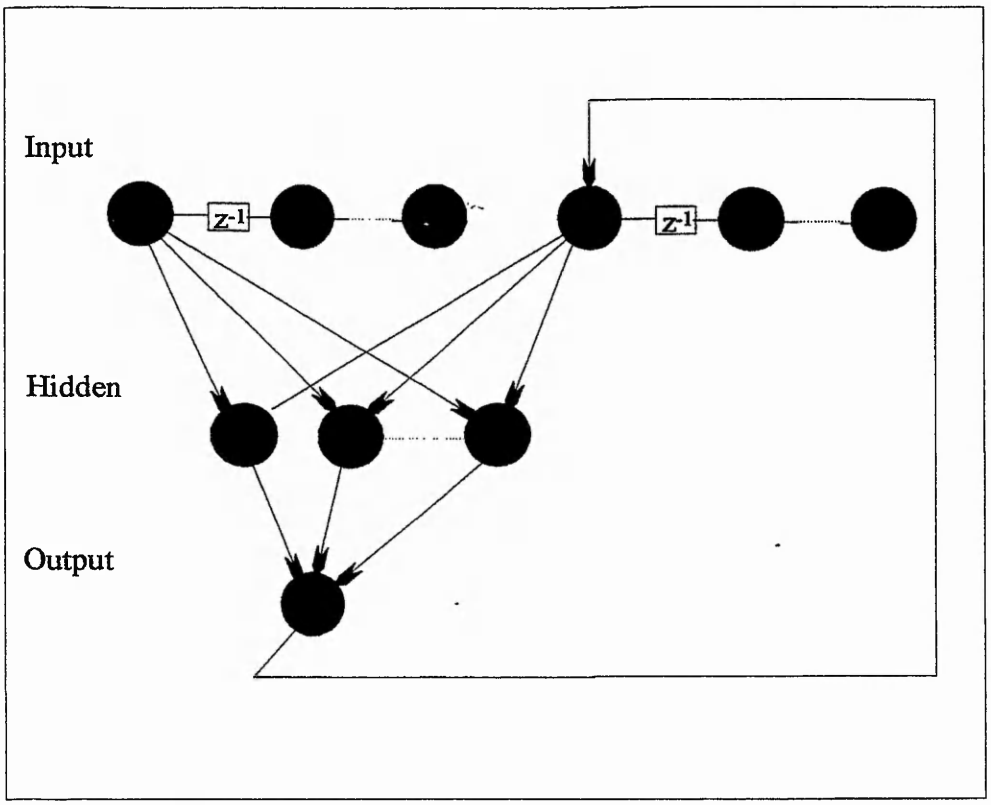


Figure 3.5: *A feedforward network with recursive connections. Input from the tapped delay line plus the previous time step's output is used for the current estimation. (Bias units are not shown in this figure.) This network requires a recursive mode of operation.*

running system assumes that the parameters will have reasonably slowly varying parameters, this information will be useful.

3.4.11 Genetic Algorithms for Parameter Estimation

This section describes the general genetic algorithm (GA). It first describes the basic algorithm, then selects each aspect to discuss in more detail. Difficulties GAs experience are then described.

A General Genetic Algorithm

The genetic algorithm (GA) is a versatile method to perform a stochastic parameter optimization search. GAs are capable of searching a large parameter space, and focus their search for optimal parameters in promising regions of the search space. They have

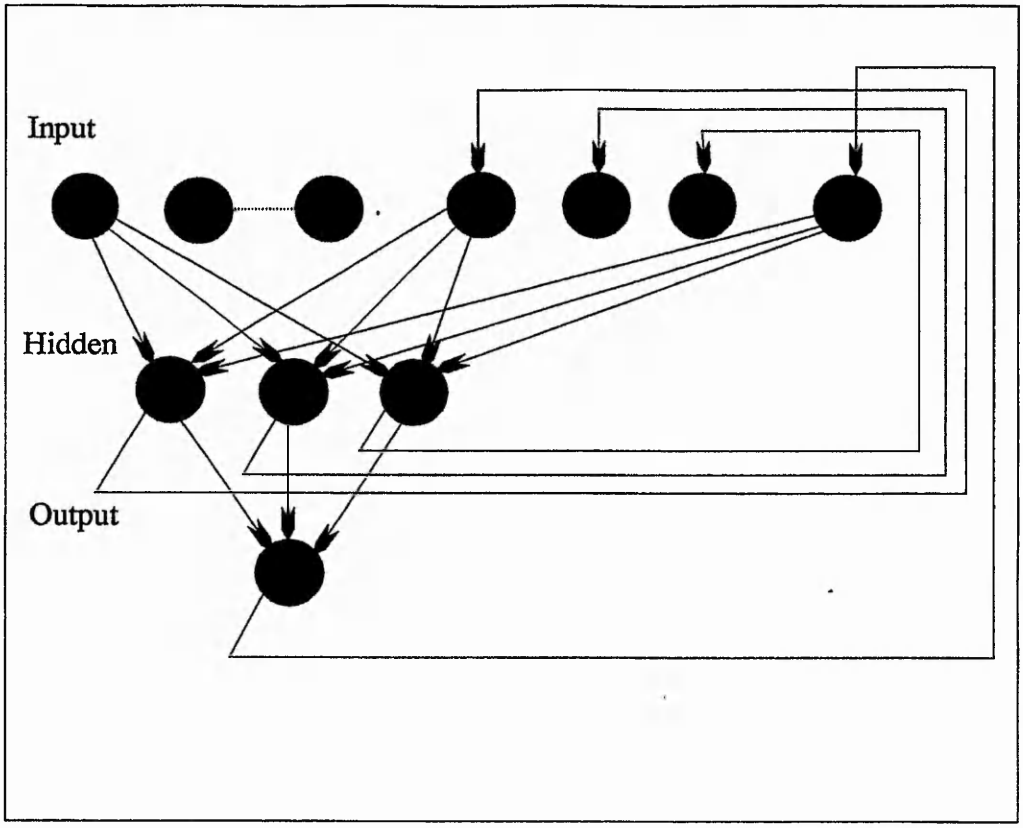


Figure 3.6: *A feedforward neural network with connections from both the hidden units and output unit(s) leading to the input layer. This gives the network temporal information and hence requires a recursive mode of operation.*

been shown to be particularly suited to multimodal solution problems (see sec. 4.6). It is inspired by the concept of evolution, and borrows several ideas from genetic reproduction. These hinge on the three basic genetic operators of reproduction, crossover and mutation.

In brief, a GA creates a population of chromosomes. Each chromosome is an encoding of the parameters to be optimized. The chromosomes are each given a merit rating, depending on how well the decoded parameters suit the problem. This figure of merit is known as a chromosome's "fitness". If a chromosome is denoted by a"C", then

$$F_c = Func(C) \tag{3.7}$$

where F_c is the fitness and $Func$ represents the fitness function. The population

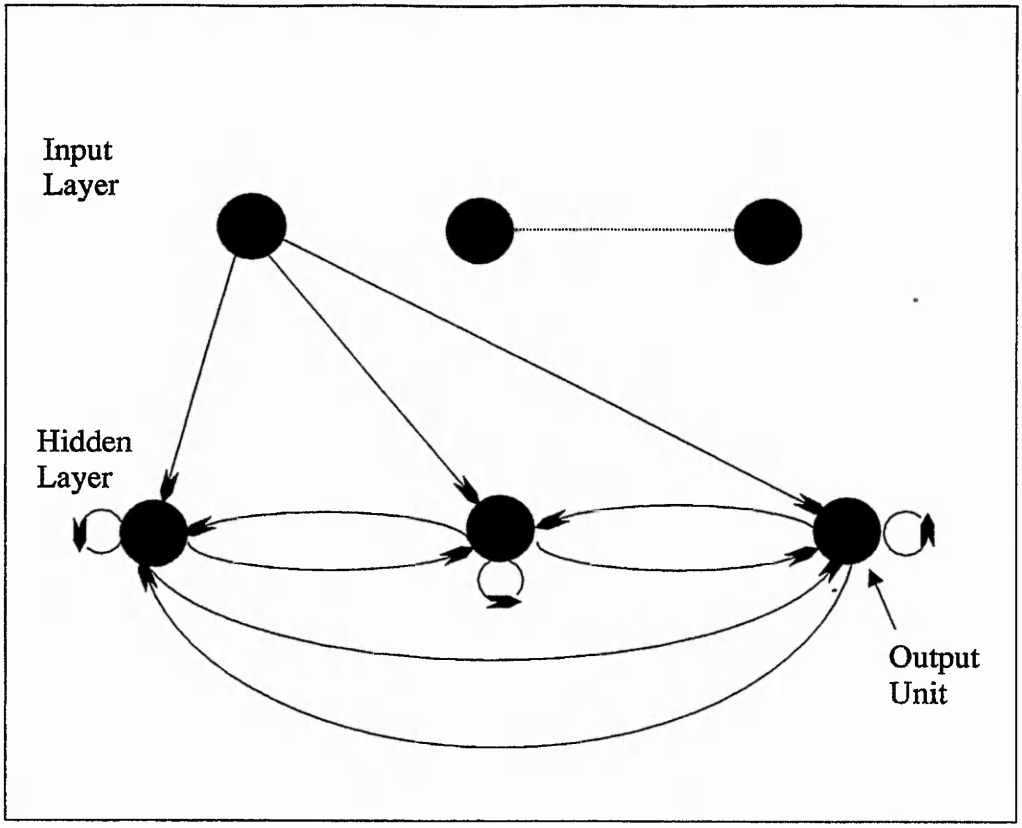


Figure 3.7: *A fully recurrent neural network. The layer of N input units feed into a cluster of fully interconnected units. Any of these units can be selected as an output. This network requires a recursive mode of operation.*

of chromosomes is ranked according to the fitness of each member. Two chromosomes are selected at random, with a preference for fitter chromosomes, to become parents. These combine via a reproduction process to produce two offspring.

$$Reproduction(C_{p1} + C_{p2}) = C_{o1} + C_{o2}$$

where $p1$ and $p2$ indicate the parents, and $o1$ and $o2$ are the offspring. The offspring have their fitness measured, and if they rank, they will be inserted into the population, displacing the weakest member of the population.

This process of selecting parents, creating offspring and ranking them in the population is known as a "generation". Many generations are performed, and with time, the fitness of the population improves with the highest ranking chromosome representing the best solution to date.

The algorithm stops when either a limit of the number of generations is reached, or

a chromosome becomes sufficiently optimal. Good results can be obtained with many variations of this algorithm. They have been shown to be highly robust to a large variation in the controlling parameters, such as population size and choice of genetic operators [24].

Population Initialization

To start a GA, a population must be created. This is usually done purely at random. The number of chromosomes in the population is chosen. Each chromosome is then created at random, and is a string of zeros and ones — a binary encoding of the desired parameters.

The chromosome is a representation of the parameter values to be searched. The number of bits used to encode each parameter must be chosen, and a note kept of the scaling required to eventually decode the chromosome. This may present a limitation for the user since the range a parameter can take must be pre-determined. Each binary string for each parameter is concatenated to form a single long chromosome.

For example, an 8 bit binary string representing one parameter can have a range of values from 00000000 to 11111111, which is decoded to be values between 0 and 255. This will allow the parameter to be described in a range with only 256 different values. A quantizing effect takes place. This may cause problems, and a balance between sufficient accuracy in the parameter's value, and overloading the GA with excessively long chromosomes.

Fitness Function

Each chromosome in the population is then passed through the fitness function. The fitness function gives a figure of merit on how well the decoded chromosome solves the problem.

The process decodes each binary number and scales it for each parameter represented. A pass of the problem in which the parameters are used is now made, and a

fitness is returned based on how well the problem was solved. This fitness is a figure of merit for each chromosome and allows the entire population to be ranked in order of its fitness. The chromosomes which best solve the problem are at the top end of the population.

Parent Selection

The first generation has now been created, and selection for the next generation can now commence. A selection procedure now starts, whereby two chromosomes are selected to act as parents to produce some offspring. The parents are reproduced to form the offspring. Then two other genetic operators, crossover and mutation act on the offspring.

Selection of the parents can be accomplished in many ways. The method used here is called tournament selection — in one round of the tournament a potential parent is selected at random from the population and its ranking in the population noted. A number of rounds are conducted, and the chromosome with the highest ranking goes on to become a parent. The greater the number of rounds, the greater the chance of having a high ranking parent. This emphasis on selecting higher ranking parents is known as the *selection pressure*, and ensures that new solutions are sought for in areas of the parameter space which are already producing good results.

Offspring Production

Once the two parents have been selected, they are copied. These copies are termed the offspring, and these go on to the processes of crossover and mutation.

There are two common types of crossover; one point and two point. In one point crossover, the offspring are placed side by side and a point is chosen at random along their length. At this point a cut is made and the tails of each offspring is swapped with the other i.e. the offspring exchange their latter sections. This is shown in Figure 3.8.

Two point crossover takes each offspring, and makes two cuts in each. The cut-out

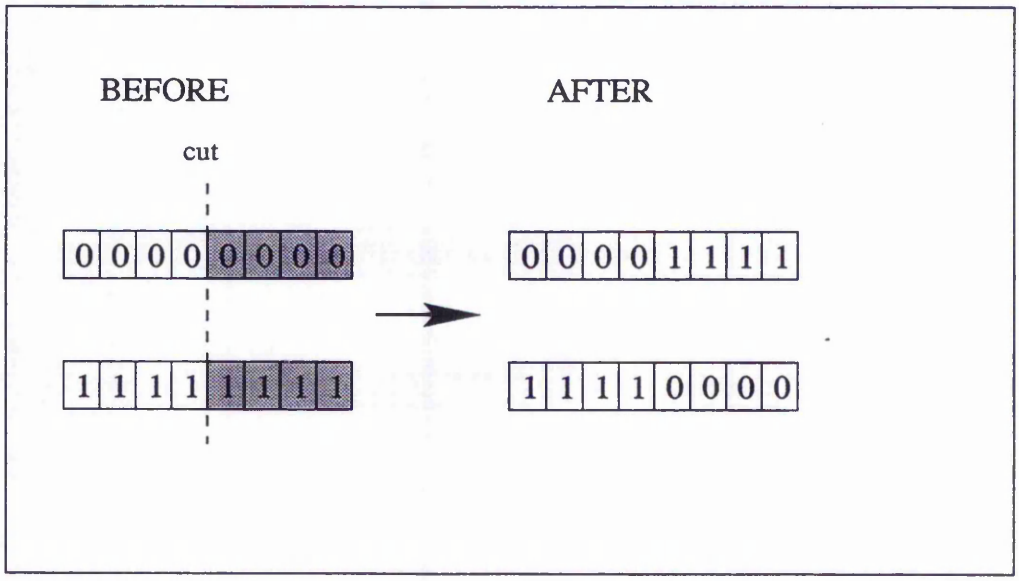


Figure 3.8: *Two offspring undergo one point crossover. A point is chosen along the length of the offspring, and the tails swapped.*

section from each offspring is then swapped (fig. 3.9). The position of the first cut is random, with the length of the removed section being fixed.

Mutation is then applied to each offspring. There is a probability that each bit in the chromosome binary string will be "flipped" i.e. a "1" will become a "0", and vice-versa (fig. 3.10). This probability is usually very low, so that there are only likely to be one or two mutations per offspring, or even less.

Once these genetic operations have been performed for each offspring, each offspring has its fitness measured. If its fitness merits it, the offspring will be inserted into the population with the bottom most member of the population hence being lost, since the population size is always kept at a fixed size. If the fitness of the offspring is worse than the weakest member of the population, it does not enter the population and is lost.

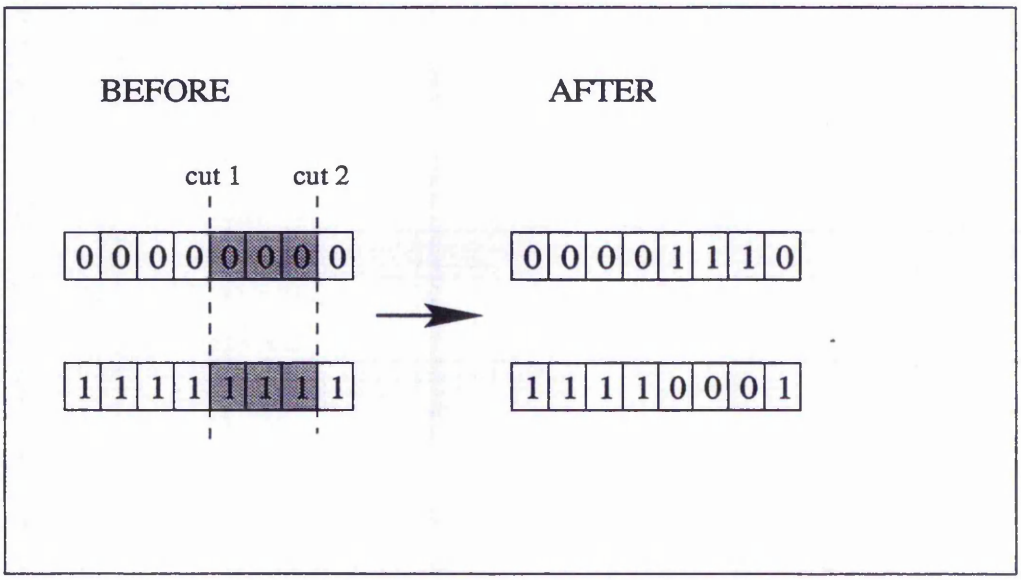


Figure 3.9: *Two offspring undergo two cut crossover. Two points are chosen along the length of the offspring, and the sections of chromosome are swapped between the offspring.*

Generations

The process of selecting parents, creating offspring, and then ranking the offspring constitutes one generation. As the number of generations increases, the weaker chromosomes are removed from the population, and the overall fitness of the population increases. When one particular solution of the problem begins to dominate in the population, the population is said to be converging. Generations continue until one of the termination criteria is fulfilled: i) The fitness of the highest ranking member falls below a pre-set critical value, ii) the maximum number of generations is exceeded, iii) the maximum number of function evaluations is exceeded.

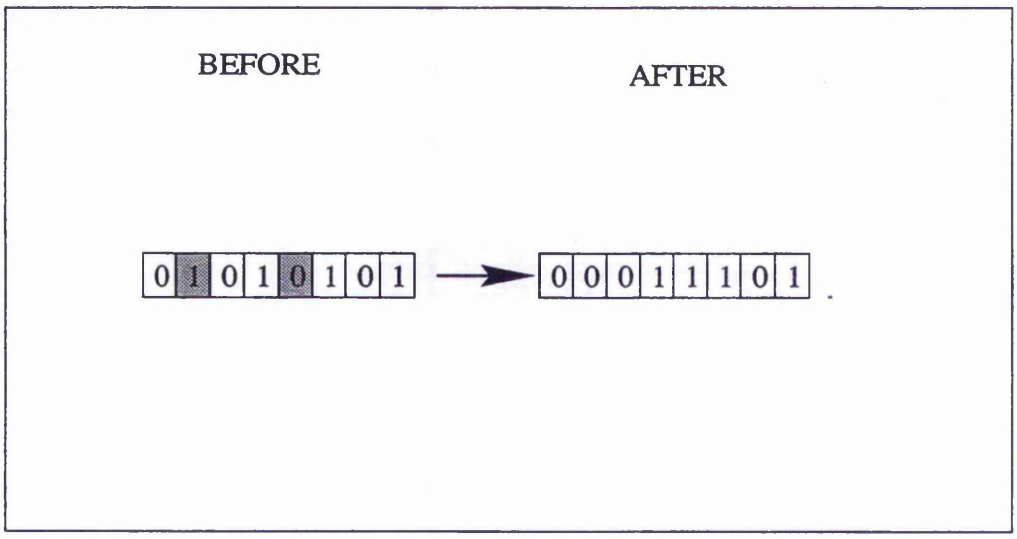


Figure 3.10: *An offspring undergoing mutation. Any point in the chromosome can be reversed.*

Genetic Algorithm Theory

Why does the GA work? How does it achieve its resistance to local minima traps, and why is it particularly suited to multimodal problems? There are several reasons for its success.

Firstly, the initialization where each chromosome is randomly generated gives the GA a wide view of the search space. The larger the population size, the greater this spread.

Crossover is a crucial genetic operator. It allows sections of chromosomes to swap. It has been extensively shown that "building blocks" form in chromosomes[21, 20]. When these blocks are beneficial to a chromosome they are more likely to be passed to the next generation. The crossover operator allows parts or all of building blocks to be swapped between different solutions and to be built upon. It is the creation or perpetration of building blocks that leads to the convergence of the GA.

Finally, the mutation operator allows for resistance to local minima. The random change of a chromosome within the local minima may push it outside and into a new and better minima [7]. It also acts to prevent premature convergence, where the genetic diversity is prematurely lost, preventing a rapid descent of the current minima. As the number of generations increases and the chromosomes become more similar, the mutation operator becomes more important in allowing escape from local minima, and it is common to increase the mutation probability as generations pass.

Variations of the Genetic Algorithm

The genetic algorithm described above is termed a "steady state" genetic algorithm[40]. This is due to the relatively slow manner in which the population will evolve. Since one generation only produces two offspring, the overall difference between adjacent generations is slight. It is also a monotonically reducing algorithm i.e. the fitness of the highest ranked chromosome never decreases.

A common variation of the GA is when a top percentage of the current population is taken, and offspring are created only from this select few until the next generation is once again full, but only with offspring[7]. The new generation is then ranked as before and the process repeats. In terms of generations, this method converges much quicker than the steady state version. There is little difference, however, in the number of offspring created and fitness tested. In this form, this variation is not monotonically reducing since there is no guarantee that any one of the offspring will definitely be fitter than the highest ranking chromosome from the preceding generation. This can simply be remedied by allowing the elite parents to also enter into the next generation.

A different and common selection process is known as "roulette' selection. In this case, the fitness needs to be one which is maximized. Each chromosome selects part of a "roulette wheel", with the amount selected proportional to its fitness function. The fittest chromosome thus takes the largest part of the wheel. A chromosome is then selected by randomly choosing a place on the wheel, and which ever chromosome

covers this point is chosen. This process allows super fit chromosomes to have a much higher chance of being selected than in tournament selection. This is not necessarily desirable, as reproduction based strongly on a single super fit chromosome is likely to force the population into premature convergence, losing genetic diversity and increasing the chance of a local minima trap.

There is a further variation of the crossover operator. Here, a chromosome is re-structured into a series of loops. Each loop represents one of the parameters. Crossover between chromosomes is then allowed only between corresponding loops. This approach makes it more likely for the creation and building of blocks[24].

There are a host of other more minor variations, but those described above cover the most common of the most major variations. Other genetic operators are often proposed, but rarely do they serve any globally applicable use. Despite all these variations, GAs remain highly robust, and good results can be obtained with almost any variation of GA, and even with a large range of controlling parameters.

3.4.12 Epistasis

Epistasis can be observed in the biological genetic process as well as in the genetic algorithm. This is the requirement that two parts of the chromosome must change simultaneously for any benefit to the chromosome as a whole to be noticed. If only one of the changes occurs, no improvement in the fitness will be noticed, or it may even decrease.

An example taken from [7] is that of a bat, with its sonar emission and hearing ability. When evolving, a flying bat with no sonar system would often collide with objects. The same can be said for a bat with either only the ability to emit sonar sound, or only the ability to hear sonar sound. It is when the two aspects occur simultaneously that any benefit is seen.

In the GA, the chance that building blocks will be created at two different parts of

the chromosome at the same time and subsequently built upon is remote, compared to the creation of a single building block. Thus, a problem that requires the development of a dual aspect can expect to do so only slowly with a GA.

Chapter 4

Software Implementation and Test Strategy

4.1 Simulation Structure

Simulation of the real time recursive block parameter estimation environment is written in the C++ programming language. This is an Object Orientated language, where classes are constructed which contain both data, and the functions that act upon the data. This language lends itself to a modular construction, and with a little care, these modules can easily reflect the problem in hand.

4.1.1 Structure Details

There are three main classes in the structure of this simulation (fig. 4.1). The most prominent class, `SOE` (Second Order Equation) controls all the data leading up to and after the estimation of parameters from a single block of target signal. This class is fed data by the `SOEdata` class, which is essentially a database to store all important input and output information for all estimates made by the `SOE` class in a run of the simulation.

At the heart of this study is parameter estimation. The estimation is performed by

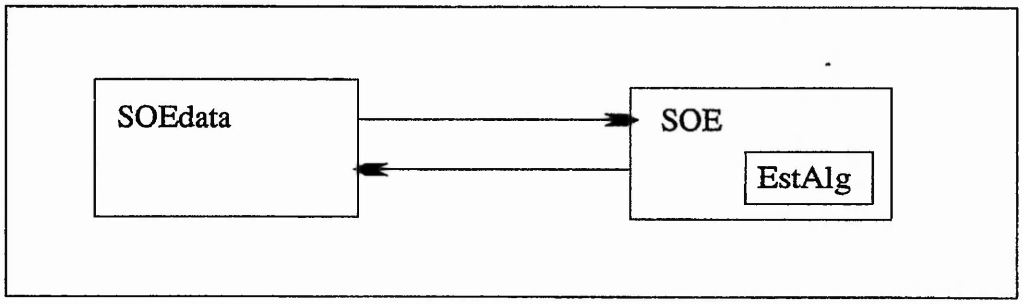


Figure 4.1: *The class structure of the estimation simulation. Data moves between the* `SOEdata` *database and the instance of the* `SOE` *process. This contains an* `EstAlg` *class which contains the estimation algorithm.*

a variety of different algorithms, but the input and output is always the same. Going into the algorithm is the target signal, and coming out are the estimated parameters. The class which actually performs the estimation is `EstAlg` . Of course, the contents of this class vary depending upon the chosen estimation algorithm. The `EstAlg` class is called within a function of the `SOE` class, when this class has prepared all the relevant data.

Each run of the simulation involves estimating one hundred parameter vectors, which may vary with time, with each parameter vector being estimated over ten consecutive blocks. The `SOEdata` class contains the initial values for the parameters for each of the one hundred test vectors, plus the two initial conditions, x_{1o} and x_{2o}, necessary to identify a unique trajectory. These vectors are passed one at a time to the `SOE` class which generates the signal on the fly for each block. The `SOE` class contains the

sampling frequency, f_s, and is thus able to feed the `EstAlg` class with data according to the adaptive sampling frequency algorithm. The parameters at the end of each block are stored, and at the end of each series of ten blocks, the history of the run is passed to the `SOEdata` class. At the end of the simulation, the `SOEdata` class dumps the history of the entire simulation to disk for analysis later by the user.

Figure 4.2 depicts the structure of the simulation in pictorial form. The steps are as follows:

1. `SOEdata` feeds in the target parameter vectors from disk.
2. Initialization of `SOE` occurs, which also includes details of the `EstAlg` initialization.
3. Start the vector number loop.
4. `SOE` receives target vector from `SOEdata` .
5. Zero the clock.
6. Start the block number loop.
7. Target signal is generated based on the current clock value, the current sampling frequency, and the target parameters.
8. Target signal is passed to `EstAlg` which returns an estimate vector.
9. Frequency estimate is passed to the adaptive sampling frequency algorithm function within `SOE` to update the sampling frequency. Clock is updated.
10. End of block number loop.
11. Pass run history to `SOEdata` .
12. End vector number loop.
13. Dump statistics to disk for analysis later.

An object design for each class is shown in figures 4.3, ẽffig:object-soe and 4.5. Each class contains both functionality and data. The first subdivision in each diagram holds the class name. The second contains the principle functions inherent to the class. These describe the most significant processes the class carries out. In the third box is listed the class's primary data structures. It is upon these data structures that the functions operate.

4.1.2 Initialization

A separate program generates the parameter data. There are limits imposed on the values of some parameters; for the frequency parameter this stems from the initial sampling frequency used, and for the other parameters and the initial conditions, these stem from a need to limit the range of the x variable to approximately ± 2. There are sometimes further limitations on the parameter and starting condition ranges when the estimation algorithm fails on the maximum ranges, and simpler signal profiles are required.

Other initialization details, such as initial sampling frequency, details for the adaptive sampling frequency i.e. sample length and n_C and the noise and magnitude of noise to be used, are kept in a `.inf` file which is read at the beginning of each simulation. Memory arrays are dynamically created so that the program does not have to be re-compiled whenever controlling parameters are varied.

Some of the estimation algorithms called in `EstAlg` also require some user defined variables e.g. for polynomial least squares fitting, the polynomial order is user defined. To save loading this information each time the `EstAlg` class is called, it is stored within the `SOE` initialization and passed as a function parameter when the estimation algorithm is called.

These measures attempt to make the simulation "user friendly" by allowing many of the simulation parameters to be set within `.inf` files rather than hardcoded into the

program. This allows for greater speed in testing the performance of the estimation algorithms under a wide range of situations.

4.1.3 Execution

In the simulations, each estimation algorithm is executed many times. There are a hundred test parameter vectors, and each of these can be estimated over a number of blocks. This situation allows for a straightforward design using two `for` loops. The inner loop cycles through the number of blocks, whilst the second cycles through the target parameter vector list.

```
SOE.Initialization()
for ( vector = 0 ; vector < NumVectors ; vector++ )
{
  SOE.ZeroClock()
  SOE.ReceiveTargetParameters( SOEdata( vector ) )
  for ( block = 0 ; block < NUmBlocks ; block++ )
  {
    SOE.CalculateSignal()
    SOE.EstimateParameters()
    SOE.UpdateClock()
    SOE.AdaptiveSamplingFrequencyAlg()
    SOE.DeliverDetails(SOEdata)
  }  // Block loop
}    // Vector loop
SOEdata.StoreAllDetails()
```

4.1.4 Peripheral Functions

There are some functions used in the simulation which do not relate directly to any specific class, since they are general mathematical functions. These include functions for calculating the RMS error between two vectors, calculating the correlation between two vectors and calculating the mean of a vector.

These functions are outside of the class system and are stand alone functions used by the various estimation algorithms. For example, the GA algorithm uses the RMS and correlation functions to compare the target and estimated signals to obtain a fitness for the estimated parameters.

4.2 Estimation Algorithms

4.2.1 General

All of the estimation algorithms are encoded as classes. Each class has only three functions called externally. These are the initialization function, the execution of the algorithm and a function to return the estimated parameters. All other functions for the algorithm class are called internally.

Several of the algorithms have a simple internal structure where the estimation requires only a single core function, maybe with a few supporting functions. These include the difference equations, the polynomial LS fitting and the Downhill Simplex.

The high level algorithm contains several important functions based around a Fourier Transform class. The target signal is used to initialize the class, which after performing an FFT, executes several other tasks e.g. PSD and dominant frequency search. The algorithm then combines the results from these functions to produce its estimate for the parameters.

Estimation using a neural network, either for derivative estimation or direct parameter estimation, follows a straightforward approach, similar to those described above,

with the addition that the network size, and other details, are loaded at the beginning of the simulation and passed in through the SOE class. The major difference with this algorithm is that a network must first be trained so it can perform the estimation task.

Training is performed in a separate simulation. Training data with specified characteristics for the signals is randomly generated. These must be different from the test signals used to measure the performance of the network in the estimation simulation. Training is conducted and monitored, and when the error of the network reaches minimum levels, the weight matrix which identifies the unique network is transported to the estimation simulation and is used in a "read only" mode.

4.2.2 Recursive Estimation

When the estimation algorithm is recursive, there are two phases to the estimation procedure. An initial estimate has to be made, which then is used to seed a recursive estimation algorithm. This can be seen in Figure 4.2.

Since all the algorithms can work as initial estimators, each recursive algorithm used itself in "initial estimator" mode to complete the first phase. The only exception to this is the Downhill Simplex method which is seeded by the high level algorithm. This simply involves inserting both the Fourier transform class and simplex class in the SOE 's *Estimate()* algorithm and keeping track of the current block number.

4.2.3 Genetic Algorithm Structure

A genetic algorithm (GA) is a more involved estimation algorithm with several steps in its initialization and its running. The structure is shown in Figure 4.6. Its steps are:

1. Receive target trajectory.

2. Create a random population.

3. Test each chromosome and obtain its fitness.

4. Rank the entire population according to fitness.

5. Select two chromosomes to be parents.

6. Create the offspring and perform the genetic operators.

7. Find the fitness of the offspring.

8. Insert the offspring into the population.

9. Test for the termination conditions. Return to step 5 if another generation is available.

10. Decode the fittest chromosome and return the estimated parameters to SOE .

When the GA is used in a recursive fashion, there is only one additional feature of the algorithm, other than keeping a track of which block the GA is operating in. This is to re-encode the chromosome sections which relate to the initial conditions. This is achieved by calculating the expected values for the initial conditions and then encoding them into binary and replacing the relevant section of each chromosome.

4.3 Test Strategy

Experiments are started with simpler problems, which are then built upon and learnt from, until the full continuously running estimation algorithm is complete. Experimental work starts with parameter estimation for an initial, single block of data. Firstly, this is achieved via derivative estimation in Section 5.1. Then other methods which estimate the parameters are directly examined. Together these sections constitute the first stage of a continuously running parameter estimation system by allowing the first estimate to be made without any previous estimates. Section 5.3 completes the system by examining all the methods for parameter estimation which can operate in a continuously running mode.

4.4 Test Parameter Generation

To measure how well each method estimates the parameters, a standard experiment is used. An experiment will consist of a single run made up of typically one hundred parameter sets which require estimating. In the first two sections where an initial estimate is required, the parameters do not vary within the time range and only one block of data is fed to the parameter estimation module for each of the one hundred parameter sets. In the last section where the system runs in a continously running mode, the parameters vary their value sinusoidally according to pre-set frequencies.

The parameter values which make up a run are chosen at random within certain ranges. It is clear from careful anaylsis of the test procedure that some methods are capable of coping with the complete parameter ranges, whilst others require the range of some of the parameters to be limited, and this is discussed in each appropriate section.

A set of parameters is chosen at random within the following ranges:

$$\begin{aligned} \frac{-0.4}{\omega_n} < \quad \xi \quad &< \frac{0.4}{\omega_n} \\ 0.01\omega_{max} < \quad \omega_n \quad &< \omega_{max} \\ -\omega_n^2 x_{max} < \quad U \quad &< \omega_n^2 x_{max} \end{aligned}$$

The reasons for the choice of these bands require some explanation. Firstly, the observed damping, Ξ, is a function of both the value of ξ and ω_n, and is $2\xi\omega_n$. Secondly, the maximum frequency, ω_{max} is a function of the initial sampling frequency, f_s used for receiving incoming data, and is set to,

$$\omega_{max} = \frac{f_s}{4} \tag{4.1}$$

This ensures that there are at least four data points per cycle of the incoming signal. This is the minimum number of points that can describe the amplitude of a sinusoidal

signal, which is necessary since some of the methods used for parameter estimation depend on fitting an internally generated estimate signal, including magnitude, to the target signal.

The value of x_{max} is an arbitrarily chosen value to limit the overall range of the variable x. x_{max} actually describes the largest magnitude for the equilibrium position of x and not its maximum value. In the generation of the tests signals, however, the values of the initial conditions x_{1o} and x_{2o} are selected so that the signals have a range in amplitude from 0.5 to 2.0. This is done by selecting at random a phase angle, ϕ, and an amplitude, A, and then setting,

$$\begin{aligned} x_{1o} &= A\cos\phi + \frac{U}{\omega_n^2} \\ x_{2o} &= -\omega_n A\sin\phi \end{aligned}$$

This entire set of conditions allows a robust set of signals to be generated, whilst keeping a high element of variation in the signals. As mentioned, some methods require these restrictions to be tightened.

In the continuously running mode experiments, damping ξ is always set to zero since running a second order system for many blocks with non-zero damping can easily arise with the signal vanishing to zero or becoming unstable and oscillating to wild values. It is still a valid task for the estimation algorithms to correctly return a value of zero for the ξ parameter.

4.5 Performance Measurement

It is necessary to have some yardstick by which to measure the performance of each run. To do this in the first two sections where the parameter estimation is for one block only, the Absolute Average Difference (AAD) vector is used. The AAD vector is $(AAD_\xi, AAD_{\omega_n}, AAD_U)$, where,

$$
\begin{aligned}
AAD(\xi) &= \frac{1}{P}\sum^{P}|\hat{\xi}-\xi| \\
AAD(\omega_n) &= \frac{1}{P}\sum^{P}|\hat{\omega}_n-\omega_n| \\
AAD(U) &= \frac{1}{P}\sum^{P}|\hat{U}-U|
\end{aligned}
$$

where P is the number of test patterns (usually one hundred). Naturally a zero AAD vector means perfect estimation.

When continuous parameter estimation is applied, the Total AAD (TAAD) vector is used to measure performance. The TAAD is simply the total of the AAD vectors for each block of data for a signal. If there are M blocks of data in the P test sets,

$$
TAAD_{\xi} = \frac{1}{T}\frac{1}{M}\sum^{P}\sum^{M}|\hat{\xi}-\xi| \tag{4.2}
$$

and similarly for the other two parameters. Again, a zero vector indicates perfect estimation.

It is important to note that when discussing sampling frequency, hertz (cycles/s) is used as the basic unit. Whenever ω_n or a related variable is discussed, angular frequency is always used. Therefore, if the sampling frequency f_s is 25Hz, and $\omega_n = f_s/4$, it has a value of $25 \times 2\pi/4 = 39.9$Hz.

4.6 Error Function Analysis

Two of the direct parameter estimation algorithms, namely the Downhill Simplex and the Genetic Algorithm (GA), have their basis in the comparison of the incoming data signal and a signal internally generated by estimated parameters. By comparing these two signals and determining how similar they are, the algorithms can give some figure of merit to the parameters that generated the estimated signal.

There are two functions used by the two algorithms to compare the signals; the first is the straightforward Root Mean Square (RMS) error,

$$RMS = \sqrt{\frac{\sum^{N}(y_{target}(i) - y_{estimate}(i))^2}{N}}. \tag{4.3}$$

When the signals are identical in all respects i.e. each data value matches, the RMS is zero.

The other combines the correlation between the signals and the difference between the signal's means,

$$Error = -Corr(y_{target}, y_{estimate}) + \frac{|\mu_{target} - \mu_{estimate}|}{W} \tag{4.4}$$

The correlation function returns values between ± 1.0. It returns $+1.0$ if the vectors are identical ignoring scale and position.

How do these functions vary? Figure 4.7 shows an input target signal of 2Hz sampled at 25Hz for one second. There is no damping and no external input. A second, estimated signal, is generated from estimated parameters. The estimated parameters need to match the target parameters and this is achieved when either of the error functions are at their global minima.

The variation of the RMS function when the phase between two otherwise identical signals is moved from zero to 2π, is shown in Figure 4.8(a). The error near zero and 2π phase exhibit a low error, whereas the error rises to a maximum at π radians phase difference. The important feature is that there is only a single peak in the error curve, and a simple gradient descent search algorithm will easily find the global minima between two signals which vary only in their phase.

When the two signals have the same phase but their frequencies are different, a more complex error profile is seen (fig. 4.8(b)). The target frequency is again 2Hz, and the estimated signal sweeps from zero to one quarter of the sampling frequency. The global minima appears as expected at 2Hz. Either side of this are a number of local minima.

A slightly different, but still multi-modal error function, is seen when the phase is

optimized at each estimated frequency (fig. 4.8(c)). Each of the minima, including the global, are slightly broader when the phase is not optimzed.

Finally, Figure 4.8(d) the correlation and mean error function is shown at each frequency when the phase is optimized. The local minima at higher frequencies still occur, but at lower frequencies than the target, there is only one minima.

The multi-modal nature of the error curves poses a particular problem for the iterative optimization algorithms. Such algorithms work by descending the error curve. If they are placed in a minima other than the global minima, the error curve will lead them to an erroneous conclusion. To combat this, either the algorithm must possess the ability to escape from non-global minima, or the algorithm must be initialized with an estimate within the global minima.

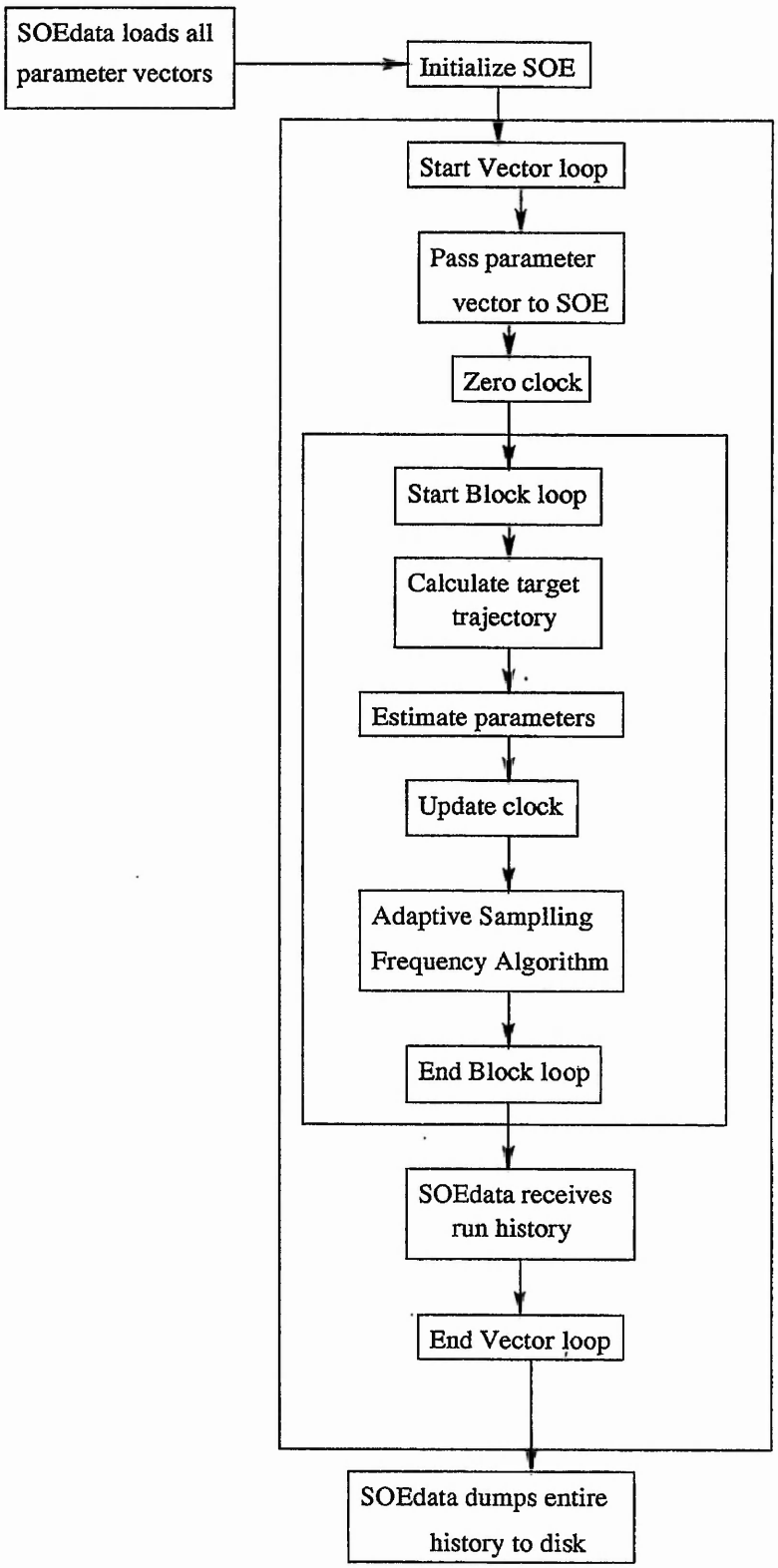


Figure 4.2: *Structural design of the real time recursive block parameter estimation simulation.*

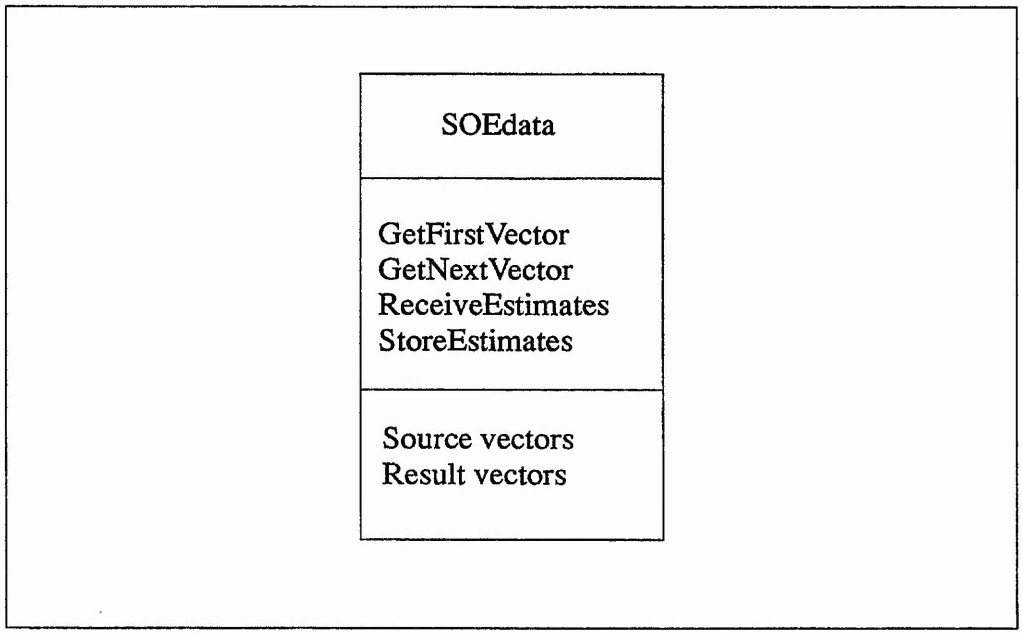


Figure 4.3: *Object design for the* `SOEdata` *class. The central division describes the principle functions and the lower the data structures.*

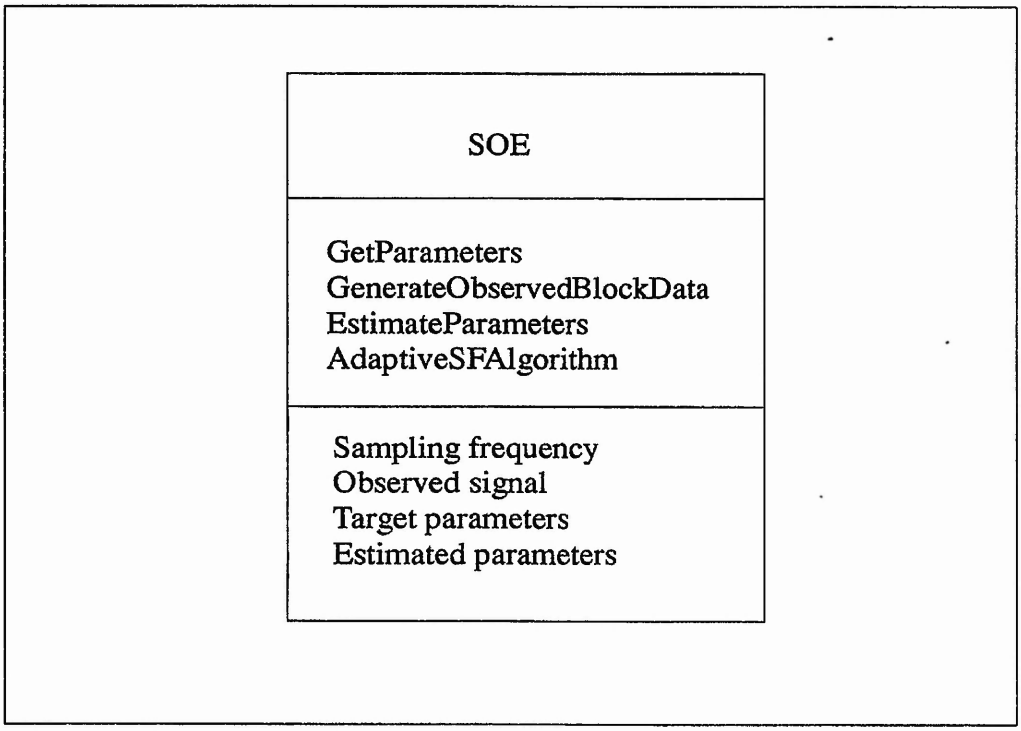


Figure 4.4: *Object design for the* `SOE` *class. The central division describes the principle functions and the lower the data structures.*

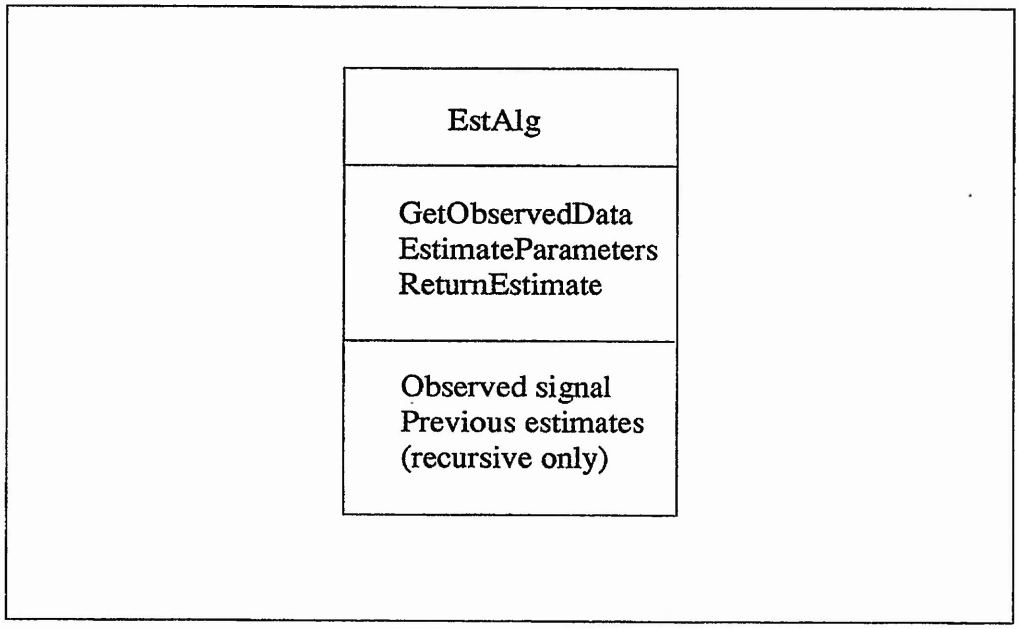


Figure 4.5: *Object design for the* `EstAlg` *class. The central division describes the principle functions and the lower the data structures.*

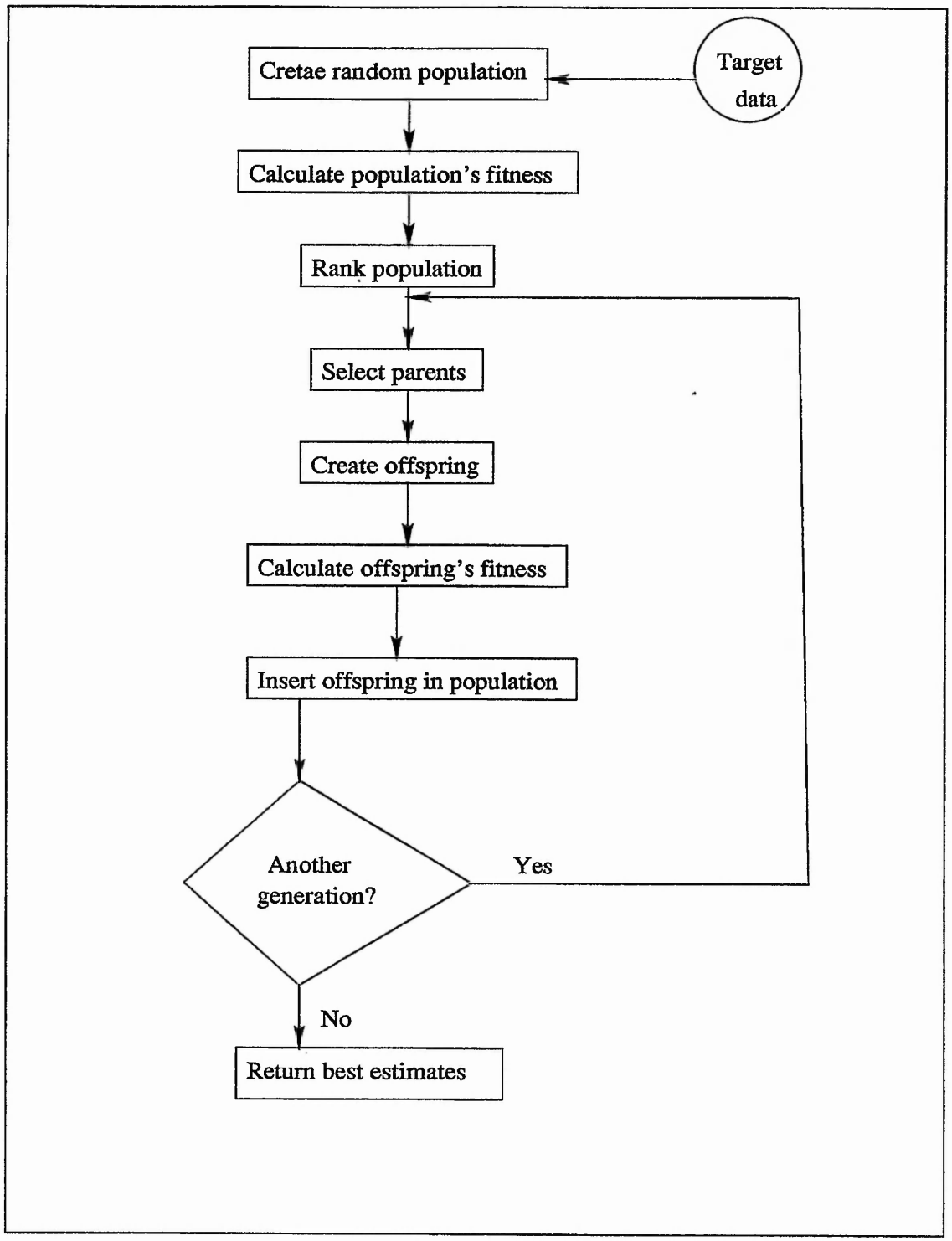


Figure 4.6: *A flow chart describing the structure of the genetic algorithm estimation process.*

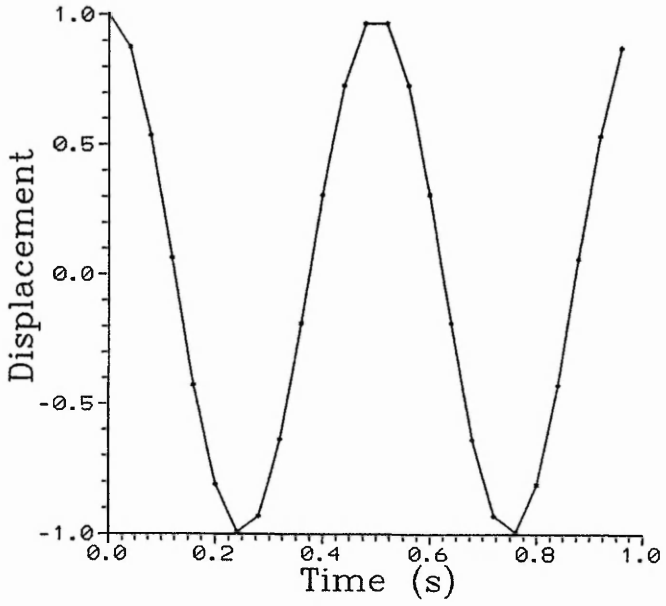


Figure 4.7: *An input target signal sampled for one second at 25Hz. Signal frequency is 2Hz, with no damping and zero mean.*

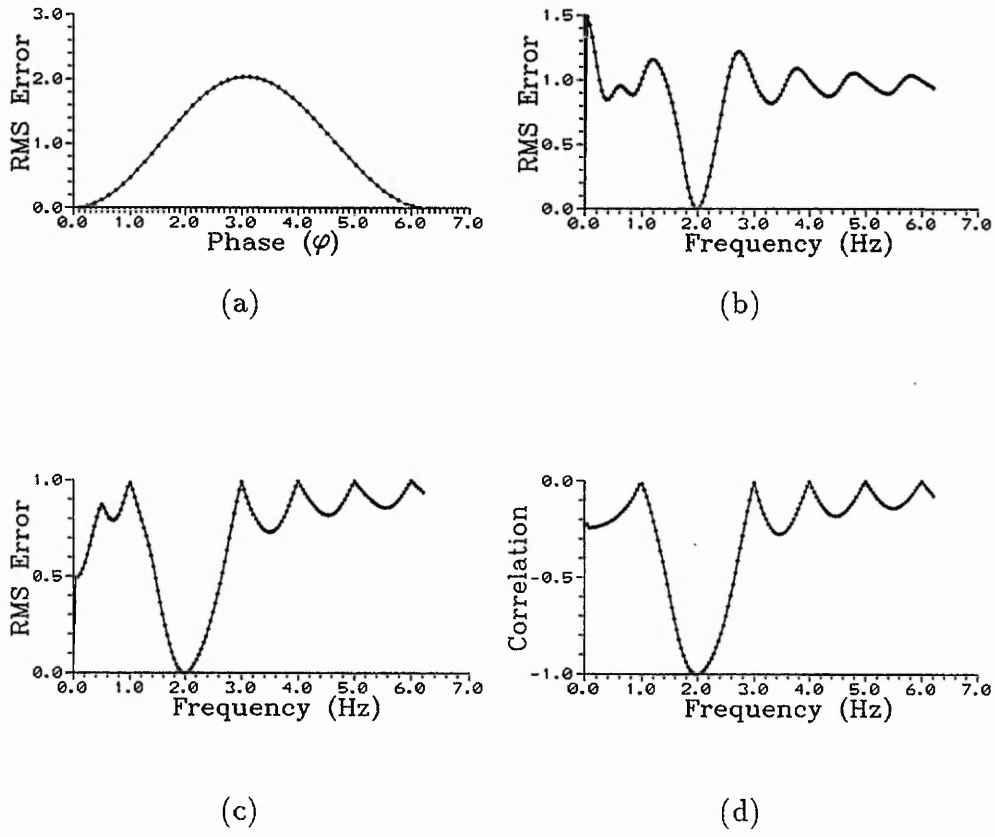


Figure 4.8: *RMS error functions between a target signal and an estimated signal. a) When the phase varies, b) When the frequency varies, c) When the frequency varies and phase is optimized, d) When frequency is varied and phase optimzed and the error function is the correlation function.*

Chapter 5

Experimental Results and Discussion

This chapter is dedicated to the reproduction of the results obtained with extensive experimental work into different estimation methods for the parameters of the general second order system. It details each stage of the experimentation, describing notable features in the results.

The overall test strategy and preliminary analysis is described within Chapter 4. A full conclusion of the work described in this chapter is contained in Chapter 6.

5.1 Time Derivative Estimation Methods

5.1.1 Introduction

This section covers parameter estimation using time derivatives, as discussed in Section 3.4.1. Once the time derivatives have been estimated, the parameters of the second order system are calculated using the equations of [1]. Firstly, the use of simple difference equations to estimate the derivative is covered, which operates by subtracting consecutive data points. A series of alternative estimation methods is then tried to counter the problem of noise. These are polynomial least squares smoothing, digital

filtering and the application of feedforward neural networks.

If x_1 is the zeroth derivative, x_2 the first derivative (velocity) and so on up to x_5, then,

$$\begin{aligned} a &= -\frac{x_5.x_2 - x_4.x_3}{x_4.x_2 - x_3^2} \\ b &= -\frac{x_5.x_3 - x_4^2}{x_3^2 - x_2.x_4} \end{aligned}$$

where $b = \omega_n^2$ and $a = 2\xi\omega_n$.

5.1.2 Difference Equation Derivative Estimation

For each parameter set in a run, six points are generated, since the difference equations require only six points to calculate all the required derivatives. The parameters ξ, ω_n and U are estimated using the equations of Section 3.4.1.

With a pure uncorrupted input signal, this method achieves an AAD vector of $(0.31, 0.27, 20.3)$. This is a run where the sampling rate is fixed at 25Hz. This is a poor estimate of the damping parameter, which only has a range of ± 0.4. The frequency estimate is good, since the range of this parameter is $3.93 - 39.3$. In this case, U can vary between $\pm w_n^2$.

The ASFA with Fixed Parameters

Application of the adaptive sampling frequency algorithm (ASFA) when the parameters remain fixed, produces more accurate parameter estimates. It only takes a few blocks of input signal for the sampling frequency to stabilize, where

$$f_S = \frac{\hat{\omega}_n n_C}{2\pi}$$

after each block. After four data blocks the AAD vector is $(0.2, 0.02, 1.4)$, clearly a marked increase in accuracy.

Figure 5.1 shows how the sampling frequency and the estimate of frequency vary over the time period of the signal. In Figure 5.1a the input signal is shown. Note that the number of points increases towards the end of the signal, indicating an increased sampling frequency. In Figure 5.1b the sampling frequency is shown to move from its initial value of 25Hz up to nearer 48Hz. The actual frequency of this signal is 2.8Hz; with the algorithm attempting to feed in 16 data points per cycle, this would require a sampling frequency of $f_D \times n_C$, or $2.88 \times 16 = 46.1$Hz. This shows the algorithm is getting close to the desired sampling frequency.

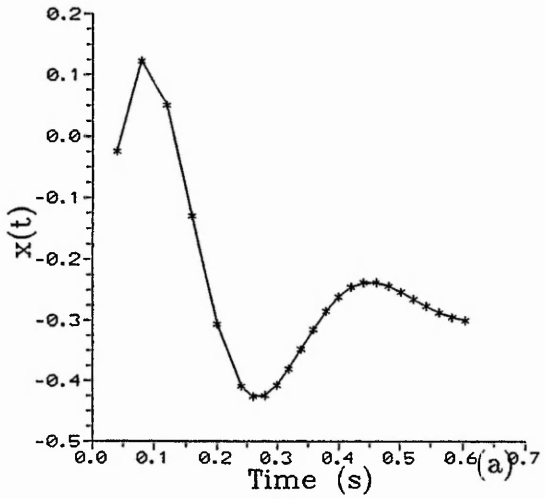
Figure 5.1c shows the estimated frequency compared to the target frequency. The first estimate is made with a sampling frequency of 25Hz. The second is made with an adjusted sampling frequency of 50Hz, allowing for a more precise estimate to be made. The feedback system quickly settles down to produce an estimate only 1Hz out.

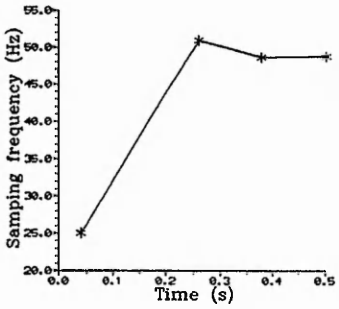
The estimates for all three parameters receive a significant improvement in their accuracy due to the use of the adaptive sampling frequency algorithm when the parameters remain fixed with respect to time. The estimate for ξ, is still however, poor at 0.2 when the actual value is zero. Despite the equations which calculate a and b (eqn.'s 3.2 and 3.3) and hence the parameters is exact, the difference equations for calculating each of the time derivatives is *not* exact. This is where an error in the parameter estimation occurs.

Noise Tolerance

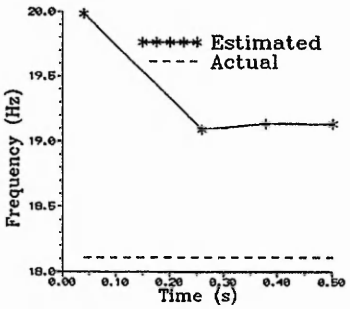
Adding noise to the input signal produces a rapid degradation in the accuracy of the estimates. Table 5.1 shows the resulting AAD vectors when noise is used. Both white noise of equal probability distribution and impulse noise are tested.

In the second column, the number of test signals where the ξ estimate is greater than 0.6 is tabled. Such signal's estimates are not used in calculating the AAD for the run, and are effectively discarded as too poor. In this case, the signals are left out for the run with 0.01 amplitude white noise have an AAD vector of $(1.76, 1.37, 236.9)$.





(b)



(c)

Figure 5.1: *A signal (a) is sampled at a varying sampling frequency. (b) shows the sampling frequency at the end of each block. (c) shows the estimated frequency of the signal compared to the actual frequency.*

Noise Type	Out of range	AAD		
	$\lvert\xi\rvert > 0.6$	ξ	ω_n	U
No noise	20	0.26	0.16	12.394007
White noise Mag=0.001	38	0.35	0.32	31.6
White noise Mag=0.01	52	0.44	1.0	193
White noise Mag=0.1	70	0.38	2.9	685
Impulse noise Prob=10% Amp=0.1	34	0.3	0.7	130
Impulse noise Prob=15% Amp=0.4	52	0.3	1.5	501

Table 5.1: *Table showing AAD vectors for the difference equation method with different levels and type of noise on the input signal.*

This effect is more pronounced when other methods are used.

With only modest amounts of noise, the estimate of ξ becomes worse than simply guessing the result. The estimate of U also suffers from noise, but despite the degradation, the estimate of ω_n is still reasonable.

Comments

These few runs demonstrate that the time derivative estimation method for parameter estimation is an accurate method when the adaptive sampling frequency algorithm is applied. Tolerance to noise is low, however, with poor results being obtained with even small amounts of either white or impulse noise.

5.1.3 Polynomial LS Fitting For Derivative Estimation

By taking a block of input signal and fitting a polynomial with the least squares (LS) criteria, a set of polynomial coefficients are generated which allow the derivatives to be calculated at any point on the polynomial. To fit a polynomial of order N, there must be at least $N+1$ points, and to evaluate x_5, the fourth derivative, the order must be at least four. There must therefore be at least five points per block in the input signal.

A further consideration is that each derivative of a second order system has a sinusoidally varying profile. The polynomial order must be able to approximate this sufficiently well. The higher the order used, the better the polynomial will be able to match the fourth derivative's profile. Low orders can still be used, however, as long as the approximation is not too coarse.

Polynomials and their derivatives are always evaluated at the centre of the input block. It can be shown that the derivative estimates for a polynomial fit with the LS algorithm are most accurate towards the centre of the vector.

It is possible to achieve an almost arbitrary accuracy when no noise is present in the system. By using the highest polynomial order possible for the length of the block used, optimal results are obtained. Increasing the block length allows for improved accuracy, since this allows for higher polynomial orders to be used, which can approximate the higher derivative orders better. For example, a block length of 6 and a 5th order polynomial, the AAD vector is $(0.03, 0.12, 31.2)$. When the block length is 8 and an order of seven used, this improves to $(0.005, 0.033, 7.22)$.

Noise Tolerance

Adding white noise to the signal disables the accuracy of this setup. A polynomial of order one less than the length of the data can match each data point exactly. Such a polynomial mimics the noise on each data point, and no smoothing occurs. Lower orders must be used if smoothing is to take place.

Noise Type	Out of range	AAD		
	$\lvert\xi\rvert > 0.6$	ξ	ω_n	U
No noise	0	0.04	0.3	51.3
White noise Mag=0.001	7	0.07	0.3	56.1
White noise Mag=0.01	16	0.08	0.4	67.3
White noise Mag=0.1	14	0.18	1.0	137
Impulse noise Prob=10% Amp=0.1	9	0.1	0.56	76.5
Impulse noise Prob=15% Amp=0.4	8	0.18	1.1	145

Table 5.2: *Table showing AAD vectors for the polynomial LS method with different levels and type of noise on the input signal.*

A sixth order polynomial is used with a block length of 12. Table 5.2 shows the AAD vectors for different levels of noise and types of noise using this arrangement.

In the run of white noise with magnitude 0.01, where 16 of the signals have ξ estimates out of range, the AAD vector for the signals that were left out is $(1.5, 2.5, 494)$. This is significantly different from those estimates within range, and demonstrates that the potential for making significantly erroneous estimates is quite rare, but severe.

The source of the high errors is test signals that have frequencies at the far low end of the scale. When ω_n is approximately less than 6.0, there is a high chance that the estimate will be significantly out. There is no simple solution to this difficulty — it is a property of the algorithm that frequencies significantly below the sampling frequency

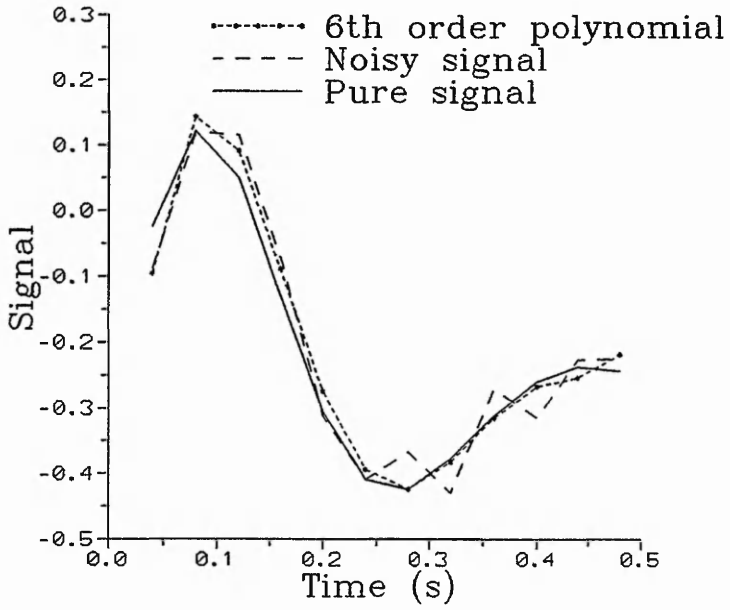


Figure 5.2: *A noisy signal is fitted with a fourth order polynomial. The polynomial matches more closely with the noise free signal, allowing more accurate derivative estimation.*

with white corrupting noise, cannot be estimated accurately. Study shows that the higher derivatives are estimated more poorly than the lower ones, which suggests that the high frequency variation of the noise is perturbing the higher coefficients of the polynomial LS fit, whereas, in high frequency signals, it is the signal itself which is dominant. it is possible to estimate the lower frequency signals accurately only by reducing the sampling frequency, at the expense of excluding higher frequency signals.

Figure 5.2 shows a noisy input signal, where the noise has a maximum magnitude of 0.1. Also shown are the 4th order polynomial fit and the noise free signal. It can be seen that the polynomial curve matches more closely the noise free signal than the noisy one.

Despite the low order, good results are obtained due to the short block length. The high derivatives are estimated well since a straight line is a reasonable approximation

over the short block length. Increasing the block length decreases the validity of this approximation. Increasing the order allows the polynomial to mimic the noise and decrease the derivative estimate accuracy.

Comments

Use of a polynomial fitted used the LS criteria to the incoming signal produces arbitrarily accurate results when no noise is present. Increased accuracy is achieved with increased sample length whilst maintaining the maximum allowed polynomial order.

Application of noise to the incoming signal disables this property and the polynomial has to be used as a smoother. As noise increases, higher order coefficients are distracted by the noise, notable when the signal frequency is low. As a result, low frequency signals (compared to the sampling frequency) cannot be estimated with certainty when noise is present. Either the sampling frequency must be reduced, or low frequencies removed from the test data.

This problem can be expected to cause difficulties when the ASFA algorithm is used when estimating signals with variable parameter values. Signals which start with a high frequency which then reduce can be expected to be estimated accurately, as the ASFA will adjust the sampling frequency. Signals with a low initial frequency will fail in the same fashion as above, and the ASFA will fail to fix an appropriate sampling frequency.

5.1.4 Pre-processing Signal Filtering

Since noise significantly reduces the capabilities of both the difference equations and the polynomial fitting approaches for derivative estimation, filtering the input signals before they are passed to the estimation algorithm is performed. With the noise partly filtered out, estimation can be expected to be more accurate.

Filtering can be any of lowpass, highpass, bandstop or bandpass. Ideally, a bandpass

filter centred on the actual input signal's frequency would remove most noise and still leave the signal substantially unaffected. Since it is the frequency of the signal which is sought, however, this is not possible. Further, such a filtering system would have to consist of a bank of bandpass filters each tuned to a separate frequency, and the best one used depending on the estimated frequency. Although this is a possible approach for an adaptive system, it is clumsy and not entirely necessary due to other considerations.

Since the input signals can range from a low frequency up to middle range frequency, and white noise has frequency content across the entire range of frequencies, whilst impulse noise has high frequency elements, lowpass filtering will remove a substantial part of the noise from any input signal.

The Moving Window Average Filter

A simple lowpass filter is the Moving Window Average (MWA), where a number of data points from the input signal are averaged to produce the output for the current time step ie.

$$y_n = \frac{1}{N} \sum_{i=0}^{N-1} x_i$$

where N is the order of the filter, or the size of the moving window. The resulting output signal y_n lags the original signal by $(N-1)/2$, but this does not affect the parameter estimation process.

MWA Filtering and Difference Equations

Application of the MWA filter with the difference equation method improves parameter estimation measurably. Table 5.3 shows the results of runs with 0.01 magnitude white noise and a range of MWA order.

Considering only those signals where the ξ estimate falls within range, the use of a MWA has no measurable effect on the ξ estimate itself. On ω_n and U, however, there is an improvement by a factor of approximately 3 with the best results obtained with

MWA order	Out of range $\lvert\xi\rvert > 0.6$	AAD		
		ξ	ω_n	U
No noise	20	0.27	0.17	12.7
1	52	0.46	1.14	206.7
3	43	0.4	0.55	73.9
5	42	0.40	0.50	42.1
7	92	0.40	1.41	86.8

Table 5.3: *Table showing AAD vectors for the difference equation method with pre-processing filtering with different MWA orders. White noise is used at magnitude* 0.01.

MWA order 5. An order of 7 results in 92% of the test patterns falling out of range. In the other cases, the out of range vectors are not excessively poor.

Since the MWA is a lowpass filter, it may be thought that only low frequency signals should be tested. This would allow maximum noise reduction, and hence a maximization of estimation accuracy. This is not the case. If only low frequency test signals are presented to the algorithm, very high numbers of estimates fall out of range because the signal does not vary over many of the six data points used, and so noise becomes a large factor.

This process can be countered by using the adaptive sampling frequency algorithm (with zero damping). The sampling frequency is adjusted so that 6/16th's of a cycle are used in the difference equations. This will mean the signal varies over a reasonable range. Experiments show, however, that even small amounts of noise (white noise of magnitude 0.01) allow the sampling frequency to explode in some cases. On cases where the sampling frequency remains stable, accuracy is greatly improved.

MWA Filtering and Polynomial LS Fitting

Applying the MWA pre-processing with the polynomial fitting always produces less accurate estimates than when no MWA filtering is used. This is with both white noise and impulse noise. Since the polynomial least squares fitting acts as its own filter, and the MWA is also a lowpass filter, it is of little surprise that no noticeable difference is observed since the two filters perform a similar job.

Comments

This section has shown that the MWA can improve the estimation accuracy when the difference equation method is used when white noise is present on the signal. It is shown, however, that the short sample length used in calculating the time derivatives led to problems when the signal moved through only a small range. It is also shown that a benefit in applying the MWA lowpass filter on low frequency signals.

Further work for parameter estimation via time derivative estimation must therefore lie in the direction of increasing the sample size used in calculating the time derivatives. The polynomial LS fitting does allow for this, and other methods such as neural networks can also be used (see next section).

5.1.5 Neural Networks for Derivative Estimation

The motivation for applying neural networks in this problem is clear; the transformation from an input signal to a time derivative is a nonlinear one, and neural networks are capable of performing such mappings. Neural networks are also well known for their tolerance to noise. This combination of properties make neural networks a good prospect for this problem.

Feedforward neural networks trained with backpropagation are used with the hyperbolic tangent function as the transfer function. This function ranges between ± 1.0, allowing easier estimation of negative derivative values.

A network is trained with signal trajectories as the input. Five output units are used, each one trained to output one of the five state variables x_1, x_2, x_3, x_4 or x_5 which are then used to calculate the parameter estimates. It is necessary to perform scaling on both the input and output.

Although networks are able to generalize, scaling of the input is performed to make the network's job as easy as possible. Firstly, the mean of the input signal is removed from the input signal. It is then scaled so the maximum input value is unity. This translation and scaling allows any offset and any magnitude of signal to be standardized, making the estimation process more tolerant of input variance.

Input and Output Scaling

Scaling of the output is necessary since derivative values can get very large, the more so for higher derivatives. The fourth derivative for example, is proportional to ω_n raised to the fourth power. Since $omega_n$ can easily have values of $6 \times 2\pi$Hz, this leads to extremely large values. Scaling is performed by taking the nth root, where n is the order of the derivative. Negative values are first made positive, rooted, and then made negative once more. This gives a maximum value of a derivative of ω_n^{max}, so dividing by this value gives a linearly varying function ranging between ± 1.0. This allows the tanh function to be used on the output units of the network. If O is the output value, then the scaled output, O_S is,

$$O_S(\frac{d^n x}{dt^n}) = \frac{1}{w_n^{max}} O^{\frac{1}{n}} \tag{5.1}$$

Linear output units with unscaled output cannot be used since the error which is backpropagated during training for the higher derivative output units would swamp that of the lower derivative output units. Linear output units can still be necessary with scaled output, whoever, since negative damping can lead to scaled derivative values outside of the range ± 1.0.

Preliminary Experiments

Two major points can be concluded from preliminary experiments. After testing a variety of networks eg. one layered linear adaptive filters, networks with linear hidden units, it is apparent that the minimum requirements for a network to perform the mapping is a neural network with nonlinear hidden units. This reinforces the principle that the mapping from input signal to derivatives is a nonlinear one. The second point is that the networks are not able to cope well with training data where the damping parameter has non-zero values. Best accuracy is obtained when all training and test patterns have zero ξ values.

Based on these principles, a suitably sized network to optimally perform the task can be performed. Training data consisting of one thousand patterns with the frequency ranging between a maximum of $f_S \times 2\pi/4$ and 1% of this and damping set to zero is used. The external input is limited so that the equilibrium position is between ± 1.0, and the amplitude ranges between $(0.5, 2.0)$. Scaling and translation is used to add to the invariance of the process.

After testing networks with between 6 and 12 hidden units and a range of input lengths, the best performing network in terms of AAD vector has 8 tanh hidden units, linear output units with 17 input units. Increasing the number of hidden units has no beneficial effect.

For all networks, the profile of the RMS curve is one that falls sharply at the beginning of training and then gradually shallows out to asymptotically approach some value. No plateaux are encountered, or local minima apparently escaped from and similar behaviour is always found regardless of the initial weight matrix. Training does benefit, however, from a reduction in the learning rate mid-training. This is shown in Figure 5.3. Here, training is carried out for 25 epochs with a learning rate of 0.1. At this epoch, it is reduced by a factor of ten to 0.01 where the dip in the RMS is observed. The same change in learning rate is made at epoch 40, and a smaller dip is

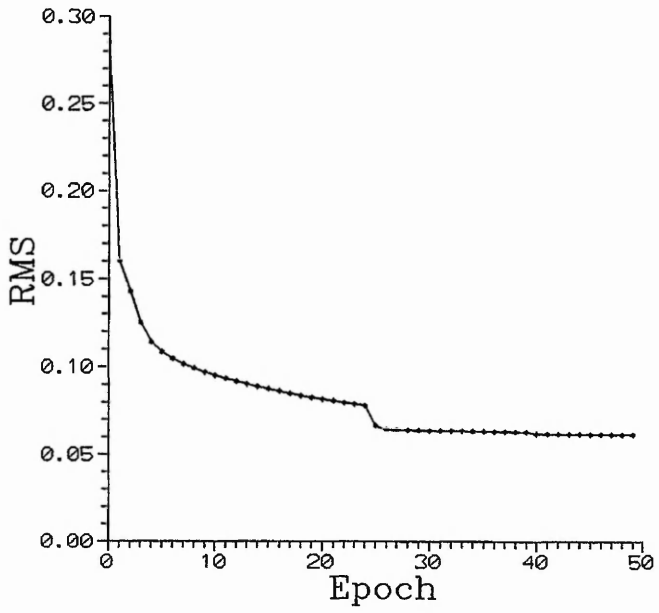


Figure 5.3: *The RMS curve for a a network with* 17 *input units,* 8 *hidden units with nonlinear hidden units. Training data varies in all respects except the damping parameter which is always set to zero. The asymptotic approach indicates no plateaux in the problem space.*

seen here also.

The AAD vector for this network over 250 test patterns is $(0.043, 0.92, 42.0)$. This compares well with the AAD for the polynomial method when no noise is present and with the same input length and a polynomial order of 6 of $(0.002, 0.04, 2.8)$. This latter case is also with a variable damping parameter, an important failing of the network method.

Alternative Training Methods

A number of alternative training methods is tested with neural networks to improve their accuracy. One method involves training five separate networks, each with one output unit — one network for each of the five derivatives. Their outputs are then

combined to produce the parameter estimates. Similar accuracy is recorded using this method as with one single network.

Another method attempts to train the network with a genetic algorithm (GA). The advantage of this approach is that the fitness function of the GA is not the RMS value of the network, but rather the accuracy in the parameter estimates. As a result, it can be hoped that only networks which give good parameter estimates go on to the next generation. Networks using this method give "average" results. That is, regardless of the input, the networks try to give outputs which result in average parameter values being generated. This is due to the fitness function being simply the sum of the AAD vectors, and average values give good all-round fitness. Reducing the number of training patterns results in networks which are unable to generalize and so fail on the test data.

Noise Tolerance

Tolerance to noise for networks trained with backpropagation is tested. White noise of different maximum magnitudes, and impulse noise of different probabilities and amplitude are used. The best tolerance to noise is obtained when a network is trained with uncorrupted noise, although training with all types of noise was tried. Table 5.4 lists the performance of A 17 input unit, 8 unit tanh function hidden layer and 5 linear unit output layer network trained with uncorrupted data and tested with different noise levels.

The neural network method appears to give accurate estimation of the ξ parameter. This is only the case when the test signals have zero value for ξ. Exposing the above network to test signals which have ξ between ± 0.4 gives an AAD of $(0.2, 4.1, 3952)$. It is important, therefore, to limit the use of the neural network to non-damped signals.

Noise Type	Out of range	AAD		
	$\|\xi\| > 0.6$	ξ	ω_n	U
No noise	0	0.04	0.91	36.4
White noise Mag=0.01	1	0.04	0.99	37.6
White noise Mag=0.1	6	0.6	2.4	69.5
Impulse noise Prob.=10% amp=0.1	4	0.04	1:6	51.0
Impulse noise Prob.=15% Amp=0.4	1	0.06	3.91	119.6

Table 5.4: *Table showing AAD vectors for a neural network trained with uncorrupted training patterns, and tested on noisy test patterns.*

5.1.6 Summary of Derivative Estimation Methods

There are three main approaches explored. Of these, the polynomial least squares fitting approach gives most accurate results for the widest range of parameters. Indeed, in the absence of noise, an arbitrary precision can be obtained. When noise is present, performance is still good, giving reasonable accuracy on the parameters. This method also allows the parameters to have almost any sensible values and any sensible starting conditions. This method benefits particularly from use of the adaptive sampling frequency algorithm, with increased accuracy in parameter estimation as the optimal sampling rate is obtained. This method is particularly suited to a continuously running mode since it gives the most accurate estimate of the frequency parameter. Its only drawback lies in inability to estimate frequencies at very low frequencies relative to the sampling frequency when noise is present.

Use of the difference equations suffers from three main problems, Two occurring

when noise is present. Firstly, the difference equations give only approximate values for the derivatives by their nature. Secondly, when any noise is present, even small amounts, the derivative values become widely inaccurate. Parameter estimation is further encumbered since the difference equations only use six consecutive data points in their calculations, and when these six points are in a part of the signal which does n to vary over a large range, the effect of noise is amplified.

Neural networks were tried, partly to overcome this last problem. A network's input can have any length, and so can be expected to cover more of the signal. Network's are unable, however, to cope with input data where the signal had a non-zero damping coefficient. Scaling of the input and output allowed for any offset and amplitude to be catered for, whilst the network learned to estimate the derivatives for any frequency and initial conditions. Given the limitation that only non-damped signals could be used, noise tolerance was good, with the U estimate being more accurate than with the polynomial method, and ω_n only a small factor different.

5.2 Direct Parameter Estimation

5.2.1 Introduction

This section covers the methods used to estimate the parameters of a second order system by direct methods, as opposed to first finding the time derivatives of an input signal. These methods are introduced in Section 3.4.6 onwards. A high level approach using signal processing methods is discussed first. In the next section, recursive neural networks are covered. Then in latter sections, iterative algorithms, such as the conventional Downhill Simplex method and the relatively new method of genetic algorithms, are covered.

For the experiments in this section, test signals are used where the parameters do not vary with time, as in the previous section for derivative estimation. It is in the

following section (sec. 5.3) that tracking a varying parameter is covered.

5.2.2 High Level Method

Introduction

In this method, features of the input signal are used to estimate its parameters. A Fourier transform is used to obtain the Power Spectral Density (PSD), or frequency content, of the signal. The dominant frequency is taken as the natural frequency of the second order system signal. The mean of the signal is used in conjunction with the frequency estimate to an external input estimate, and the magnitude of the dominant frequency over consecutive blocks allows the damping coefficient to be estimated.

The Fast Fourier Transform (FFT) algorithm is used to obtain the PSD. This algorithm always uses a sample length which is a power of two. If the incoming signal is not of such a length, it is padded out with zeros to the next power of two. This can lead to inaccuracies in the frequency estimate, and so in general, it is best to use sample lengths which do not require padding.

If the number of points in a sample is N, then the PSD consists of $(N/2) + 1$ positive values. The first is the absolute value of the mean, and the remaining $N/2$ points correspond to frequency bins equally spaced from $\frac{f_S/2}{N/2}$ up to $f_S/2$. The spacing between frequency bins is inversely proportional to the number of points in the sample. The frequency bin with the highest value is taken as the frequency estimate of the input signal. It follows that the more data points in the sample, the higher the accuracy of estimate ie. $\frac{f_S/2}{N/2}$.

Calculation of the mean of the input signal leads to an estimate of the external input as the mean is a reflection of the equilibrium position of the signal, and

$$x_{eq} = \frac{U}{\omega_n^2}$$

For the mean to reflect the equilibrium position most accurately, the signal needs to be a whole number of cycles. If this is not the case, the estimate is still good if the

signal consists of several cycles. The mean can mis-represent the equilibrium, however, if only part of a cycle is present in the signal. This is likely for shorter sample lengths and lower frequencies. The adaptive sampling frequency method is usually setup to attempt to include exactly one cycle in the input signal, however, and use of this signal can be expected to improve the estimate of the external input.

A value for ξ is obtained with successive estimates of the magnitude of the dominant frequency from one block of input signal to the next. If A_b is the magnitude of the dominant frequency in block b, and A_{b+1} the amplitude in the next adjacent block, ξ is approximated by,

$$\xi = \frac{1}{\hat{\omega_n}(t_b + t_{b+1})/2} \ln \frac{A_b}{A_{b+1}} \tag{5.2}$$

where $\hat{\omega_n}$ is the estimated frequency, and t_b is the time between blocks of incoming data. It is necessary to have more than one block to make an estimate for ξ, over which time the sampling frequency, and hence t_b may change. In this case, the average of the period of each block is used. Note that the better the estimate for frequency, the more accurate the ξ estimate will be.

Noise Free Experiments

A set of runs where the test data had zero damping, but frequencies between $f_S/4$ and 1% of this, random amplitudes and offsets, produces the results shown in table 5.5. Despite the zero damping used to allow for the long block lengths used, the accuracy of ξ is still important from this run. The AAD vectors are shown after one block (no damping estimate can be produced on a single block) and after three blocks with the ASFA applied (with no time variation of the parameters).

As theory predicts, the accuracy of the estimates increases with increased block length. Accuracy can also be increased by application of the adaptive sampling frequency algorithm (ASFA). When the ratio of sample length to n_C is greater than 2,

Block length	AAD after 1 block			AAD after 3 block		
n_C ratio	ξ	ω_n	U	ξ	ω_n	U
64/8	-	0.59	15.1	0.0005	0.44	12.4
32/8	-	1.28	30.0	0.0018	0.89	29.1
16/8	-	3.08	68.2	0.0129	2.06	61.6
8/8	-	7.22	170.1	0.0305	7.21	192.1
16/4	-	3.08	68.2	0.0041	1.39	42.8

Table 5.5: *Table showing AAD vectors for parameter estimation with FFT method after one and three blocks using the ASFA. Damping is set to zero to allow for long block lengths.*

there is approximately a 30% improvement in the frequency estimate, whilst there is a smaller improvement in U. Note, that when the ratio is less than 2, the algorithm performs poorly, and also the ASFA has no effect.

Noise Tolerance

Noise tolerance can be expected to be high for this method. The estimate for the frequency parameter is taken as the frequency bin of the largest value in the PSD. Only if noise of a different frequency has a greater magnitude will the estimate be wrong. This is unlikely, since white noise will approximately add equally to all frequency bins. Impulse noise may well produce a large high frequency content. The accuracy of the damping coefficient also relies to an equal extent on whether the correct frequency bin is selected. Additionally, the estimate for U depends on the mean of the signal. This too should not be adversely affected by white noise if its mean is zero. Impulse noise may again cause difficulties since it will offset the mean in one direction. Results are shown in table 5.6.

The ξ estimate is largely unaffected by noise, as is the frequency estimate even at

Noise Type	AAD after 1 block			AAD after 3 blocks		
	ξ	ω_n	U	ξ	ω_n	U
No noise	-	3.08	68.2	0.004	1.39	42.8
White noise Mag=0.1	-	3.08	68.3	0.004	1.39	42.6
White noise Mag=0.4	-	4.27	78.9	0.006	2.29	103.7
White noise Mag=0.8	-	6.31	221.5	0.009	3.81	143.9
Impulse noise Prob=10% Amp=0.1	-	3.08	67.7	0.004	1.40	42.9
Impulse noise Prob=15% Amp=0.4	-	3.36	76.5	0.007	1.37	51.0

Table 5.6: *Table showing AAD vectors for the high level method with a variety of noise levels and types.*

white noise magnitudes of 0.4. Only the offset is affected significantly by the noise. This high tolerance to noise can be understood with inspection of the PSD (fig. 5.4). The white noise provides some frequency content across the board, but the dominant frequency remains a high peak.

Comments

The approach used in this method yields accurate estimates of the unknown parameters, even with high levels of noise. It is also fast, with the main computational load being the FFT. For parameter estimation of a second order system it is highly suitable. The methods are not easily transferable, however, to other parameter estimation areas, since they are specific to this particular problem. For example. this approach could

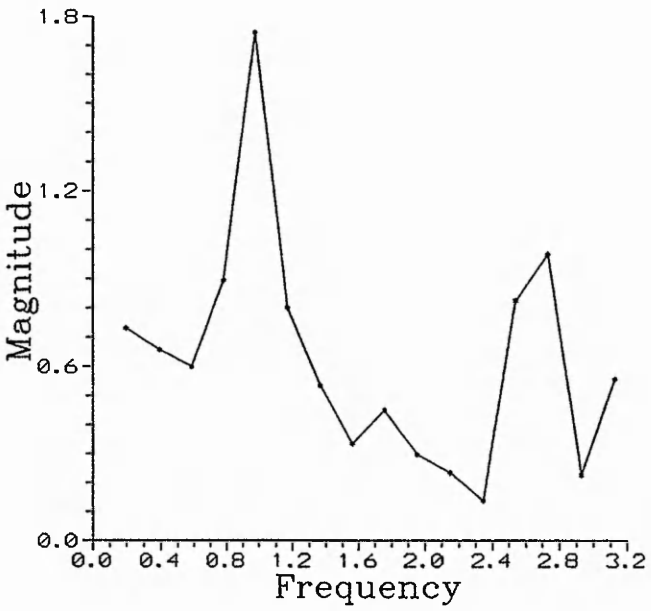


Figure 5.4: *The PSD of an input signal with white noise. The dominant frequency remains significantly proud of the background interference.*

not be used on the filter coefficient estimation [45].

5.2.3 Downhill Simplex Method

The Downhill Simplex method attempts to optimize a function by varying the dependent parameters in an iterative manner. It needs to be seeded with an initial guess on the N target parameters; from here, it generates a further N guesses which can be visualized as $N+!$ points in parameter space. These points enclose a volume known as a simplex. By repeatedly moving the point with the highest error the simplex can move through parameter space, following a downhill path on the error curve.

This algorithm is known to be quite slow but robust. It has to evaluate the error function each time a new estimate is made, and several hundred estimates per parameter set is not unusual. The algorithm finishes either when the error falls below a critical value or a pre-set maximum number of iterations is exceeded.

Two error functions are tested. The first is the RMS between the target trajectory and the trajectory generated by the estimated parameters. Use of the RMS does force the algorithm to also estimate the initial conditions of the target trajectory in addition to the main three parameters. The second error function is the sum of the correlation between the target and estimate trajectories, plus the difference in the means. The correlation function is scale and position invariant, but returns a value of unity is the two trajectories have the same shape. Adding the mean term allows the external input parameter to be optimized. Use of this error function imposes less restriction on the initial conditions, since only their ratio has to be correct, rather than their exact value as with the RMS function.

This algorithm is heavily dependent on the initial guess, and it is therefore important to make this guess as accurate as possible. To do this, the high level approach of Section 5.2.2 is employed, and as a result, the simplex method becomes an augmentation of this method. The estimated damping is always set to zero. The frequency is estimated using a Fourier transform and taking the dominant frequency as the frequency. Calculation of the mean in conjunction with the estimated frequency gives an estimate of U. The initial conditions are estimated using the phase information of the dominant frequency obtained from the Fourier transform.

$$\begin{aligned} x_{1o} &= A_D \cos \phi_D + \mu \\ x_{2o} &= -A_D \hat{\omega_n} \sin \phi_D \end{aligned} \tag{5.3}$$

where A_D is the amplitude of the dominant frequency,and ϕ_D is the phase of the dominant frequency. μ is the mean of the target trajectory.

The successful use of the simplex method will need a good initial estimate. Section 4.6 examines how the RMS error function varies between a target and estimated signal. It shows a series of local minima with one global minima when the two signals match. Initializing the simplex anywhere but in this global minima will almost

certainly result in an inaccurate estimate.

Preliminary Results

This algorithm is an augmentation of the high level approach of Section 5.2.2 which results in good parameter estimation in itself. With 100 test patterns where the damping parameter ξ ranges between $(-0.4/\omega_n, 0.4/\omega_n)$ (to prevent excessive overall damping or instability), an AAD of $(0.03, 3.2, 63.4)$ is obtained without use of the simplex. Use of the simplex algorithm subsequent to this gives and AAD of $(0.11, 0.78, 37.1)$. 15 of these have an RMS of less than 0.0001, which collectively have an AAD of $(0.03, 0.01, 0.2)$.

These results show that this method improves the estimate on frequency and external input, and when the RMS is very low, the estimate is very accurate. This is undoubtedly an improvement on the high level approach except for the damping parameter. The estimate is poorer for ξ than when a guess of zero is made! This is a reflection that the RMS is not particularly suited as a mechanism by which to determine this parameter. One pattern with an eventual RMS of 0.0002 has an AAD of $(0.8, 0.5, 1.8)$. Another way of describing the problem is that for a certain RMS error, there is a range of combinations of the parameters and starting conditions which will produce such an RMS. It is just that the lower the RMS limit gets lower, the smaller the number of possible values of the parameters which are dramatically different from those that created the target data.

Use of the correlation and mean error function does not produce good results. The AAD is $(0.031, 3.1, 64.5)$. This does suggest that the correlation function is more sensitive to damping, although the estimate is really still poor.

Adding noise to the input signal decreases the accuracy of the simplex method. Table 5.7 shows the results.

These results can be compared with table 5.6 to see what improvement the simplex algorithm gives over the plain high level approach. In all cases, the accuracy is greater with the simplex augmenting the high level algorithm. For example, with 0.1

Noise Type	AAD		
	ξ	ω_n	U
Initialization algorithm	0.03	3.16	63.4
No noise	0.11	0.78	37.1
White noise Amp=0.1	0.18	1.38	46.1
White noise Amp=0.4	0.16	3.71	161.1
Impulse noise Prob=10% Amp=0.1	0.12	0.95	37.9
Impulse noise Prob=15% Amp=0.4	0.13	1.76	59.9

Table 5.7: *Table showing AAD's for the Downhill Simplex method with different levels of noise on the input signal.*

magnitude white noise, the plain high level approach gives an AAD of $(-, 3.08, 68.3)$ (no value available for the first term), with a sample length of 16. The same sample length and noise level with the simplex gives an AAD of $(0.18, 1.38, 46.1)$, a significant improvement.

There is a distinct disadvantage to using the simplex algorithm despite its obvious benefit in estimate accuracy. It is an algorithm iterates, and so takes a significant amount of time to run, many more times, say, than the high level approach alone.

Comments

As with many of the algorithms in this study, the Downhill Simplex method can be expected to improve in performance when the adaptive sampling frequency algorithm

is used. Unlike other algorithms, this is not because a single cycle will be included in the input data block (although this may well aid accuracy), but because of the final and best estimate from a previous block can be used to seed the algorithm for the next block of input signal. This will work best when the parameters vary least over time. This is covered in Section 5.3.4.

5.2.4 Neural Networks

Neural networks can be used for the parameter estimation task since they are well-known mappers between nonlinear input and output relationships. The mapping from an input signal to a parameter such as damping or frequency is nonlinear, and so it is reasonable to use this approach. Further, neural networks are well-known for their noise tolerance, an important element of this study's investigations.

A feedforward neural network can be used to estimate the parameters of a second order system by presenting the trajectory of the signal to the input layer of the network, and training it to output the desired parameters using an algorithm such as backpropagation. By adding connections from either or both of the hidden and output layers, extra information is available for the network. When feedback connections lead from the hidden units to the input layer, they are known as Elman connections, and supply the network with a view of its own internal representation of the problem. Feedback from the output units to the input layer allows the network to view its own estimates for the problem.

To allow networks with feedback to learn, they must be presented with a series of temporarily adjacent patterns. The network can then learn to generalize over the entire length of the sequence. Training data is created, therefore, in groups, where each group consists of a number of patterns to be presented to the network, which are part of the same sequence of input signal.

Preliminary Results

Experiments quickly show that neural networks are limited in their ability for solving this problem, as found in parameter estimation with networks via derivative estimation (see Section 5.1.5). Training data that uses input signals which have a non-zero damping are not learnt, and a network trained with zero damping does not output accurate parameter values if the test data has non-zero damping. Further, input signals with external inputs which are non-zero are also not learnt well. Only frequency is estimated with any real accuracy, and this only under limited conditions of zero damping and external input.

Neural networks do benefit marginally, however, from having recursive connections. A network with no recursive connections estimates frequency less accurately than a network with connections from the output units to the input layer. Recursive connections from the hidden layer to the input do not aid frequency estimation. The AAD_{omega_n} for a non-recursive network with data with zero damping and offset, and fixed amplitude signals, is 1.2, whilst that of a recursive network is 1.1. Training is difficult, with momentum a necessary tool to obtain convergence. There is also a high sensitivity to the initial weight matrix, and several runs have to be made for each network. Adding white noise to the test data gives an AAD_{ω_n} of 1.3 for both types of network.

Varying the size of networks gives a reasonable and predictable result; that is, with more hidden units the higher the accuracy of the output, and the more likely that training will be successful. Eight hidden units is found to be the minimum number of hidden units that give good accuracy. Increasing the number of input units also improves accuracy, but again, the effect gets less as the number increases.

Experiments show that use of feedforward neural networks for direct parameter estimation, even with recursion, is limited. The problem is strongly nonlinear, and training is difficult being highly dependent on the initial weight matrix. Only frequency can be estimated with any reasonable accuracy and networks with recursive connections

aid estimation only mildly.

Use of the adaptive sampling frequency algorithm can be expected to aid the ability of networks to estimate input signals with non-zero external input and varying amplitude. Scaling of the input by first removing its mean, and then normalizing to unity, will become more accurate when one cycle of the input signal fits within one input pattern. This is covered in Section 5.3.5.

Fully Recurrent Neural Networks

It is also possible to test a fully recurrent neural network, where there are only two layers to the network. The first input layer leads as normal to a second layer. This layer is fully recurrent, where every unit is connected via a weight to every other unit, including itself. The Real-Time Recurrent learning (RTRL) [44] is used to train such networks. A network with ten input units, and nine fully recurrent units trained with blocks of input signal and scaled frequency as the output, however, fails to make any learning.

Comments

Use of neural networks for direct parameter estimation is limited. The input signals for both training and testing must be relatively simple, having fixed amplitude, zero damping and zero external input. Networks with feedback connections, and also the fully recurrent neural network do not lead to any improvement in the parameter estimation.

5.2.5 Genetic Algorithms

A Genetic Algorithm (GA) is a search method particularly suited to multimodal parameter space and also to problems which have highly nonlinear or even discontinuous solutions. Section 4.6 shows how matching an estimated signal to a target signal is multi-modal for both the RMS and correlation functions. It is also clear that the

parameter estimation problem lies in the nonlinear domain.

A population of "chromosomes" encoding the parameters in a binary sequence combine to form offspring which can assert themselves in future generations of the population, depending on how well the offspring solve the desired problem. Essentially, the GA searches for solutions to a problem near other solutions with a bias towards those solutions which appear promising. They are capable, however, of leaping from local minima to local minima in their search for an optimal solution.

Fitness Functions

A GA is used here to search for the three parameters of the second order system. The fitness of an individual in the population is determined by recreating a signal from the parameters encoded in the individual, and comparing this with the target trajectory. The most obvious method of comparison, the RMS, cannot be used without also estimating the initial conditions x_{1o} and x_{2o}. This is also true for the fitness function based on the correlation between the two signals and the difference in their means.

it is possible to devise a method where the initial conditions do not have to be estimated. This is performed by sweeping through possible combinations of the two variables, and recording, say, the fitness at each combination. Then after the sweep, the combination with the best fitness is taken as the fitness for this set of estimated parameters. This method, however, increases the computational load considerably

A critical facet of GA's is selecting a suitable fitness function. The two tested here are the RMS between the target and estimated signals, and a second function based on the correlation between the trajectories and the difference in their means. If χ is the fitness, then,

$$\chi = Corr(y_{actual}, y_{estimate}) + \frac{|\mu_{actual} - \mu_{estimate}|}{W} \tag{5.4}$$

where μ is the mean of a signal, and W is some weighting value.

To perform an optimization without directly estimating the initial conditions, each parameter estimate set is tested by sweeping the initial conditions. This is performed with,

$$\begin{aligned} x_{1o} &= \cos(\phi) + \mu_{target} \\ x_{2o} &= -\omega_n \cos(\phi) \end{aligned}$$

where ϕ is varies from 0 to 2π in a fixed number of equally spaced steps. The greater the number of steps, the higher the accuracy, but the greater the computational load.

Results

The GA is run using the one hundred test data sets and an AAD vector evaluated in the normal fashion for each run. Runs where the correlation function and mean function were used had a weighting value for W of 30.0. Discussion of the W variable is given in sectionsec:ch5-crga.

With the RMS fitness function, an AAD of $(0.031, 8.91, 269.2)$ with a sample length of 16. When the correlation fitness function is used this improves to an AAD of $(0.021, 1.79, 180.3)$. This is a predictable result; the RMS fitness function demands that the initial conditions match those of the target signal's initial conditions, whereas the correlation and mean fitness function require only that they are in the correct ratio — a more flexible criteria. This is because it is the profile of the signals which provides a low error, rather than matched values.

Attempting to optimize the estimation by sweeping through values for the initial conditions, thus avoiding the necessity to estimate them at all with the GA, does not produce beneficial results. When the initial conditions are generated by sweeping through the phase divided into twenty, the AAD with the correlation and mean fitness function is $(0.023, 2.31, 72.5)$, and the RMS fitness function is AAD is similar. This

is an improvement for U by a factor of approximately 2, but does deteriate the ω_n estimate by approximately by 35%. It can be expected that the frequency estimate will improve if the number of divisions for the phase is increased, but the algorithm will take proportionally longer to run.

It is possible to run the GA using the ASFA, even though the parameters do not vary, and still expect some benefit. The GA is run in a mode where the initial conditions *are* estimated. At the end of each block, each chromosome has the part of the bit string which represents the initial conditions replaced with binary representations of the final conditions of the estimated signal. Then, at the beginning of each block after the first, the population will be primed with chromosomes which already have a high probability of representing good solutions.

With the correlation and mean fitness function, an AAD of $(0.027, 1.93, 95.2)$ is obtained. This is slightly poorer for the frequency estimate, but as with the sweep method, an improvement of nearly a factor of 2 for the external input.

Noise Tolerance

Noise tolerance is tested with a GA using the correlation and mean fitness function over a single block of incoming data. The sample length is 16, and the frequency can range between $f_S/4$ to 1% of this value. The GA has a population of 50 chromosomes and runs for 300 generations. The results of adding different levels and types of noise are shown in table 5.8.

Noise Type	AAD		
	ξ	ω_n	U
No noise.	0.021	1.79	180.3
White noise Mag=0.1	0.026	2.24	167.5
White noise Mag=0.4	0.030	3.10	186.5
Impulse noise Prob=10% Amp=0.1	0.031	2.00	114.1
Impulse noise Prob=15% Amp=0.4	0.028	2.24	157.1

Table 5.8: *Table showing AAD vectors with the GA method using the correlation and mean fitness function, when different types and levels of noise are added to the input signal.*

5.3 Continuously Running Block Parameter Estimation

5.3.1 Introduction

In subsequent sections, experiments and results for parameter estimation of a second order system are given when performed in a continuously running mode. Importantly, the parameters of the system vary smoothly with time. A block of data is fed into the parameter estimation algorithm and an estimate is generated by this module. The Adaptive Sampling Frequency Algorithm (ASFA) is applied to adjust the sampling frequency to increase the accuracy of the estimates. The next block is then fed in, and the parameter estimation repeated.

There is an element of feedback, when the estimate for the frequency of the input

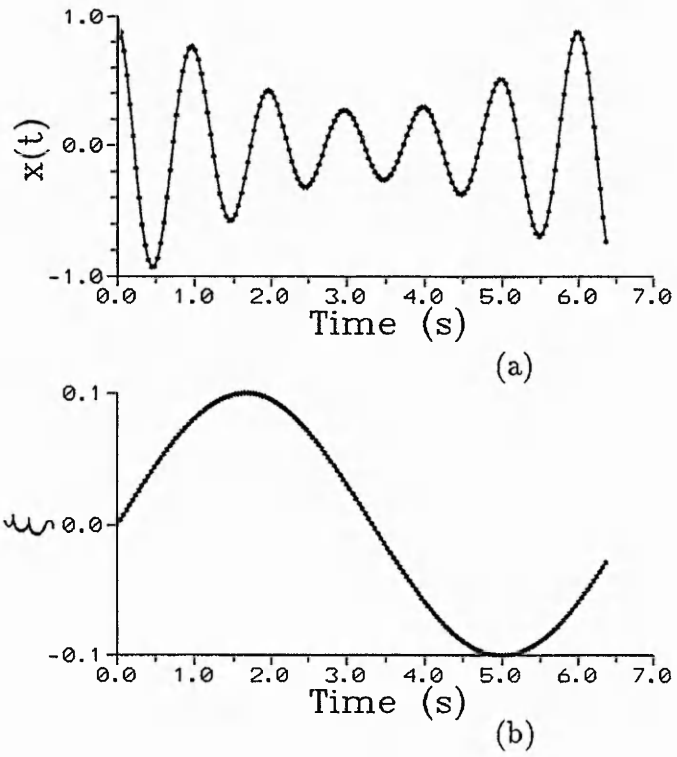


Figure 5.5: *Profile of an input signal where just the ξ parameter is adjusted in a sinusoidal fashion. a) The second order system signal, b) The ξ parameter variation.*

signal is used to adjust the sampling frequency with which the next block is read in. This is the adaptive sampling frequency algorithm (ASFA). As described in previous sections, significant improvements in the accuracy of the estimates can be expected when the ASFA is applied.

When running in the continuously running mode, the data generation module can be set to vary the value of the parameters. This will affect the profile of the generated signal, and the parameter estimation methods will be expected to track the parameter variation.

How the variation of a parameter during the course of the data generation is shown in the the following figures. Adjusting the ξ term in a sinusoidal fashion results in the profile of Figure 5.5, in the absence of other parameter variation. The amplitude modulates at the same rate as the ξ variation.

Varying the frequency ω_n (fig. 5.6) produces a double effect. Naturally, the fre-

Noise Type	AAD		
	ξ	ω_n	U
No noise.	0.021	1.79	180.3
White noise Mag=0.1	0.026	2.24	167.5
White noise Mag=0.4	0.030	3.10	186.5
Impulse noise Prob=10% Amp=0.1	0.031	2.00	114.1
Impulse noise Prob=15% Amp=0.4	0.028	2.24	157.1

Table 5.8: *Table showing AAD vectors with the GA method using the correlation and mean fitness function, when different types and levels of noise are added to the input signal.*

5.3 Continuously Running Block Parameter Estimation

5.3.1 Introduction

In subsequent sections, experiments and results for parameter estimation of a second order system are given when performed in a continuously running mode. Importantly, the parameters of the system vary smoothly with time. A block of data is fed into the parameter estimation algorithm and an estimate is generated by this module. The Adaptive Sampling Frequency Algorithm (ASFA) is applied to adjust the sampling frequency to increase the accuracy of the estimates. The next block is then fed in, and the parameter estimation repeated.

There is an element of feedback, when the estimate for the frequency of the input

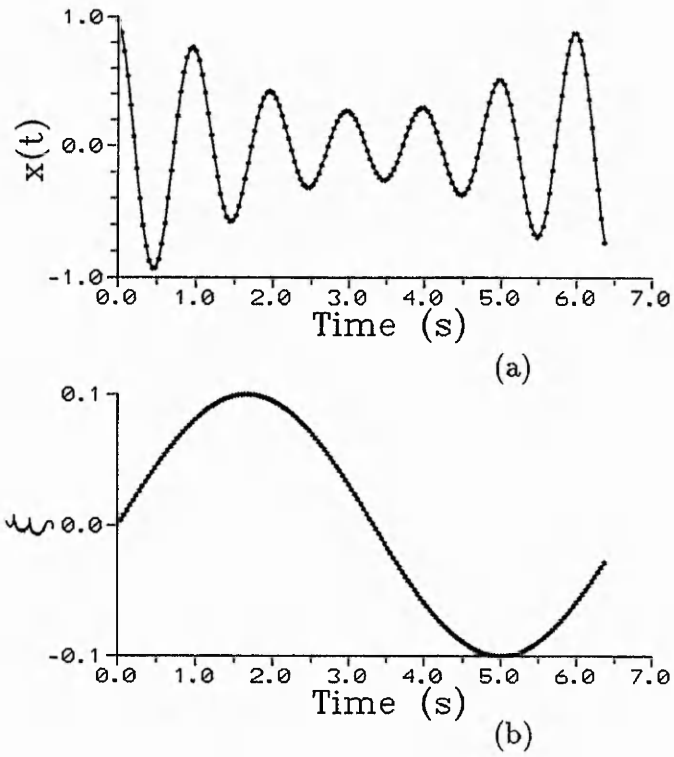


Figure 5.5: *Profile of an input signal where just the ξ parameter is adjusted in a sinusoidal fashion. a) The second order system signal, b) The ξ parameter variation.*

signal is used to adjust the sampling frequency with which the next block is read in. This is the adaptive sampling frequency algorithm (ASFA). As described in previous sections, significant improvements in the accuracy of the estimates can be expected when the ASFA is applied.

When running in the continuously running mode, the data generation module can be set to vary the value of the parameters. This will affect the profile of the generated signal, and the parameter estimation methods will be expected to track the parameter variation.

How the variation of a parameter during the course of the data generation is shown in the the following figures. Adjusting the ξ term in a sinusoidal fashion results in the profile of Figure 5.5, in the absence of other parameter variation. The amplitude modulates at the same rate as the ξ variation.

Varying the frequency ω_n (fig. 5.6) produces a double effect. Naturally, the fre-

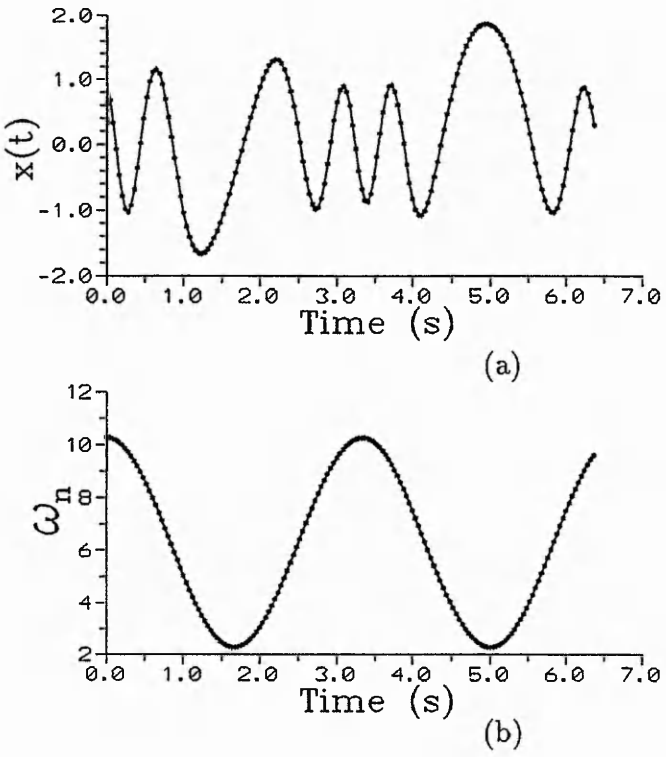


Figure 5.6: *Profile of an input signal where just the ω_n parameter is adjusted in a sinusoidal fashion. a) The second order system signal, b) The ω_n parameter variation.*

quency increases and decreases with the parameter variation. Secondly, the amplitude increases with lower frequency and vica-versa. This can be understood by considering the second order system of a pendulum in the form of a weight on a string. When set swinging it will oscillate with a fixed amplitude. If the string is shortened, the frequency will increase, and also, the amplitude will decrease. Increasing the string length will result in a lower frequency but a larger amplitude.

Varying the external input only has the effect of shifting the mean position of the signal (fig. 5.7).

Although varying each parameter individually, whilst the other two remain constant, gives rise to simple wave forms, combining variations in the parameters gives more complex profiles since both the overall damping and the mean position are functions of ω_n as well as the more obvious ξ and U respectively. This can be seen in Figure 5.8 where the variations of the three previous examples are combined in one

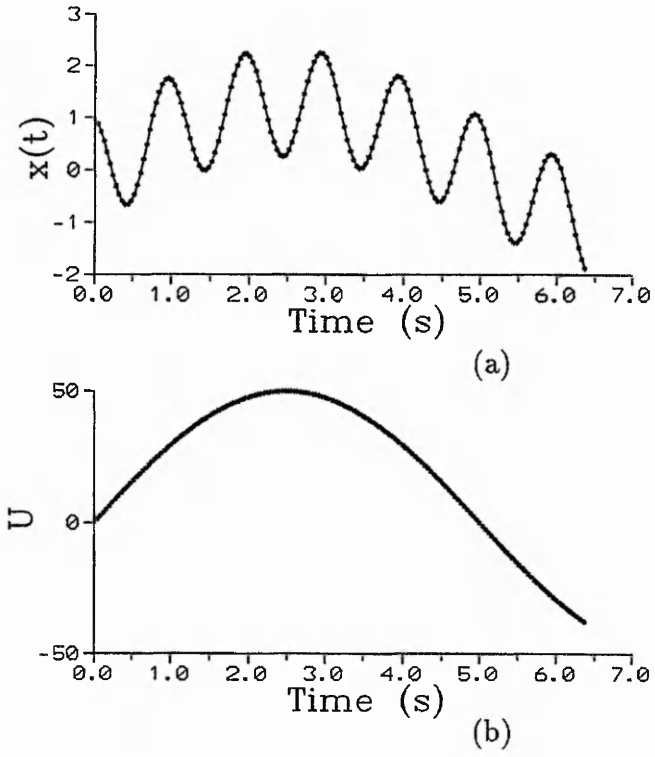


Figure 5.7: *Profile of an input signal where just the U parameter is adjusted in a sinusoidal fashion. a) The second order system signal, b) The U parameter variation.*

input signal. It is possible to tell by inspection that the frequency starts off high, grows lower, and then again higher in the course of this sample. It is not so trivial to determine, if indeed possible by eye, that the damping and external inputs are also varying.

It is the task of the parameter estimation module to give as accurate an estimate of the parameters at the end of each block.

Performance Measurement

To measure the performance of each method, an extension of the AAD vector is used whereby for a run consisting of N_b blocks of input signal, the total AAD, or TAAD, is evaluated as,

$$TAAD = \frac{1}{N_b} \sum_{i=0}^{N_b-1} AAD_i. \tag{5.5}$$

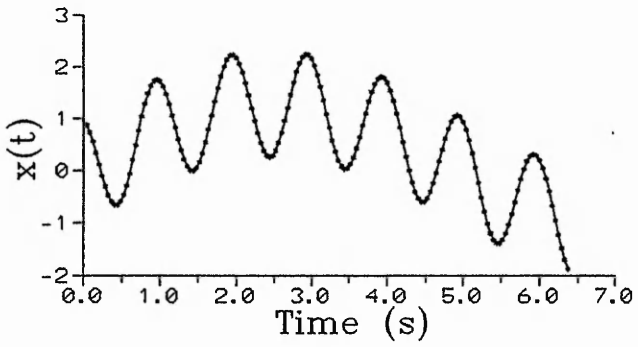


Figure 5.8: *Profile of an input signal where each parameter is varied in a sinusoidal fashion.*

This is covered fully in Section 4.5.

Test Data

Test data consists of one hundred initial parameter settings, including initial conditions. These allow a test signal to be generated, as with previous experiments. Variation of the parameters is achieved by a sinusoidal pattern proportional to the frequency parameter. It is necessary, however, to maintain the ξ parameter at zero to ensure that the signal does not become either damped to a zero magnitude, or expand to excessive values. At a given time t, the frequency and external input are based on their initial value as,

$$\omega_n(t) = \omega_n(0) + 0.3 \times \omega_n(0) \sin(0.05\omega_n(0)t)$$

$$U(t) = U(0) + 0.1 \times U(0) \cos(0.04\omega_n(0)t)$$

Hence, the frequency will change in magnitude by 30% with at 1/20th of its actual frequency. Similarly for the external input, it will vary by 10% of its initial value at 1/25th of the initial frequency. This variation in parameters gives significant alteration to the initial values which the estimation algorithms must be able to cope with. Making the variations proportional to the initial frequency ensures that no excessive or too rapid a change in the parameter values is seen. With the random nature of the initial values,

the test signals therefore represent a constrained but variable, and meaningful test set.

Unless otherwise stated, experiments for evaluating a TAAD vector run for ten blocks.

5.3.2 Derivative Estimation For Continuous Parameter Estimation

This section continues the work of Section 5.1.3 where the parameters of the second order system are estimated based on a polynomial least squares fit to the input trajectory. This method is the most effective tested and so is passed on for continuous parameter estimation.

Preliminary Results

As shown in Section 5.1.3, it is possible to get an arbitrary accuracy on the parameter estimation using polynomial least squares fitting when no noise is present. This is also true for continuous parameter estimation when the parameters vary with time, except that to achieve the same degree of accuracy with time varying parameters, the polynomial order (and hence the sample length) must be increased. For example, when there is no parameter variation, a sample length of 7 and maximum order 6, results in a TAAD of $(0.002, 0.22, 9.6)$. To achieve the same order of accuracy when the parameters vary with time a sample length and corresponding maximum order of twelve is required.

This can be explained by recalling that the derivatives are calculated at the midpoint of the signal, since this is where the higher orders of the polynomial have their most accurate values. The TAAD takes the target parameter values from the end of the block, and since these are different in the parameter varying case from the centre of the signal, the TAAD will be greater.

Noise Type	Out of range	AAD		
	$\|\xi\| > 0.6$	ξ	ω_n	U
No noise	0	0.008	0.36	13.2
White noise 0.1 mag	0	0.065	1.73	64.9
White noise 0.4 mag	4	0.23	76.0	59,000
Impulse noise 10% prob 0.1 mag	0	0.028	1.37	54.7
Impulse noise 15% prob 0.4 mag	1	0.104	8.00	1519

Table 5.9: *Table showing TAAD vectors for parameter estimation using derivative information obtained from polynomial least squares fitting. Noise of different types and magnitudes is added to the input signal.*

Noise Tolerance

In a more realistic scenario, noise will be present on the input signal, and in this situation, the maximum order for a given sample length cannot be used, as described in Section 5.1.3. Table 5.9 shows the results of running test signals where noise is added to the incoming signal. Here, the sample length is 12, with a polynomial order of 7, and n_C set to 8.

Experiments show that when noise is added to the signal, even relatively small amounts eg. 0.1 magnitude white noise, this algorithm cannot estimate with any certainty low frequency signals in the first data block. As a result, the ASFA sets an inappropriate sampling frequency and subsequent blocks also give inaccurate estimates. With an initial sampling frequency of 25Hz, signals above approximately 1Hz are estimated correctly, whilst those below result in inaccurate estimates. The data

sets for table 5.9 use frequencies between 1 and 6.25Hz. If frequencies below this are required when noise is present, then a lower initial sampling frequency must be used.

Comments

When white noise of magnitude 0.4 is added, the TAAD has unreasonable values. Although many of the test signals have much more reasonable AAD's, this result does indicate the instability of this algorithm with these parameters. Increasing the sample length does allow for improved performance - increasing from 12 to 24 gives a TAAD of $(0.167, 17.2, 310)$. Again, overall unacceptable, but it does indicate that more of the test signals are being correctly estimated and "locked onto" by the ASFA.

5.3.3 High Level Method

This method, using a Fourier transform to extract features of the signal, is an accurate parameter estimation approach as demonstrated in Section 5.2.2. It is also particularly resilient to noise corruption.

Since the accuracy of parameter estimates in this method are dependent upon the frequency estimate, with both U and ξ being functions of ω_n, higher accuracy can be expected with a long sample length allowing the FFT to return a more precise value on the dominant frequency. If the ASFA were not being used, the benefit obtained from long sample lengths would have to be countered with the fact that the frequency in the sample is changing with time and a short sample would give the most accurate result. Due to the ASFA where n_C points occur per cycle, a suitable value for the ratio of N/n_C must be maintained, where N is the sample length.

Preliminary Results

If the ratio N/n_C is high eg. $N = 64$ and $n_C = 4$, then after a few blocks, approximately eight cycles would be in the sample the FFT performs on. In this time, a significant

N/n_C	TAAD		
	ξ	ω_n	U
16/4	0.009	2.5	47.4
32/8	0.009	2.6	46.0
64/16	0.0083	3.1	56.4

Table 5.10: *Table showing the TAAD of one hundred test signals with different sample lengths, but kept in a fixed ratio of N/n_C. Ten blocks are used.*

change in the actual frequency can occur and so an erroneous result returned. With a ratio of one, however, the ASFA would be not be able to operate properly since the sampling frequency could not be adapted to low frequencies.

A ratio of two or four is suitable. Improved results are obtained by increasing the sample length. This is not a problem after a few blocks, since the ASFA adjusts the sampling frequency so that only a sample where the frequency change is small is used. Results are shown in table 5.10.

There is a danger which must be avoided. If the sample length is too large, the first block will cover a large segment of time, over which the frequency variation itself may complete at least one cycle. This can result in a low frequency oscillation becoming dominant in the Fourier transform, giving a grossly inaccurate result. This problem is due partly, however, to the specific manner in which the test data is created, and may not always apply in a real world situation.

Although the accuracy of the damping and frequency estimates improves as the sample length improves with the fixed ratio, the estimate of external input does not follow the same pattern. The estimate for U is based on the mean of the signal segment, as well as the frequency estimate. The variation of the two are independent and it can be only this which explains the reduction in accuracy.

Figure 5.9 shows the actual and estimated frequency of a signal over the course of

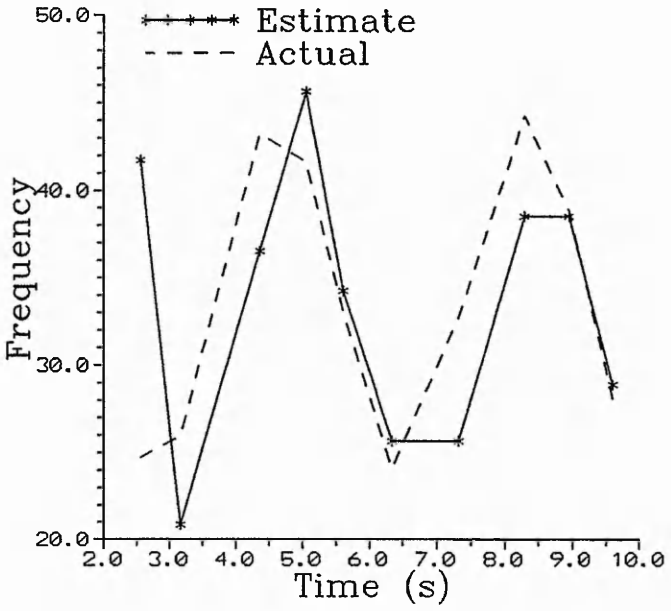


Figure 5.9: *Figure showing the estimated and actual frequency of a signal over the course of a run.*

a run, with a data point collected at the end of each block. It can be seen how the estimated frequency lags actual frequency.

Noise Tolerance

Noise tolerance is very good with moderate noise levels (see table 5.11). When white noise of magnitude 0.3, or impulse noise of magnitude 0.1 and probability 10% is applied, the AAD at the end of each run is almost unchanged from the tests where noise is absent, and indeed, is slightly lower.

As the noise levels increase, however, the chance of a different frequency other than the actual frequency being dominant in the PSD becomes more probable. When this happens it is a catastrophe for the algorithm in its current form. If the wrong frequency is returned as dominant, the sampling frequency for the next block of data is inappropriate. This easily leads to the actual signal being lost to aliasing or swamped

in a low frequency bin. It is possible for the estimated frequency to explode into unrealistic values.

When the parameters are varying, the system is more susceptible than when the parameters do not vary. This is because the dominant frequencies of the signal may cross frequency bin boundaries, and so the power of the signal will be spread across more than one frequency bin. As a result, noise does not have to be of such a great magnitude to become dominant. So, although in Section 5.2.2 white noise levels could rise up to 0.4 and above without problems, the level is much lower in this situation.

Safeguards could be put in place which check that the estimated frequency is not going beyond expected bounds, or that the dominant frequency is not too different from the dominant frequency in the previous block. Once noise levels have risen above a certain level, however, it is unreasonable to expect this approach to be able to cope consistently. It can be pointed out though, that once noise has risen to such levels, it is unreasonable to expect any method to extract a signal from so much noise.

5.3.4 Downhill Simplex Method

In Section 5.2.3, the Downhill Simplex algorithm is applied to the initial stage of parameter estimation when an estimate is required for the first block of incoming data. In many cases, results were very good, with highly accurate results being obtainable where the RMS fell below 0.001 and parameter estimation is approximately to three significant figures. There are also failures amongst the successes, however, where the simplex sought a solution in a non-global minima. Within the permitted number of function evaluations, it was not able to find a solution near the actual.

The task for the algorithm is made more difficult in the continuously running mode of operation. Previously, the function that the simplex algorithm used to generate its estimate signal from the estimated parameters was identical to that which generated the target signal from the target parameters. In continuously running mode, this is no

Noise Type	TAAD		
	ξ	ω_n	$\dot{U}$
No noise	0.009	2.48	47.4
White noise Mag.=0.1	0.009	2.49	47.8
White noise Mag.=0.4	0.010	3.63	79.0
Impulse noise Prob=10% Amp=0.1	0.009	2.48	47.7
Impulse noise Prob=15% Amp=0.4	0.010	2.61	66.3

Table 5.11: *Table showing TAAD for test signals estimated using the continuously running High Level method, with noise added to the incoming data.*

longer the case, since the actual signal is generated using two additional parameters — the frequency and external input variation parameters. This study does not use a simplex which attempts to find the variation rates, although it is capable and could lead to a more accurate parameter estimation method.

There are further problems for this algorithm when used in the continuously running mode; a run made up of say, ten blocks of data, has ten opportunities to go wrong. This is significant because at the end of each block, the estimated parameters and final conditions are used to seed the algorithm at the start of the next block. Since the performance of the algorithm is heavily dependent on the starting estimated parameter values, when these are wrong, the algorithm has little chance of getting back on the right track and there is a high probability that all subsequent estimates will miss the mark. The whole system may also go unstable if the ASFA is used and inappropriate sampling frequencies are used to read in the next block. In brief — if the algorithm gets the estimates significantly wrong on one data block, it will probably get all subsequent estimates wrong.

As in the previous section for this algorithm, the initial estimate to seed the simplex is based on the results of applying the high level approach of Section 5.2.2. At the end of the algorithm's iterations, its estimates for the parameters and the final state of x_1 and x_2 are used to initialize the algorithm for the next block of input data.

Preliminary Results

In Section 5.2.3, 3% of the test signals concluded with an estimate of frequency which was over a factor of two out. An equivalent run where the parameter values were changing with time, 5% of the signals were incorrectly estimated by a factor of two or more. All such cases are less than approximately 1Hz. The problem is due to the initialization method which uses an FFT, which can only return quantized values for the frequency, and at the low frequency end this means that frequencies are estimated incorrectly by over a factor of two. It has be shown that it is difficult for this method

Number of blocks	TAAD		
	ξ	ω_n	U
1	0.002	2.29	46.7
10	0.006	1.92	48.7

Table 5.12: *Table showing TAAD in non-continuously and continuously running mode.*

to escape such local minima (sec. 4.6). The effect can be eradicated by limiting the frequency allowed in the test data to greater than 1Hz or by reducing the initial sampling frequency, in which case the maximum frequency tested must be reduced. Alternatively, the effect can be tempered by using a longer sample length so the FFT can return a more accurate frequency estimate.

Table 5.12 shows the TAAD vectors for the test data in both the non-continuous and continuously running modes. There is no great difference in performance between the simplex method applied to a single block of data or after ten consecutive blocks, except in the case of the frequency estimate.

Figure 5.10 shows the frequency estimate for one test vector over the ten blocks. Although the estimate tracks the actual value for the majority of the run, at the end the estimate is nearly half of the target. With the penultimate block's estimate beginning to wander, this confirms the difficulty that this algorithm can experience when seeding with the previous blocks estimate. It is also an important indicator, that the failure of the algorithm occurs on the most rapidly varying section of the graph.

Noise Tolerance

Tolerance to noise is tested by adding white noise and impulse to the test signals as they are generated. Each signal lasts for ten blocks and the ASFA is used. Results are shown in table 5.13.

It is evident that noise does result in a reduction in performance for this method.

Noise Type	TAAD		
	ξ	ω_n	U
No noise	0.006	1.92	48.68
White noise 0.1 mag	0.007	2.22	62.6
White noise 0.4 mag	0.006	2.74	89.1
Impulse noise 10% prob. 0.1 mag	0.006	1.98	50.1
Impulse noise 15% prob. 0.4 mag	0.006	1.91	60.5

Table 5.13: *Table showing TAAD for test signals estimated using the continuously running Downhill Simplex method, with noise added to the incoming data.*

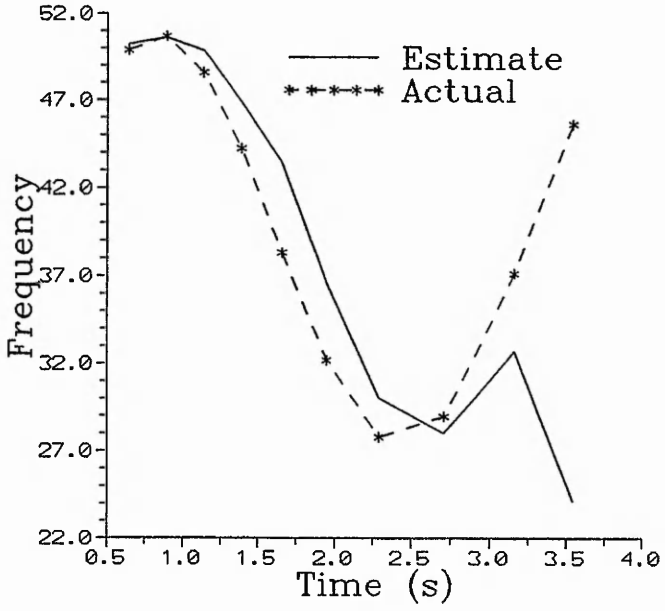


Figure 5.10: *The actual and estimated frequency of a test signal. The last two blocks are poorly estimated.*

It is interesting to note that the performance degrades proportionally with noise magnitude, as one would expect.

Comments

This algorithm works well when the initial estimate for seeding is within the global minima. When this is not the case, however, it cannot remove itself from the incorrect local minima. This poses particular problems when consecutive blocks depend upon prior estimates for seeding purposes.

Otherwise, this algorithm has been robust, and behaves well when noise is added. It is, however, slow. Many evaluations of the functions require evaluating for the this algorithm.

5.3.5 Neural Network For Continuous Parameter Estimation

In Section 5.2.4 it is shown that neural networks can be used successfully for direct parameter estimation with certain restrictions. These are that the input patterns need to be of constant amplitude and zero damping and external input. The networks are unable to generalize with training data that has inputs patterns which do not comply to these restrictions, even with pre-scaling.

It is reasonable to expect a neural network to have further difficulty with parameter estimation when the frequency parameter varies with time. In training, the neural net is exposed to signal data that has a fixed frequency. Yet, when used in this application, the input pattern is no longer of a single fixed frequency, but rather, in some fashion varies.

Results

A feedforward neural network with sixteen input units, twelve hidden units and one output unit to estimate the frequency is trained with training data which conforms to the restrictions. Further, the training data consists of signals with a fixed frequency. The network is then used in forward pass mode. The estimated frequency is used in the ASFA, where the estimated frequency is adjusted by the ratio of f_T/f_S, where f_S is the current sampling frequency, and f_T is the sampling frequency used to generate the training data for the network. This is in addition to the scaling of the frequency output.

In the tests, only the frequency is varied in a sinusoidal fashion as before. This is compared with similar test signals whose parameters do not vary with time. The former set is also tested when noise is added to the signals. The results are shown in table 5.14. Only the TAAD of the frequency parameter is shown since both ξ and U remain zero throughout all tests.

When there is no variation in the frequency parameter over the ten data blocks, the

Signal Type	ω_n TAAD
No freq var., No noise	0.96
With freq var. No noise.	2.70
White noise 0.1 mag	2.81
White noise 0.4 mag	4.32
Impulse noise 0.1 mag 10% prob	2.77
Impulse noise 0.4 mag 15% prob	3.74

Table 5.14: *Table showing TAAD for different test signals, using a neural network for parameter estimation.*

network is able to estimate the frequency with good accuracy (TAAD= 0.96). When the signal's frequency varies with time, however, the TAAD degrades to 2.7. In this situation, the network is being presented with subtly different signals to those it is trained with, and the difference manifests itself in the reduced performance. It would be possible to train the network with test signals which did vary in time in the same fashion as the test data. This is only a practicable idea if the manner of the parameter variation is known in advance since there is literally an infinite number of different ways the signal could vary, and it is not sensible to propose to train a network for each of many possibilities.

Noise tolerance is reasonable, with only significant degradation in performance occurring when noise levels are high.

Comments

Use of neural networks for parameter estimation is limited to a narrow range of possible situations. Damping and external input must remain zero, as well as the signal amplitude remaining constant. Noise tolerance is, however, reasonable.

5.3.6 Genetic Algorithms For Continuous Parameter Estimation

Section 5.2.5 uses a genetic algorithm (GA) to estimate the parameters of a second order system for a single data block, and when the parameters do not vary with time. There are only minor adjustments which are required for the algorithm to work effectively on a set of continuous blocks, and employing the ASFA.

As with the Downhill Simplex method, a GA estimates not just the parameters, ξ, ω_n and U, but the initial conditions too, x_{1o} and x_{2o}. This is necessary since the GA works by comparing a signal generated by the estimated parameters with the original input signal. Whereas the parameters are slowly varying with time, the initial conditions from one block to another are wildly different. As a result, the estimates for x_{1o} and x_{2o} need updating in the GA population when a new block is started. The encoding of the parameters in the population remains unadjusted. Although it is possible to have a fitness function that doesn't directly depend on the GA to estimate the initial conditions, the results of Section 5.2.5 indicated that the performance of such a fitness function is not high, and it increases the computational load of the algorithm considerably.

The updating of x_{1o} and x_{2o} is achieved by decoding the each chromosome in the GA population, and calculating the final conditions. These are then encoded and replace the old initial conditions in the chromosome.

It is important to note a problem also experienced by the Downhill Simplex method, which is that the fitness function fits a second order system signal to a target signal

where the parameters are varying in time ie. it attempts to match the target signal with something only similar to it.

GA Parameters

There are a large number of factors controlling a GA, although they are well known for their robustness, and can perform well with a wide range of settings. For example, the mutation and crossover probabilities can have a wide range of values and yet the GA will still produce good results.

Despite this, some experiments are designed to test the difference in performance of the GA with a range of varying factors. As well as the study of the influence of noise, five other points are investigated;

1. Number of generations. Runs are made to confirm that increasing the number of generations increases convergence.

2. Checking whether maintaining the same population throughout a series of blocks is better than starting with a new population at the start of each block.

3. When a population is maintained throughout a parameter set, does adding a perturbation to the initial conditions aid performance?

4. How can the fitness function be varied to improve performance?

Generations

There are two parts to finding the influence of more generations. Firstly, a run is performed where only one block per set is made, and then secondly, a run with ten blocks is performed. Each is done for a variety of generations. The results are shown in table 5.15.

It can be easily understood that when only one block per target parameter set is performed, the estimation becomes more accurate as the number of generations increase

Number of generations	TAAD 1 block				TAAD 10 blocks		
	ξ	ω_n	U		ξ	ω_n	U
100	0.006	3.49	279.2		0.015	2.71	147.6
200	0.012	3.13	98.4		0.016	1.89	81.8
300	0.011	3.03	72.7		0.016	1.79	67.5
400	0.015	2.97	69.8		0.022	2.52	114.90

Table 5.15: *Table showing the TAAD vectors for one block and ten block runs where the number of generations of the GA are increased.*

— the extra generations allow a lower error to be achieved in the population. When the mode of operation is continuous, however, and the population is maintained throughout the ten consecutive blocks, the performance drops between 300 to 400 generations.

This phenomena is due to a too high a degree of convergence in the population. With each consecutive block the population becomes less diverse, and the chromosomes in the population represent highly similar solutions. Then, as the next data block brings in a new signal with different characteristic parameters, the population is unable to adjust via the main operator of crossover. Only the weakly contributing mutation operator can shift the population. When the number of generations is lower eg. 200, the population is not able to converge to the same extent and stagnate in one particular solution. Instead, a diverse selection of chromosomes is maintained which are able to contribute usefully by supplying a range of chromosome bits in the crossover operator.

Continuous Populations

In the previous section where the number of generations is adjusted, the population could stagnate and would fail to adapt if the number of generations grew too high. This effect is caused by the use of a continuous population ie. one that is created at the beginning of a parameter set search, and remains until all through the blocks. The

alternative is to completely re-create a population at the beginning of each block. This does not incur any extra computational work, since even in the continuous population each chromosome needs its initial conditions updating and re-ranking.

A run where the population is maintained has a TAAD of $(0.016, 1.79, 67.5)$. A TAAD of $(0.014, 1.95, 152.1)$ is obtained when the population is re-created at the beginning of each block. This indicates a benefit in using a continuous population.

Here, there were 300 generations. Improved accuracy may be obtained by using a re-created population and increasing the number of generations. This approach will not incur the problems of stagnation but will require greater computational time.

Initial Condition Perturbation

When running a continuous population, the initial conditions for each chromosome are updated at the end of each block so that they represent the new initial conditions for the start of the next block. Due to the parameter variation, however, these will not be wholly accurate and some benefit could be obtained by perturbing the new values represented.

This is achieved at the end of each block by adding a small random value proportional to the initial conditions' value. ie.

$$x_{1o}(new) = x_{1o}(old) + p \times x_{1o}(old). \tag{5.6}$$

Table 5.16 shows a table of results where p varies. It is clear that this approach does not improve GA performance.

Fitness Function Variation

In Section 5.2.5 the RMS function is shown to be a poor fitness function. This is true also in the continuously running case. There is also the possibility of varying the correlation and mean function.

P	TAAD		
	ξ	ω_n	U
0%	0.016	1.79	67.5
1%	0.018	1.72	71.0
5%	0.024	2.27	88.8
10%	0.014	2.30	93.5

Table 5.16: *Table showing the TAAD vectors when the perturbation percentage varies.*

The variable W from Equation 4.4 can be varied to adjust the relative importance of the mean function to the correlation function. Figure 5.11 shows a graph describing the TAAD values for the frequency and external offset values for a range of values for W. When the value of W is large, the significance of the mean function is less, and the profile of the target and estimated signals is more important. As a result, the frequency TAAD is smaller. The inverse of this affect is observed when W becomes smaller.

Noise Tolerance

Based on the experiments described above, a GA can be produced which will give a high performance, and can be tested when noise isp resent on the input signals. The GA parameter settings are to have a continuous population over the ten blocks of each test set, no initial condition perturbation, a fitness function utilizing the correlation and mean functions with W set to 30, and a maximum number of generations of 300. Table 5.17 shows the results when various types and magnitudes of noise are imposed on the input signals.

Comments

A genetic algorithm (GA) is a general optimization method that is capable of estimating the parameters of a second order system to a good degree of accuracy. It is able to

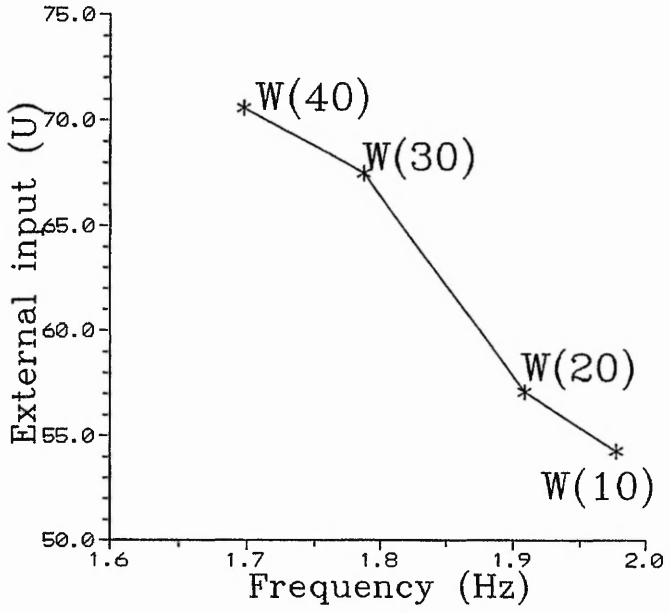


Figure 5.11: *Variation of TAAD's for frequency and external input when the W variable is adjusted for the fitness function.*

produce good estimates under a wide range of settings.

It has been shown that the best fitness function combines the correlation function between the target and estimated signals, and the difference between the means. For continuously running, the chromosomes in the population need to be updated to represent the last known conditions in preparation for the next block.

5.4 Discussion

As described in Chapter 3, the continuously running parameter estimation algorithm requires an estimation algorithm to seed a recursive method. It turns out that all algorithms can be used for the initial estimation, whilst only a few can be adjusted to benefit from a recursive mode of operation.

Given below is a summary of the results obtained by the experimentation discussed in this chapter. The first section describes the performance of algorithms for a single

Noise Type	AAD		
	ξ	ω_n	U
No noise	0.022	1.74	82.5
White noise Mag=0.1	0.020	2.23	86.3
White noise Mag=0.4	0.019	3.41	121.3
Impulse noise Prob=10% Amp=0.1	0.024	2.04	80.0
Impulse noise Prob=15% Amp=0.4	0.023	2.44	114.3

Table 5.17: *Table showing TAAD vectors for parameter estimation using a genetic algorithm (GA). Noise of different types and magnitudes is added to the input signals.*

block of incoming data where the parameters are time invariant. This gives a measure of which algorithms are best suited to the first stage of the complete continuously running algorithm. Both noisy and noiseless input signals are summarized. Following this is a summary of results obtained for time varying parameters, and where the adaptive sampling frequency algorithm is used.

5.4.1 Initial Parameter Estimation

Each estimation algorithm is tested for parameter estimation with a single block of incoming signal. The signals are generated from parameters that do not vary in time. Table 5.18 shows the results of each method when no noise is present on the signal. The ranking is dependent on the accuracy of the frequency estimate.

All the algorithms had only a short sample of data to make the estimation with. Typically the length is 16 data points sampled at 25Hz. The sample is kept short since

Estimation Algorithm	Out of range $\lvert\xi\rvert > 0.6$	AAD		
		ξ	ω_n	U
Difference equations	20	0.26	0.16	12.4
Polynomial LS	0	0.04	0.3	51.3
Simplex with RMS	0	0.11	0.78	37.1
FFNN's via deriv. est.	0	0.04	0.91	36.4
FFNN via direct est.	-	-	1.2	-
GA with corr+mean	0	0.021	1.79	180.3
FFT (Block length=16)	-	-	3.08	68.2

Table 5.18: *Summary of AAD vectors for estimation algorithms with noise free input signals and non time-varying parameters after one block. Ranking is in order of accuracy of frequency estimate.*

the methods may wish to be used in a parameter varying situation, and short samples are required so that the variation can be tracked and updated frequently. It can be expected for some methods, however, that longer samples will improve accuracy, for example the high level approach using an FFT will increase its frequency estimate in proportion to the length of the sample.

Although the difference equation method produces the most accurate frequency and external input estimates, its damping estimate is poor. More importantly, 20% of the estimates have such poor estimates for damping they are excluded.

Polynomial LS fitting gives the highest damping accuracy, and the next best frequency estimate, although the external input estimate is relatively poor. It may be recalled that an arbitrary accuracy can be obtained with polynomial LS fitting when no noise is present.

Use of neural networks for derivative estimation produces relatively good results across all three parameters. However, it should be noted that the network doesn't per-

form well with signals that have non-zero damping, and the results shown are generated when such signals are used in the testing.

Another consideration is the computational complexity of each algorithm. This will be dealt with in depth in Section 5.4.3.

Table 5.19 shows a table of results for the estimation algorithms when noise is present on the incoming signal. Results for white noise of magnitude 0.1 and impulse noise of amplitude 0.1 and probability 10% are shown.

Performance of each algorithm is similar for each type of noise with only slight variations between them. Polynomial LS fitting ranks top in both situations due to its good frequency estimation. Its damping and external input estimation is poor, however, relative to the other methods. Further, it also fails on its damping estimate completely in a good proportion of cases. Difference equations have an even greater failure rate when noise is present.

It is interesting to note the accuracy with which the GA can estimate the damping coefficient despite its relatively mediocre ranking. This is most likely due to the correlation function used in the fitness function of the GA, which takes into consideration the differences in the profiles between the target and estimated signals.

In considering which of the algorithms is "best", it must be remembered the purpose of using the algorithms in this fashion, which is to give an estimate of the parameters which can then be fed to another algorithm, which will probably operate in a recursive mode. This algorithm will require a good estimate of all three parameters. Possibly more important than this, however, is to consider the effect of the adaptive sampling frequency algorithm. This depends only on the frequency estimate, and a good frequency estimate will provide the following algorithm to operate optimally. This algorithm, with a good estimate for frequency, can then make accurate estimates of the other two parameters even if the seed value is poor.

As a result, despite the polynomial LS algorithm's lack of accuracy in the damping

field, since it produces the most accurate frequency estimate when noise is present, and the second most accurate when noise is absent, it is better that it should be selected as the algorithm used in the first stage of the continuously running block algorithm. It is possible though, that specific applications will require more accurate initial estimates of damping and external input.

5.4.2 Recursive Block Estimation

Continuously running parameter estimation differs from the initial parameter estimation in two important ways; Firstly, the parameter values vary with time. Secondly, several of the estimation algorithms use information from prior blocks to aid estimation in the current block.

In the simulations, the parameter values vary sinusoidlally, except for the damping parameter which is kept at zero to prevent the signal value either vanishing to zero or exploding. Each run lasted ten blocks and the adaptive sampling frequency algorithm is used to increase the accuracy of the estimates. Measurement of performance uses the TAAD (Total Absolute Average Difference) which compares the estimates with the actual values after each block in a run.

Tables 5.20 shows a summary of the TAAD vectors obtained for each of the methods used under these conditions under three noise situations. Namely, no noise present, white noise of magnitude 0.1 and impulse noise of amplitude 0.1 and probability 10%. Ranking is in order of accuracy of the frequency parameter.

In all situations the polynomial LS method gives better results by a significant margin. When noise is absent in particular the estimate of all three parameters is high.

Both the Simplex and GA methods have broadly similar results, although the GA appears less able to estimate damping. Interestingly, the Simplex is superior to the High Level approach. This is because the since the Simplex initializes itself with the

High Level algorithm.

As presented, the summary results do not give full note of the ability of the High Level method. This algorithm is particularly resilient to noise and can give results similar to those presented here even when noise levels are four times, or even more, the level.

5.4.3 Computational Complexity

In computation quantities, the algorithms vary significantly (fig. 5.12). All the algorithms take more computations the longer the block length, but in different proportions. The Simplex and the GA varies linearly with block length, although they do have significant other overheads. The NN algorithm varies depending on the size of the network used and lies somewhere close to the proportion of the square of the block length. The High Level method which uses an FFT has a workload proportional to the $N \log N$, where N is the block length. Finally, the polynomial LS method's computations is proportional to the square of the block length.

Having stated these figures, however, it must be noted that the FFT, the LS fit and the NN are fast algorithms, especially when compared to the Simplex and GA methods. These latter two make many repetitions of calculations per block, whereas the others make only a single sweep per block.

In practice, therefore, the Simplex and GA methods are very slow to run and could not realistically be expected to run in a real-time mode. They could be used in a post-processing scenario, and with increased iterations, could possibly outperform the polynomial LS method.

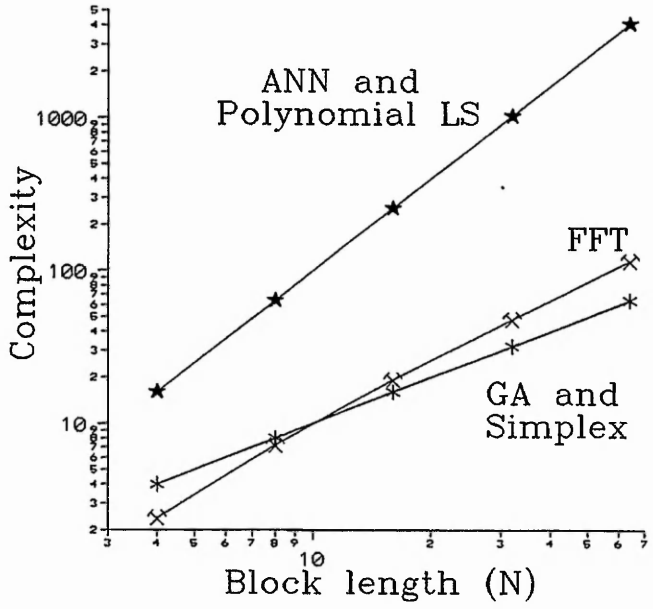


Figure 5.12: *Graph showing the complexity of each algorithm as a function of block length. N.B. Each axis is nonlinear.*

White noise Mag= 0.1				
Estimation Algorithm	Out of range $\lvert\xi\rvert > 0.6$	AAD		
		ξ	ω_n	U
Polynomial LS	14	0.18	1.0	137
NN direct	0	-	1.3	-
Simplex	0	0.18	1.38	46.1
GA with corr+mean	0	0.026	2.24	167.5
NN deriv. est.	6	0.6	2.4	69.5
Difference equations	70	0.38	2.9	685
FFT Method	0	-	3.1	68
Impulse Noise Amp= 0.1 Prob= 10%				
Polynomial LS	9	0.1	0.56	76.5
Difference equations	34	0.3	0.7	130
Simplex	0	0.12	0.95	38
NN deriv. est.	4	0.04	1.6	51.0
GA with corr+mean	0	0.031	2.0	114
FFT Method	0	-	3.1	68

Table 5.19: *Summary of AAD vectors after one block for estimation algorithms with white noise magnitude* 0.1, *and impulse noise with amplitude* 0.1 *and probability* 10% *on input signals. Parameters are non time-varying. Ranking is in order of accuracy of frequency estimate.*

White noise Mag= 0.1			
Estimation	AAD		
Algorithm	ξ	ω_n	U
No Noise			
Poly LS	0.008	0.36	13.2
GA	0.022	1.74	82.5
Simplex	0.006	1.92	48.7
High Level	0.009	2.48	47.4
NN via direct	-	2.7	-
White Noise Mag=0.1			
Poly LS	0.065	1.73	64.9
Simplex	0.007	2.22	62.6
GA	0.020	2.23	86.3
High Level	0.009	2.49	47.8
NN for direct	-	2.81	-
Impulse Noise Amp=0.1 Prob=10%			
Poly LS	0.028	1.37	54.7
Simplex	0.006	1.98	50.1
GA	0.024	2.04	80.0
High Level	0.009	2.48	47.7
NN via direct	-	2.77	-

Table 5.20: *Summary of TAAD vectors after ten blocks for estimation algorithms with no noise, white noise magnitude* 0.1, *and impulse noise with amplitude* 0.1 *and probability* 10% *on input signals. Parameters are time varying. Ranking is in order of accuracy of frequency estimate.*

Chapter 6

Conclusions and Further Work

6.1 Conclusions

6.1.1 Stereo Camera Vision System

This study has examined the feasibility of a stereo camera vision system that could be used in a real-world situation to monitor parameters of objects it observes. Consideration has been given to the types of noise that may be experienced by the vision system which has been reflected in the extensive number of experiments carried out.

As described in the first chapter, the vision system consists of two "pin-hole" cameras situated at known positions in the camera reference frame. Variation of the cameras' angles allows different parts of the world to be viewed. Such a system as this is able to pin-point the location of a point object within view of both cameras. A method for performing this triangulation is given.

This study has identified a number of options for improving the quality of information extracted from a vision system by a post-processing approach. Once the location of the tracked object within the stereo images is found, and their position in the real-world reference frame calculated, the methods in this study can be used to reduce the corrupting effect of noise when estimating the parameters of the object. Some methods, such as the polynomial least squares, is fast, and could be performed in real-time.

Other methods, such as the iterative simplex and GA ones, would need to be run off-line.

As covered below, the sampling rate of the cameras is required to vary to optimize results. In conventional cameras this will not be possible and so it is envisaged that any such vision system would use some electronic means of image storage as used, for example, by CCD's.

Although not dealt with here, it is assumed that the processing knowledge and power exists to extract the location of objects within a camera's image. This information can then be passed to other processing algorithms to obtain characterizing parameter values for observed objects.

6.1.2 Second Order Systems

The algorithms presented in this work are not, however, limited to input from a vision system. Any measurement device that read a signal at a known sampling rate will present a data stream that can be used. This means that the work conducted here has enormous scope and could be used in a vast range of applications and fields.

Many dynamic systems can be approximated with a second order system. Such a system is sufficiently complex to display the significant features of higher order systems but can be analyzed without excessive computation.

A second order system is determined by three parameters and two variables. The parameters are the damping, the natural frequency and the external input. The precise trajectory is determined by the initial values of the signal and its first derivative. This study concentrates on determining the parameter values, although some of the algorithms necessitate the evaluation of the initial conditions also.

6.1.3 Estimation Accuracy

To measure the accuracy of the algorithms the AAD and TAAD measures were introduced. Secondly, the algorithms must be able to cope with varying noise such as white noise and impulse noise. Thirdly, the algorithms must be able to track the parameter variation. This requires that the incoming signal blocks must represent only a small variation in parameter value. In the simulations, parameter variation is proportional to the frequency parameter, and this allows the introduction of an adaptive sampling frequency algorithm.

A major aim of this work was to devise an algorithm that would monitor the value of the parameters through time via examination of the incoming signal, given also that the value of the parameters will vary slowly with time. The problem is first broken down by determining the parameters based on blocks of data of known and fixed length. There are then two stages to a continuously running system. Firstly, an initial estimate of the parameters is required given no other information than the incoming signal and knowledge of the sampling frequency. Once this is achieved, a second stage starts which may use previous estimates of the parameters in addition to the incoming signal to aid estimation for the current block. Table 6.1 lists the algorithms according to whether or not they can be used in a recursive fashion. All the algorithms can be used in a non-recursive manner, and so can all be used in the initial stage of the continuously running algorithm to make an initial estimate of the parameters from the first data block.

Two methods use estimation of the time derivatives of the incoming signal to make the final parameter estimation. These are the difference equation and polynomial least squares fitting algorithm. The method for obtaining the parameters from the time derivatives is given in a paper by Al-Dabass [1].

Estimation Algorithm	
Non-recursive	Recursive
Difference Equations	Downhill Simplex
Polynomial LS fitting	Genetic Algorithm
High Level	Neural Networks

Table 6.1: *List of algorithms based on whether they can be used in a recursive mode.*

6.1.4 Adaptive Sampling Frequency Algorithm

Many of these algorithms benefit from use of the adaptive sampling frequency algorithm (ASFA). This was developed so that the number of points per cycle is approximately constant. Another way of viewing this is that the sample covers a period of time over which the parameters do not vary significantly. This is achieved by examining the estimated signal frequency and then adjusting the sampling frequency so that a fixed number of points is present per cycle.

6.1.5 Estimation Algorithms

Difference Equations

Use of difference equations is a basic method for obtaining the time derivatives (see Section 3.4.1). It requires only five data points to obtain the fourth time derivative needed. It is therefore computationally very inexpensive, and because the number of points is low, can monitor the parameter variation closely, especially when the ASFA is used. It suffers from two drawbacks, however. Firstly, using difference equations to evaluate time derivatives is only approximate. Secondly, this methods suffers greatly when noise is present. This is observed in the results when this method ranks most accurate method when no noise is present, but falls to almost last when noise is present.

High Level Approach

Application of an FFT in the High Level approach is surprisingly a poor estimation method for frequency (although see Further Work) because the block length used is so short. Use of a higher band of sampling frequency would allow longer samples to be used whilst each block covered the same period of time. This was not done in this work to maintain consistency between methods. This algorithm is good, however, in its resistance to noise corruption. Accuracy is maintained almost at the same level as when noise is absent up to a high level. Above a certain noise level the method collapses and highly erroneous estimates are made. This is due to noise becoming the dominant frequency in the spectrum.

Downhill Simplex

The Downhill Simplex method is a method that takes the result from the High Level algorithm to act as a seed. It then uses its parameter estimates to seed the next block and start the Simplex. This algorithm ranks well in both noisy and noiseless situations and improves on the High Level algorithms estimates. It is, however, slow, performing many iterations and comparisons each block.

Genetic Algorithms

A similar situation is true with the Genetic Algorithm. Demonstration of its abilities and properties are shown in experiment and some GA parameter estimation is performed to optimize results. It is shown that the GA benefits from working in a recursive manner. It ranks well, but again, is computationally expensive performing many comparisons per block.

FFT Plus Iterative Algorithm

It is shown that the FFT algorithm can be used to give an estimate of the parameters, and this can then be used to give initial conditions for either the Downhill Simplex or the GA approaches. This can be termed *seeding* the algorithms. This approach is shown to improve the estimates given by the FFT approach. This hybrid does reduce the speed with which estimates are generated, since both the simplex and GA are iterative, and hence slow algorithms.

Neural Networks

Neural networks perform poorly in this study. It is shown that they cannot perform well when the incoming signal is position and scale variable. Scaling of the input and output is also necessary. Reasonable frequency estimate are only achieved with a damping of zero. Although training is slow and must be performed off-line, it is a quick method since it requires only a single pass in operation. Although neural networks can be trained and operated in a recursive mode, little to no benefit is observed in this work from doing so. Indeed, a fully recurrent neural network trained with RTRL fails with even the basic mapping from signal to frequency.

6.1.6 Overview

Overall, the estimation of parameters is best achieved with the polynomial least squares fitting algorithm which determines the time derivatives of the signal block. These are then translated into parameter estimates. As previously mentioned, in a noiseless situation this algorithm can obtain an arbitrary accuracy given an increasing block length. This increases the accuracy of the higher derivative estimates. This algorithm is successful because it fulfils the least squares criteria exactly, and also within one sweep of the algorithm. The GA and Simplex methods attempt to descend the LS criteria, and only approach the optimal solution.

The Simplex and GA methods do have their place, however. In situations where the system is not known, they can match a function to the incoming data regardless of the function.

This study has produced two previously undocumented hybrid systems. These are the application of the simplex algorithm after an initial estimate using an FFT, and also by following the FFT using a GA to improve the estimates. The combination allows a the relatively quick FFT to seed the simplex or GA. It is usual to give random initial conditions for these two algorithms, but it is shown here that seeding causes a more rapid improvement in the estimates.

Block parameter estimation and the simulation thereof enabled the use of the Adaptive Sampling Frequency Algorithm (ASFA). The combination of the former and the new algorithm allowed improvement in the estimates to be made. This approach is also previously undocumentated.

6.2 Further Work

In this work a system is developed for tracking and monitoring the value of time varying parameters of a second order system. It can be concluded that the best method to do this is with a polynomial least squares approach.

In this study only simulated data has been used. It would be beneficial to see how accurately the algorithms can cope with real data. This would require a real-life situation which can be approximated to a second order system to be identified. Such systems may include vehicle suspension systems and the wave-form of human speech. This may well involve adjustment to cope with different types of noise eg. white noise with a non zero mean. It will also be important to see how well each method copes when the system being compared is not exactly a second order system, as in the simulations carried out so far.

The adaptive sampling frequency algorithm is currently quite crude, jumping to

precise values for the sampling frequency. Since the parameters are expected to vary slowly, a more intelligent system could be developed which allows only small changes in the sampling frequency to be made. This would result in a more stable system, and prevent large variations in the sampling frequency value.

This approach could be expanded to cover all of the parameters. In this work the parameters were simulated to vary sinusoidally with time. It would therefore be possible to perform some prediction on the expected value of each parameter. Although real data may not conform to sinusoidal variation, it may show continuous and smooth variation in the absence of catastrophes.

Fourier transforms are used in this study to indicate the natural frequency of the signal. A limitation of this approach is the accuracy of the estimate is governed by the sample length. This problem could be avoided by use of the chirp-z transform. This algorithm is based on the FFT, but is able to return a more accurate estimate in frequency, for a reduction in the range covered by the transform. This algorithm could be used, therefore, to lock onto the natural frequency giving a higher accuracy on its estimate. This would also improve the estimates of the other parameters since these are dependent on the natural frequency estimate when usnig the high level approach.

Observations of an object using a camera pair allows its path to be tracked in each dimension. From these the characteristics of the motion can be estimated. These can be used to predict the motion of the object.

It is also possible to track line objects from images. Once lines can be extracted from images, their motion can be analyzed and their properties estimated. Line objects may can be rigid, flexible, or compressible. This idea can be extended further to areas of a surface which can be tracked. Again, rigid, non-rigid and compressible areas can be analyzed and parameters estimated.

References

[1] D. Al-Dabass, *"Characteristics Estimation of Point Objects using Stereo Vision"*, IEE Colloqium on Digital Signal Processing, Savoy Place, London, 1985.

[2] B.Anderson and D.Montgomery, *"A Method for Noise Filtering With Feedforward Neural networks: Analysis and Comparison with Lowpass and Optimal Filtering"*, International Joint Conference on Neural Networks, Vols 1-3, 1990, Ch. 430, pp.A 209-A 214.

[3] P.J.Angeline, G.Saunders and J.Pollack, *"An Evoluationary Algorithm that Constructs Recurrent Neural Networks"*, IEEE Transactions on Neural Networks, January 1994, Vol.5, Issue 1, pp.54-65.

[4] S.N.Balakrishnan and J.Rainwater, *"Use of Time Varying Dynamics in Neural Network To Solve Multi-target Classification"*, Proceeds of the IEEE 1992 National Aerospace and Electronics Conference, 18th-22nd May 1992, Vol.1, pp.414-20.

[5] S.Baluja and D.Pomerleau, *"Using the Representation in a Neural Networks Hidden Layer For Task Specific Focus of Attention"*, International Joint Conference on Artificial Intelligence, 1995, Ch.278, pp.133-139.

[6] Sa H. Bang and Bing J. Sheu, *"Neural Network Communication Receiver Based on the Nonlinear Filtering"*, International Joint Conference on Neural Networks, 1992, Vol.2, pp.999-1004.

[7] D.Beasley, D.Bull and R.Martin, *"An Overview of Genetic Algorithms: Part 1 Fundamentals"*, University Computing, 15(2), pp.58-69, 1993.

[8] D.Beasley, D.Bull and R.Martin, *"An Overview of Genetic Algorithms: Part 2 Advanced Methods"*, University Computing, 15(40, pp.170-181, 1993.

[9] M.J.Boek, *"Experiments in the Application of Neural Networks to Rotating MAchine Fault Diagnosis"*, International Joint Conference on Neural Networks, 18th-21st November 1991, Vol.1, pp.769-74.

[10] C.H.Chen, *"Neural Networks for Financial Market Prediction"*, 1994 IEEE International Conference on Neural Networks, Vol 1-7, 1994, Ch.881, pp.1199-1202.

[11] T.Chen, W.C.Lin and C.T.Chen, *"Artificial Neural Networks for 3D Nonrigid Motion Analysis"*, IEEE Transactions on Neural Networks, November 1995, Vol.6, Issue 6, pp.1394-1401.

[12] J.M.Conell and C.S.Xydeas, *"A Comparison of Acoustic Noise Cancellation Techniques for Telephone Speech"*, 6th International Conference on Digital Processing of Signals in Communications, 2nd-6th September 1991, pp.320-325.

[13] J.Dayhoff, *"Neural Network Architectures: An Introduction"*, 1990, Van Nostrand Reinhold, ISBN: 0-442-20744-1.

[14] Hugo de Garis, *"GenNETS: Genetically Programmed Neural Nets"*, International Joint Conference on Neural Networks, 1991, Vol.2, pp.1391-6.

[15] J. Elman, *"Finding Structure in Time"*, Cognitive Science, No. 14, pp. 179-211, 1990.

[16] T.Fechner, *"Nonlinear Noise Filtering With Neural Networks: Comparison with Weiner Optimal Filtering"*, 3rd International Conference on ANN's, 25-27th May 1993, pp.143-7.

[17] *"Feedback and Control Systems"*, Schaum's Outline Series.

[18] G.Foresti, V.Murino, S.Regazzoni, G.Vernazza, *"A Distributed Approach to 3D Road Scene Recognition"*, IEEE Trans. on Vehicular Technology, vol.43 No.2, pp.3890-406.

[19] T.W.Frison, *"Controlling Chaos with a Neural Network"*, International Conference on Fuzzy Systems, 20th-24th March 1995, Vol.4, pp.1943-8.

[20] D.E.Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.*, 1989, Addison-Wesley, ISBN 0-201-15767-5

[21] J.H.Holland, *"Adaptation in Natural and Artificial Systems"*, Ann Arbor, MI:University of Michigan Press 1975.

[22] E.C. Ifeachor and B.W. Jervis, *"Digital Signal Processing: A Practical Approach"*, 1993, Addison Wesley ISBN 0-201-54413-X.

[23] K.Jundi, T.El-Ali, P.Eloe and F.Scarpino, *"Introduction to Neural Networks and Adaptive Filtering - 3 Illustrative Examples"*, IEEE 1993 National Aerospace and Electronics Conference, Naecon 1993, Vols 1 and 2, 1993, Ch. 180, pp.904-912.

[24] P.G.Korning, *"Training Neural Networks By Means of Genetic Algorithms Working On Very Long Chromosones"*, International Journal of Neural Systems, 1995, Vol.6, No.3, pp.299-316.

[25] J.R. Koza and J.P. Rice, *"Genetic Generation of Both the Weights and Architecture for a Neural Network"*, International Joint Conference on Neural Networks, 8th-14th July 1991, Vol.2, pp.397-404.

[26] K.P.Li and J.A.Naylor, *"A Whole Word Recurrent Neural Network for Keyword Spotting"*, ICASSP, 23-26th March 1992, Vol.2, pp.81-84.

[27] P.Matteucci, C.Regazzoni, G.Foresti, *"Real Time Approach To 3D Object Tracking In Complex Scenes"*, Electronics Letters, vol.30 No.6, pp.475-7, March 1994.

[28] J.R.McDonnell and D.Waagen. *"Evolving Recurrent Perceptrons for Time Series Modelling"*, IEEE Transactions on Neural Networks, January 1994, Vol.5, Issue 1, pp.24-38.

[29] A.V.Medvedev and H.T.Toivonen, *"A Systematic Synthesis of a Neural Network Based Smoother"*, Proceeds of 1992 IEEE International symposium on Intelligent Control, 11th-13th Auguest 1992, pp.147-51.

[30] B.Mulgrew and C.Cowan, *"Adaptive Filters and Equalizers"*, Kluwer Academic Publishers, 1988, ISBN: 0-89838-285-8.

[31] J.Ortega, W.Rheinboldt, *"Iterative Solution of Nonlinear Equations in Several Variables"*,New York Academic Press, 1970.

[32] W.H.Press, W.T.Vetterling, B.P.Flannery, S.A.Teukolsky, *"Numerical Recipes in C: The Art of Scientific Computing: 2nd Edition"*, 1992, Cambridge University Press, ISBN: 0-521-43108-5.

[33] G.V.Puskorius and L.A.Feldkamp, *"Neurocontrol of Nonlinear Dynamical Systems With Kalman Filter Trained Recurrent Networks"*, IEEE Transactions on Neural Networks, March 1994, Vol.5, Issue 2, pp.279-97.

[34] M.T.Rahman, G.L.Lebby and E.E.Sherrod, *"Noise Cancellation in Time and Frequency Domain Using Neural Networks"*, Proceeds of the 26th South-Eastern Symposium on System Theory, 20-22nd March 1994, pp.634-7.

[35] A.N.Refenes and M.Azema-Barac, *"Neural Network Applications in Financial Asset Management"*.

[36] D.Rummelhart and J.McClelland (editors), *"Parallel Distribuated Processing"*, Vol.I and II, MIT Press, Cambridge MA, 1986.

[37] M.D.Schuster, *"A Comprehensive Analysis of Neural Solution to the Multi-Target Tracking Data Association Problem"*, IEEE Trans. on Aerospace and Electronic Systems, Vol29, No.1,Jan.1993, pp.260-267.

[38] J.J.Shynk, *"Adaptive IIR Filtering"*, IEEE ASAP Magazine April 1989.

[39] W.Sohn and N.Kehtarnavaz, *"Analysis of Camera Movement Errors in Vision Based Vehicle Tracking"* , IEEE Trans. on Pattern Analysis and Machine Intelligence vol.17 No.1 Jan 1995.

[40] D.Whitley, T.Starkweather, C.Bogart, *"Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity"*, Parallel Computing, 1990, Vol.14, No.3, pp.347-361.

[41] B.Widrow, M.Lehr, F.Beaufays, E.Wam and M.Bilello, *"Learning Algorithm for Adaptive Signal Processing and Control"*, 1993 IEEE International Conference on Neural Networks, Vols 1-3, 1993, Ch. 360, pp.1-8.

[42] B.Widrow and D.Stearns, *"Adaptive Signal Processing"*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[43] P.Wieland, *"Evolving Neural Network Controllers for Unstable Systems"*, International Joint Conference on Neural Networks, 8th-14th July 1991, Vol.2, pp.667-677.

[44] R.Williams and D.Zipser, *"Experimental Analysis of the Real-Time Recurrent Learning Algorithm"*, Connection Science,

[45] L.Yao, W.A.Sethares, *"Nonlinear Parameter Estimation Via the Genetic Algorithm"*, IEEE Transactions on Signal Processing, April 1994, Vol.42, Issue 4, pp.927-35.

[46] H.Zhuang, *"A Self Calibration Approach to Extrinsic Parameter Estimation of Stereo Cameras"*, IEEE Int.Conf. Robotics and Automation, vol.4, pp.3428-33, 1994.

No.1, pp.87-111,1989.

Bibliography

1. S.Barker, *"High Speed Face Location at Optimal Resolution"*, World Congress on Neural Networks, 1995.

2. D.A.Castelow,P.F.Buxton, M.Rygol, P.Courtly, S.B.Pollard, *"A Plotform for the Development of a Machine stereo Vision System for the Control of A Robot Vehicle"*, IEEE Colloqium of Autonomous Guided Vehichles, 7th November 1991, pp.61-63.

3. T.Catfolis, *"A Method for Improving the Real-Time Recurrent Learning Algorithm"*, Neural networks, V6, pp807-21, 1993.

4. A.Cichocki and R.Unbehauen, *"Neural Networks for Optimization and Signal Processing"*, 1994, Wiley, ISBN: 3 519 06444 8.

5. Daw-Tung Lin, J. Dayhoff and P. Ligomenides, *"Trajectory Recognition With a Time Delay Neural Network"*, International Joint Conference on Neural Networks, 7th-11th June 1993, Vol.3, pp.197-202.

6. M. Dorigo and Machine Learning, *"ALECSYS and the AutonMmouse: Learning to Control a Real Robot by Distributed Classifier Systems"*, Vol.19,Part 3,page 209-40, 1995.

7. C.K.Goodwin, D.Al-Dabass, K.Sivayoganathan, *"Simulation of a Vision Steering System for Road Vehicles"*, Eurosim '95 Simulations Congress, pp. 1247-1252, Sept. 11-15th 1995, Technical University of Vienna, Austria, ISBN 0-444-822-410.

8. C.K.Goodwin, D.Al-Dabass, K.Sivayoganathan, *"Feasibility of a Vision Based Steering System"*, 28th International Symposium on Automotive Technology and Automation, 18-22nd Sept. 1995, Stuttgart, Germany. ISBN 0 9477 19741.

9. P.Husbands, I.Harvey,D.Cliff, *"Circle in the Round: State Space Attractors for Evolved Sighted Robots"*, Robotics and Autonomous Systems, 1995, Vol.15, No.1-2, pp.83-106.

10. K.Kristinsson and G.A.Deumont, *"System Identification and Control Using Genetic Algorithms"*, IEEE Transaction on Systems, Man and Cybernetics, September 1992, Vol.22, Issue 5, pp.1033-46.

11. A.Kuntman, N.Uyanik, B.M.Baysal, *"A Novel Method To Estgimate Various Equation of State Parameters"*, Polymer, 1994, Vol.35, No.15, pp.3356-3358.

12. B.B.Litkouhi, A.Y.Lee, D.B.Craig, *"Estimator and Controller Design for LaneTrack: A Vision Based Automatic Vehicle Steering System"*, Proceedings of the 32nd IEEE Conference on Decision and Control, Vols 1-4, 1993, Ch.819, pp.1868-1873.

13. L.R.Lopez and H.J.Caulfield, *"A Principle Complexity in Evolution"*, Lecture Notes in Computer Science, 1991, Vol.496, pp.405-9.

14. M.B.Matthews and J.S.Moschytz, *"Neural Network Nonlinear Adaptive Filtering Using the Extended Kalman Filter Algorithm"*, International Neural Network Conference, Vols 1 and 2, 1990,

    Ch. 216, pp.115-118.

15. J.R.McDonnell and D.Waagen. *"Evolving Recurrent Perceptrons for Time Series Modelling"*, IEEE Transactions on Neural Networks, January 1994, Vol.5, Issue 1, pp.24-38.

16. A.V.Medvedev and H.T.Toivonen, *"A Systematic Synthesis of a Neural Network Based Smoother"*, Proceeds of 1992 IEEE International symposium on Intelligent Control, 11th-13th Auguest 1992, pp.147-51.

17. *"Neural Networks Summer School: Theory, Design and Applications"*, Cambridge University, 19-22nd Sept. 1994.

18. F.Mondada and D.Floreano, *"Evolution of Neural Control Structures: Some Experiments on Mobile Robots"*, Robots and Autonmous Systems, 16,1995,pp.183-195.

19. D.Ngeyen and B.Widrow, *" The Truck B acker Upper : An example of self-learning in neural networks "*, Neural Networks for Control, MIT Press, Cambridge MA, 1990.

20. J.Ortega, W.Rheinboldt, *"Iterative Solution of Nonlinear Equations in Several Variables"*,New York Academic Press, 1970.

21. S.Papert and M.Minsky, *"Perceptrons: An Introduction to Computational Geometry"*, MIT Press, Boston, 1969.

22. D.Pavisic, L.Blondel, J.P.Draye, G.Libert, P.Chapelle, *"Efficient use of Dynamic Recurrent Neural Networks for Active Noise-Control"*, Proceedings of the 15th International Congress on Acoustics, Vol II, 1995, Ch.173, pp.243-246.

23. F.Rosenblatt, *"The Perceptron, A Probabilistic Model for Information Storage and Organisation in the Brain"*, Psychological Review, No.65, pp.386-404, 1958.

24. D.Ruck, S.Rogers, M.Kabrisky, P.Maybeck, M.Oxley, *" Comparitive Analysis of Backpropogation and the Extended Kalman Filter for Training Multilayered Perceptrons"*, IEEE Trans. on Pattern Analysis and Machine Intelligence, V14, No.6, pp.686-90, 1992.

25. M.Schoenauer and E.Ronald, *"Neuro-Genetic Truck Backer Upper Controller"*, Proceeds of IEEE Conference on Computation, June 1994, Vol.2, pp.720-723.

26. K.Sekihara, H.Haneishi, N.Ohyama, *"Details of Simulated Annealing Algorithm To Estimate Parameters of Multiple Current Dipoles Using Biomagnetic Data"*, IEEE Transactions on Medical Imaging, June 1992, Vol.11, Issue 2, pp.293-299.

27. M.K.Sen and P.L.Stoffa, *"Rapid Sampling of Model Space Using Genetic Algorithms: Examples from Seismic Waveform Inversion"*, Geophysical Journal Int., 1992, Vol.108, Part 1, pp.281-92.

28. H.Sito, M.Mori, *"Application of Genetic Algorithms To Stereo matching of Images"*, Pattern Recognition Letters, 1995, Vol.16, Part 8, pp.815-21.

29. G.Sun, H.Chen, Y.Lee, *"A Fast On-line Learning Algorithm for Recurrent Neural Networks"*, International Joint Conference on Neural Networks, Vols 1 and 2, 1991, Ch. 290, pp.B 13-B 18.

30. S.Tsugawa, *"Vision Based Vehicles in Japan: Machine Vision Systems And Driving Control Systems"*, IEEE Transactions of Industrial Electronics, Auguest 1994, Vol.41, Issue 4, pp.398-405.

31. R.Williams, *"Training Recurrent Networks Using the Extended Kalman Filter"*, Int. Joint Conference on Neural Networks , Baltimore 1992, V4, pp.241-46.

32. Xin Yao, *"A Review of Evolutionary Artificial Neural Networks"*, International Journal of Intelligent Systems, 1993, Vol.8, No.4, pp.539-567.

33. D.Zipser, *"A Subgrouping Strategy That Reduces Complexity and Speeds Up Learning In Recurrent Networks"*, Neural Computation, 1,552-558, 1989.

Appendix A

Derivation of Complex Solution to the Second Order System

This appendix derives the solution $x(t)$, given the initial conditions x_{1o} and x_{2o}, and input (constant) U, of the characterizing equation,

$$ax'' + bx' + cx = U \tag{A.1}$$

where $a = 1.0$, $b = 2\xi\omega_n$ and $c = w_n^2$. Each apostrophe given to each x indicates a time derivative e.g. $x'' = \frac{d^2x}{dt^2}$.

The roots of the characterizing equation are,

$$r_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{A.2}$$

Note that,

$$\begin{aligned} r_1 \times r_2 &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} \cdot \frac{-b - \sqrt{b^2 - 4ac}}{2a} \\ &= \frac{b^2 - (b^2 - 4ac)}{4a^2} = \frac{c}{a} \end{aligned}$$

The Free solution can be written as,

$$x(t) = c_1 e^{r_1 t} + c_2 e^{r_2 t} \tag{A.3}$$

To find the values for c_1 and c_2, we examine the initial conditions for $x(t)$ and $x'(t)$. The initial conditions are,

$$\begin{aligned} x_{1o} &= x(0) \\ x_{2o} &= x'(0) \end{aligned}$$

We can therefore write that,

$$x_{1o} = c_1 + c_2 \tag{A.4}$$

Now since,

$$x'(t) = c_1 . r_1 e^{r_1 t} + c_2 r_2 e^{r_2 t} \tag{A.5}$$

we can write,

$$x_{2o} = r_1 c_1 + c_2 r_2 \tag{A.6}$$

Inserting Equation A.4 into Equation A.6, we get,

$$\begin{aligned} x_{2o} &= r_1(x_{1o} - c_2) + c_2 . r_2 \\ &= r_1 . x_{1o} - r_1 . c_2 + c_2 . r_2 \end{aligned}$$

Re-arranging,

$$x_{2o} - r_1 . x_{1o} = c_2(r_2 - r_1) \tag{A.7}$$

Further re-arranging for c_2, we get,

$$c_2 = \frac{x_{2o} - r_1.x_{1o}}{r_2 - r_1} \tag{A.8}$$

Now inserting Equation A.8 into Equation A.4,

$$\begin{aligned} c_1 = x_{1o} - c_2 &= x_{1o} - \frac{x_{2o} - r_1.x_{1o}}{r_2 - r_1} \\ &= \frac{x_{1o}.r_2 - x_{2o}}{r_2 - r_1} \end{aligned} \tag{A.9}$$

We now calculate the forced solution which is then added to the free solution.

$$x(t) = \int_0^t w(t-\tau) * U(t).d\tau \tag{A.10}$$

where $w(\cdot)$ is a free response of the system $= c_1 e^{r_1 t} + c_2 e^{r_2 t}$.

Therefore, for $U(t) = U$,

$$x(t) = U \int_0^t c_1 e^{r_1 t} e^{r_1 \tau} + c_2 e^{r_2 t} e^{-r_2 \tau} d\tau \tag{A.11}$$

with $w(t) = c_1 e^{r_1 t} + c_2 e^{r_2 t}$ we can deduce $c_1 + c_2$ from $w(0) = 0$, and recalling that $w'(0) = 1$ (universal initial condition for weight functions).

$$\begin{aligned} 0 &= c_1 + c_2 \\ c_1 &= -c_2 \end{aligned}$$

We can now write,

$$\begin{aligned} w'(t) &= c_1 r_1 e^{r_1 t} + c_2 r_2 e^{r_2 t} \\ w'(0) &= 1 = c_1 r_1 + c_2 r_2 \\ 1 &= c_1 r_1 - r_2 c_1 = c_1 (r_1 - r_2) \\ c_1 &= (r_1 - r_2)^{-1} \\ c_2 &= -(r_1 - r_2)^{-1} \end{aligned}$$

Therfore $x(t) = U\int_0^t$ as before Equation A.6,

$$\begin{aligned} &= \frac{Ue^{r_1 t}}{r_1(r_1 - r_2)} - \frac{Ue^{r_2 t}}{r_2(r_1 - r_2)} - \frac{U}{r_1(r_1 - r_2)} - \frac{1}{r_2(r_1 - r_2)} \\ &= \frac{Ue^{r_1 t}}{r_1(r_1 - r_2)} - \frac{Ue^{r_2 t}}{r_2(r_1 - r_2)} + \frac{U}{r_1 r_2} \end{aligned}$$

The total solution is then,

$$\begin{aligned} x(t) &= c_1 e^{r_1 t} + c_2 e^{r_2 t} + \frac{Ue^{r_1 t}}{r_1(r_1 - r_2)} - \frac{Ue^{r_2 t}}{r_2(r_1 - r_2} + \frac{U}{r_1.r_2} \\ x(t) &= \left(c_1 + \frac{U}{r_1(r_1 - r_2)}\right) e^{r_1 t} + \left(c_2 - \frac{U}{r_2(r_1 - r_2)}\right) e^{r_2 t} + \frac{U}{r_1 r_2} \end{aligned}$$

Appendix B

Backpropagation

Backpropagation [36] is the most widely used of the algorithms to train feedforward neural networks. There are many variations, each claiming an improvement to the basic algorithm in either speed of training or quality of training, but almost invariably at some other cost. The use of momentum alone to augment the basic algorithm is covered here, as it remains one of the best ways of improving overall performance.

Figure 3.3 shows a general feedforward neural network. It is made up of a number of layers of units (also known as artificial neurons). Between the layers of units are weighted connections, allowing information to pass from the top layer, down the connections and onto the next layer. Each unit has a value called its *activation*, and the weight matrix is also known as the impulse response of the network.

Backpropagation (BP) uses error gradient information to obtain minima in the search space of the impulse response. It is closely related to the Widrow-Hopf rule for adaptive linear filters, and indeed, the LMS algorithm is a special case of the BP algorithm when the network has only an input and an output layer.

When training a feedforward network with backpropagation, there are two clear phases. The forward phase is when the input signal is propagated forward through the network to produce values for the output unit(s). In the back progagation phase, this output is compared with the desired output for the current input pattern, and an error

is propagated back through the network, which updates the weights in such a way as to make the network output nearer to the desired output.

At the beginning of the forward phase, units in the input layer are given activations from the outside world. Each unit in the layer below then conducts a summing process. When each layer is complete, the next layer below performs the summing process. For a general unit i in layer l, the summing process is,

$$s_i = \sum_{j=0}^{N_l - 1} w_{ij}^l x_j^{l-1} \tag{B.1}$$

where N is the number of units in the lth layer, and $l = 0$ is the input layer, and there are $L - 1$ layers. The weight w_{ij}^l refers to the weight leading from the jth unit in the $(l-1)$th layer to the unit i in the lth layer.

Once the summing is complete, the unit performs a transfer function on the summed weighted input to arrive at the unit's activation,

$$x_i = f(s_i) \tag{B.2}$$

where x is the activation of a unit and $f(\cdot)$ is the transfer function.

The actual form of $f(\cdot)$ can be any function that is differentiable as all algorithms using error gradient descent require $f'(\cdot)$ in calculating weight updates. Common functions include the sigmoid,

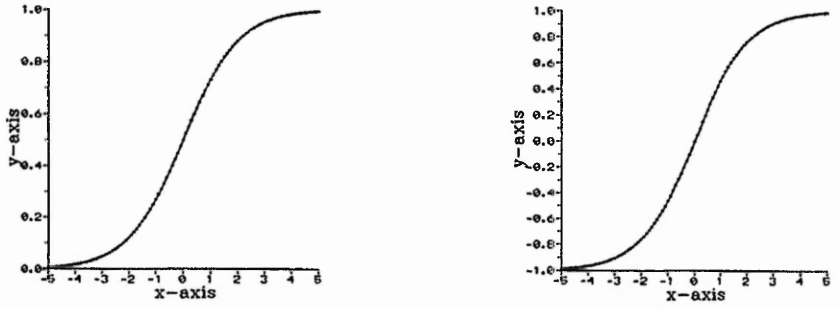
$$y = \frac{1}{1 + \exp^{-\gamma x}} \tag{B.3}$$

and the hyperbolic tanh function,

$$y = \tanh(\gamma x) = \frac{1 - \exp^{-2\gamma x}}{1 + \exp^{-2\gamma x}} \tag{B.4}$$

The variable γ allows the steepness of the curve to be varied, but is usually simply set to unity. Figure B.1 show the forms of the two transfer functions. The significant difference between them is that the sigmoid ranges between zero and one, and the

tanh function ranges between ±1. Many other functions have been used, with success commonly dependent on the application. Functions are usually nonlinear, to maintain a network's nonlinear mapping abilities, and often limit both the minimum and maximum values.



(a) (b)

Figure B.1: *Two common transfer functions. a) The sigmoid, b) the* tanh *function.*

The activation of the output unit(s) is named y_i for each of the i output units. Once values for all y_i have been evaluated by the forward phase, the backpropagation phase may begin. This starts with a calculation of an error vector,

$$e_i = d_i - y_i \tag{B.5}$$

This is performed for the output layer only, and d_i represents the desired output of the ith output unit for the current input pattern. To update the weights connecting the output and the last hidden layer the Widrow-Hopf rule is applied,

$$dw_{ij}^{L-1} = f'(s_i)2\mu e_i x_j^{L-2} \tag{B.6}$$

where j is the unit in the $L-2$th layer, $L-1$ is the total number of layers, $f'(\cdot)$ is the derivative of the transfer function of the output unit i, μ is a learning rate, e_i is the error of the ith output unit and x_j is the activation of the jth unit in the $L-2$th layer.

For all other weight layers above this ie. for updating weights connecting *to* a hidden layer,

$$dw_{ij}^{l} = f'(s_i)2\mu\delta_h x_j^{l-1} \tag{B.7}$$

where δ_h is calculated for each unit in the current lth hidden layer,

$$\delta_h = \sum_{i=0}^{N_{l+1}-1} e_i w_{hi} \tag{B.8}$$

where N_{l-1} is the number of units in the $(l-1)$th layer. Once the weight update has been calculated for each unit in each layer, layer by layer, then for all i and all j,

$$\boldsymbol{W_{new}} = \boldsymbol{W_{old}} + \boldsymbol{dw} \tag{B.9}$$

Thus, the error is propagated back through the network for each layer in the network. Once complete, a new input pattern is presented and a forward phase can start.

The learning rate variable, μ, is user set. If set to unity, each input pattern would remove any useful adjustment of weights already performed. Too small a value of μ would result in the weights being adjusted too slowly, and an excessive number of presentation of input patterns would be needed before the network accurately output the desired values. In practice, the value of μ can be 0.3 and can go as low as 0.1. Values outside this range are by no means prohibited, but would be used only in special cases.

One of the major problems with neural networks is their susceptibility to local minima. This means that the network is not performing at its optimal rate, and the

output it provides is substandard. One of the most commonly used techniques for preventing capture in local minima is the use of a momentum term.

Adding a momentum [13] term to the plain BP algorithm takes into consideration the weight update from the previous propagation. If a weight update occurs at a time step n, then it will also include information from the weight update at time $n-1$. Thus, the equation for a weight update becomes,

$$\boldsymbol{W}_{new}(n) = \boldsymbol{W}_{old}(n) + \boldsymbol{dw}(n) + \alpha \boldsymbol{dw}(n-1) \tag{B.10}$$

Here, α is a parameter between zero and one which determines the importance given to the previous weight change. High values of α of around 0.9 are common.

Momentum can be understood by visualizing the error surface as an undulating surface over which the network searches for a minima. The gradient descent rules of BP always move it downwards. This means that if the current position on the error surface is in a shallow hollow, then the network will not be able to leave this hollow and move on to a deeper, globally minimal position. Adding a momentum term allows the motion of the network's position to pass on through a local minima and up over a nearby brow, and down into another minima. The greater the value of α, the more able it is to do this. In this fashion the network can escape a local minima.

Implementation of momentum in a BP algorithm can frequently improve convergence rates by a significant factor and reduce the chance of the network being caught in a local minima whilst only a reasonable increase in storage and calculations is required.