Enabling Rapid Iterative Model Design within the Laboratory Environment

Thomas F. Clayton

A thesis submitted for the degree of Doctor of Philosophy

Institute for Integrated Micro and Nano Systems

School of Engineering

The University of Edinburgh

2009

Abstract

This thesis presents a proof of concept study for the better integration of the electrophysiological and modelling aspects of neuroscience. Members of these two sub-disciplines collaborate regularly, but due to differing resource requirements, and largely incompatible spheres of knowledge, cooperation is often impeded by miscommunication and delays. To reduce the model design time, and provide a platform for more efficient experimental analysis, a rapid iterative model design method is proposed.

The main achievement of this work is the development of a rapid model evaluation method based on parameter estimation, utilising a combination of evolutionary algorithms (EAs) and graphics processing unit (GPU) hardware acceleration. This method is the primary force behind the better integration of modelling and laboratory-based electrophysiology, as it provides a generic model evaluation method that does not require prior knowledge of model structure, or expertise in modelling, mathematics, or computer science. If combined with a suitable intuitive and user targeted graphical user interface, the ideas presented in this thesis could be developed into a suite of tools that would enable new forms of experimentation to be performed.

The latter part of this thesis investigates the use of excitability-based models as the basis of an iterative design method. They were found to be computationally and structurally simple, easily extensible, and able to reproduce a wide range of neural behaviours whilst still faithfully representing underlying cellular mechanisms. A case study was performed to assess the iterative design process, through the implementation of an excitability-based model. The model was extended iteratively, using the rapid model evaluation method, to represent a vasopressin releasing neuron. Not only was the model implemented successfully, but it was able to suggest the existence of other more subtle cell mechanisms, in addition to highlighting potential failings in previous implementations of the class of neuron.

Acknowledgments

Many people have contributed to this work, in discussions, guidance, aid, and most importantly motivation when the 2nd and 3rd year blues were upon me. I would like to thank my supervisors, Prof. Alan Murray, Prof Gareth Leng, and Dr. Iain Lindsay; those that have enlightened me in the ways of electrophysiology and biological modelling, Dr. Nancy Sabatier and Dr. Ruth Durie; and those that have been constantly supportive or were my partners in crime, Dr. Katherine Cameron, Dr. Leena Patel, Dr. Simeon Bamford.

This thesis is a monstrously large body of work, and because I am terrible at writing coherently, several people proof read it for me. These people I hold in the highest regard, most notably because they are family members who felt compelled to aid me when they didn't really have to. Thank you Mom! Thank you Dad! Without your constant support (and criticism) I would not have been able to finish.

Finally, thank you Louise Albert. You have proofread nearly every piece of writing that I have written in the past 5 years. You have kept me going when all I wanted to do was throw my thesis in the bin and go get a real job. I only hope that I can return the favour some day.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Thomas F. Clayton)*

Contents

Chapter 1 Introduction

At present, the field of neuroinformatics suffers from a divide between its practitioners. This stems from a natural disparity in the knowledge bases possessed by the two primary types of specialist who practice the field: electrophysiologists and neural modellers. The former tend to have a background in biology, while the latter more often come from mathematics or computer science. The two parties use different terminology and concepts, and thus difficulties in communication can arise. This is aggravated by the fact that the majority of electrophysiology is performed within a laboratory environment, to which modellers often do not have access, whilst the modellers use their own set of resources, such as server clusters and super computers, which are unfamiliar to electrophysiologists.

These issues greatly impede collaborative efforts within the field, limiting the extent to which the two parties can work alongside one another, instead promoting a model design process of two largely independent stages. Typically, the first stage involves the collection of cellular recordings by electrophysiologists. These are then passed to the modeller, who performs the second stage by seeking to explain the recorded behaviour with a neural model. Through modelling, hypotheses belonging to both parties regarding the function of the cell, can be proven or disproven. This is a slow process, as the biological experimentation tends to be unpredictable, especially when working *in-vivo*, and some data sets can take years to collect. Only once a significant quantity of data has been collected and analysed can the modelling process begin, which means that often the modeller is working on a different project from the electrophysiologist, who will have started to collect data from the next cells of interest.

1.1 Objective

This thesis presents the initial steps to better integrate the electrophysiological and modelling aspects of neuroscience, to reduce the model design time and thus provide a platform for more efficient cell analysis.

Integration in this manner would traditionally require that the practitioner be a double specialist, having access to both knowledge bases, and the ability to construct neural models whilst concurrently interacting with the biology. The alternative considered here is to create a toolset that masks the technicalities of one of the knowledge bases,

providing a friendly interface for a single specialist to access. The modelling process, being largely deterministic, is a better candidate for packaging in this way. In contrast, electrophysiology, consisting mainly of biological experimentation, is dominated by highly unpredictable non-deterministic processes, which are better guided by human judgment. This thesis therefore explores the development of such a toolset to aid the modelling process.

The goal of this research was to create a platform for rapid iterative design, a concept that has been used extensively in many other fields, such as computer science and electronic engineering. It is currently used within neuroinformatics to develop mathematical models, but not in electrophysiology. Rapid iterative design represents a "trial and error" approach to the design process, which is universally intuitive. It is important to use intuitive processes when attempting to bridge the gap between knowledge bases; otherwise the target audience will have difficulty in accessing the packaged domain. Intuitive processes adopted by one group can often be difficult for another, and so it is vital that methods be tailored for specific audiences, in this case experimental electrophysiologists with little computer science or mathematical knowledge.

Any rapid iterative model design tool to be used by electrophysiologists within a laboratory must meet the following criteria. As part of the proof of concept, the thesis focuses on the latter two of these criteria to determine their feasibility:

1. The tool must include an electrophysiologist-friendly graphical user interface (GUI) that facilitates rapid and simple model construction.

2. Models developed by the tool must modular, insofar as they can be made from pre-defined functional blocks.

3. The tool must include functionality allowing models to be rapidly evaluated.

This project explores the overarching hypothesis that a rapid model evaluation tool, based upon parameter estimation, utilising evolutionary algorithms, and implemented using graphics card hardware acceleration, can provide a modelling framework that is flexible, cost effective, and suitable for a laboratory environment.

The thesis also examines the particular hypotheses that:

- Excitability-based models can be useful in the context of the second criteria, as they are modular in nature and, whilst simple in structure, can be made to mimic the behaviours of a wide range of cell types though parameter adaptation.

- Through the addition of simple functional blocks, excitability-based models can be easily extended to fit more complex long term behaviours.

1.2 Thesis Structure

This thesis is split into seven chapters. Chapter 2 provides an introduction to electrophysiology, covering the relevant regions of the brain, and techniques for data collection and analysis. Chapter 3 is a summary of neural modelling, as it relates to this thesis, and evolutionary algorithms (EAs). Several archetypal neural models are introduced, along with an excitability-based modelling paradigm that is subsequently used throughout this thesis. Issues relating to the combination of modelling and electrophysiology, as well as the suitability of each of the archetypal models to an environment tailored to electrophysiologists, are discussed. This section is followed by a review of the modelling environments, highlighting the need for a platform tailored to laboratory-based electrophysiologists. Section 3.3 introduces the three archetypal EAs, whose viability as the core of our rapid model evaluation method will be evaluated in Chapter 4. Chapter 3 is concluded with an overview of single- and multi-objective fitness evaluation.

Chapter 4 discusses the development of a rapid model analysis methodology, focusing on its two main components: evolutionary algorithms (EAs), and hardware acceleration. The choice of hardware acceleration is discussed in relation to laboratory experimentation, for this project as well as for future possible commercialisation. The final section of Chapter 4 presents a comparative study of the three archetypal evolutionary algorithms introduced in Chapter 3 and one algorithm of our own construction. This comparison takes into account the environment of massive parallelisation promoted by hardware acceleration.

Chapter 5 investigates the suitability of excitability-based models as candidates to fulfil the requirement of a modular model framework. During this chapter, the basic excitability-based model is used to recreate behaviours from 72 cells from the ventro-medial hypothalamic (VMH) region of the brain. The model is then tested for reproducibility, robustness, and ability to create hypotheses through the explanation of cell behaviour.

Chapter 6 performs a case study to test the extensibility of the basic excitability-based model, and its ease of use as the basis of an iterative design method. This case study takes the simple excitability-based model further, showing that through the addition of a new, but simple, functional block, the model can be made to fit the complex phasic firing patterns of vasopressin releasing neurons. In this chapter, specific sub-behaviours of vasopressin cells, such as phasic response and noise characterisation, are analysed to test the reproducibility of the model.

Chapter 7 summarises the thesis, draws conclusions, and provides avenues for future work.

Chapter 2 Basic Electrophysiology

This thesis covers a number of disciplines, including electrophysiology, neural modelling, evolutionary computation, and electronic engineering. Of these four domains, the author has the least experience with electrophysiology, but as the focus of this work is centred on providing tools for this discipline, some basic understanding of electrophysiology is required. For the latter half of this project, the author worked closely with electrophysiologists to gain some understanding of the field. This chapter is presented to provide the reader with this knowledge, highlighting the difficulties electrophysiologists face when performing laboratory experimentation, as well as introducing the cell analysis techniques that are used in this project.

Electrophysiology is the study of cells and tissues through their electrical properties. Within neuroscience, this involves the measurement of the change in voltage and current flow caused by neural activity. Experimental procedures have been developed to explore the electrical properties of neurones on a number of different scales, from the change in membrane potential caused by a single ion channel, to the action potential activity from small populations of cells.

This study focuses on action potential activity of single cells (single unit recording), and hence this chapter, although providing a brief background of other methods, mainly focuses on the experiments conducted to collect data of this form. Future studies may include working with small network or ion channel models, for which intracellular recordings and multi-unit recording of simultaneous network activity may be required.

2.1 The Hypothalamus

All the cell recordings used within this study have been collected from within the hypothalamus. This section provides a general overview of this area of the brain, explaining its function, as well as its importance to the scientific community.

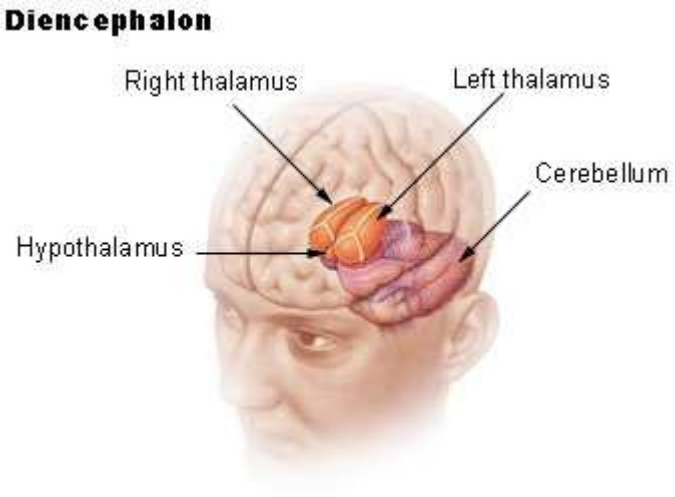


Figure 2-1 – Location of the Hypothalamus. Image taken from http://upload.wikimedia.org/wikipedia/commons/b/b2/Illu_diencephalon_.jpg

The hypothalamus is found beneath the thalamus, just above the brain stem, as shown in Figure 2-1. It is responsible for regulating many of the body's metabolic and autonomic nervous system processes, and is commonly linked with basic animal instincts, such as the fight or flight response, dietary, and sexual activities. Control is facilitated through the release of hormones, which trigger responses in other areas of the brain as well as organs around the body. Because the hypothalamus is responsible for so many bodily functions, it is complex, with many sub sections as shown in Figure 2-2. The areas focused on in this work are the Supraoptic nucleus (SOH), Paraventricular nucleus (PVH), and Ventromedial (VMH) nucleus.

2.1.1 Arginine Vasopressin Release Neurones

The Supraoptic (SO) and Paraventricular (PV) nuclei are responsible for, amongst other things, Arginine Vasopressin (AVP) secretion, which is triggered by changes in blood plasma osmolarity (Robertson, 2001). AVP is responsible for the control of water retention by the kidneys (an anti-diuretic), and thus greater quantities are released when the body is dehydrated. Plasma osmolarity is a measure of the number of soluble molecules per unit volume of solution that are able to pass through the membrane of a cell, and hence create an osmotic pressure across it (sodium is the main component). Hence, both a decrease in plasma water content, as well as an increasing concentration of these particles, acts as triggers for AVP release. A secondary threat that provokes the

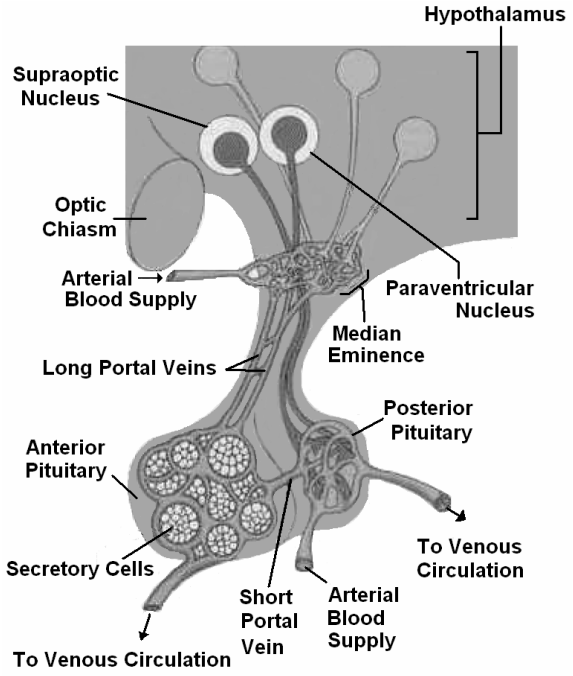


Figure 2-2 – Sub sections of the Hypothalamus and Pituitary. Image adapted from (Durie, 2007), original source could not be found.

release of AVP is the decrease in blood pressure. Although small in a healthy subject, the effect of AVP is an important part of the body's response to blood loss, and in this case trades an increase in blood pressure, to account for haemorrhage, with a decrease in plasma osmolarity achieved by triggering water retention. As blood plasma osmolarity increases, there is a proportionally linear response in AVP release. There is also a less sensitive response to blood volume depletion, as shown in Figure 2-3.

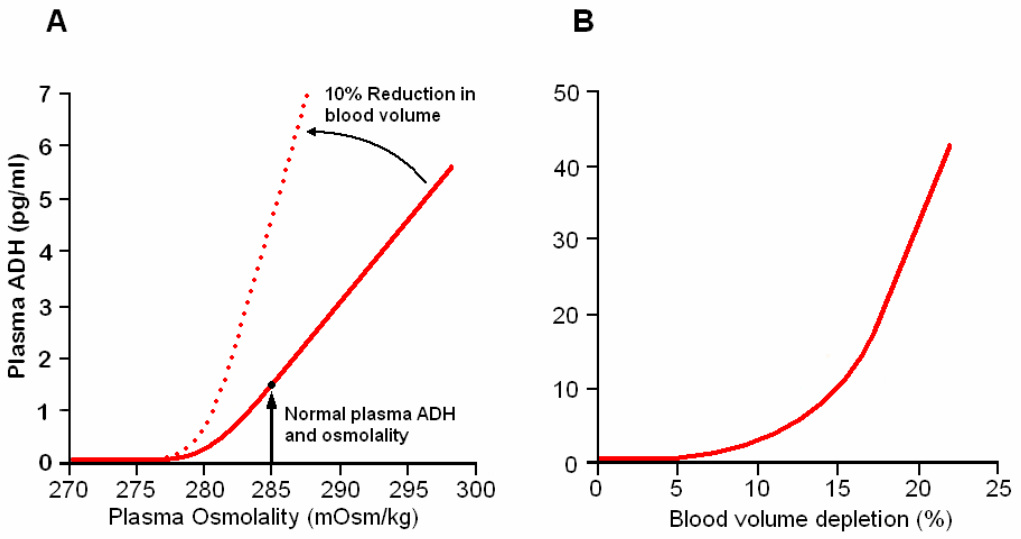


Figure 2-3 Effects of (a) increase in blood osmolarity and (b) blood loss on AVP secretion. Image adapted from (Davies et al., 2001)

The feedback system that protects blood osmolarity is extremely sensitive, responding to as little as a 1% change in osmotic pressure. Hence the possible ranges of AVP secretion are large (Arnauld et al., 1975) and are reflected by the equally large range of spiking behaviour produced by the AVP secretion neurons.

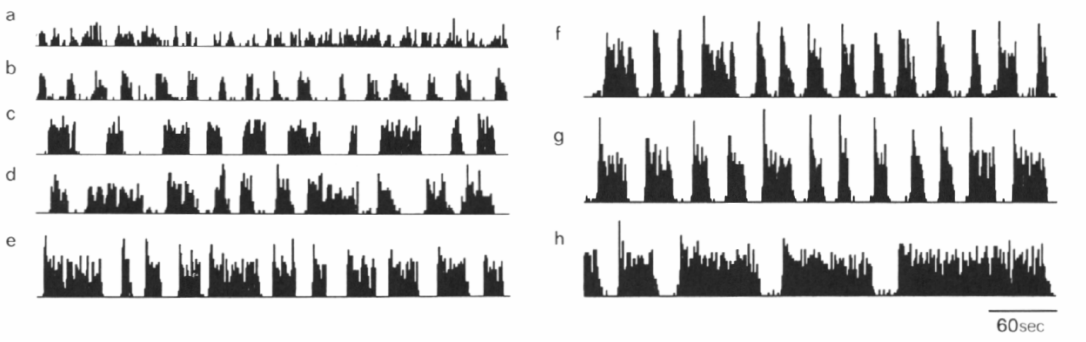


Figure 2-4 Range of vasopressin releasing neuron behaviours (spike count in one second bins with time). Image adapted from (Poulain et al., 1988)

These behaviours range from the sparse sporadic firing shown in Figure 2-4a, through a range of phasic firing, characterised by regions of activity followed by regions of silence (Figure 2-4b-h), to continuous firing (Brimble and Dyball, 1977, Wakerley et al., 1978).

As a population, AVP secretion neurons form a load balancing network that, through the combination of feedback mechanisms, respond to the various conditions that threaten normal blood osmolarity and pressure. These feedback mechanisms exhibit themselves as intracellular electrical responses (a slow afterhyperpolarisation (AHP) that suppresses over-activity that is dangerous to a cell), extracellular electrical responses (PSPs), and the release of various peptides, such as Dynorphin, Adenosine and Vasopressin. There are additional chemical interactions, but the three mentioned above represent the most significant contributing factors for the regulation of individual and population activity, which facilitates constant AVP secretion and threat management. Chapter 7 expands upon this section, providing information regarding modelling approaches and current theory relating to the cause of this phasic behaviour.

2.1.2 The Ventromedial Hypothalamic Nucleus

The Ventromedial nucleus of the hypothalamus (VMH) has many functions, being known to assist in the regulation of blood pressure, pain pathways (McClellan et al., 2006), glucose homeostasis (Song et al., 2001), regulation of oestrogens (Pfaff and Keiner, 1973), but most commonly being linked to satiety (King, 2006) and sexual regulation (Davis and Barfield, 1979). The VMH consists of many sub-populations that differ structurally, chemically and behaviourally, whilst also projecting into many different areas outside of the VMH. With the increase in obesity in the USA and Western Europe, there is increasing interest in the VMH, and particularly in identifying the mechanisms that regulate appetite. A recent study has focused on the classification of VMH cells on the basis of their firing patterns (Sabatier and Leng, 2008), of which nine groups have been defined by a combination of statistics derived from their inter-spike interval (ISI) combined with the shape of their hazard distributions (these measures are explained in Section 2.3). This is explored in greater detail in Chapter 6, which utilises an excitability-based model to gain insight into the cause of four of the nine behavioural groups.

2.2 Methods of Data Collection

The majority of experimental methods (Bretschneider and Weille, 2006) currently in use involve the placement of electrodes near, on, or within the target neurones. Different electrodes, ranging from insulated needles with un-insulated tips to liquid electrolyte filled glass pipettes that are micrometers in diameter, are used to detect different levels of electrical detail. The size of probe determines the level of detail recorded, with larger probes measuring the electrical activity of entire populations, and the smallest able to isolate and record from individual sections of a cell membrane. Experimental methods can be split into two groups: intracellular and extracellular. All the data used throughout this thesis was collected extracellularly. A brief description of some intracellular methods is included to show how complex and difficult these processes are to perform, and therefore why new methods that can make better use of successful experimentation are important. Future work may well use intracellular data, such as membrane potential recordings, but because of the difficulty in making these

recordings, combined with the fact that large sets of pre-recorded extracellular data were already available, this work has avoided this approach.

2.2.1 Intra-cellular Methods

Intra-cellular methods (Gurney, 2000, Kornreich, 2007) focus on recording directly from single neurons or their elements (such as ion channels). Although it is possible to perform many intra-cellular techniques *in-vivo*, the complexity of integrating the apparatus with the *in-vivo* environment can make this task difficult. Because of this, the majority of intra-cellular techniques are performed in more controllable *in-vitro* environments. This is acceptable for exploring ion channel characteristics and simple cell behaviours, but when trying to understand more complex behaviours, especially inter-neural interactions, it is important to take the cell's environment into consideration. This is especially true of hypothalamic vasopressin releasing neurons, recordings of which are examined in this work.

Patch Clamping

Patch clamping was originally developed by (Sackmann and Neher, 1976), and is performed with probes that are approximately a micrometer in diameter. These probes, which consist of a miniature pipette with a liquid electrolyte core, are brought into contact with the target cell and, through suction, tightly adhere to it, creating a giga-Ohm seal between the electrolyte and the cell exterior. Once this is done, several different types of analysis can be performed, depending on the aspect of the cell the experimentalist wishes to explore. A good paper covering a range of approaches was written by (Hamill et al., 1981).

Single ion channels within the patch of membrane can be examined through direct measurement of the change in current, and the function of the ion channels may even be tested by the addition of drugs to the electrolyte solution. The probe can also remove a section of membrane whilst still maintaining the seal. In this manner, the experimentalist can control the environment on both sides of the ion channels.

Whole cell recording, where the current flow within the cell cytoplasm is measured directly, is possible by puncturing the cell through excessive suction, thereby letting the

liquid electrolyte come into direct connect with the cell cytoplasm. This method provides a much clearer picture of a cell's electrical activity due to the greatly decreased resistance between the cell and the recording apparatus. However, it is important to note that this is a destructive form of analysis. Because of the greater volume of the pipette compared to the cytoplasm, the soluble parts of the cell will slowly spread to the pipette solution, an affect know as "dialyzing" or "wash-out", which can cause modifications to the normal cell behaviour (Horn and Marty, 1988).

An alternative to whole cell mode is to perform perforated patch recording (Rae et al., 1991), where antibiotics that have the effect of punching holes in the membrane are added to the electrolyte solution. This reduces the amount of dialysis, but also increases the resistance of the seal, reducing the resolution of the recorded current by clouding it with noise. This experiment also has a longer setup time, as it takes up to 30 minutes for the antibiotics to perform the perforation.

Sharp Microelectrode Recording

Sharp microelectrode recording (Brown et al., 1981) involves piercing a cell with an electrode, in order to record its internal electrical properties directly. The electrode is of similar design to the patch clamping pipette, but with a much finer tip (<0.5μm) so as to greatly reduce dialysis, and therefore increase the possible length of the experiment. A Sharp electrode has a much higher resistance compared to a patch clamp. However, because the electrode is imbedded in the cell, large voltage changes can be recorded compared to those available within a small section of the cell membrane available during patch clamping. In addition, due to its simplicity sharp electrode recordings can be performed more easily *in-vivo.*

Voltage and Current Clamping

Because many of the ion channels are voltage dependant, it is important to have a way to set the membrane potential. This can be done using similar techniques to patch clamping, but rather than measuring current flow, the membrane is connected to a negative feedback circuit to fix the potential at the desired level (Finkel and Redman, 1983). Current flow can then be measured. Similarly, current flow can be manipulated

(using a technique such as current injection), and the resulting changes in membrane potential recorded.

2.2.2 Extra-cellular Recording

This type of data acquisition encompasses methods ranging from the electroencephalogram (EEG), which measures changes in electrical activity over large areas of the brain, to placing one or more electrodes near individual cells to record their electrical patterns. All the electrical data used in this thesis was collected via the latter method, where a glass microelectrode filled with 0.9% NaCl is introduced to the target area of the brain; in this case, the supraoptic (Sabatier et al., 2004) and ventromedial nuclei (Sabatier and Leng, 2008). Because the probes are outside the cells, the electrical activity is greatly reduced (~1mv→3mv peak-to-peak), so only action potentials can be reliably recorded. When recording extra-cellularly, the electrode is inserted amongst a population of neurones, and hence there is a possibility of detecting multiple neurones at once. It is possible to determine, through the different spike shapes, the origin of each action potential. This technique is known as spike sorting, and can be used to explore small network activity. Using multiple electrodes inserted near one another, it is easier to determine the number of neurons, and the origin of each action potential through relative time delays. An excellent introductory review of basic single- and multi- electrode spike sorting methods has been written by (Lewicki, 1998).

2.2.3 In-vivo Extra-cellular Recording Method

The following sections detail the tools that were used to collect the *in-vivo* neural data which forms the focus of the electrophysiological aspects of this study (Chapters 6, 7). It should be noted that there exist multiple alternatives to the hardware and software platforms used here, but it is their specific function that is important.

Through the use of modelling, neuroscience has been brought nearer to the field of computer science, but even with a large influx of computer scientists into the field, there is still a significant difference in mindset between computer scientist and experimental biologist. Typically, a computer scientist expects academic software to be open source, and would prefer an off-the-shelf hardware solution in order to have total control over the implementation. In contrast, an experimental biologist would ideally want a black

box solution that fulfils their specification. Hardware and software would be bundled as a single package, and thus whether an off-the-shelf solution is used is unimportant. A reliable source of support is also important. The reason for this disparity is twofold:

1) Most experimental biologists do not have an understanding of the hardware or software involved, or of programming.

2) Biology is non-deterministic and unpredictable. Therefore experimental biologists need to be able to rely on their hardware to make the most of their successful experiments, and are willing to pay significantly for this.

2.2.4 Hardware

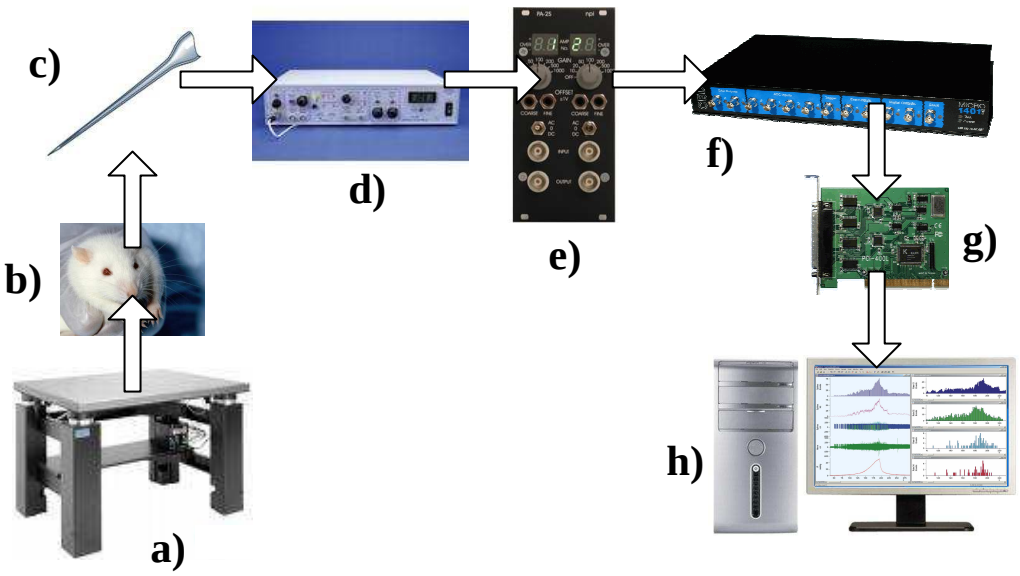


Figure 2-5 Experimental setup for the collection of spike traces *in-vivo*; a) Vibration isolation table; b) Spraque-dawley male rat; c) NaCl filled glass micropipette; d) Pre-amplifier; e) Filter; f) 1401 data acquisition unit, 500KHz 16-bit analogue to digital converter (ADC); g) PCI interface card; h) Desktop machine running Spike2. The last three components f), g) and h) are all supplied by (Cambridge Electronic Design, 2008) .

Figure 2-5 depicts the basic experimental setup required to collect neural spike timestamps in-vivo. A vibration isolation table is required because without it, movement in laboratory, such as footsteps, can cause the end of the probe to deviate away from a cell during recording. To record vasopressin releasing neurons *in-vivo*,

female Sprague-Dawley rats were placed under urethane anaesthesia, and had their Supraoptic nucleus and neural stalk exposed by transpharyngeal surgery. A glass microelectrode filled with 0.9% NaCl was introduced into the supraoptic nucleus under direct visual control. When recording from the VMH, male rats were used. At the end of the experiments the rats were killed with an overdose of anaesthetic. A set of in-vitro vasopressin cell data was also used, and in this case it was collected from coronal slices, also from Sprague-Dawley male rats. All experimental procedures were approved by the local ethical panel of the University of Edinburgh and were carried out by Dr N. Sabatier in compliance with current UK Home Office (Animals (Scientific Procedures) Act) legislation.

The electrical signal from the probe is first amplified, and then filtered to remove background nose. It is then passed to the 1401 analogue-to-digital converter (ADC), which samples the incoming signal, and timestamps the action potentials. This device is connected by a PCI interface to a desktop PC running the software Spike2. The software, PCI interface card and 1401 ADC are all supplied by (Cambridge Electronic Design, 2008).

2.2.5 Software

The Spike2 (Cambridge Electronic Design, 2008) software package was used to collect electrophysiological data within our *in-vivo* laboratory. Spike2 is supplied with hardware that acts as an interface between the pre-amplifier (which is in turn connected to the probe) and the PC. The hardware interface consists of a multi-channel high-resolution analogue-to-digital converter (ADC), the output of which is fed into a custom add-in PCI board that works with any off-the-shelf desktop computer. The software package receives the streams of neural data and stores them, as well being able to perform a number of data analysis techniques on the distribution of spikes, such as “Instantaneous Frequency” and “Inter-Spike Interval” analysis, or even the shape of the action potentials. If required, these data can be output to text file in order to perform further analysis with more specialised statistical packages.

2.3 Methods of Analysis

Due to the relatively small size of action potentials compared to the noise from neighbouring cells, recording cell membrane potential in an extra-cellular manner is unreliable. Because of this, the data used in this study consisted of time-stamped action-potentials received from the cell over a period of time. A series of statistical tests performed on the data were used to classify each cell. The following sections detail the various forms of analysis that were used throughout this thesis, the majority of which are used throughout the neuroscience community.

2.3.1 Inter-Spike Interval

The inter-spike interval (ISI) is a histogram-based method for studying neuron behaviour. The time between event timestamps is calculated and placed into bins, typically 1ms to 5ms in size. Figure 2-6 shows the ISI histograms of a regular firing, and a vasopressin releasing neuron respectively, both found within the hypothalamus.

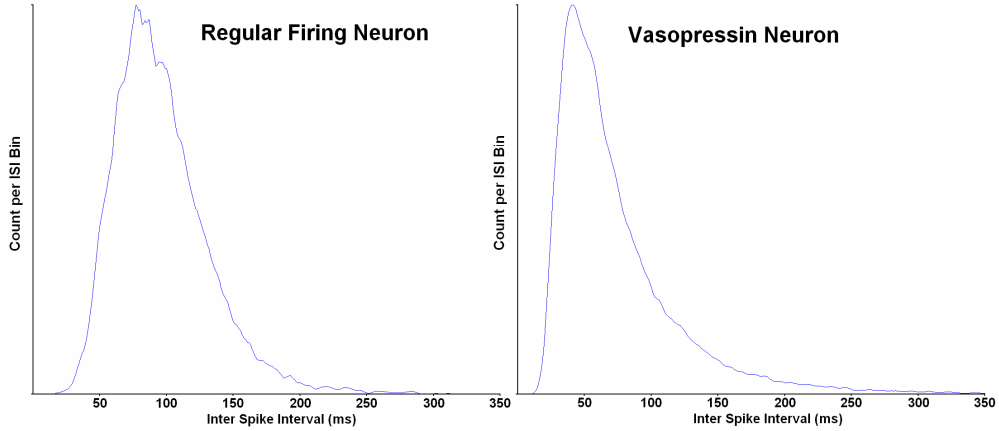


Figure 2-6 Inter-Spike-Interval histograms of Regular firing and Vasopressin releasing neurons. The time between sequential spike events is calculated and binned into 1ms bins. Note the subtle difference between the two traces; the vasopressin neuron has a more rapid rise to the peak interspike interval period, and a slower decay rate at the tail end of the distribution. This shows Data collected by (Sabatier et al., 2004, Sabatier and Leng, 2008)

Once this histogram has been constructed, several statistical tests can be performed upon it. The mode, mean, and coefficient of variance can be used to classify the shape of the histogram. (An average of the 'N' bins each side of the true mode is taken to get an approximation of the mode of noisy distributions that show clusters of peaks near

one another; where, for this thesis, N equals five. This helps to smooth out histograms that are noisy due to lack of data). These values are used to provide information about the skewness and width of the distribution. The skewness and kurtosis may also be calculated. However, when examining cells that exhibit more extreme behaviours, such as the oscillatory firing neurons shown in Figure 2-7, the skewness and kurtosis are much less useful.

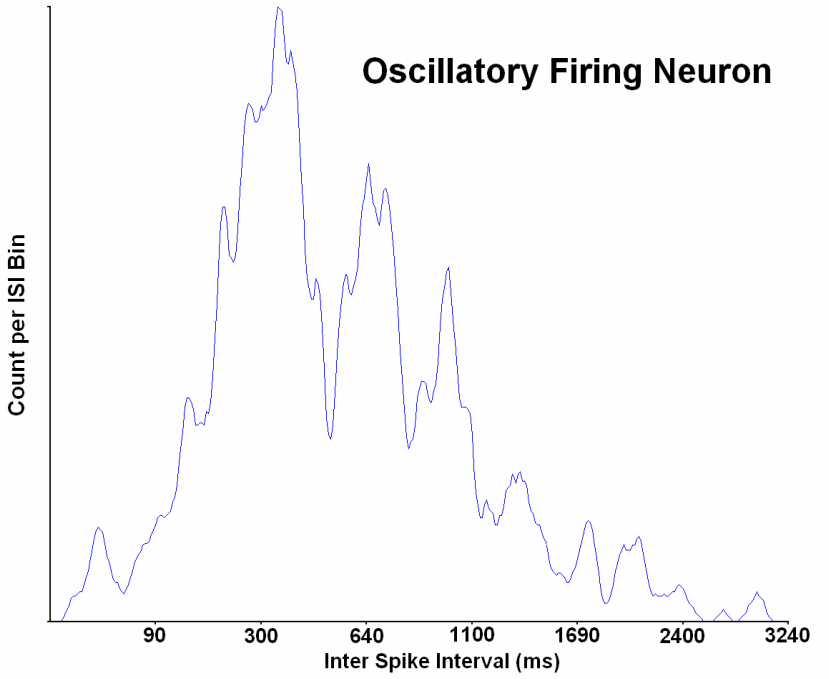


Figure 2-7 Oscillatory firing neuron. This histogram was constructed with linearly increasing bin widths, where the width of the bin increases by 1ms every 20 bins. This oscillatory behaviour is thought to be caused by an external network generated 3-5 Hz rhythm. Data collected by (Sabatier and Leng, 2008).

2.3.2 Hazard function

The hazard function (Luce, 1986, Janssen and Shadlen, 2005), H(n), is a modification of the ISI histogram, I(n), from which it is calculated as follows.

$$H(n) = \frac{I(n)}{\sum_{n}^{\infty} I(n)} \qquad \textbf{2-1}$$

The hazard function represents the probability of the neuron firing after a specific interval following a preceding action potential. The hazard function tends emphasises the tail end of the ISI distribution, and is used to highlight activity caused by specific firing characteristics of the cell, such as a cell's depolarising after potential (DAP). Figure 2-8 shows the Hazard function for the regular and vasopressin neurons whose ISI is shown in Figure 2-6.

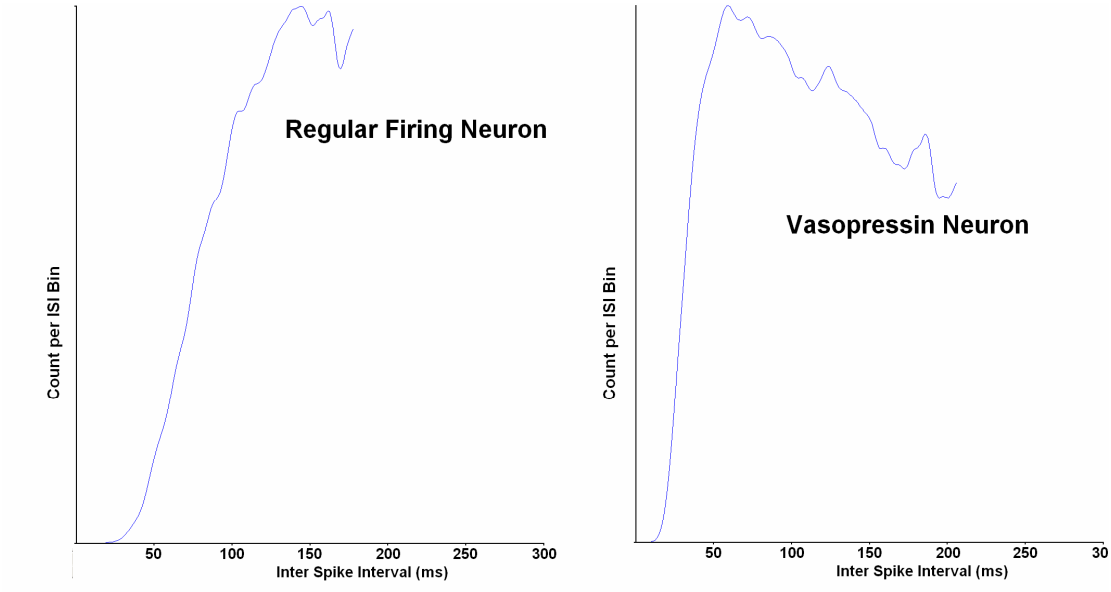


Figure 2-8 Hazard functions for the regular and vasopressin neurons shown in figure 2.6. The hazard function much better highlights the difference between the two traces, with the vasopressin neuron having a significantly increased probability of firing for a short period of time after firing. Data collected by (Sabatier et al., 2004, Sabatier and Leng, 2008)

The DAP is visible in the vasopressin hazard as an area of increased likelihood of firing at around 60ms after the previous spike. (Sabatier and Leng, 2008) used the Hazard function to classify 272 cell recordings, collected from the VMH in the hypothalamus, into 9 groups.

2.3.3 Rate recording

Rate recording is a useful tool for analysing cell characteristics that involve longer timeframes. One example of this is the firing pattern of vasopressin cells, which consist of bursts of activity, up to several minutes long, followed by periods of silence (Figure 2-9).

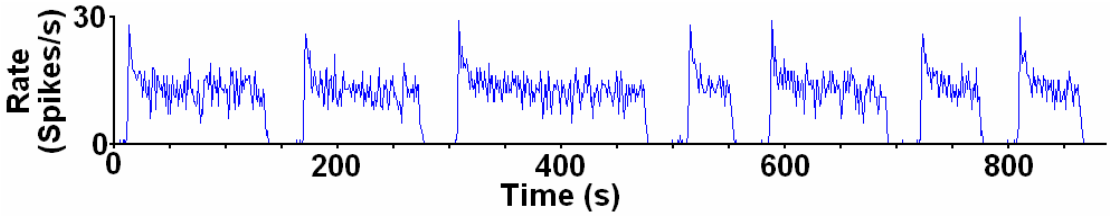


Figure 2-9 Rate recording of a vasopressin releasing neuron. Data from (Sabatier et al., 2004).

Rate recording allows the calculation of burst and silence periods, as well as providing an insight into firing behaviour within bursts. Bursting behaviour has been previously analysed by locating burst initiation and termination times, where a burst has considered to have started after three consecutive bins of activity, and terminated with three consecutive bins of inactivity (Wakerley et al., 1978). This method is prone to errors when analysing noisy cells, and so new more reliable methods are discussed in Chapters 4 and 6.

2.4 Summary

This chapter has provided an introduction to electrophysiology, focusing on the hypothalamus (the area of the brain explored in this study), laboratory-based experimental methods (how data is collected from the hypothalamus), and data analysis techniques. All cellular recordings were collected from the supraoptic and ventromedial hypothalamic regions, extra-cellularly, using a sharp microelectrode. The data from the ventromedial hypothalamic region was collected *in-vivo* from male rats under urethane anaesthesia (Sabatier and Leng, 2008), whilst the *in-vivo* vasopressin cell activity was recorded in a similar manner from female rats (Sabatier et al., 2004). The remaining vasopressin recordings were taken *in-vitro* from coronal slices of male rats (Sabatier et al., 2004). All experimental procedures were approved by the local ethical panel of the University of Edinburgh and were carried out by Dr N. Sabatier in compliance with current UK Home Office (Animals (Scientific Procedures) Act) legislation.

Chapter 3 Introduction to Neural Modelling and Evolutionary Computation

Where Chapter 2 introduced the electrophysiological aspects of this multi-disciplinary thesis, this chapter presents a review of two of the remaining disciplines, namely modelling and evolutionary computation. It is split into four main sections. The first two introduce modelling, with a focus on single cell neural models, and the tools currently available to aid in their construction, simulation, and evaluation. The latter two sections focus on evolutionary optimisation methods, with Section 3.3 introducing evolutionary computation, focusing on a range of archetypical methods. The following section then emphasises the importance of fitness evaluation as a function to guide the adaptation process, specifically highlighting the two main forms of multi-objective fitness evaluation in use today.

3.1 Modelling

Modelling has become increasingly popular over the past 20 years (O'Reilly and Munakata, 2000, Dayan and Abbort, 2001, Feng, 2003), partly due to the increase in computational resources resulting from the processor industry's constant pursuit of Moore's law (Schaller, 1997). Interestingly, even though available computational resources have dramatically increased, the general principles of modelling have remained largely the same, with the only major difference being the size and complexity of models investigated. Within the field of neuroscience, biological models are a useful tools for the conceptualisation of hypotheses involving the function of a neural unit (Koch and I., 1998, French and Pavlidis, 2007), be it a single synapse, neuron, or neural network (Deco et al., 2008). Through simulation (hardware or software), the behaviour of the model can be tested against the behaviour of the real biology, furthering our understanding of the biological system (Silver et al., 2007) through the verification of hypotheses.

The development of the Hodgkin and Huxley model is a good example of successful modelling, showing the archetypical way in which modelling is integrated with experimentation. Hodgkin and Huxley, (1952) measured the properties of the squid giant axon and then derived a mathematical description for the recorded behaviour. This mathematic description (model) had three terms in it, but they didn't know why at the time. This prompted more experimentation which ratified their model, as it emerged that you could map the terms of their equation onto the Sodium, Potassium, and leakage currents. The Hodgkin and Huxley model is discussed in Section 3.1.3.

3.1.1 Model Design Goals

Six basic model design goals were first formalised by Jacobs and Grainger, (1994), and later explored in more detail by Pitt et al., (2002), with many of their arguments summarised in (Pitt and Myung, 2002). When designing a model to test a hypothesis it is important to keep these six goals in mind. These are paraphrased from Pitt et al., (2002) below with examples.

Plausibility

Are the base assumptions that the model builds upon plausible? *e.g. is it plausible to represent collections of ion channels as a single quantifiable variable conductance?*

Explanatory Adequacy

Is the theoretical explanation of the data, and hence the basis of how the model functions, consistent with what is known? *e.g. is the electrophysiological behaviour of a neuron largely determined by its ion channels?*

Interpretability

Are the components of the model linked to known processes? Is the practical implementation of the base assumptions, especially those of the smallest sub-components, sound? *e.g. is the implementation of ion channel collections a reasonable representation of their behaviour?*

Descriptive Adequacy

Can the model reproduce the observed data? *e.g. can the model recreate the original membrane potential dynamics?*

Simplicity

Is the model's description of the observed data, and hence its form, as simple as possible without sacrificing descriptive adequacy? *e.g. could the model produce the similar performance with fewer ion channel representations?*

Generalisability

Can the model correctly predict future observations, both within and outside the range of the original data set? *e.g. can the model produce a distribution of behaviours consistent with the original data set? Can the model predict behaviours outside of the original data set, which are then ratified by experimental methods?*

Of the six measures, the first three require expert knowledge, and are generally not quantifiable, as they depend largely on user preference. However, it should be noted that, as these three measures suggest, it is important to create biological models using concepts that are tied to realistic biological sub-systems. While a rationalist model can be created without these ties, if it does manage to represent the biological system successfully, it will be more difficult to explain the cause of behaviours within the underlying and largely empirical biology. This is because the control parameters will have no relevance to the better-understood biological subsystems. A model that is rooted in accepted biological theory, created from a combination of biological subsystems, will offer greater insight to a wider audience.

The latter three of the six measures are quantifiable, and hence should be used when choosing between two model implementations. As modelling has become more popular, the number of available models and modelling tools has increased dramatically, and hence there is a requirement for an empirical approach for model evaluation. One of the most popular methods is the use of adaptive algorithms to find control parameters that enable the model to produce as close a fit to the observed data as

possible. The model that produces greatest goodness of fit (GOF) after adaptation is classed as the superior model. More importantly for the work described later in this thesis, model parameter adaptation can be used as a null hypothesis, highlighting aspects of current understanding that are faulty. Ultimately, the best approach to testing a hypothesis through modelling is to use the model to create predictions that can be verified through experimentation (Steuber et al., 2007).

Obtaining good fits to observed data through parameter estimation is an intuitive method, fulfilling the requirements of the fourth measure, d*escriptive adequacy*. However, by itself, this measure is often flawed (Roberts and Pashler, 2000). This form of *descriptive adequacy* can be reasonable for analysing systems that exhibit no variation or noise on top of the underlying mechanisms the model is attempting to represent. However, the majority of applications involve modelling systems where the observed data is collected via methods that introduce noise to the observed data set. For instance, noise can be introduced through variation in experimental apparatus, variability across biological specimens, or even sporadic environmental conditions such as variable levels of electromagnetic radiation from external sources. In these cases it is important to identify that the optimal model is the one that mimics the underlying system, not the system plus the noise.

The principle of Occam's Razor is often used to guide the selection process, where the *descriptive adequacy* is combined with a measure of the model's complexity to create a metric that favours models that can be adapted to produce a closer match to the observed data, but not at the expense of model *simplicity*. These new metrics, such as Minimum Description Length (Hansen and Yu, 2001), tend to provide superior measure of a model's *generalisability*, as shown in Figure 3-1.

In addition to having a reduced chance of over-fitting, simple models tend to also be more robust, in that they are less susceptible to extreme behavioural shifts as a consequence of minor control parameter variation. This is desirable in a model, as ultimately, for the model to prove useful in providing an explanation of behaviour, it must show how areas within the parameter space correspond to broad types of behaviour. A chaotic model that shows very little in the way of correlation between

parameter space and behaviour is most likely the product of over-fitting, and is far too complex to be useful.

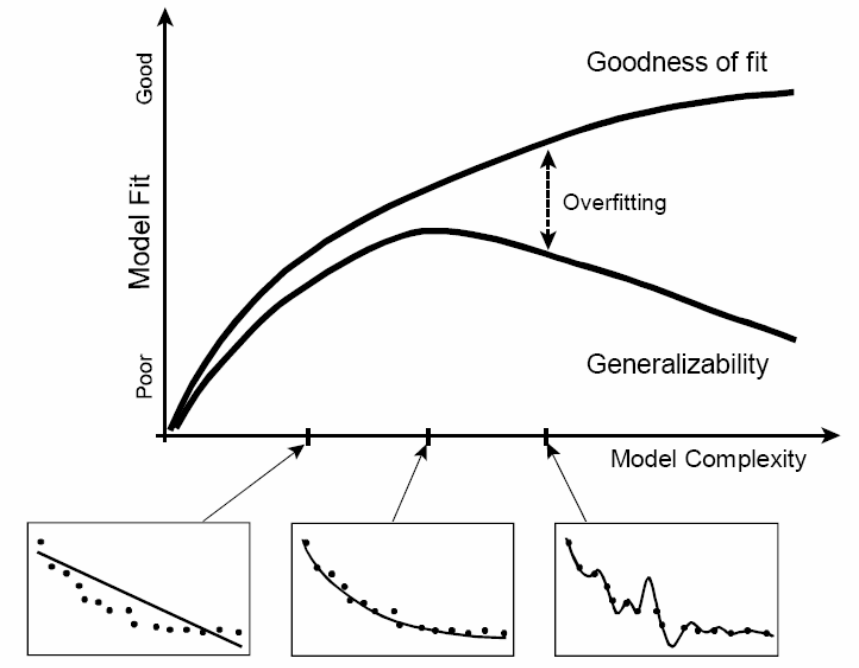


Figure 3-1 Figure and caption re-printed from (Pitt and Myung, 2002). Relationship between goodness of fit and generalizability as a function of model complexity. The y-axis represents any fit index, where a larger value indicates a better fit (e.g. percent variance accounted for). The three smaller graphs along the x-axis show how fit improves as complexity increases. In the left graph, the model (represented by the line) is not complex enough to match the complexity of the data (dots). The two are well matched in complexity in the middle graph, which is why this occurs at the peak of the generalisability function. In the right graph, the model is more complex than the data, fitting random error. It has better goodness of fit, but is over-fitting the data.

3.1.2 Introduction to Single Cell Neural models

Many different types of neural model exist, all of which can be classified by the level-of-detail (LoD) they incorporate when representing the biology (Herz et al., 2006). This level-of-detail can be defined by the smallest instanceable model component, also called a primitive. For a model of a neural network, a highly abstracted behavioural model of a neuron may be the primitive, whereas for a single cell model, the primitive components may represent the summed behaviour of classes of ion channels. For ion

channel models, the behaviour of individual molecules can be used. Ultimately, the LoD represented in a model is limited by the computational power available for simulation, and so as the quantity of available computational power increases, more complex neural models have emerged. This thesis is primarily concerned with single cell neural models, where the level of detail is limited to the effects of groups of ion channels, essentially membrane sub-behaviours. This thesis seeks to aid in the construction of neural models that can explore the cause of neural behaviour through the analysis of experimental data. Because the data collection method (Section 2.1), only provides timestamp data, and not an analysis of synaptic inputs or changes in cell membrane potential, there is not enough data to verify correct sub-behaviour implementations, and so increasing the LoD of proposed models will provide little benefit. In addition, simpler models tend to be more robust, and if a simpler model can be found to fit the behaviour of the biology, then it is more likely that the underlying mechanisms of the biology are similar to that of the model (Occam's Razor). The following sections detail several archetypal single cell neural models with a level of detail that is suited to explaining timestamp data.

3.1.3 Hodgkin-Huxley

Initially developed from data collected by Hodgkin and Huxley, (1952) from the squid giant axon, conductance-based models represent the membrane as the sum of many voltage-dependant ion channels or currents. Each channel is controlled by its conductance, the current value of the membrane potential, and a gating variable that represents the voltage-dependant activity (Skinner, 2006). Figure 3-2 shows the circuit diagram of the basic Hodgkin and Huxley model.

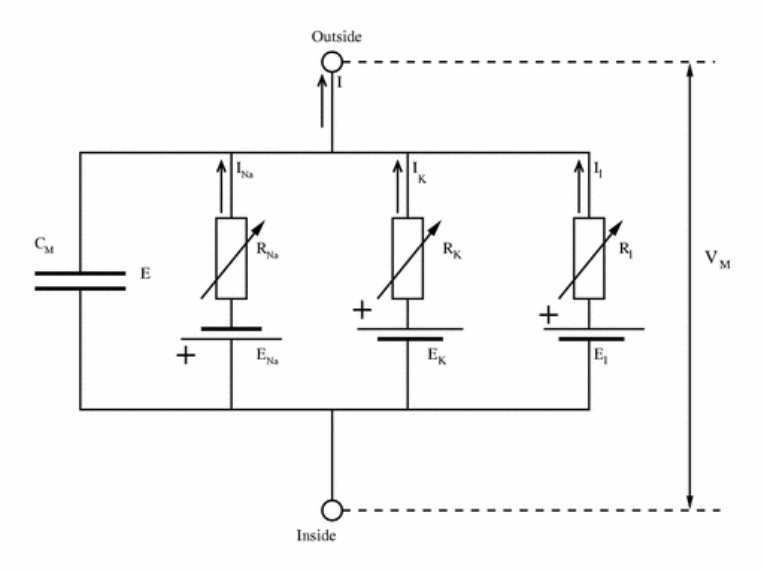


Figure 3-2 A Simple Hodgkin-Huxley circuit diagram. Total ionic current is represented by the sum of the three most prominent current types; Sodium (NA), Potassium (K), and a small leakage current (L). These currents are accumulated on the membrane, which is represented as the capacitor C_M. Each current represents the combined effects of a single type of ion channel, which are individually modelled as voltage controlled current sources, and therefore represented by a voltage source (E_{NA}, E_K, E_L) and variable resistor (R_{NA}, R_K, R_L) in series. Figure based on Figure 1 from (Skinner, 2006).

Even in its simplest state, the original model requires three ion currents to model the flow of sodium and potassium ions, and a leakage current that approximates the sum of the passive properties of the cell. Additional ion channels can be instantiated to better reflect behaviours from different cells, thereby moving away from the traditional H-H model to a generic Conductance based model. The following equation formally describes the function of the basic Hodgkin-Huxley model.

$$I = C_M \cdot \frac{dV_m}{dt} + i_{Na} + i_K + i_L$$

Where

$$i_{Na} = g_{Na,\max} m^3 h \left(V_m - E_{Na}\right)$$

$$i_K = g_{K,\max} n^4 \left(V_m - E_K\right)$$

$$i_L = g_L \left(V_m - E_L\right)$$

Equation 3-1 – Hodgkin-Huxley Equation for the previous circuit. *m*, *h*, **and** $g_{Na,max}$ **are respectively the sodium activation and deactivation coefficients, and maximum conductance.** *n* **and** $g_{K,max}$ **are the potassium activation coefficient and maximum conductance.** g_L **is the leakage conductance.**

Simple conductance-based models are easily implemented within silicon (Lazaridis et al., 2007, Toumazau et al., 1998). However, enabling the model to mimic biological behaviour is considerably more difficult, as even the simplest version has a large number of control parameters to be configured. Additionally, the large capacitances required to enable the model to run in a biological time scale necessitate a large foot-print within the integrated circuit. Despite this, simple single cell models have been successfully adapted to the behaviour of a real cell through in-vitro patch clamping (Saighi et al., 2006).

Although the H-H model has undoubtedly moved the field of neuroscience forward, it should be questioned as to how good the model, which has essentially been untouched for 50 years, is. One serious problem with the H-H model, which has led to incorrect predictions, is that its behaviour regarding ion channel collections is not actually based upon the underlying mechanics of individual ion channels (Bezanilla and Armstrong, 1977). Additionally, full conductance-based models tend to be avoided when exploring network activity, as the number of control parameters can be unacceptably large, resulting in a computationally intractable problem (Lytton and Stewart, 2005). This has led to a situation where the standard H-H model may be perceived as not accurate enough for some single cell models, and is not tractable enough for the study of network activity. Meunier and Segev, (2002) produce an excellent article discussing this topic.

3.1.4 Condensed Conductance-based models.

Originally proposed and developed by FitzHugh, (1961) and Nagumo et al. (1962), condensed conductance-based models are reduced forms of the Hodgkin-Huxley model, capturing the majority of functionality, but making sweeping simplifications, thereby reducing its overall complexity. The resulting simplified model comes in the form of two simultaneous equations, one representing membrane potential, the other a recovery variable that is used to reset the membrane potential post-action potential.

$$\dot{V} = V - \frac{V^3}{3} - W + I$$

$$\dot{W} = 0.08(V + 0.7 - 0.8W)$$

Equation 3-2 The FitzHugh-Nagumo model is a two dimensional simplification of the Hodgkin-Huxley model; V → Membrane potential; W → Recovery Variable; I → Magnitude of Stimulus Current.

In addition to being more computationally tractable than the Hodgkin-Huxley model, the FitzHugh-Nagumo model behaviour can be represented and tracked within a two dimensional space (phase portrait, see Figure 3-3). This allows the investigation of specific biological phenomena that are linked to spike generation, through the analysis of the visually changing patterns within the phase portrait.

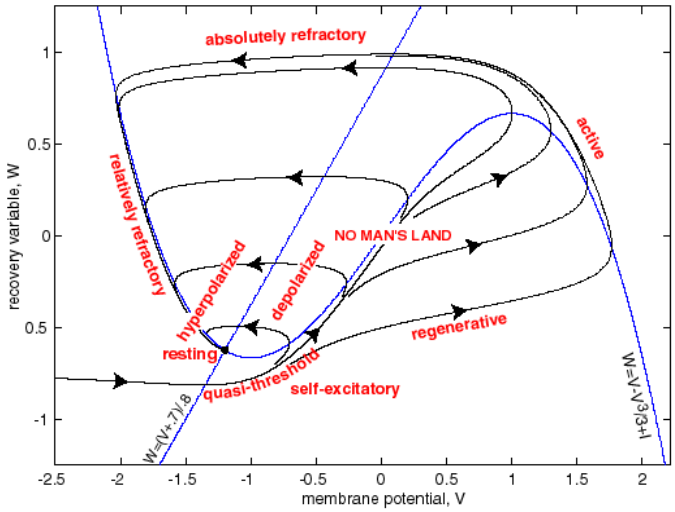


Figure 3-3 Phase portrait and physiological state diagram of FitzHugh-Nagumo model (modified from FitzHugh 1961)

The Izhikevich model (Izhikevich, 2003) is a further reduction of the FitzHugh-Nagumo model. Through the examination of biological data, many of the control parameters are fixed to values that limit the model to behaviours of specific neurons, such as cortical neurons. This specialisation reduces the number of control parameters to just four, whilst still enabling production of an extensive range of behaviours from the modulation of these parameters (Izhikevich, 2006). A summary of the Izhikevich model is given in Figure 3-4.

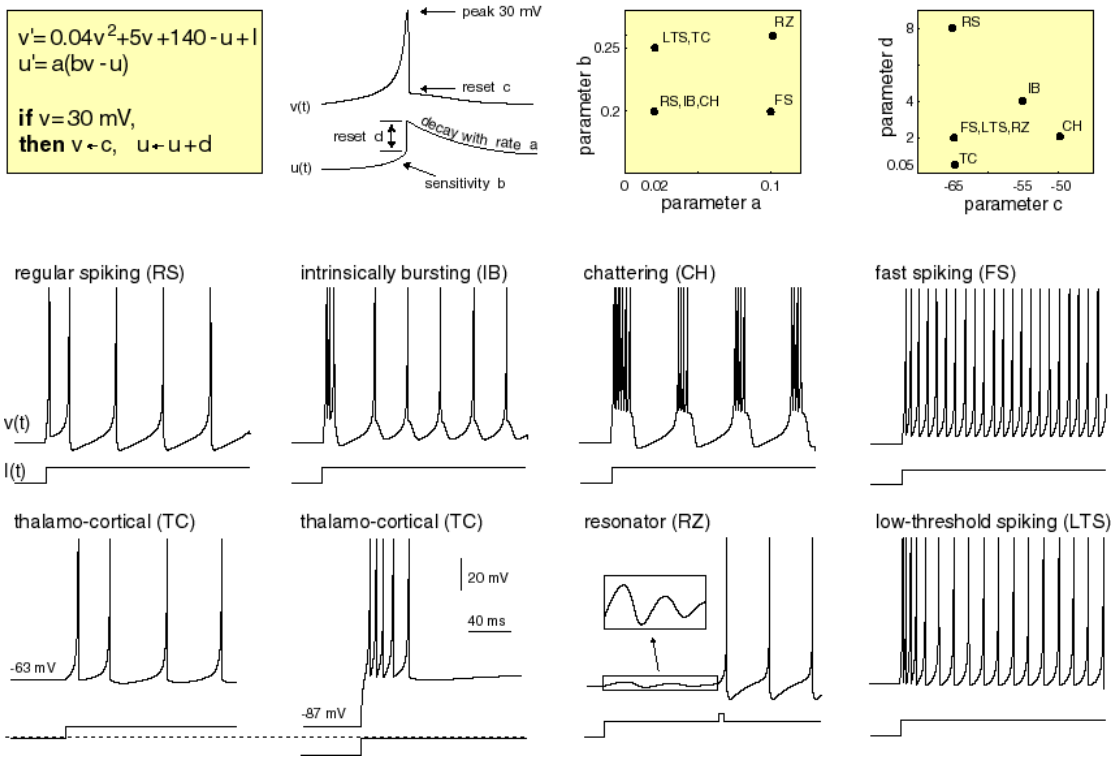


Figure 3-4 Summary of Izhikevich's model showing the large range of behaviours that it can produce. Electronic version of the figure and reproduction permissions are freely available at www.izhikevich.com

3.1.5 Integrate and Fire (IF) models

Developed by Lapicque in 1907 (Abbott, 1999), the Integrate and Fire (IF) model treats the neuron as an electrical circuit that accumulates incoming electrical impulses (Pre Synaptic Potentials or PSPs) onto a capacitor (the membrane), until a threshold voltage is reached. At this point, the neuron fires, and the membrane is reset to its rest value. To make the model more biologically-similar, the capacitor can be made leaky, and a refractory period can be enforced after firing to prevent immediate re-activation. Figure 3-5 shows the circuit diagram of an IF model without a refractory period. This model is commonly used (Burkitt, 2006) because of its simplicity, only requiring a few control parameters. Indeed, its simplicity, and its construction from electronic primitives, facilitates easy fabrication onto silicon (van Schaik, 2001), allowing for the creation of large neural networks by instantiating these IF modules, as well as creating add-on functionality. An example of this modularity is the ability to easily place synapse models between the neuron and the PSPs (Bofill-i-Petit and Murray, 2004).

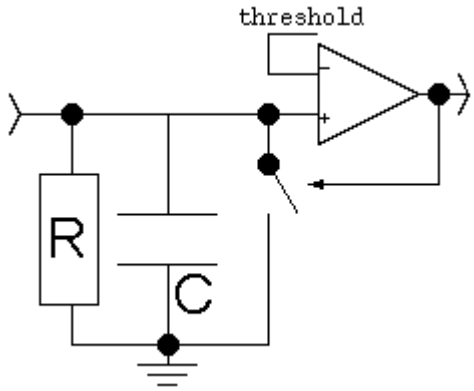


Figure 3-5 A Leaky Integrate and fire neural circuit

Due to the simplicity of the IF model, it does not capture many of the more complex cell dynamics that other models can provide. As a result, this style of model is usually only used in systems where an understanding of population dynamics is the goal of the modelling procedure. An example of such a system is the task of modelling the mapping processes that occur between different areas of the brain. In this case, a simple Hebbian learning scheme is applied to the synapses of each neuron, along with a mechanism that mimics chemical gradient behaviour to represent the facilitation of axon growth (Bamford et al., 2008).

3.1.6 Spike Response Models

The Spike Response Model (SRM) (Gerstner et al., 1993, Kistler et al., 1997, Jolivet et al., 2003, Gerstner and Kistler, 2002) is an extension of the leaky IF model, where the membrane potential of the cell is derived from the time since the last action potential initiation, and the time since each incoming synaptic event. The impact of both the cells own action potential, and incoming synaptic events are defined by two kernels that describe the shape of their impact with time, η and κ respectively. In this way, the SRM introduces a measure of refractoriness to the IF model, that describes a combination of effects, such as increasing threshold, hyperpolarizing afterpotential and post action potential reduced responsiveness.

$$u(t) = \eta(t - \hat{t}) + \int_0^\infty \kappa(t - \hat{t}, s) I(t - s) ds \qquad \textbf{Equation 3-3}$$

Equation 3-3 describes the membrane potential as a function of time. It is constructed through the sum of influence from the kernel that describes the shape of the membrane potential, both during and post firing, and the sum of the influence from each synaptic input. An example of the two kernels is shown below in figure 3.6.

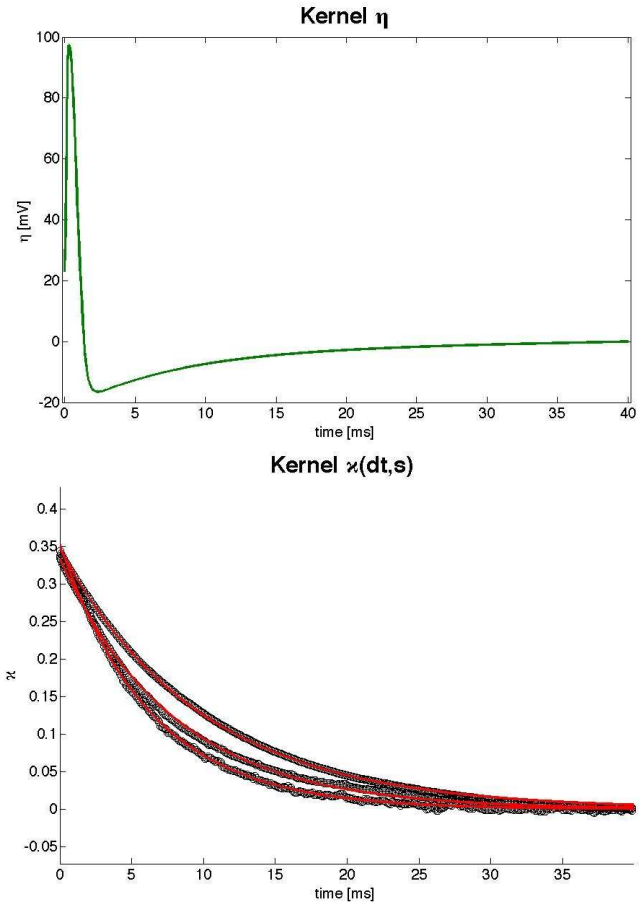


Figure 3-6 Graphs showing the kernels that control the model's response over time to (top) an outgoing action potential and (bottom) three incoming synaptic events. The outgoing action potential consists of an initial peak (the spike) followed by a refractory period which greatly decreases the probability of re-firing. The incoming synaptic events are represented by a single exponential decay (the red lines are a best fit to different values of this exponential) Both images re-printed from (Gerstner, 2008).

The advantage of this form of model is that it can be made to produce a biologically realistic response to incoming synaptic events, whilst still being extremely simple in structure. This makes the SRM an ideal candidate for network models (Gerstner, 2000), as well as being easily adaptable to experimental data (Jolivet et al., 2006).

However, the basic SRM cannot represent longer term behavioural patterns, such as adaptation and bursting, as the probability of firing is determined only by the effects of the most recent action potential. The cumulative spike response model (CSRM) (Gerstner and van Hemmen, 1996) provides the solution to this problem. Where the SRM is only interested in the most recent action potential, the CSRM calculates its membrane potential from the combined effects of several spikes, with synaptic events treated in the same manner as within the original SRM (Equation 3-4).

$$u(t) = \sum_{t^k} \eta\left(t - t^k\right) + \int_0^\infty \kappa\left(t - \hat{t}, s\right) I\left(t - s\right) ds \qquad \textbf{Equation 3-4}$$

3.1.7 Noise and the SRM

While the models mentioned previously were based on mimicking cell membrane potential, those used in this study instead track the cell's probability of firing, also known as excitability (Paninski et al., 2004). The SRM and CSRM can be simplified by replacing the synaptic input with a noise source, which then allows their study through cell excitability. This model, which has been called stochastic threshold model (Gerstner and Kistler, 2002), is a significant improvement for applications that involve parameter estimation, as the noise source can be reduced to a single parameter, thereby reducing the dimensionality of the model and greatly simplifying the adaptation process. Additionally, the model's ISI distribution can now be calculated analytically as a probability density function (PDF). The same process can be performed for CSRMs but the calculation of the PDF is significantly more complex.

3.1.8 Excitability-based Models

The models presented in this thesis are based on a cumulative spike response model with noise, using a series of summed exponential decays to represent the accumulation of all previous spike responses. Each exponential corresponds to an excitatory or inhibitory post action-potential membrane dynamic (the Hyperpolarising After Potential (HAP) (Roper et al., 2004), Depolarising After Potential (DAP), and AfterHyperPolarization (AHP) (Andrew and Dudek, 1984)). When Combined with the noise source control parameter, this creates a 7-dimensional parameter space where

pairs of parameters can be directly tied to the magnitude and duration of specific identifiable intrinsic cell characteristics.

Excitability-based models are computationally simple because the model's spike response is a sum of exponentials, where each can be calculated at every time step by a single multiplication. Additionally, the response to the entire spike history, instead of just the most recent spikes, can be calculated at the same time using the same method. This is because the overall spike response over time is split by individual post action-potential membrane dynamics (formally described in discrete and continuous forms in Equations 3-5a and 3-5b).

$$\frac{\Delta M}{\Delta t} = -M\left(1 - e^{-\frac{\ln(2)\Delta t}{M_{HL}}}\right) + M_{ST}.H\left[E_F - E_{TH}\right] \qquad \textbf{Equation 3-5a}$$

Where M = Magnitude of the post action potential membrane dynamic; Δt = size of the simulation time step; M_{HL} = Half-life of the membrane dynamic; M_{ST} = Step increase triggered by cell firing. $H\left[E_F - E_{TH}\right]$ = Heavyside function that resolves to 1 when the overall membrane excitability E_F is greater than the membrane threshold E_{TH}.

$$\frac{dM}{dt} = -\frac{M}{\tau_M} + M_{ST}\sum_{t^S<t}\delta\left(t - t^S\right) \qquad \textbf{Equation 3-5b}$$

The above equation represents the behaviour with the standard continuous formal representation. The exponential decay is reduced to the time constant τ_M, and the Heaviside function is approximated by the Dirax delta function δ, where t^S is the time of spike production, which are generated when the total membrane excitability, E_F exceeds the threshold E_{TH}.

With fixed control parameters for M_{HL} and M_{ST} the bulk of the function can be simplified prior to simulation to a single multiplication and a possible addition:

$$\frac{\Delta M}{\Delta t} = -M.\tau_M + M_{ST}.H\left[E_F - E_{TH}\right] \qquad \textbf{Equation 3-6}$$

The bulk of the model complexity is tied to an easily conceptualised membrane response to cell activation. Therefore this model's computational simplicity and intuitive nature makes it an ideal candidate for implementation within a laboratory along side time sensitive experimentation.

3.1.9 Discussion of Model Suitability

For the most part, electrophysiologists are biologists with some degree of statistical knowledge and do not specialise in computer science or mathematics. Hence many are uncomfortable with models whose description and operation are mathematically challenging, or are presented in unalloyed mathematical language. Thus, it is important to use models that are

(a) Directly linked to biological phenomena that they can visualise, recognise and understand, and

(b) Presented with biological terminology as much as is possible.

From a biological standpoint, conductance-based models are best suited to the biological mindset, as they are essentially a weighted sum of different ion channel representations, which are well understood biological subsystems. However, due to the large number of control parameters and the resulting "curse of dimensionality", (Powell, 2007) optimisation algorithms can find it difficult to find and reproduce specific behaviours. More importantly, over fitting becomes more likely, as un-biological parameter sets can be found that produce unstable, but biologically realistic behaviours.

The Izhikevich model overcomes this problem by using a reduction of a condensed conductance-based model, the result being a four dimensional parameter space, within which it is possible to segregate areas of specific behaviour. This sounds ideal, but in fact the Izhikevich model has a flaw; due to its lossy method of parameter reduction, each of the control parameters represents a combination of many different factors that cannot be easily re-separated. This means that there is no hard link between a parameter and a single cell dynamic.

The ideal model would be one rooted in biological theory, with few control parameters that are all directly linked to biological phenomena. These two goals are obviously

contradictory, and therefore the ideal model where complex neuronal behaviour is controlled by a single biologically inspired control parameter is impossible to implement. Currently, the Izhikevich model, due to its small number of control parameters and easily conceptualised variable space, is one of the best models for the development and exploration of network activity. This is because in this situation only the behaviour of the individual units, not an explanation of their function, is required.

The advantages of a spike response model (SRM) based approach over the Izhikevich model are that its parameters (spike response kernel) are tied to a quantifiable cell characteristic, and therefore predictions made by the model can be tested directly by in-vitro intracellular whole cell recordings. The drawback of SRM based approaches are their relatively large number of input parameters compared to the Izhikevich model, where even the basic excitability-based model consists of seven control parameters. However, for the purposes of behavioural examination, this can be reduced by fixing the HAP control parameters at biologically plausible values, and treating the DAP and AHP as multipliers of the HAP control parameters. This reduces the model down to a 5-dimensional space, which is much more easily analysed through an exhaustive search, allowing firing behaviour to be explained as a function of relative post-action potential membrane dynamics. It should be noted that this 5-dimensional search space cannot be used to fit experimental data, as the biology will have a range of HAP sizes and half-lives. However, the full model can be made to mimic a wide range of firing behaviours.

Excitability-based models offer a convenient middle ground between the simplicity of the Izhikevich model and the more biologically structured conductance based models. They still represent an abstraction of the underlying cell mechanisms, but do not simplify to the degree that many internal cell mechanisms are reduced to single mathematical equations. Instead, control parameters are tied to easily separable functional blocks that represent behavioural cell mechanisms, which have quantifiable characteristics that can be identified through laboratory experimentation. Additionally, excitability-based models have a memory of cell activity that allows the re-creation of more complex firing patterns over a longer timeframe, an example being doublet firing activity recorded from the VMH, which is further explored in Chapter 5.

The work presented in this thesis is based upon the modelling of neuron firing patterns, and is therefore not interested in mimicking membrane potentials, just the probability of firing. Additionally, this work aims to bring modelling within the laboratory, which imposes time limitations, hence computational complexity is an issue. Therefore all models within this thesis are derived from the basic excitability-based model, which is further explored in Chapter 5. Future work may involve the use of patch clamping (Section 2.2.1) where accurate membrane recordings are taken, and therefore a model that can more closely mimic this data would be required, such as the FitzHugh-Nagumo model.

3.2 Modelling Platforms

Many software packages exist to aid the modelling and simulation processes. Some of these, such as NEURON and GENESIS, have been in use for decades, with their developers constantly improving and adding new features as the cost of computation diminishes. This section provides an introduction to a range of modelling packages commonly used in computational neuroscience laboratories. This is done to determine if there currently exists a platform that already has the functionality required to be implemented within a laboratory alongside experimentation and used by an experimentalist with no programming or algorithmic expertise.

3.2.1 NEURON

The NEURON simulation environment (Carnevale and Hines, 2006, Hines and Carnevale, 1997) is the oldest and most established neural modelling environment available. It was initially developed over twenty years ago by John W Moore from Duke University (Hines, 1989), as a neural specific framework for the simulation of biologically realistic models. Since its inception, it has been constantly improved, guided by feedback from its community of users.

One of NEURON's initial design goals was to provide neuroscientists with a simple interface to high-level modelling by removing the mathematical aspects of model design, thereby allowing neuroscientists with a more biological mindset to better understand and utilise the environment. One of the ways this was accomplished was through the use of *natural syntax*, an example of which is a *section*, which is a

computationally efficient representation of a single continuous un-branched cable (Hines, 1984). *Sections* can be connected together to create complex neural structures, such as dendritic trees, without the modeller having to understand cable mathematics. This is termed "compartment modelling", where each of the sections is evaluated independently, and its electrical properties are represented as a single isopotential. Through the use of Sections and other similar constructs, it is possible to create realistic and complex neural models relatively painlessly.

Because the NEURON environment is tightly coupled with biologically-realistic functional blocks, it would seem highly suited for use by biologists within a laboratory environment. As part of the constant modification and improvement of functionality that NEURON undergoes, new tools have been bolted on to the existing system (Channel Builder, Cell Builder, Import3D, Linear Circuit Builder, Network Builder, ModelView, Multiple Run Fitter, Impedance Tools). Although these provide an impressive variety of up-to-date functionality (Hines and Carnevale, 2005), the application is built primarily on a command line interface called Hoc, which is less accessible to non-expert users. The Graphical User Interface (GUI), while flexible and robust, is not user friendly, suffering from a lack of clarity and consistency with modern computer applications (Dix et al., 2003). In an effort to correct these failings, the NEURON development team have decided to incorporate the Python programming language instead of the original Hoc language. This move to a more standardised and well known language will aid in attracting users, some of which may have been alienated by the old Hoc interface.

Recent advances in the NEURON framework now allow computation to be split across many processors allowing for a near linear speedup in simulation time (Migliore et al., 2006, Hines and Carnevale, 2008). However, at the time of writing, no move has been made to explore incorporation of other hardware acceleration platforms, such as GPUs.

3.2.2 NEST

While NEURON focuses primarily on biologically-realistic models of single neurons or very small networks, the NEural Simulation Tool (NEST) (Gewaltig and Diesmann, 2007) is a software package for large network simulation. NEST is a framework that is

designed to handle large networks of simple neuron models (Morrison et al., 2005), using *nodes* and *connections* that represent the soma and dendrites of each neuron. Communication is facilitated through the use of *events* (Spike, Current, Potential) that traverse the connections between *nodes*. As NEST is specifically aimed at exploring network behaviour, the exploitation of parallelisation has been a design goal from the offset. Because of this, NEST is designed to efficiently capitalise on multi-processor systems, where the processors may be distributed over a network.

NEST currently provides an assortment of premade neural models that can be instantiated to create networks, but if a practitioner wishes to alter these models, they are required to edit the computational code, written in C++. As much of the user interface is command line based, and network configuration requires coding, NEST is not currently suitable for use by experimental biologists, though this may change, as the NEST initiative is in the process of implementing a Python interface. Even so, as the majority of applications within a laboratory involve single neurons and not networks, NEST was not considered a suitable development platform for use in this research.

3.2.3 GENESIS

The General Neural Simulation System (GENESIS) (Bower and Beeman, 2007, Bower and Beeman, 1998) was the first attempt at a truly generic neuronal modelling and simulation platform. Currently on version 2.3 (Bower et al., 2003), and developing version 3 from the ground up, GENESIS is a Unix based platform designed to support biologically-realistic models ranging from subcellular components all the way up to neural networks. Modelling is facilitated by an object oriented design approach, which allows customisation of both the models and the simulation environment. Model functionality can also be extended through the creation of new functional blocks.

GENESIS has become a popular choice of modelling framework, both because of its ease-of-use (Hucka et al., 2002), and the level of support available for it. It has a very adaptable GUI, and was designed to be used with biologically-realistic neural models. The drawback of the GUI flexibility is it needs to be customised to produce the desired interface, and this customisation is done through code. GENESIS 3 plans to extend its framework making GENESIS platform independent, as well as including interfaces to

the NeuralML standard (Goddard et al., 2001), and the ability to utilise many different forms of hardware acceleration.

Due to the platform limitations, the current GENESIS framework could not be used at Edinburgh because our laboratory, like many others, is predominantly Windows based, and GENESIS is designed to run on Unix or Linux. However, future work should investigate, when it is released, the use of GENISIS 3 in providing the functionality required, such as the GUI elements and NeuralML interface.

3.2.4 SNNAP

The Simulator for Neural Networks and Acton Potentials (SNNAP) (Ziv et al., 1994) was developed to facilitate the rapid development of realistic single- and multi- neuron models. SNNAP calculates the electrical properties of individual neurons through the use of conductance based methods, similar to those used in the Hodgkin-Huxley model (Voltage dependant ion channels). SNNAP is also capable of modelling the changes in, and the corresponding effects of, intracellular secondary messenger ions, such as Ca+ stores. Connections between neurons are modelled through either electrical or chemical synapses.

The array of functionality that SNNAP provides is impressive, almost equalling that of NEURON and GENISIS. SNNAP, although initially designed to model neurons as a single isopotential compartment, can also produce multi-compartment models, thereby allowing the creation and simulation of complex dendritic structures. The advantage of SNNAP over both NEURON and GENESIS is its graphical user interface, which enables a user with no programming skill to construct and evaluate neural models through, amongst other methods, the creation and tuning of control equations, Figure 3-7. The GUI also provides an excellent tool for the representation of simulation data, being able to create and save various plot types including variable phase plots.

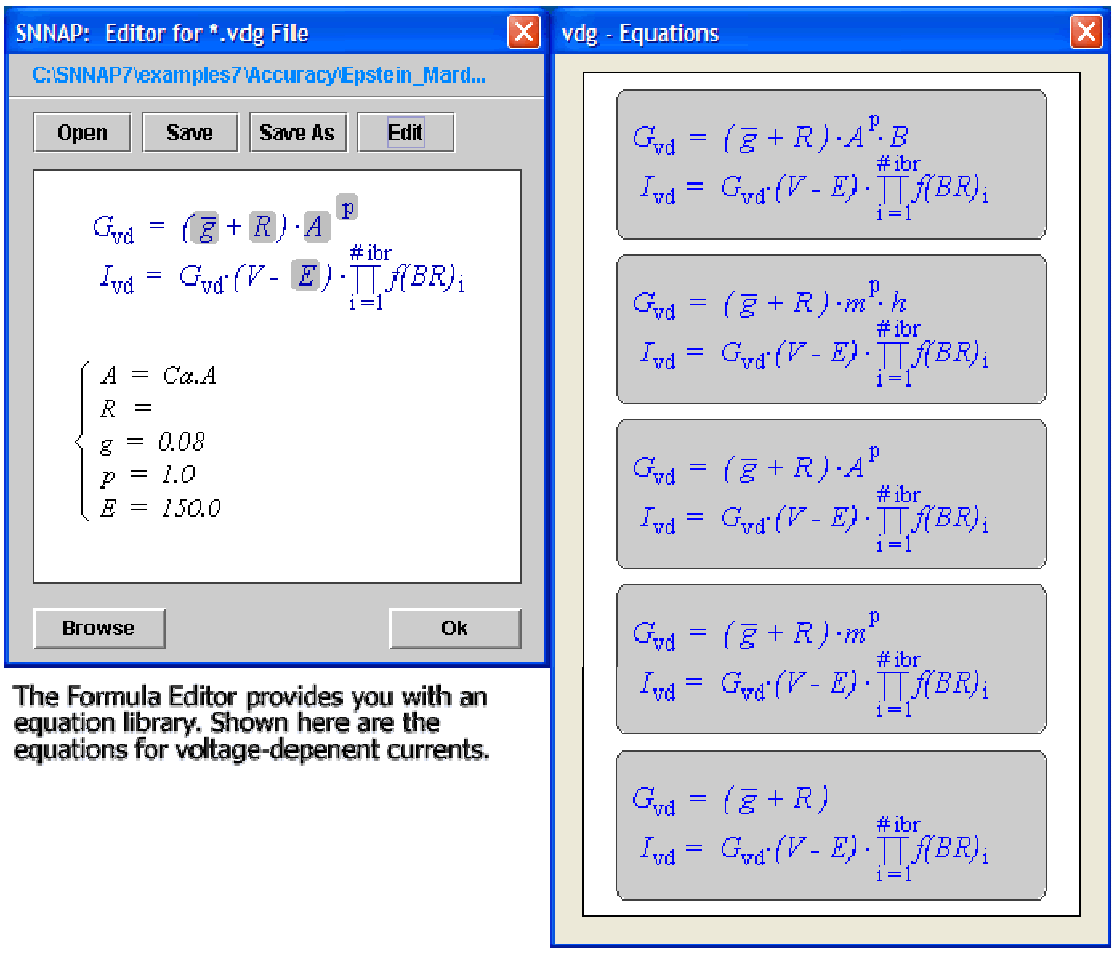


Figure 3-7 The SNNAP formula editor provides an intuitive interface for controlling the various neural equations. Image taken from the SNNAP website http://snnap.uth.tmc.edu/ .

SNNAP's strength is by far its usability, which is currently one of the greatest of all generic single neuron and small network modelling and simulation platforms. SNNAP's ease of use coupled with its platform non-specific nature has helped gather a strong and diverse user base, where SNNAP is used for both teaching and research (Av-Ron et al., 2007). However, although there has been a poster presentation on the parallelisation of SNNAP, focusing on server clusters and super computers (Av-Ron et al., 2005), there is no indication that the current release includes these features, which therefore limits the size and complexity of SNNAP models.

3.2.5 SURF-HIPPO

Surf-Hippo (Graham, 2004) is a neural modelling platform designed to create and simulate biophysically detailed compartmental models of neurons and neural networks,

and is therefore functionally similar to both NEURON and GENESIS. Surf-Hippo, however, is programmed fully using the Lisp language, which is its main selling point. Lisp is a hi-level functional programming language designed to handle symbolic representations, which therefore facilitates the creation of models with complex interactions. The syntax of Lisp is also perceived to be more natural, and the creators state that this natural syntax, combined with the Surf-Hippo GUI, allows the efficient construction and analysis of neural models by users that do not have advanced programming skills. Similar to GENISIS, Surf-Hippo is a built upon a high level interpreter, and therefore benefits and suffers for this. Interpreted modelling environments tend to be less optimised as the hi-level program does not have direct access to the system processor. However, the major advantages of using an interpreted language are the flexibility and simplicity that allows swift model revision and analysis.

Although Surf-Hippo includes a GUI, it is not as intuitive as other modelling platforms such as SNNAP and EONS (below), partly because although the main underlying functionality has seen revisions and improvement, the GUI was mostly left in its initial state. Hence over the years, as with many of these modelling platforms, the GUI has become dated and less intuitive. Additionally, it should be noted that although Surf-Hippo can re-create the majority of the functionality that GENESIS and NEURON can provide, Surf-Hippo does not have a community of users with a size comparable to that of both NEURON and GENESIS, hence there is less overall support in the form of forums, user guides, demonstrations etc.

3.2.6 EONS

Elementary Objects of Neural Systems (EONS) (Bouteiller et al., 2006) is platform for the creation and simulation of synaptic models. EONS allow the creation of synapses from the ground up, starting with ion channels, release sites, and postsynaptic receptors, working up to more complex synaptic mechanisms. EONS is a centralised Java WebStart application, where all the models and synaptic sub elements are stored in a central database, and therefore requires an internet connection to use.

Currently on version 2 (Bouteiller, 2009), which supports the creation of custom equations to control the various model elements, EONS represents an interesting

alternative to GENESIS for the exploration of synaptic behaviour. Unlike the current version of GENESIS, EONS is not limited to any specific operating system. However, because the majority of the functionality is controlled offsite, the utilisation of hardware acceleration is currently impossible. Additionally, the software is not open source, which depending on the author, may limit its extension to be used within the laboratory. One of the advantages of EONS is its excellent graphical user interface (GUI), which is designed to accommodate non-expert users through an intuitive computer aided design interface (CAD).

3.2.7 COPASI

COmplex PAthway SImulator (COPASI) (Hoops et al., 2006) is the successor to GEPASI (Mendes, 1993), both of which offer a platform for modelling biochemical networks, and can hence represent electrophysiological dynamics as chemical reactions. As with the majority of modelling platforms COPASI supports a range of modelling methods, including multi-compartmental models. COPASI utilises the Qt toolkit (Eng, 1996) for its GUI elements, and is therefore platform independent, as well as intuitively styled, see Figure 3-8.

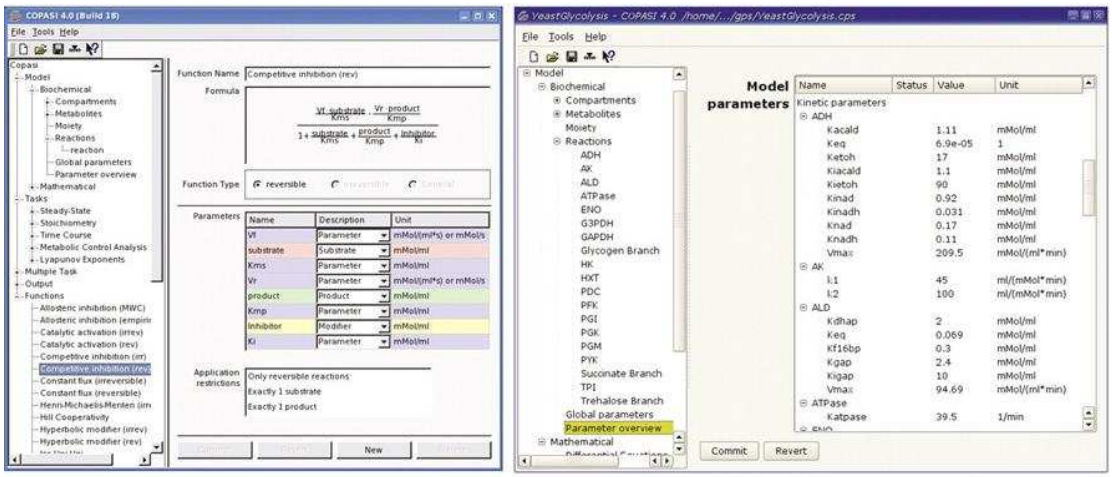


Figure 3-8 Two snapshots of the COPASI user friendly GUI. Image adapted from (Hoops et al., 2006).

Because this software is aimed towards biochemists, it does not involve coding, instead relying on the formulation of mathematical equations that dictate the changes in concentrations of chemicals over time. COPASI also includes a range of optimisation

algorithms for fitting models to experimental data, including several evolutionary approaches as well as a standard array of gradient decent algorithms.

Ultimately COPASI was not designed with neural modelling in mind, instead aiming for a much broader audience of computational biochemists. However, the target audience of the work presented in this thesis is primarily biologists, and hence may be more accepting to a modelling platform that is based on chemical rather than electrical interaction. A major drawback of COPASI is its apparent lack of ability to utilise hardware acceleration, limiting the size of created models, and the speed of parameter fitting.

3.2.8 Specialist modelling and simulation platforms

Because of the size of the field of neuroscience, and the time limitations set by the average length of a PhD or post-doctorate position, the majority of practitioners tend to write their own tools for the exploration of their particular subsection of neuroscience. These usually consist of a focused simulation environment that allows minor modifications to the structure, and parameter control, of a particular class of model. Additionally a specifically tailored user interface is usually supplied that highlights the relevant model behaviours. Some of these specific platforms are discussed below.

The Auditory Image Model of peripheral auditory processing (AIM) (Patterson et al., 1992, Patterson et al., 1995, Patterson, 2000) is designed to explore how the brain processes complex sounds like music and speech. AIM has been extended from its initial design to work with Matlab, using Matlabs user interface features, as well as to run in C for distribution of workload across server clusters.

SpikeNET (Delorme et al., 1999, Delorme and Thorpe, 2003) is a platform designed specifically for the implementation and real-time simulation of very large (10,000,000+) asynchronous integrate and fire neural networks. SpikeNET is heavily focused towards image processing applications, specifically object recognition, with its main goal being the understanding of the human visual system, and its implementation within software recognition systems.

"MCELL: A Monte Carlo simulator of cellular microphysiology" (Stiles et al., 1996) is a modelling environment designed to explore biological signal transactions, namely those linked with synaptic transmission. MCELL is designed to take advantage of the increasing quantity of computational power to simulate large-scale 3D microphysiological models (Casanova et al., 2004), which can then be visualised through impressive 3D snapshots and animations. Unfortunately, MCELL is limited to practitioners with at least a small amount of programming experience as MCELL has its own model description language (MDL) which is essentially a scripting language.

3.2.9 Enabling Platforms

The field of neuroscience is very broad, and the aspects of this field that make up the sub-field of computational neuroscience are diverse. Because of this, for a long time, the majority of research involved the creation of stand-alone tools. Recently there has been a push to standardise much of the field, or at least to provide methods to tie the various tools together. The aim is to limit the amount of replication that researchers perform when entering the field or developing tools of their own. The following sections highlight the three main areas that have been targeted; code redundancy; overarching support platforms; and standardised data communications methods, which tend to focus on mark-up languages. Many of the concepts described have been implemented within the field of systems biology, which shares much in the way of computational approaches, but have only limited interaction with computational neuroscience. (De Schutter, 2008) provides an excellent review of the two subjects, focussing on their communities and the sociological aspects that contribute to their separation.

Code Redundancy - MIIND

Multiple Interacting Instantiations of Neuronal Dynamics (MIIND) (de Kamps et al., 2008) is a collection of pre-designed neural components, the functionality of which is greatly desired by practitioners that wish to model higher cognitive functions, such as neural networks. The general philosophy behind MIIND is to provide this functionality in a standardised and easily implementable form, without enforcing any particular modelling strategy. As a result the programming tools provided are designed to

function as programming shortcuts, and thereby increase the productivity of cognitive neuroscientist.

MIIND is a strong contender to be used as part of a future network model as it can provide much of the underlying network infrastructure without requiring the understanding of a scripting language to facilitate network construction. Additionally, the MIIND libraries do not presume, and therefore limit development to, any specific underlying hardware acceleration. This is useful when facing a constantly evolving and turbulent hardware acceleration market.

Overarching Support Framework - Bio-SPICE

Biological Simulation Program for Intra- and Inter- Cellular Evaluation (Bio-SPICE) (Kumar and Feidler, 2003) provides a platform for the support and intercommunication of various modelling tools. Bio-SPICE is open source, with the core application being the 'Dashboard', that allows the downloading and integration of new tools with existing functionality. These tools, which are generally designed or adapted to become plug-in modules, can then be arranged as part of a design flow, where the Dashboard handles the dataflow between applications.

Bio-SPICE, and frameworks like it, answer a growing problem of functionality bloat, where many tools are developed independently but in fact have similar aims. Developers tend to focus their research so much that often there are no tools in existence to provide the exact combination of functionality they require. Bio-SPICE instead provides a central resource that enables users to download specific tools tailored to specialised areas. This thereby often reduces the quantity of software development required, as subsections of functionality may already exist, and therefore, the user only needs to fill in the gaps.

Markup Languages

Models represent the conceptualisation of a hypothesis, and analysis of model behaviour is used to ratify that hypothesis. Hence, one of the most important aspects of modelling is the ability for models to be re-created and analysed by other parties independently to test their performance. Early on within the field of computational

neuroscience, multiple different modelling environments were created, often to test specific hypotheses. This posed a problem for other academics who wished to incorporate previously published work with their own, as there existed no real standard for the transfer of models, other than the unalloyed formal mathematical language (Van Horn and Ball, 2008). Furthermore, models programmed for one application would rarely work with another.

The proposed solution is the use of a unified data communication method to store model implementations, such that although different simulation environments would function differently upon the model, the basic model structure would stay the same across modelling platforms. The first of these was the Systems Biology Mark-up Language (SBML)(Hucka et al., 2003), followed by CellML (Lloyd et al., 2004) and NeuroML (Goddard et al., 2001). Each of these mark-up languages provide a means for information transfer and model exchange; however, the different languages focus on different areas, providing extra information that relate to specific fields such as the distribution of ion channels and receptors within neuronal morphology.

Currently, the dominant simulation platforms (NEURON, GENESIS, SNNAP) all support one or more of these languages. Some mark-up languages, such as NeuroML, have tools that can convert files to and from other formats to increase compatibility and knowledge transfer.

Recently, the developers of NEURON have identified a similar problem which relates to understanding data that has been shared. They developed a tool to aid in the understanding of different model implementations through visualisation called ModelView (Hines et al., 2008). This tool is primarily designed to work with models published in ModelDB (Peterson et al., 1996), a database for the sharing of model implementations. But, through the interface NEURON has with mark-up languages, it should be possible to use this tool to visualise other models developed using different tools.

3.2.10 Summary

It is important to note that the majority of software modelling packages are aimed at practitioners with at least some understanding of computer science and mathematics.

This makes them unsuitable to be used directly by biologists, and therefore these platforms are typically used by computational neuroscientists in collaboration with experimental electrophysiologists. After a superficial study of a range of modelling platforms it is possible to come to several conclusions regarding the current environment for the development of new modelling tools.

The sheer number of currently available tools is staggering. Only the main platforms, and a subset of more focused platforms, have been described in this section. The reason for this number is linked to the typical resources available to the average computational neuroscientist, which are in turn dictated by funding cycles. Essentially there is only so much that a PhD student or post doctorate academic can do within their averagely allotted 3 year period. Hence, many projects are started and brought to the proof of concept stage before being dropped leaving only a website with a link to download the core program and a users manual (if lucky). In an almost Darwinian process, some of the modelling platforms find a user base that helps it become self sustaining, which leads to the solidification of the now major development platforms such as NEURON, GENESIS, SNNAP and COPASI. Additionally, as these platforms were initially competitive, they tended to absorb the features of each other leading to the majority of modern modelling frameworks having similar functionality. The obvious exception to this are frameworks that function in completely different ways such as COPASI and GENESIS (biochemical and electrophysiological respectively).

With the maturing of the field of computational neuroscience, there is less scope for new or more specialised platforms (AIM, SpikeNET) to compete with the older more established ones other than to integrate themselves with a larger framework such as Bio-SPICE. However, due advancements in graphical user interface (GUI) design, new tools, which tend to take advantage of the more modern GUI frameworks such as Qt, are starting to bring modelling practices outside of computational neuroscience, tapping new user bases. This is done by providing an interface that masks the technicalities of a programming environment where the user does not require any programming expertise to construct, simulate and evaluate models; SNNAP is a great example of this. Although in the last decade great leaps have occurred in incorporating modern interface design, there is still no framework that could be used by an experimental biologist that does not have programming experience and is not comfortable with unalloyed mathematics.

The utilisation of hardware acceleration is another factor that helps to separate development platforms. The majority of older platforms support the use of server clusters to accelerate simulation, partly because at the time of conception, this was simply the only way to perform the more complex simulations in a timely fashion (NEURON, GENESIS, Surf-Hippo). Interestingly the newer platforms have shown a tendency to forgo acceleration (SNNAP, COPASI), instead relying on the large increases in power that the newer standard desktop processor has attained, and focusing instead on improving the application interface. Only a few platforms are specifically attempting to take advantage of new forms of hardware accelerations, generally for the simulation of network models, but often the individual implementation is left to the user (NEST). Although there are probably tools available under support frameworks (Bio-SPICE) that enable the simulation of neural models on local massively parallel hardware such as FPGAs or GPUs, these types of hardware accelerations are not supported by the mainstream development environments.

In conclusion, there currently exists no modelling platform that caters specifically for experimental electrophysiologists who wish to incorporate modelling with laboratory experimentation. The specific criteria for such a platform would be:

- User friendly Computer Aided Design (CAD) interface for the construction of models without the need to touch upon unalloyed mathematics, algorithm theory, or requiring programming expertise.

- The ability to incorporate a wide range of hardware acceleration to meet time constraints imposed by laboratory experimentation; specifically local forms that can be user controlled such as Field Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs).

- The ability for models to interact with biological behaviours through parameter estimation. Working directly with laboratory equipment would be ideal.

3.3 Evolutionary Computation

Evolutionary algorithms were first discovered over 50 years ago with the implementation of the basic Genetic Algorithm (Holland, 1975). Since their inception,

and with the dramatic increase in readily available computational power, much research has been done to understand, improve, and implement these algorithms to solve optimisation problems in a massive range of applications (Back et al., 1997, Fogel, 2000).

Evolutionary algorithms (EAs) are an approach to optimisation inspired by natural phenomena such as natural selection and swarm behaviour (Eiben and Smith, 2008, Jong, 2002). While individual EAs can be inspired by widely differing natural systems, all evolutionary approaches share a commonality in the way that they function. EAs are population based, which is to say that they function by defining a population, and then evolving that population over many iterations. This iterative process has the same structure for all EAs, with the only difference being the core algorithm that creates new positions to test, based upon the current population.

Evolutionary algorithms are known to perform well in complex and unknown parameter spaces (Schwefel, 2000), and so are especially suited to the optimisation problems presented in this thesis. During model development the shape of the parameter space can change not only between iterations of a model's design process, but also during optimisation, if the model development is performed whilst recording from the biology (Back, 1998). However, this robustness to the adaptation environment comes at a cost; many more function evaluations are required in comparison to the more general optimisation approaches such as gradient decent. Function evaluations for parameter estimation involves minimising the difference between the behaviour of the model and that of the biology, and hence has a tendency to involve a simulation run followed by an analysis of that run.

Fortunately the majority of EAs are "embarrassingly parallel" tasks where each function evaluation from a single iteration of the population can be carried out simultaneously. As the cost of computational power has reduced, there has been an increasing interest in implementing EAs at least partially within or around various types of hardware acceleration such as Field Programmable Gate Arrays (FPGAs) (Shackleford et al., 2001) and Graphics Processing Units (GPUs) (Wong et al., 2005). A discussion of various forms of hardware acceleration is performed in Section 4.2.

A plethora of different EAs have been developed over the past 50 years. These are based upon on a range of biological principles, but they can generally be grouped into a relatively small set of archetypal mainstream approaches. The central approaches are: Genetic Algorithms (GAs), Particle Swarm Optimisers (PSOs), Estimation of Distribution Algorithms (EDAs), Ant Colony Optimisers (ACOs), and hybrid algorithms that combine various aspects of the above. The following sections introduce three of these evolutionary approaches with a brief review of the literature in readiness for Chapter 4, where a comparative study of these algorithms is performed to determine the correct algorithm for use in this work.

3.3.1 Genetic Algorithms

First discovered by (Holland, 1975), GAs are inspired by natural selection in nature, and are the birth place of the population based evolutionary algorithm paradigm. Because GAs were discovered such a long time ago, there is a staggering quantity of literature on the subject. Hence instead of pointing towards hundreds of different papers, the following text books are suggested. (Goldberg, 1989) presents an introduction and overview of GA theory, with (Goldberg, 2002) providing a more in-depth mathematical and theoretical analysis. (Goldberg and Sastry, 2009), when published, will cover much of the research performed in the last 8 years, including the study of very large problem spaces. All three of these books refer to a mathematical concept called the Building Block Hypothesis (BBH), which although commonly thought of as true, is still under much scrutiny by minority forces within the field (Burjorjee, 2008;), and so this section steers clear of this topic, instead focusing on practical implementation.

GAs represent possible solutions to a problem as chromosomes, which consist of many genes. With binary GAs (BGAs), each of these genes is a single bit (0 or 1), and many genes can be combined to makeup the full representation of the value of a single dimension within the problem space. This is different in real valued GAs (RGAs), where each gene represents a dimension of the problem. This simple difference between RGAs and BGAs becomes quite complex when the genetic operators are taken into account.

There is no real canonical GA due to the range of applications they have been integrated with of over the years since their discovery. However, a typical GAs tends to follow a general set of rules when creating new trial solutions: Selection; Combination; Mutation; and Insertion.

Parent Insertion and Selection

Two or more solutions are selected from the population. Depending on how the population is managed, the selection scheme may be biased towards fitter solutions. For example, if new solutions have a random chance of replacing previous solutions based upon their fitness score, there is already a selection pressure towards fitter solutions, and so the population can be randomly sampled. Similarly, if new solutions are combined with the existing population, sorted by fitness, and the population is then trimmed back to the actual population size by removing the worst solutions, a second selection pressure is not required during the selection phase. Ultimately, when designing a GA, the only requirement of the re-insertion and selection mechanisms is this pressure that favours fitter solutions in some way. The intensity of this pressure is responsible for the elitism shown by the algorithm and therefore greatly effects the algorithms trade-off between exploration and exploitation.

Cross-over

Once the parents are chosen, combination occurs. At this stage the algorithm has chosen two or more parents to combine to make new solutions. In the canonical form of recombination (note that this would be a BGA), the two solutions are selected; a crossover scheme is applied, then each gene within the chromosome has a chance of mutating. There are many types of crossover mechanism, though typically crossover is performed by placing the two chromosomes next to each other and then randomly selecting one or more points along its length. The section of genes between alternating points on both chromosomes is then exchanged. This is called N point crossover, where the two point version is depicted below in Figure 3-9.

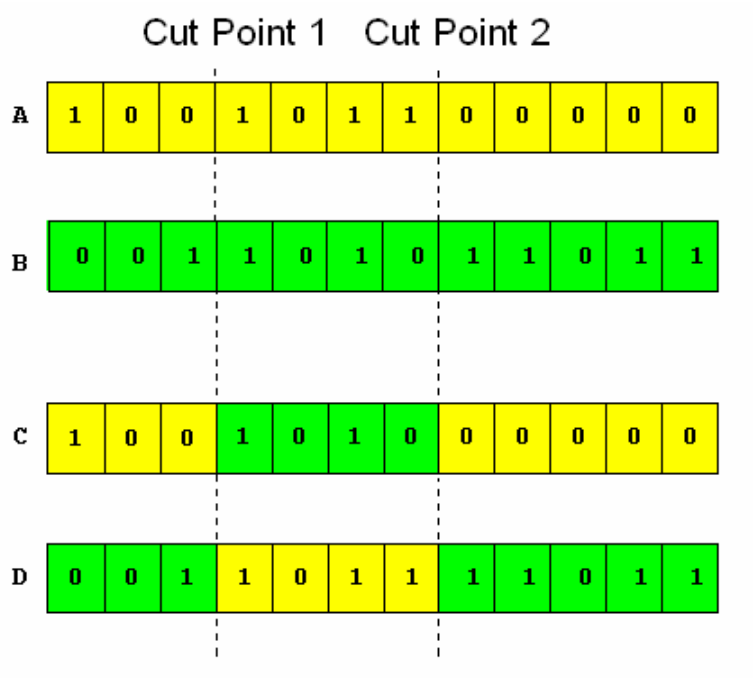


Figure 3-9 Two point cross-over. Two parents (A & B) are randomly selected using a selection scheme, and two positions within the chromosome are randomly chosen. Two new children (C & D) are generated with exactly the same genetic material as their parents except that the region between the two randomly selected points is swapped between them. As the GA gets closer to an optima, the parents will become more similar, and hence the effects of crossover will be reduced.

This can be extended to any number of points, with the only limitation being the number of sections relative to the number of genes within each chromosome. Additionally, multiple parents can be used, with different sections of the new chromosomes being selected from different parents.

Mutation

Mutation is an optional, but usually present operator, and is extremely important for the robustness of the algorithm. Because new solutions can only be a combination of the initial population, without mutation the algorithm cannot fully explore the parameter space. So for any particular gene position, if all the values within the initial population are ‘0’ then without mutation, all new solutions will be forced to have a ‘0’ in that position as well. This becomes even more important when considering RGAs where each gene represents a whole number, and not part of one. In this case, even within small parameter spaces, not including mutation severely limits the algorithms performance. Mutation within RGAs is doubly complicated, as the amount of mutation

becomes a factor, where in BGAs it is just bit twiddling. Again there are a wide range of options. Explorative algorithms may randomly select a new value across the entire range of values with a uniform random number generator, whilst more exploitative algorithms would use different distributions, such as a Gaussian centred upon its original position. They both have their strengths, with the first enabling the quick localisation of the algorithm to the areas of best fit whilst being robust to traps, and the second providing much greater convergence performance once the rough location of the global optima is found.

Simple Concept, Complex Behaviour

While genetic algorithms are mechanically simple, their behaviour is quite complex. To create a good GA the cause and effect of the various operators must be understood. At their core GAs function by optimising a population of partial solutions, where a partial solution is a subset of a full solution; for instance, two consecutive bits out of a ten bit long chromosome. Many partial solutions of genetic code are created when the population is first randomly initialised. The selection pressures, which are a combination of the cross-over, insertion and selection operators, all work to eliminate poorer solutions as the population evolves, thereby removing the partial solutions that these full solutions are comprised of. As partial solutions are discarded the pool of genetic material is reduced, such that more frequently, solutions will be selected for recombination that share one or more partial solutions. Therefore, all offspring from these parents will also include this genetic material. As the population converges on optima, the majority of the solutions will tend towards a similar configuration, with the size of the clustering within the parameter space determining how much of the genome is still in question. Mutation does not play a part in the steering of the genetic process; instead its purpose is to create new genetic material, and hence new partial solutions. In this way mutation is a guard against algorithmic stagnation, whereby the GA get trapped in local optima.

It was the study and understanding of partial solution interaction that lead researchers into the realm of EDAs, where the interaction between partial solutions are studied and directly considered in the combination operators. An example of this is the Expanded

Compact Genetic Algorithm (ECGA), which uses a simple modelling technique to relate clusters of genes with one another.

How partial solutions impact RGAs

The concepts of partial solutions relate to Binary GAs, where individual partial solutions can start as sub-sections of individual dimensions. For instance, a 3-bit, high-quality partial solution could represent the three most significant bits (MSBs) of one dimension of a solution. This would indicate that good solutions could be found within an area dictated by a range of values in this particular dimension that are bounded by the three bits. An example being the three MSBs of a 5 bit dimension valued 101 (20) would dictate that good solutions would be found between 10100 (20) and 10111 (23). This presents a problem for real valued GAs, as their genes do not represent individual bits of a dimension, but instead the entire dimension. To account for this change in problem structure, mechanisms are required to mimic the effects of evolving and enlarging building blocks. Without such a mechanism, the number of searchable locations within the parameter space is limited by the number of different values for each gene present in the initial population. One suggested mechanism replaces the standard crossover mechanic by instead randomly selecting new solutions from the edge between the two selected parents, thereby creating new genetic material. (Wright, 1991) found that a combination of both this method and normal crossover mechanic provided the greatest performance over a series of test problems.

3.3.2 Particle Swarm Optimisation

The PSO (Poli et al., 2007) was first conceptualised by (Kennedy and Eberhart, 1995) in a paper that focused on re-creating flocking and swarming behaviours seen in nature. The resulting understanding was then re-focused towards global optimisation problems. The last decade has seen the creation of a wealth of literature, applying, improving and understanding the mechanisms that enable this form of optimisation to function (Poli, 2007), to the extent that a central website is maintained with the aim of collating as much of this literature as possible, as well as defining a yearly standard (Particle Swarm Central, accessed 2009).

The basic PSO

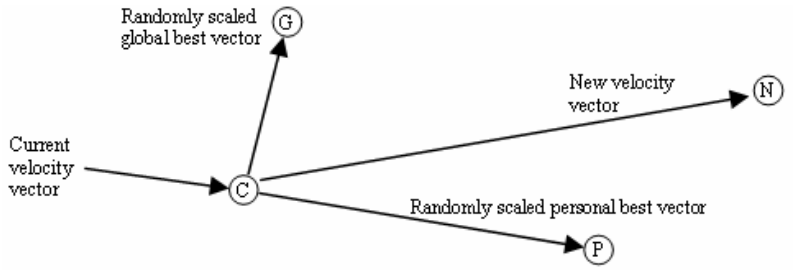


Figure 3-10 The behaviour of an individual agent within a PSO; C → Current Position; G → Global best position found so far by the population; P → Personal best position found by this agent; N → New position of the agent after the current iteration.

PSOs treat their working population as a collection of agents that travel through multi-dimensional space; hence they keep a record of their current location and velocity. The agents then determine their course through the parameter space by combining aspects of their own history with that of one or more members of the collective. In the canonical version of the PSO, each agent records the best location that it has found within the parameter space so far, and the fittest solution from these is recorded as the global best. Each agent's velocity is re-calculated at every population iteration by combining the agent's momentum (a portion of its previous velocity) with nostalgia forces that draw the agent towards its personal, and the global, best solutions by random magnitudes. These magnitudes, R_1 and R_2 in the formal representation, are independently generated, uniformly distributed, random numbers. This is shown in Figure 3-10.

Or more formally:

$$V_{i+1,j} = \omega V_{i,j} + \psi_1 R_1 \left(x_{G,j} - x_{i,j}\right) + \psi_2 R_2 \left(x_{P,j} - x_{i,j}\right)$$
$$x_{i+1,j} = x_{i,j} + V_{i+1,j}$$

Equation 3-7

Where $x_{i,j}$ = current value of the j[th] dimension; $x_{G,j}$ = global best value of the j[th] dimension found so far by the population; $x_{P,j}$ = personal best value of the j[th] dimension found so far by this agent; $V_{i,j}$ = component velocity of the agent for the j[th] dimension. ω = inertia weight; ψ_1, ψ_2 = acceleration coefficients; R_1, R_2 = uniform random numbers.

One of the advantages of the core algorithm is its simplicity, whereby it only has three control parameters apart from its population size. ω is known as the *inertia weight*, and is used as a constant dampening effect on the momentum, by determining how much of the previous iterations velocity is transferred over to the next iteration. This does not have to be < 1.0, and in some algorithms ω is modulated throughout the optimisation process with an initial value of > 1.0 that promotes instability, but greater exploration. ω is then reduced incrementally over the optimisation processes, producing more exploitative behaviour. ψ_1 and ψ_2 are known as the *acceleration coefficients*, and control, by way of there ratio to one another, how much the agent relies upon its own experiences, or that of the swarm, to determine its course through the parameter space. Interestingly, ψ_1 and ψ_2 can be seen to represent the mean stiffness of two springs between the current point and global and local best. In such a way it is possible to predict behaviour by depicting the system as an integration of Newton's second law, with higher values of ψ_1 and ψ_2 (> 2.0) producing over responsive and unstable[1] behaviours. When used in an unstable fashion the algorithm must be controlled. An initially popular method was to limit the velocity of each agent. However, care must be taken when selecting this limit as it directly impacts upon the algorithms explorative nature.

Analysis of Swarm Behaviour

Although simple from an algorithmic viewpoint, the PSO exhibits an interesting range of swarming behaviours that was not initially understood, even though the core algorithm was quickly integrated into a wide range of applications. Clerc and Kennedy (Clerc, 2002) delved into the underlying algorithm to explore swarming behaviour, finding that it could be depicted as five dimensional space. This analysis focused on the convergence behaviour of the PSO and explains how specific behaviours such as agent

[1] Many combinations of control parameters create an incremental increase in the excitability of the swarm, resulting in the population scattering towards infinity, or the edge of the parameter space if it is bounded.

explosion[2], stability, and rate of convergence, are effected by the control parameters. Using this knowledge they suggest a variant of the basic PSO that is structured in such a way as to more easily facilitate the selection of control parameters that produces stable, but still explorative, convergent swarm behaviours. This variant is formally described in Equation 3.8 and 3.9.

$$V_{i+1,j} = \mathrm{X}\left(V_{i,j} + \psi_1 R_1\left(x_{G,j} - x_{i,j}\right) + \psi_2 R_2\left(x_{P,j} - x_{i,j}\right)\right)$$
$$x_{i+1,j} = x_{i,j} + V_{i+1,j}$$

Equation 3-8

Where

$$\psi = \psi_1 + \psi_2 > 4$$
$$\mathrm{X} = \frac{2}{\psi - 2 + \sqrt{\psi^2 - 4\psi}}$$

Equation 3-9

Swarm Topology

While Clerc and Kennedy focused upon the behaviour of individual agents, other research has concentrated on the dissemination of knowledge throughout the swarm. This is known as population topology (Mendes, 2004), and is often seen as a social network. It is important to note that without the interaction with the rest of the swarm, the algorithm performs abysmally, and so it is obvious that agent interaction is vital to a PSO's success. Initial exploration into this area of research focuses around fixed topologies, and is concerned with the optimal size, structure and distribution of sub-sets of agents, or local neighbourhoods. Such topology can been seen when an agent is influenced by its own experience, as well as the experience of K pre-defined neighbours. In its simplest form this topology can be represented as a ring lattice where each agent is connected to two others (Eberhart and Kennedy, 1995).

[2] Explosion behaviour is caused by over responsiveness, where the population is rapidly drawn to an optima but overshoots due to momentum carried over from the previous step.

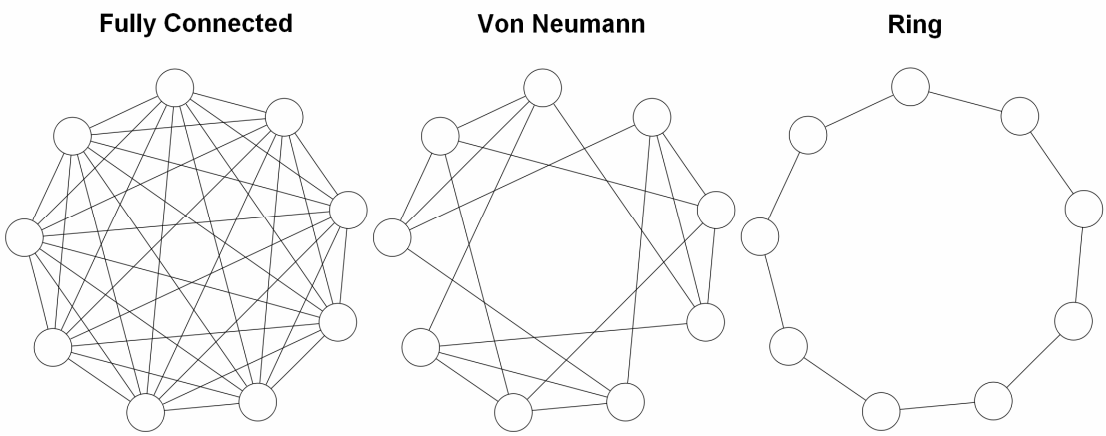


Figure 3-11 Swarm topologies

This lattice allows the swarm to divide itself over several optimal areas simultaneously although over time the area with the best fitness will "rain in" the rest of the population. The topology can, of course, be increase beyond K = 2, with the canonical version of the PSO being fully connected, where K is the size of the population – 1. Other topologies such as the Fully Informed Particle Swarm (FIPS) suggest different velocity update rules alongside their proposed topology, which instead takes into account all members of a neighbourhood, not just the best (Mendes, 2003). Interestingly, investigations into static topologies have shown that the past experience of an individual may not be required for optimisation to function (Mendes, 2003). Topologies can also be dynamic, where the algorithm seeks to optimise the social network during optimisation. A simple example would be to slowly, over the course of the optimisation, morph a K=2 topology into a fully connected one. Sparsely connected topologies are more explorative, whilst fully connected ones converge much faster. Thus by changing the topology during the optimisation process it is hoped that the resultant algorithm will gain the advantages of both at the required time (Suganthan, 1999). Other dynamic topologies include; distribution of subpopulations with random connections that are re-randomised at regular intervals (Liang and Suganthan, 2005); dynamic hierarchy, where fitter agents rise in the hierarchy tree and therefore influence the less fit agents (Janson and Middendorf, 2005).

Effects of Population Size

In the majority of previous studies, the population size of the swarm is kept constant across different trials. This is done in an effort to maintain fairness during comparisons of different swarming approaches. Even so, there have been some studies that take population size into account, showing that, at least for a small range of problems[3], the canonical PSO provides a marginal increase in performance with larger population sizes (Shi and Eberhart, 1999, Trelea, 2003), especially when dealing with problems with higher dimensionality. It should be noted, though, that no studies have increased the population sizes to the levels that are enforced by the GPU architecture described in this thesis. However, a more thorough examination of the PSO algorithm, which involved optimising the population size and update order with a genetic algorithm (GA), has shown that there is an optimal population size for each problem (Diosan and Oltean, 2007). Additionally, because this study does not update the whole population at the same time, instead selecting a sub-population that may consist of multiple updates for individual agents, the significance of each agent when compared to their fitness can be explored. It was found that the PSO gives priority to the fittest agents over less fit ones, which are often updated many more times, suggesting that even when using the optimal population size, compressing the population to a smaller size will not heavily impact the performance of the algorithm.

Although the majority of initial studies into PSOs used a simultaneous update scheme, it has been shown that PSOs perform best when the population is informed of the result of each agents movement before the next member of the population updates (Carlisle and Dozier, 2001). This suggests that PSOs are best implemented as a series task and not a parallel one, requiring relatively small population sizes for problems of high dimensionality, compared to more traditional approaches such as GAs. This is not to say that they are less computationally intensive. Indeed, PSOs tend to require orders of

[3] The usual test fitness landscapes are the Sphere, Rosenbrock, Rastrigin, Griewank, and Schafer functions.

magnitude more iterations (depending on control parameters) of the population to converge to the global optima.

It is important to note that for this work, the interest lays the total number of function evaluations required to meet a goal. Thus as the population size increases the ideal situation would be to see an overall improvement in the total required number of iterations (population size multiplied by the number of population iterations plus one). The results from (Trelea, 2003) show that this improvement is indeed seen but only when increasing from small population sizes (15 → 30 in most cases), and only for the more complex fitness landscapes. However, there is sufficient evidence to suggest that as the population size increased (30 → 60) there was a significant increase in the number of function evaluations, essentially showing diminishing returns for increase population size.

3.3.3 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) (Larranga and Lozano, 2002) describe a more general approach to optimisation that is a subset of evolutionary algorithms. Much like traditional Genetic Algorithms (GA), they maintain a population of solutions that parents are selected from in order to facilitate the creation of offspring. The difference however, is how these selection and combination routines function.

The basic EDA

In an EDA, a subset of the total population is taken, which usually consists of the fittest solutions, and they are all combined in the form of a probabilistic model that describes their distribution. Offspring are then generated probabilistically from this model, and evaluated in the usual manner. As an increasing number of fitter solutions are created, the distribution will converge upon optima, as the average fitness required to remain within the population increases. The two variables to this process are the selection scheme and how the model is constructed, with model construction methods dominating the literature. The main advantage of estimation of distribution algorithms is their ability to directly model interactions between input parameters, which enables a more efficient search by allowing the algorithm to isolate and explore multiple optima simultaneously. This become increasingly useful with the increase in dimensionality,

where traditional combination operators such as crossover, can cause the algorithm to repeatedly search in unrelated areas when the partial solutions are split, see Figure 3-12

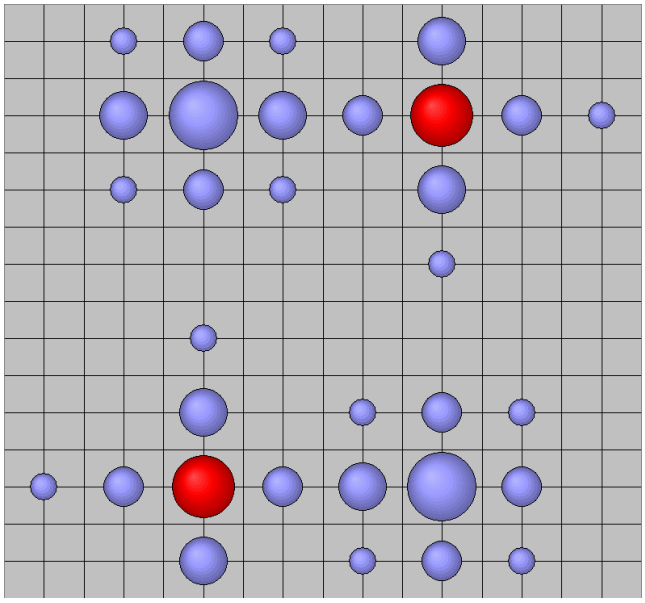


Figure 3-12 Illustration of the probability of producing offspring (larger circles represent a greater chance) by using cross-over on two parents (red) selected from a bi-modal distribution. The resulting distribution of possible children includes many in areas away from the two modes.

EDAs and the Building Block Hypothesis

EDAs are a natural progression from GAs, taking the concept of partial solution interaction further. The analysis of partial solution interaction is codified within either a Bayesian or similar parametric structure, or using some form of mixture model, be that analytical, neural, or otherwise inspired. This method is therefore immune to the splitting of building blocks by the effects of cross-over, as it does not use such operators; however, there are other issues that a practitioner should be aware of. Because new solutions are generated from a statistical model based on a sub-set of previous solutions, it is important to make sure that the sub-set properly represents the underlying distribution, especially the less salient parts of the problem. Without accounting for the less salient areas of the problem space, the algorithm may be directed towards, and get stuck in, local optima (Chuang and Chen, 2008).

Modelling Methods

Some of the initial work with EDAs involved the construction of Bayesian trees to model the interactions between control parameters (Larranaga et al., 2000). In the case where a solution is represented in a binary fashion, each bit would be modelled as its probability of being a 1 or a 0, which if the problem is multi-model, will depend on the value of other bits in the solution. As the algorithm processes, the complexity of the Bayesian model tends to simplify as local optima are discarded to hopefully resolve into one mode, and therefore no dependency amongst control parameters.

The simplest approach to model construction is to use a kernel based mixture model. This involves the placement of probability density function (PDF) kernels, which are usually uniformly Gaussian in nature, centred on the fittest solutions found within the parameter space. New solutions are then drawn randomly from the resulting mixture model, which is the sum of all the PDF kernels. This approach is computationally simple, but can suffer from a lack of convergence when the kernels are large in comparison to the distribution of fit solutions. This generally occurs during the final stages of adaptation when the algorithm nears an optima. To account for this behaviour, each kernel's covariance matrix can be modified to better fit the local distribution of fit solutions. An alternative but similar method is to use expectation maximisation (EM) to fit a set of Gaussians to distribution of fit solutions. This can produce an extremely accurate model of the fitness landscape, but this method is usually not used due to the computational intensive nature of the EM algorithm.

Neural networks (Martí et al., 2008) have also been used to model parameter distributions as they are known to be able to capture complex relationships within high dimensional spaces . The application of neural networks is especially suited to true multi-objective optimisation tasks (Section 3.4.2), where the network must model the relationships of not only inter-parameter interactions, but also between these interactions and the resulting multidimensional solution space.

There is a wide range of modelling methods used in EDAs, with the usual trade-off of computational complexity vs. accuracy in representing parameter interactions. EDAs are a useful tool in tackling multi-model parameter spaces. However, care must be

taken, as they tend to be aggressive in their optimisation, quickly focusing on optima, with a greatly reduced chance of searching in areas that are not related to the fit solutions set. This means that the state of the initial population is even more important than in the GA or PSO, because in there basic form, there is no mechanism to help EDAs navigate out of local optima once it is trapped. However, this behaviour makes EDAs more efficient at solving simpler optimisation problems.

3.3.4 Summary

This section has introduced three archetypal evolutionary algorithms (EAs), each of which is used as part of a comparative study in chapter 4; genetic algorithms (GAs), particle swarm optimisers (PSOs), and estimation of distribution algorithms (EDAs). Each class of algorithm functions differently, being inspired by different natural process or behaviour. The sub-fields relating to each of these algorithms show that there are a wide range of successful algorithmic implementations for each class of EA, but also that each algorithm is better suited towards different types of optimisation problem, or problem environments.

It is because of this realisation that hybrid algorithms were implemented. There exist a very large number of publications that report good performance from hybrid algorithms in comparison to various archetypal approaches (Grosan and Abraham, 2007). A popular combination is combining an EA with a gradient descent or similar approach. This suggests that hybrid algorithms are superior, which in many cases is they may be. However, the writer suggests that the reason for the large number of publications on hybrid algorithms is twofold.

1. Incremental progress, coupled with a need to publish.

2. The focus of hybrid algorithms on limited and specific tasks.

The “no free lunch theorem” states that all algorithms can be considered to provide equal performance over the set of all problems. So therefore, these more focused algorithms will perform more poorly on other problems when compared to the more generic archetypical algorithms. However, the majority of these hybrid papers are conference papers, and therefore do not have the space to show performance over a

wide range of problem types, so it is easy to create situations where the trial hybrid algorithm will outperform the more general archetypical algorithm. Although this may appear to be a rather disparaging view of the literature, it does highlight a key point about the practical implementation of EAs. Generally it is beneficial to create a hybrid algorithm that specialises in solving a specific class of problem than opting to use the archetypal method.

3.4 Fitness Evaluation

Fitness evaluation is the most important part of any practical implementation of an evolutionary algorithm, as it is the way in which the EA gains information about the form of the function for which solutions are sought. Without a suitable fitness function the optimisation will perform at best badly, or at worst not at all. The following sections discuss two different fitness evaluation techniques, single- and multi-objective. Ultimately, the work within this thesis requires a multi-objective approach as the behaviour of neural firing patterns is typically analysed by several methods at once. However, all the issues regarding single-objective optimisation are equally valid when performing multi-objective.

3.4.1 Single-objective

Single-objective fitness evaluation defines the performance of a trial solution by a single value that is calculated from the trial solution's position within the parameter space. Although conceptually simple, an understanding of single objective fitness function is useful when dealing with the more complex multi-objective approaches.

For a fitness function to be viable, it must present a gradient within the parameter space for the evolutionary algorithm to exploit. Without such a gradient, it is impossible for the algorithm to determine the difference in fitness between solutions unless the optimum has already been found randomly, in which case there is no need to optimise. For this type of problem, a random search is just as efficient as any other search heuristic, as no information can be gained by probing the parameter space. This will rarely be the case for the proposed model analysis method as all evaluations represent percentage deviations from target behaviours, and are hence real numbers. However, if the function is made too specific, by re-enforcing success and punishing failure

excessively, then the steepness of the resulting gradient may cause the heuristic to become analogous to a random search.

Figure 3-13 shows three fitness functions, where only the second is viable. The first is a broad distribution, and while an EA will easily find a broadly optimal area, the true optimum is difficult to distinguish from its surroundings. The third represents a search space where the optimum is unlikely to be found except by a random search, as the only information that can be gained from the parameter space is whether the optimal solution has already been found. The distribution of the second fitness function can both guide an algorithm to the broadly optimal area, and also highlight a much more focused representation of the optimal solution.

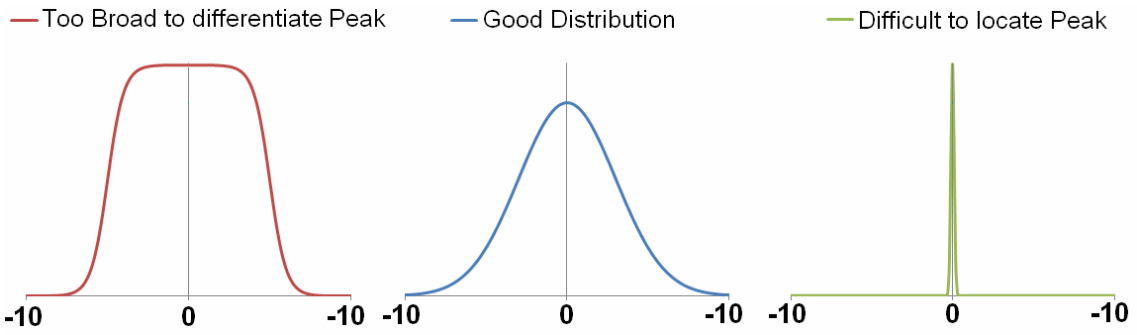


Figure 3-13 A range of fitness functions. The optimum is at 0 in all three functions, but it is difficult for a search heuristic to find in the red and green distributions because in both cases, there is not a consistent selection pressure across their length.

3.4.2 Multi-objective

The majority of real optimisation problems are multi-objective in nature, as many aspects of a problem are quantifiable and must be optimised simultaneously. This is also the case for the proposed optimisation problem as each solution produces a simulation that is analysed in a manner similar to the electrophysiological practises discussed in section 2.2.3. Therefore each solution will have a measure of fitness determining how close it matches various aspects of a cells firing pattern, such as mode, mean and covariance of the ISI.

Combinatorial Fitness

Traditional methods of multi-objective optimisation seek to reduce the multi-objective problem to a single objective by placing weights upon the performance of each objective, which are then combined to produce a single, easily-comparable value (Jaszkiewicz, 2002). This combinatorial method is used in this thesis, as it allows a user to signify the relative importance of each measure. In this way, combinatorial fitness analysis allows the user to input domain specific knowledge about a biological system, and is one of the interfaces used to incorporate modelling in an intuitive manner.

The disadvantage of this form of fitness analysis is that it relies upon a certain level of user training to understand the relevance of the weighting system, and how it impacts upon the search algorithm. Without this knowledge, users tend not to use the functionality, and hence are left with an un-optimised search heuristic. An alternative solution is one where a minimal amount of fitness function adjustment is required. This falls into the realm of a true multi-objective solution, as discussed below.

Pareto Space

A Pareto space (Deb, 2001) is a representation of the found solutions, from the point of view of the various measures of fitness, not the parameters that control the outcome of a fitness function. To understand Pareto space, it is first important to understand Pareto dominance. This is now explained by a brief example:

When searching for a car to buy, one can define the strengths of any particular car in terms of its cost and comfort, looking to minimise cost and maximise comfort. If one looks at a subset of the market one can find that there is a large range of cars with varying degrees of cost and comfort, from sports cars with a high cost and comfort to small commuter vehicles with a low cost and comfort. When comparing two vehicles, there are three outcomes that can be defined:

1. The first car is cheaper and more comfortable than the second. The first car is said to *Dominate* the second car.

2. The first car is more expensive and less comfortable than the second. The first car is said to be *Dominated By* the second car.

3. The first car is cheaper but less comfortable or more expensive but provides greater comfort than the second car. In these two cases the two cars are *Incomparable*.

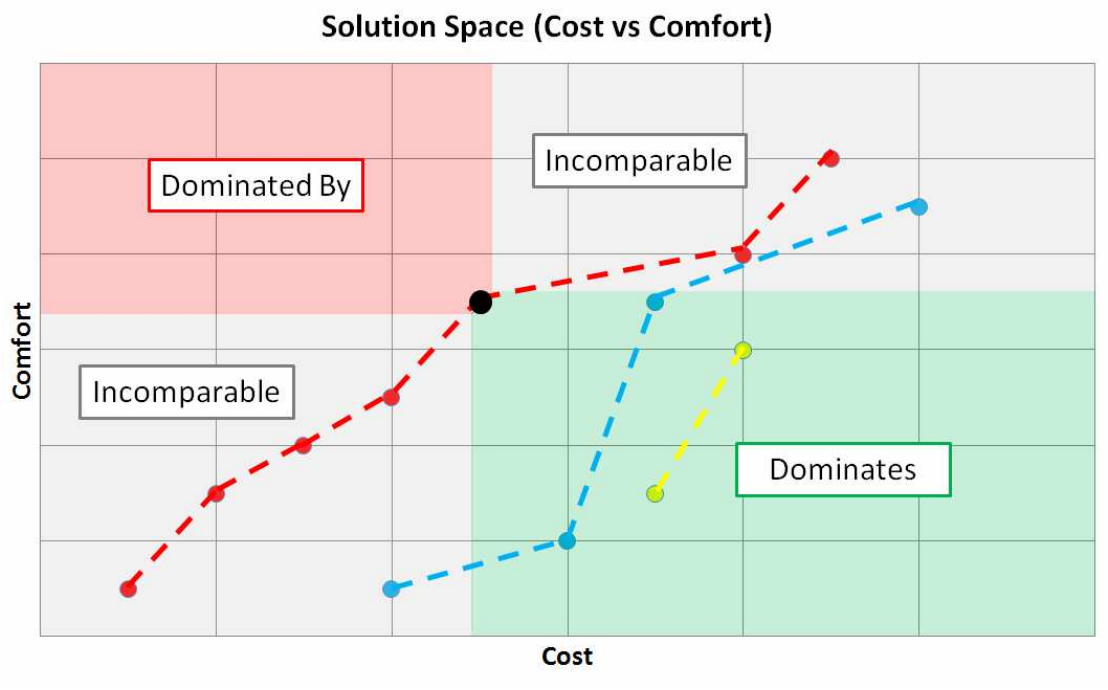


Figure 3-14 The solution space of a set of cars, showing their cost vs. comfort. The red dotted line designates the Pareto optimal front. The blue and yellow dotted lines show the second and third Pareto fronts respectively. The black dot, along with the highlighted areas of the graph, provides an example of how solutions within the space are compared. This particular point is on the Pareto optimal front and is therefore not dominated by any other solution.

Solutions are sorted by how many times they are dominated, with the set of non-dominated solutions making up the *Pareto Optimal Front*. From a multi-objective point of view, all these solutions are optimal, as each provides an alternate optimal configuration of the fitness function.

In comparison to combinatorial analysis, Pareto-based methods provide the user with a simpler interface, as no weight adjustment is required. They can also offer more information than the combinatorial approach, which effectively directs the search to a single point within the solution space. The flexibility offered by a Pareto approach allows post-optimisation analysis to be performed on the non-dominated solution set. This allows better identification of the links between control parameters and resulting

behaviours, as a greater spread of optimal solutions are available. Unfortunately, Pareto analysis has practical limitations, due to the exponential increase in algorithmic complexity with the increase in the Pareto space dimensionality (Fonseca et al., 2005, Khare et al., 2003). This means that, for practical applications, it is difficult to go beyond a handful of dimensions and their referred measures of fitness.

The implementation of Pareto fitness evaluation techniques alongside evolutionary algorithms, known as Evolutionary Multi-objective Optimisation (EMO) (Deb, 2001), has been explored for over 20 years. It was quickly realised that EAs are naturally suited to searching Pareto spaces, as their population-based approach is able to search and represent the entire Pareto optimal front simultaneously (Fonsecay and Flemingz, 1993). Additionally, their generic nature, which is a product of their non-assumption of the underlying fitness and parameter spaces, provides a platform that can be used as a foundation for any multi-objective task. A range of algorithms have been developed to combat the commonly identified, but usually conflicting issues surrounding multi-objective optimisation. These are:

1. Searching for a hyper-plane instead of a point results in a lack of *elitism*.

2. Elitism promotes clustering behaviour, which is counter to creating a spread of results that can properly represent the hyper-plane of the Pareto optimal front.

The most commonly accepted and proven EMO algorithms are NSGA-II (Deb, 2002) and SPEA2 (Zitzler, 2001, Kim, 2004), which have been used consistently as benchmarks. Both algorithms include mechanisms that seek to overcome the previously mentioned issues. Lack of elitism is tackled by conserving the fittest points that make up the Pareto optimal set between iterations, enabling them to be selected during recombination in addition to the normal population. This is done to reinforce the selection pressure towards the true Pareto optimal front. In both algorithms, clustering is dispersed by providing a metric for crowding. NSGA-II calculates the distance between a target point and its adjacent points, with a large value being superior to a smaller one. Selection is determined by Pareto dominance first, and then by the crowding metric. SPEA2 works differently, instead combining its crowding metric with the dominance count. This is done by counting the number of times a solution is

dominated, and combining this with the number of solutions it in-turn dominates. Selection then treats a lower dominance count as desirable. In this way, isolated but non-dominated solutions are more likely to be selected over solutions that are crowded, and hence dominate many other trial solutions.

More recently, researchers have focused on the core algorithm of multi-objective optimisation, rather than the fitness analysis and population management aspects. The resulting algorithms tend towards an *Estimation of Distribution* approach, usually utilizing the selection schemes from NSGA-II and SPEA2. Estimation of Distribution Algorithms (EDAs) are especially suited to multi-objective problems, as they create a Probability Density Function (PDF) of the fittest solutions found, in this case the Pareto optimal front, and then select new test points from this distribution. In this way, they can characterise inter-parameter dependencies, providing a more efficient framework than traditional operators such as crossover and mutation, which do not take this into account. A number of algorithms that utilise an EDA for the recombination step of the EMO are summarised below. The algorithms are categorised by the way they construct their probabilistic models.

- mrBOA Bayesian Model (Ahn, 2004, Zitzler et al., 2001).
- MIDEA PDF created through a combination of different probability distributions (Bosman and Thierens, 2005).
- MOPED PDF created by placing a probability distribution on each point within the selected subset of fittest points (Costa et al., 2003).
- RM-MEDA Principal component analysis (Zhang et al., 2008).
- MONEDA Growing gas neural network (Martí et al., 2008).

3.5 Summary

This chapter has provided an introduction to modelling within neuroscience. It has introduced several archetypal neural models: integrate and fire (IF); Hodgkin and Huxley; Fitzhugh Nagumo; Izhikevich; Spike Response Models (SRM), of which excitability-based models are a sub category. It has discussed how modelling is used

alongside electrophysiology, specifically in relation to practitioners with little or no computer science expertise, as well as how models are evaluated.

Typically, experimental electrophysiologists are uncomfortable with models whose description and operation are mathematically challenging, or are presented in unalloyed mathematical language. Thus, it is considered important to use models that are

(a) directly linked to biological phenomena that they can visualise, recognise and understand, and

(b) presented, as much as possible, with biological terminology.

This research focuses on excitability-based models because they represent a balance between the high complexities but realistic behaviour of conductance-based models, such as Hodgkin-Huxley and Fitzhgh-Nagumo, and the computationally efficient but functionally and behaviourally simplistic integrate-and-fire models. Additionally, excitability-based models have the advantage that their control parameters directly represent quantifiable neural sub-behaviours in the form of each cells post action-potential membrane excitability pattern. This pattern is much more intuitive than the series of simultaneous equations that form the core of the conductance based models, and is therefore more suitable to be used by laboratory-based electrophysiologists.

The second section of this chapter provided an overview of the range of modelling platforms currently available. Over the last three decades, over thirty different neural modelling and/or simulation platforms have been developed. Of these, only a small subset has managed to survive, by growing a self sustaining user base and constantly providing functionality updates. This section discussed how the tools for model development had changed over the past 20 years, highlighting the advantages and disadvantages of both new and more established platforms. Hardware acceleration, usability and the ability to perform parameter estimation are discussed, resulting in the conclusion that there currently exists no development environment that is tailored for laboratory-based experimentalists without programming experience, who are uncomfortable with formal unalloyed mathematical algorithms.

The third section of this chapter was a review of the literature on three broad archetypes of evolutionary algorithms: Genetic Algorithms (GAs), Particle Swarm Optimisers (PSOs), and Estimation of Distribution Algorithms (EDAs). These algorithms are further discussed in Chapter 4, where they are implemented as part of a comparative study to determine their suitability for parameter estimation of neural models.

The chapter concludes with a brief section introducing the various broad types of fitness evaluation. Because of the range of analysis methods currently used by electrophysiologists, any fitness evaluation method used to steer parameter estimation is likely to be multi-objective. The section introduces two methods of multi-objective fitness evaluation, combinatorial and Pareto non-dominance, both of which have their advantages. Combinatorial methods provide an interface for users to inject their own domain specific knowledge. This is done through the modification of fitness weights that alter the importance of each individual method, and hence can be used to simplify the adaptation problem. Pareto non-dominance methods do not require any user knowledge, and are hence more suitable for users that have difficulty in understanding the relevance of fitness weights. Additionally, Pareto non-dominance can be used to explore the correlations between control parameters and behaviours, by analysing the outliers of the solutions space where single measures of fitness dominate. Unfortunately, Pareto non-dominance methods are much more computationally intensive, becoming progressively more so with an increase in the number of independent measures of fitness.

Chapter 4 Development of a new rapid model evaluation method

The objective of this work was to contribute to the rapid iterative design of neural models by electrophysiologists within a laboratory, through the development of a rapid model analysis technique. The aim is to greatly accelerate the process of hypothesis creation and evaluation, by providing a platform that allows electrophysiologists to develop neural models within a laboratory environment, using biological terms and concepts. It is expected that the eventual toolset will package the required computer science knowledge within an expert system, resulting in the removal of the current delays that result from slow communication between modeller and experimentalist. Additionally, errors introduced through miscommunication of biological or modelling terms and concepts will be eliminated, as the toolset will mask the technical aspects of modelling, instead providing the experimentalist with an intuitive interface to that knowledge.

This chapter describes the development of an analysis method which combines the power of graphics processing units (GPUs) with the flexibility and robustness of evolutionary algorithms (EAs), to create a rapid model analysis method that focuses on a model's reproducibility. Other model aspects are examined in Chapter 5, where a model's robustness and its ability to extrapolate are discussed.

In order to facilitate rapid model development, where the target parameter space may change with the model structure from iteration to iteration, a method of analysis that can either predict parameter interactions and impose limits for each batch of simulations, or that does not require understanding of the shape of the parameter space, is required. The ability of a model to reproduce experimental behaviour is traditionally evaluated by one of two methods.

The first method involves comparing the behaviour of a set of recorded neurons against a distribution of behaviours created by running the model with randomly distributed control parameters. This is ideal if the set of solutions is suitably sized to allow a full exploration of the range of behaviours a model can produce (Prinz et al., 2004, Prinz, 2008). More often, however, trial models include large numbers of control parameters,

and hence a thorough exploration of the parameter space is a computationally intractable problem. As an alternative, if the parameter interaction of a model is known, a smaller but more targeted distribution of parameters can be used in order to minimise the number of simulations that produce biologically implausible behaviour. However, if this method is used with a poorly understood model, only very broad limits can be applied to the control parameters, which can result in the control parameter distribution not matching the equivalent biological distribution of values. This lack of prior information becomes increasingly problematic with the increase in dimensionality of the parameter space (Powell, 2007). For many neural models there exists a large amount of inter-parameter interaction which, when combined with an increase in dimensionality, causes large subsections of the parameter space to produce at best unrealistic, and at worst, un-analysable, behaviours.

The second method involves adapting the control parameters to produce behaviour similar to that of the target biology. With a suitably robust algorithm and a "correct" model, it is possible to build up a distribution of control parameters that produces behaviours within the same range as those collected from the biology. This method does, however, have its problems, mainly caused by computational complexity and over-fitting. Biological models usually involve large complex parameter spaces, where the complexity can increase dramatically with the model's ability to match underlying biological subsystems. Because of this, any algorithm that is used to adapt the model needs to be particularly robust against multi-modality, in order to cope with the inter-parameter interaction that is the staple of the feedback mechanisms seen with neural models. Unfortunately, increased robustness results in increased computational cost, which has been the main limiting factor of this type of model analysis.

This work adopts the latter approach. Instead of exploring the distribution of behaviours produced from randomly-selected control parameters, the proposed model evaluation method involves adapting the parameter values to test whether the model can produce behaviour similar to that of the target neuron. If a model can closely match the behaviour observed in many different samples from a class of cell, then that model may be considered to have good *descriptive adequacy*. Large deviations from the cell behaviour, if still present after a thorough search, can be attributed to flaws in the model structure, and hence also in our understanding of the internal function of the neuron.

These flaws can be identified by where the deviations between model and target behaviour occur, and future iterations can attempt to remove them through modifications of the model structure. It is important to note that, as stated at the beginning of Section 3.1, goodness of fit (GoF) cannot itself be used to fully evaluate a model. This is because GoF presumes a noiseless parameter space, not taking into account any measure of model complexity, which if excessive, can cause over-fitting. However, the concept of rapid iterative design allows for incremental increases in complexity as a model's ability to fit experimental data is tested. Therefore, a top down model construction method should be used, where the model is initially extremely simple, and additional complexity is added iteratively as it is needed.

The concept of parameter fitting is not new, and has been used successfully within neuroscience for many years (Geit et al., 2008, Schutter and Cannon, 2000, Segev et al., 1989). It is an intuitive approach, as the experimentalist does not need to know how the model structure and the various measures of fitness shape the parameter space, thus removing the need for the user to have mathematical expertise. The adaptation algorithm, if suitably reliable, will quickly isolate and search within analysable areas. Furthermore, if the adaptation can be performed quickly, then there exists the possibility to evaluate a model within the laboratory, alongside the biological experiment.

Rapid model evaluation offers many benefits by providing a framework with which other methods can be performed in a timely fashion. Some of these methods, such as cell classification and stimulus causality, are discussed under the future work section in the conclusions of this thesis. Another method, the analysis of model parameter distribution, is implemented in Chapter 5, where the distribution of model control parameters, which are the result of adaptations to sets of neurons, are compared in order to find a cause of a range of observed behaviours recorded from VMH neurons. This method is particularly useful when combined with a model that consists of parameters that tied directly to measurable cell characteristics such as the magnitude and decay of HAP, DAP and AHP, as the result of such an analysis creates hypotheses that can then be tested experimentally. Ultimately, the final test of any model will always be through the creation and validation of predictions. For neuroscience this generally involves returning to the laboratory to perform experimentation based upon information gained from the model, and this is another area where miscommunication can occur between

modeller and experimentalist. By enabling the experimentalist to perform the modelling within their laboratory, typical experimental procedures can be carried out in parallel, where model predictions are immediately tested. It is important to note that, all the methods mentioned above can, and are in this thesis, performed on pre-recorded data, but the real value of rapid evaluation comes from providing feedback during experimentation. Electrophysiology, like most laboratory experimentation, is difficult and unpredictable, and so all the functionality discussed in this work is designed to provide guidance during laboratory experimentation, in order to make the most of any data that are successfully acquired.

As described in Section 3.2, the majority of the large modelling and simulation platforms include functionality that enables parameter estimation. In NEURON this is supplied by the "Multiple Run Fitter" (Holmes et al., 2006), which uses the principal axis search method (PRAXIS). However, as with the other environments, there are several problems with this tool and its environment.

The Multiple Run Fitter was designed to be used with biological data, but not within a laboratory, and not to a specific time frame, and although it can take advantage of hardware acceleration, only one type – server clusters – is currently supported. This is typical of the all the main stream modelling platforms which have yet to adopt hardware accelerations methods beyond server clusters. Various forms of hardware acceleration are discussed in more detail in Section 4.2. Additionally, the PRAXIS algorithm, being a gradient decent method, has a greatly reduced performance when working in more complex parameter spaces (higher dimensions and multiple modes), and so some practitioners resort to hybridisation with more robust methods (Gurkiewicz and Korngreen, 2007). Because of the open source modular nature of nearly all the modelling environment (EONS is the exception), many include alternative adaptation methods for parameter estimation such as evolutionary approaches (GENESIS, COPASI), although gradient decent is still popular.

This thesis explores the use of evolutionary algorithms (EAs), see Section 3.3, which have been shown to be robust in difficult and/or unknown search environments (Schwefel, 2000). Unfortunately, EAs are computationally expensive due to the large number of fitness evaluations that are required, which in this work involves a simulation

followed by an analysis of that simulation. However, EAs are also highly parallelisable, and therefore they can be made to perform within the time limitation of in-lab experimentation by using hardware acceleration.

The remainder of this chapter is divided into four main sections. The first discusses the requirements for the rapid model evaluation method developed in this thesis. Section 4.2 briefly examines four broad forms of hardware acceleration, which is a requirement of the proposed methodology, before selecting Graphics Processing Units (GPUs) as the chosen solution. Section 4.3 briefly discusses the general requirements of the measures of fitness used to evaluate how close an instance of a model is to experimental data. This section specifically focuses on interpretability, as ultimately both adaptation algorithm and experimentalist will use this information to improve the model. Section 4.4 presents a comparative algorithmic analysis that was carried out to determine the type of EA that provides the greatest performance at optimising neural models when accelerated, but also constrained, by GPU architecture.

4.1 Requirements

Because the results of the adaptation will be used as a guide for further experimentation and model improvement, the adaptation algorithm is required to be robust. If this is not the case, then potentially good model configurations may be discarded, and sub-optimal configurations selected as a result of the algorithm being trapped by local optima. The adaptation process must also be able to function without significant prior information that would normally be supplied by the practitioner. This is because the aim is to provide a tool that does not require mathematical expertise to use. In fact, the only interaction allowed between the practitioner and the adaptation process is through the makeup of the post simulation run statistical analysis, which forms the multi-objective fitness function (Chapter 3.4.2). Additionally, to fully integrate modelling with the experimental process, there is a requirement for this method to function in the laboratory alongside the biology whilst it is being recorded. However, this presents an additional problem in the form of drifting target behaviour, which will result in a dynamic parameter space.

Evolutionary algorithms (Chapter 3.3) were chosen because they have been shown to perform well in complex multimodal multivariate parameter spaces (Schwefel, 2000), and do not rely upon prior knowledge, only guidance in the form of the fitness function. Additionally, they have been shown to be able to track optima through a dynamic parameter space (Back, 1998). However, there are several problems to be overcome when using evolutionary algorithms.

Evolutionary algorithms are computationally intensive, especially in an application such as this that can require long simulation runs to provide fitness calculations for certain types of longer term behaviour. Additionally, this intensive computation must run within the time limitations set by the experimental procedures in the laboratory. The typical length of an *in-vivo* laboratory session used to collect the type of data analysed within this thesis is approximately six hours; however, the majority of this time is spent finding the right cell, not recording it. Cells can be unstable, and even when an acceptable cell is found, there is a good chance that it will be lost again, either through death or changing behaviour. This means that analysis must be as quick as possible, a good benchmark being around five minutes, although faster speeds would be advantageous. If the process takes longer than this then the user could be tempted to start another task whilst waiting, diverting their attention to the detriment of the modelling process.

It is important to note that analysis can only begin once a significant amount of data has been recorded. The amount that can be deemed significant is dependent on the behaviours studied; for example, vasopressin cells require between 10 and 20 minutes of data before the adaptation can begin, whereas only 3-5 minutes of data are required to adapt to the cells from the VMH. This difference is due to the variation in timescales of the different behaviours: vasopressin cells (Chapter 2.1.1) express a phasic behaviour with a timescale in the order of 10s of seconds, whereas the VMH cells (Chapter 2.1.2) tend not to exhibit any trend in long term behaviour, hence they require only enough data to characterise the inter-spike-interval (ISI) histogram (Chapter 2.3.1). Hardware acceleration is a requirement of this evolutionary approach; the different types of hardware acceleration available are discussed in section 4.2.

The second problem that needs to be addressed is the choice of evolutionary algorithm. There are hundreds of freely-available EAs, each with their own strengths and weaknesses (Jong, 2006). The choice of algorithm is discussed in Section 4.4, where comparative algorithmic analysis of a number of archetypal EAs is performed.

4.2 Hardware Acceleration

Hardware acceleration is required in order for the adaptation to be performed within the timeframe of the experimental process. This is due to the high computational cost of the many fitness evaluations, and hence simulations, that are required by evolutionary approaches. An initial trial of an evolutionary approach, a genetic algorithm[1], took over eight hours to run on a desktop machine. It was, however, largely successful, being able to reproduce a near replica of the target's ISI.

There are a number of different hardware acceleration strategies that could be taken; this section explores four broad categories, before selecting one of them, Graphics Processing Units (GPUs), for development. GPUs were chosen for the extremely small initial investment required to produce a workable solution, in terms of both money and time, and for the ease with which the hardware can be integrated into an electrophysiology laboratory.

4.2.1 Application Specific Integrated Circuits (ASICs)

Application Specific Integrated Circuits (ASICs) (Golshan, 2007) present the most optimised form of hardware acceleration available. The hardware is customised to meet the exact demands of the problem, which in this case would be a neural model-specific simulation test bench on a chip. ASICs can provide significant performance gains over standard single core CPU approaches by only including required functionality, and hence having little or no redundancy in the chip. For this application, the test bench

[1] The genetic algorithm was a RGA with simple 2-point cross-over and mutation and a population size of 1000. Mutation had a 30% chance of occurring, and if this happened, a new value was randomly selected from the allowed range set by the maximum and minimum gene values.

would consist of a series of simple floating point processors wrapped with the exact amount of on-chip memory required to perform the simulations. These functional simulation engines would then be replicated and distributed across the chip area. The chip itself would be integrated with an add-in board, such as a PCI express card, to provide support and access to the simulation engines. Thus, simulation tasks sent to the board could be distributed among the engines, each of which would perform the simulation and return an analysis of the results.

There are, however, many disadvantages to this approach. Firstly, ASICs are expensive to develop, both in terms of time and money, with even the simplest chip taking several months to design. An ASIC meeting the requirements of this application could take over a year to develop. Additionally, unless the developer is working under or alongside a large IC manufacturer, the use of the state of the art fabrication techniques will be costly. Furthermore, even if the latest technology was utilised, the highly optimised nature of ASICs mean that all potential functionality must be included in the design phase, as it is almost impossible to add more features post-production. If additional functionality is required, a new expensive and time consuming IC must be developed.

Finally, ASICs are a poor choice for a proof-of-concept task such as this, as the majority of effort would be spent producing the simulation engine, rather than testing the methodological concepts. If this research is fruitful then there may be a point in the future where ASICs will have a role to play; however, because of the time, money, and flexibility limitations, they are not a viable choice of hardware acceleration at present.

4.2.2 Field Programmable Gate Arrays (FPGAs)

Originally Field Programmable Gate Arrays (FPGAs) were developed to provide a collection of reconfigurable logic gates. However, over the last ten years, an increasing number of application-specific functional blocks, such as multipliers, block RAM and even processors, have been added to FPGAs (Xilinx, 2008a). Because of this, there tends to be a great deal of redundancy as most applications will not require all the functionality available. This extra functionality takes up space, power, and time (if signals have to be routed around them on a chip), and so for the purposes of this study,

FPGAs can be considered a more flexible but less efficient variety of ASIC. Typically, FPGAs can provide an increase in performance of two to three orders of magnitude in comparison to a single CPU approach, approximately three to four times slower than an equivalent ASIC (Kuon and Rose, 2006).

Because of their flexibility and low development cost, FPGAs are increasingly used in proof-of-concept applications, especially as the cost in development time and money when dealing with an un-optimised design on an FPGA is a fraction of the requirements for the same task implemented by an ASIC. Because of the reduced cost of errors, many ASIC manufacturers use FPGAs to test functionality before the chip layout begins. Additionally, many manufacturers now use FPGAs instead of ASICs where time-to-market is of great importance, as their re-programmability will often allow "patching" of the hardware after distribution. An excellent review of FPGAs and their use has been written by (Rodriguez-Andina et al., 2007).

The main reasons for using FPGAs in a proof of concept project such as this is for their flexibility and ease of development. Additionally, their low cost and availability are advantageous. As with ASICs, an add-in board is required to act as an interface to the hardware acceleration, but in many cases the FPGA can be bought off-the-shelf pre-integrated with an add-in board, (Xilinx, 2008b) being an example. The final advantage that FPGAs provide is that because they are produced by major IC manufacturers, they utilise up-to-date fabrication techniques, which can make an FPGA advantageous in comparison to an ASIC created with older fabrication technology, such as 35µm.

However, FPGAs are not without fault. While it is relatively easy to produce a working prototype, fully utilising FPGA functionality and producing optimised code requires a detailed understanding of the underlying architecture. Due to the rapid evolution of the fabrication industry, it is difficult to stay on the cutting edge of the market, especially when it may take up to a year to optimise code for a device which will be surpassed by a newer model soon after the development process has finished. Because of this, developers tend to use FPGAs as brute force solutions, relying upon devices that are "large enough", "fast enough" and have "enough features" rather than performing optimisation, not caring if large portions of the device are not utilised.

During the initial stages of this work, FPGAs were the hardware of choice, where they were used to create a real-time implementation of the vasopressin releasing neuron model developed by Durie, (2007), and used in this chapter. The control parameters of the model were configurable by software, and the internal model variables were transmitted in real-time back to the software for analysis. This was the initial prototype for implementing a neuron model alongside biological experimentation before the problem of rapid parameter fitting was tackled. The details of this work are in Appendix C. By the time the project started to look at rapid parameter fitting (November 2006), the GPU manufacturers had moved towards a unified architecture that made GPUs an attractive alternative to FPGAs for the initial proof of concept. Future endeavours may move back towards FPGAs, especially if the performance/cost ratio increases.

4.2.3 Server Clusters

Server clusters are the type of hardware acceleration most commonly used by academics when performing massively parallel tasks. As most academic institutions have a hardware replacement period of around two years, these clusters of workstations tend to track performance increases associated with Moore's Law (Moore, 1965), staying if not on, then at least near the cutting edge. However, one of the problems with server clusters is that they are a shared resource, usually managed by a separate body. This means that although a large quantity of computational power is available, it will usually be shared amongst multiple practitioners using the cluster at the same time. Additionally, because practitioners do not have direct control over the shared resource, instead relying on a third party for support, disruptions to the service can result in substantial resource downtime for all involved.

For the rapid analysis proposed in this application, the individual simulations and the resulting statistical analysis would be farmed out across the cluster. This is more efficient than the approach used with other form of hardware acceleration, where only the simulations would be run in parallel and the analysis of each simulation performed on a host computer. This change in approach highlights the main strength of the server cluster; flexibility. Because each node of the cluster has all the functionality of a desktop computer, more of the more complex computation can be parallelised.

Additionally, the innate flexibility of the server cluster makes it better suited to the development of any framework, as new concepts can easily be trialled without having to account for limitations of additional hardware, such as limited FPGA resources.

Server clusters also have benefits as a final solution. Many of the evolutionary approaches have definable characteristics, with optimal values for specific tasks, see Section 3.3. The flexibility of a server cluster framework allows these characteristics to be chosen without considering the limitations of additional hardware. An example, which is discussed in detail in Section 4.4, is population size, a characteristic that is relevant to all evolutionary approaches. Some algorithms, such as the particle swarm optimiser (PSO), only require a small population size, with many iterations of that population. With many forms of hardware acceleration, such as ASICs or GPUs, the amount of parallelism is fixed, which in turn fixes the population size. This is not the case with a server cluster, and hence it may provide, at best, greater performance when utilised alongside these algorithms, and at worst, a more efficient implementation of that algorithm with no wasted resources.

The main drawback of using server clusters for the proposed task is the already identified lack of resource control. Because of the unpredictability of biological experimentation, the computational resources may be required at any time during the experiment, which could be up to six hours for neural recording. When required, the pre-defined amount of computational power must be provided or experiments will fail due to their time sensitive nature. Hence resource control is very important for rapid adaptation, and sever clusters may not always be able to meet these demands.

To enforce this requirement, the cluster could be made unavailable to other users, their processes being put on hold for the required time, but this would be a great waste of resources, and could create contention amongst users, especially if this method was being utilised by two independent groups on the same cluster. It is for this reason that this method was discarded for a more controllable and integrated source of computational power.

It should be noted that recently at least one company has been formed to provide computational resources “on tap” (Parabon, 2008). To do this they buy unused

computational power from idle machines (Dominques et al., 2005), typically from universities with server clusters, and then re-distribute the work time to remote massively parallel tasks. This is essentially a commercialisation of projects such as SETI@home (Anderson et al., 2002), but performed on a grander and more flexible scale. Using such a system could provide the resources when required, in a more efficient manner and at lower overall cost. However, the issue of having no direct control of the resource is magnified, as not only is there no control, but in addition the end user does not know were the resources are located. This may not be an issue though, as the company reports that due to the extremely large amount of resources they have at their disposal, they can guarantee the supply of "on-tap" computational power.

4.2.4 Graphics Processing Units (GPUs)

While the performance of 3D graphics cards has been driven heavily by the aggressive nature of the computer gaming industry, it is only recently that development platforms and programming languages designed to exploit the raw power available within Graphics Processing Units (GPUs) have been made available. Late in 2006, Microsoft released the DirectX 10 Application Programming Interface (API), which included a new version of Direct3D (Blythe, 2006), requiring specific functionality within all new GPU architectures that claim compatibility. DirectX 10 enforces a "Unified Architecture", where different tasks, such as vertex, pixel, and geometry shading, are now handled by a series of general purpose Single Instruction Multiple Data (SIMD) processors. In earlier architectures, they were processed by separate dedicated elements within the GPU. This new architecture provides enhanced flexibility, and allows the full computational power of the GPU to be utilised by distributing tasks directly to the SIMD processors to be run in parallel.

Both ATI (AMD, 2008) and (NVIDIA, 2008b) offer DirectX 10 compatible GPUs, but NVIDIA also released an API in 2006, which is heavily grounded in the C programming language. ATI has recently (AMD, December 2008) also released a similar API based around the BROOK programming language. One feature of both APIs is their ability to streamline the memory access operations within the graphics card, allowing multiple processes to be pipelined on a single element of an SIMD processor. However, even with the flexibility offered by a C-based API, the GPU

hardware is not amenable to some approaches which may have been optimal in pure software, such as the branching of “if” and “switch” statements. Furthermore, the availability of onboard memory limits the quantity and type of data that can be output from each process.

For the proposed application, it was anticipated that simulations could be farmed out in batches across the GPU, similarly to the approach that would be taken with FPGA architecture. The advantage of a GPU over an FPGA or ASIC is that the GPU architecture is designed to perform floating point calculations in a fast and efficient manner, whereas this functionally would have to be implemented manually within an FPGA or ASIC. Additionally, the GPU is already housed on an add-in card, which again removes the need to implement this functionality for each application. However, these features also result in a lack of flexibility for the GPU architecture, which means that many of the more complex analysis functions must be performed centrally on the host processor. This does not pose a problem for simple analysis, but as the analysis become more involved, this can become a significant overhead.

GPUs would appear to represent a near-optimal trade-off for the development of the proposed rapid model analysis method. Their massively parallel architecture of SIMD processors can provide near-supercomputer-like performance (the latest GPU from ATI, the HD 4870 X2, has a theoretical maximum output of 2.4 teraflops), while many of the tasks associated with implementing custom hardware acceleration come pre-packaged (API and Hardware interface). Because GPUs are designed with desktop computers in mind, they are easily implemented within the laboratory environment, and in many cases the hardware infrastructure is already available, with the insertion of the GPU being the only requirement for the proposed model evaluation software to run.

However, due to the application specific nature of GPUs, many of the post simulation analysis stages of the proposed model evaluation method must be performed on the host, and simulations must be coded with the architecture of the GPU in mind. Nevertheless, the ability to gain powerful hardware acceleration, which is easily applied within a laboratory environment, and is cheaply and readily available off-the-shelf, offsets the coding-based drawbacks substantially.

The GPU was therefore selected as the chosen form of hardware acceleration. In particular an 8800GTX graphics card, and NVIDIA's CUDA API (NVIDIA, 2008a) was used. This card can simulate a maximum of 4096 neuronal experiments simultaneously; this was therefore chosen as the population size for each evolutionary algorithm tested in Section 4.4. (Due to the parallel nature of the hardware acceleration, reducing the number of experiments provides very little performance increase.) This method provided a simulation throughput increase of 150 times, compared to the pure software single CPU approach[1]. It should be noted that the performance gain will change with the complexity of the model, the complexity of the fitness function, and the length of the simulation. However, it can be stated with confidence that a performance gain of two orders of magnitude can be easily obtained. Future GPU performance gains can be benefited from simply by replacing the graphics card(s) in the host machine. A more in depth discussion of GPUs, specifically the architecture and constraints, both coding and memory based, of the 8800 GTX used in this work, can be found in Appendix D.

4.2.5 Summary

Four broad types of hardware acceleration have been discussed in this section; Application Specific Integrated Circuits (ASICs); Field Programmable Gate Arrays (FPGAs); Server clusters; and Graphics Processing Units (GPUs). GPUs were chosen as the hardware acceleration platform for the development of the rapid model analysis method. GPUs represent an attractive middle ground between the flexibility of a software server cluster approach and the high performance achieved through using either FPGAs or ASICS. Additionally, GPUs require an extremely small initial investment in terms of both time and money to produce a workable solution, and are easily integrated within an electrophysiologist's laboratory.

[1] After the optimal archetypal algorithm was chosen, the optimisation process was run with and without the GPU acceleration to compare throughput.

Commercialisation

It should be noted if this work were developed commercially, GPUs may not be the optimal solution; the greater resources available for development may allow more application-specific hardware to be employed, such as FPGAs or even ASICs. Additionally, due to the skill sets of laboratory experimentalists, the off the shelf "Do It Yourself" approach favoured by computer scientists might not be well received, see Section 2.2.3. Whichever hardware acceleration platform is chosen, it may need to be completely hidden from the experimentalist.

4.3 Fitness Evaluation

Measures of fitness are required to guide the parameter estimation process. To better integrate the modelling with the electrophysiologist's skill set, it is important to select measures of fitness that are both functionally sound for the purposes of guiding the algorithm, and familiar to the user. For a suitable fitness function, the standard statistical tests used in electrophysiology, which are covered in Section 2.3, will be used. It is important for the electro-physiologist to be able to emphasise various aspects of the fitness function, in order to guide the adaptation process to test their hypotheses.

There are several key elements that must be accounted for when developing a metric for guiding an EA, as well providing information to an end user that is likely to be an electrophysiologist.

1. **The metric must be robust against unexpected behaviours.**

One of the praised features of EAs is their ability to innovate (Kicingera et al., 2005), producing novel, if sometimes somewhat inexplicable, solutions to problems. Therefore, the measures of fitness must be robust against this innovative behaviour, guarding against exploitation of loopholes.

2. **The metric must represent changes in behaviour as a gradient.**

As stated in Section 3.4, all optimisation algorithms require a gradient to follow to reach the optimal solution. If no such feature is present in the fitness landscape, then all adaptation methods inevitably tend towards the behaviour of a random search.

3. **The metric must be compatible with the domain of electrophysiology.**

Whilst designing these measures of fitness, it is important to keep the intended users in mind. If the researcher does not understand a metric, then they will tend not to use it. This would be detrimental to the current approach, where the weights of the measures of fitness are treated as one of the interfaces that allow the domain specific knowledge to be injected by the user. Thus, it is good practice to base new methods of evaluation on existing concepts already in the field. For instance, instantaneous rate is a measure currently in use, and so the concept of instantaneous acceleration is a logic step. This measure is used to determine local spike ordering by calculating the difference in ISI between subsequent pairs of spikes. This measure is formally introduced in section 5.2.

4. **The metric must be implementable within the chosen form of hardware acceleration.**

Many of the measures of fitness that are used in electrophysiology were designed to be used outside of the laboratory with statistical analysis packages, and are therefore flexible in terms of dataset types and sizes. In contrast, when using a rapid population-based adaptation method, thousands of simulations must be performed in parallel. Therefore, the limiting factors are memory space and time – what can be measured, how much data can feasibly be collected, and how much time is available to collect it. Therefore, it is important to choose analytical methods based upon the collection of finite, and usually small, quantities of data.

An example of this is the collection of initial ISI data, where individual spike times are not recorded, instead using a form of pre-processing to collate the spikes directly into a histogram. If the researcher wished to interact with the individual spikes for a different form of analysis, then either a way to extract the data from the histogram (Hazard Function) would need to be found, or additional pre-processing would need to be performed on the hardware acceleration during the simulation to make a new form of data available (Instantaneous Acceleration).

4.4 Comparative study of evolutionary algorithms

As stated in the previous sections, the proposed rapid model analysis approach consists of two main parts; hardware acceleration, for which we have chosen a GPU approach, and the use of evolutionary algorithms (EAs).

Using a GPU as a hardware acceleration platform imposes limitations as to how optimisation can be performed. The first limitation, which is linked to trial fitness evaluation, is caused by the limited resources and application specific nature of the GPU architecture. The effect is that only a limited amount of data can be recorded during simulation, and furthermore, the majority of post-simulation analysis must be done centrally on the host machine. The second limitation is linked to the fixed, but highly parallel nature of the GPU architecture, which in turn fixes the number of simulations that can be simultaneously performed. This also has the effect of fixing the population size of any evolutionary algorithm used, as the hardware cannot support a larger number of parallel simulations, and a smaller number would waste computational resources.

There exists an overwhelming number of evolutionary algorithms (EAs) inspired, and hence constructed around, a variety of concepts derived from nature. As no one algorithm can ever outperform another over all problems, characterised as the “No free lunch theorem” (Wolpert, 1997), specific algorithms will perform better in specific situations, specialising at optimisation within different types of fitness landscapes. Because of this, it was decided that a comparison of several archetypal evolutionary algorithms should be performed within the environment set by the GPU architecture. Specifically when adapting to fitness landscapes created by the combination of the various measures of fitness and the structure of an excitability-based neural model.

This section compares four evolutionary algorithms, three of which were introduced in Section 3.3, by analysing their ability to adapt an excitability-based neural model to pre-recorded data collected from vasopressin secreting neurons. The first three are core algorithms, Genetic Algorithm (GA), Particle Swarm Optimiser (PSO), Estimation of Distribution Algorithm (EDA). These have been proven over many years, and have a wealth of literature pertaining to their function, performance, and optimisation, see Section 4.3.

The fourth algorithm is a modified version of one of the core algorithms, the Genetic Algorithm (GA), specifically designed to test a hypothesis regarding the performance of EAs within the environment set by the GPU architecture, which promotes large population sizes but a small number of generations. As population sizes increase, the benefit gained from the increase in size diminishes (Goldberg, 1985). Different types of algorithms use their populations in different ways, and hence it is hypothesised that there will be a large range of performances, linked to how well each algorithm utilises the population resource. The modified algorithm, which is a form of Adaptive Genetic Algorithm (AGA), uses the large population to search the algorithm's own configuration space, using the metric of performance over time as a guide to optimisation. It is demonstrated that this modified algorithm provides equal or greater performance when compared to the GA it was based upon, which reinforces the views presented in the hypothesis.

The following section is split into five parts. Section 4.4.1 describes the implementation of each of the evolutionary approaches, followed by predictions of their performance. Section 4.4.2 describes the method of analysis, which is followed, in Section 4.4.3, by an overview of the measures of fitness used to guide the algorithms during the comparison. The result of each algorithm's comparative performance is presented in Section 4.4.4, followed by a discussion of these results in Section 0.

4.4.1 Implementation of Evolutionary Algorithms

This section introduces and discusses the three archetypal evolutionary algorithms that were chosen for this comparative algorithmic analysis, as well as an algorithm developed in an attempt to exploit the large population sizes promoted by the GPU hardware acceleration. These algorithms are the; genetic algorithm (GA), particle swarm optimiser (PSO), estimation of distribution algorithm (EDA), and the distributed adaptive genetic algorithm (DAGA), the first three of which are briefly reviewed in Section 3.3,. The algorithmic implementations are intentionally not heavily customised towards this parameter fitting task, although the fact that the optimal fitness is known (0% deviation from the target) is utilised in all but the EDA to improve convergence during the later stages of the algorithms progression. The main reason for minimal algorithmic customisation is the lack of prior knowledge regarding the shape of the

parameter space, and so there is little information to exploit. Additionally, this study is mostly interested in determining the class of algorithm to use in combination with the chosen hardware acceleration with this type of problem, not a specific implementation, of which there is a near infinite number.

4.4.1.1 Genetic Algorithm (GA)

A real-valued genetic algorithm (RGA) was used for the test algorithm, with two-parent selection, two-point crossover and a scaled mutation. The chosen population was sorted by fitness, and the parents were also selected based on their fitness, with the fittest parents having a greater chance of selection, Figure 4-1.

Prior knowledge of the problem helps with convergence. Because the fitness function describes the difference between biological and model behaviours as the percentage deviation between the two, the optimal fitness is always 0. Because of this, the mutation can be constrained as the algorithm progresses closer to optimum, thereby creating solutions that are more closely packed around the areas of best fit, aiding convergence. There is always a trade-off between convergence and the desire to find the global, not local, optima. Reducing a GAs probability, or amount, of mutation focuses the algorithm more tightly around the current areas of greatest fitness instead of searching the less explored parts of the parameter space. If mutation is limited, then an algorithm may get stuck in local optima, however, with an overly aggressive mutation operator, a GA will have difficulty converging. One of the main goals when developing an evolutionary algorithm for a specific optimisation task is to determine the optimum ratio of exploration verses convergence. Prior knowledge of the problem can often be used to gauge this ratio, and as with this implementation, guide the ratio during optimisation. It should be noted that even though the reduction in mutation is guided by how close, in terms of fitness, the GA is to finding a perfect fit, this application of prior knowledge does not guarantee that the algorithm will not get trapped in local optima, just that the algorithm will have an exploration/convergence behaviour that is proportional to population fitness

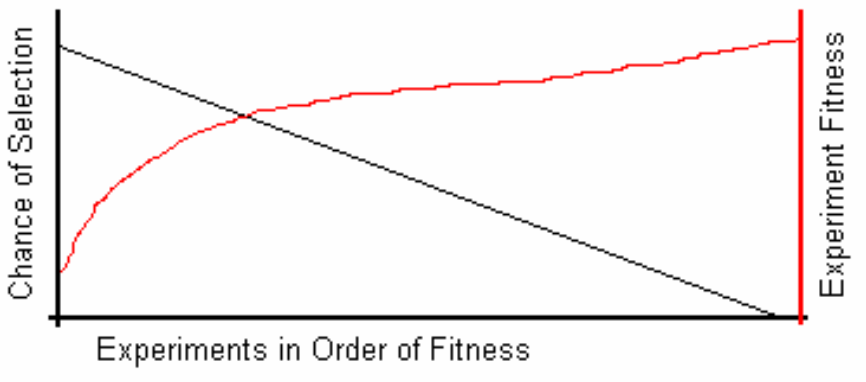


Figure 4-1 Selection method used by GA implementation. As the aim is to minimise the deviation between target and model, a lower fitness score is desired.

Because the population size represents the number of initial and working building blocks that can be represented simultaneously, as well as maintained throughout the optimisation task, the GA should provide superior performance compared to the PSO, as it is much better able to utilise the additional population members. Even so, this population size is still large compared to the dimensionality of the problem (Alander, 1999). The estimation of distribution algorithm (EDA), which is described next, would probably have an advantage over the GA, with its ability to identify and exploit inter-parameter interaction. However, this may indirectly work against the EDA by enabling it to more aggressively focus on likely areas of the parameter space before mutation has properly been applied, and therefore the algorithm would be missing the necessary partial solutions required to find the true global optima.

4.4.1.2 Particle Swarm Optimisation (PSO)

There are a very large range of PSOs to choose from, so it was decided that the PSO used in this comparative study should follow the 2006 standard from the PSO website (Particle Swarm Central - Programs, accessed 2009) making it as archetypal as possible. The only deviation from the standard was the population size, which was fixed to 4096, defined by the GPU architecture. It is important to note that the default population size is defined with the standard by

$$Size = 10 + 2 \times \sqrt{Dimensionality} \qquad \textbf{Equation 4-1}$$

For our 15-dimensional problem, this results in population size of 16. This is 256 times less than the population set by the GPU.

As stated in Section 3.3, PSOs perform best as a when each agent is analysed and re-introduced in series so that the following population members can gain immediate advantage of any beneficial positions found within the parameter space. Also, it should be noted that Diosan and Oltean, (2007) have shown that for a given problem, there is a optimum population size, which is usually small in comparison to the problem space, and that, within this population, the greatest performance is obtained by updating the fittest agents; not treating all agents equally. All these factors are directly at odds with the parallel nature of the GPU architecture and so, based upon the population size being so far from its optimal value, and the parallel nature of the hardware acceleration, extremely bad performance was predicted.

A number of different population topologies were considered, with the two main ones being fully connected, and a set of 256 independent populations of 16 members. The merit of the independent population topology is that it would allow the simultaneous analysis of subsections of the parameter space, with each independent population being initially constrained to an area surrounding one of 256 randomly chosen positions. One problem with such a topology is that within a highly multi-modal parameter space, many of these subpopulations will get stuck in local optima, and thereby not contribute to future iterations of the algorithm. The main problem is however, linked to the number of generation iterations that are to be performed. PSOs perform best with small populations, but require large numbers of population iterations (in the order of thousands). By splitting the population there is essentially 256 independent PSOs with only 21 generations each. This is far too small a number of generations to converge upon any optima, even if a PSO instance was placed directly upon one, and so it was decided that a fully connected topography should be used to better propagate fitness information among the agents and increase convergent behaviour.

4.4.1.3 Estimation of Distribution Algorithm (EDA)

The EDA used in this work was non-parametric, using the Parzen method (Costa and Minisci, 2003) with uniform Gaussian kernels. The kernel-based EDA adopted here

preserves an external subset of the fittest solutions, which is constantly updated, to help convergence and stop good solutions from being lost from generation to generation. Additionally, the EDA utilizes a nearest neighbour algorithm to control the variance of the multidimensional Gaussian kernels, as shown in Figure 4-2.

This allows the algorithm to search areas of good fitness more closely – an effect similar to the scaled mutation in the GA (Section 4.4.1.1). If points are clustered tightly, the variance of the Gaussian kernel is reduced, increasing the probability that new solutions are selected towards the centre of the clusters. Such an EDA is therefore a greedy algorithm that focuses quickly on the centre of experimental clusters. It was thought that this algorithm would perform well within this trial because of the EDA's reputation with highly multimodal problems. However, there were also concerns relating to its elitist nature. A system where good solutions are never lost from generation to generation was required, but this form of reinforcement made the algorithm overly aggressive, thereby getting stuck in local optima. To counter this, the base variance of the Gaussian kernels was increased, which proved detrimental to its ability to converge, but did provide better overall performance within the allotted computational time.

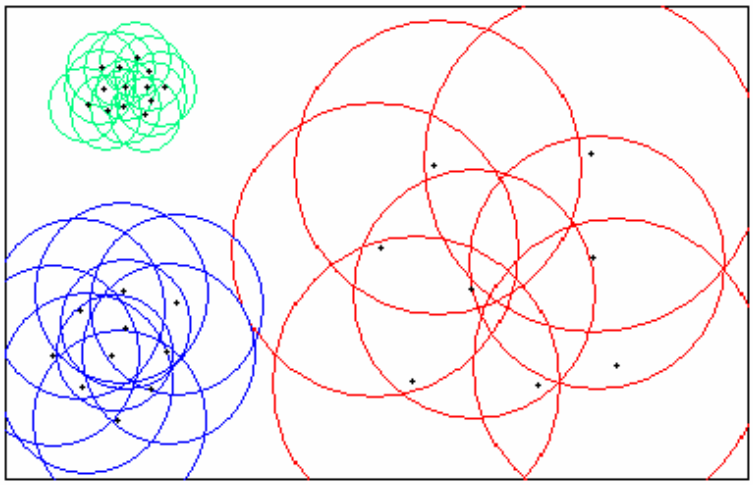


Figure 4-2 Kernel variance control through clustering analysis. The average distance between a solution and its nearest 5 neighbours is calculated. This information is then used to size any Gaussian kernel that centred upon the solutions position.

4.4.1.4 Distributed Adaptive Genetic Algorithm (DAGA)

With the dramatic increase in parallel computation that has become available to the end user, in the form of GPU and many-core architectures, it is inevitable that a time will come when the population size required by the algorithms will be much smaller than the resources available. This opinion is reinforced by the fact that fabrication methods are getting very close the minimum size of transistor, and hence in recent years, processor architectures have becoming broader. This started in earnest with Intel's push to multi-core processors, but has rapidly expanded, such that, at the time of writing, Intel's budget processor will include a minimum of three cores. Within the GPU market this parallelism creep is even more apparent, with the number of processors on chips increasing from 128 (nVidia, November 2006) to 800 (ATI, 2008) (with two chips per card), representing an increase in theoretical floating point operations (Flops) from 345.6 GFlops to 2.4 TFlops, in under two years. This presents an interesting conundrum, as not utilising this extra computational power is wasteful, and thus raises the issue of how to properly utilise these resources.

DAGA was developed to test the impact of large population sizes on algorithmic performance. As discussed in Chapter 3.3, as an EAs population size is increased, the performance per unit population decreases (diminishing returns). The DAGA was developed to test if, for large populations, greater performance could be obtained by utilising the population to, not only search the problem space for fit solutions, but also optimise the way it searches in terms of the ratio of exploration to exploitation. This is important for EA applications that utilise massively parallel hardware acceleration with fixed resource topologies (GPUs, ASICs, and to an extent FPGAs), as the EA will be configured to best fit the resource, not the other way round, and hence any improvement in resource utilisation can only improve the optimisation process.

The DAGA algorithm is based upon the previously detailed GA, using it as its core. The difference comes in the form of an adaptive wrapper that maintains a large range of GA configurations that promote exploration, and therefore retention of partial solutions, as well as exploitation. It does this by maintaining a second evolutionary approach that seeks to optimise a two dimensional configuration space defined by the rate of mutation, and the parent selection range. The population of the GA is then split into

two groups, with the first being randomly assigned a position within the configuration space, called *watchers*, and the second being randomly assigned to mimic members of the first group, called *mimics*. It is the job of the first group to act as informants as to the performance of population members at each watchers position, and this performance metric is re-evaluated after each iteration of the population. The second group then selects which configuration it wishes to mimic at the beginning of each iteration based on the performance of past iterations with an emphasis on recent activity. In this way the algorithm can change how it searches over the course of the optimisation process.

The metric for algorithmic performance is similar to Ant Colony Optimisation (ACO), where consistent performance is rewarded by the deposit of a decaying pheromone. This pheromone then draws more of the adaptable portion of the population to the areas that show consistently better performance, as shown below in Figure 4-3

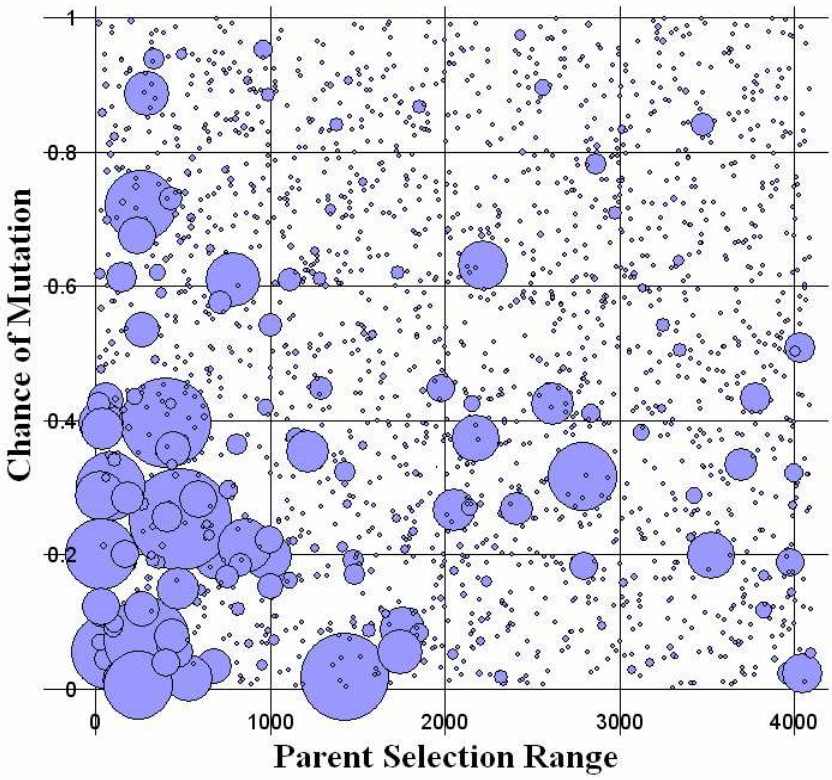


Figure 4-3 Snapshot of the configuration space; the size of the circles represent the fitness of that configuration (larger → better). The algorithm currently favours a small selection range (less than 1000) but a larger chance of mutation (less than 50%).

Unlike ACO the first section of the population is kept static. This is because the optimisation problem is dynamic, in that the optimal configuration is likely to change

with the required ratio of exploration to exploitation. The amount of pheromone dumped is also variable, and is determined by how much better a trial solution is compared to its peers. This is done to create a pressure that will spread the mimics more evenly across the watchers when the algorithm stagnates, in an effort to create more explorative behaviour to overcome traps.

This is similar to the island model GA implementation where the population is subdivided into several subpopulations, or islands, which each have their own control parameter values that dictate different degrees of exploration to exploitation. In the island model, successful subpopulation members have the ability to migrate to different subpopulations, and thereby spread favourable genetic material. The difference between the island model and the DAGA is twofold. Firstly, the DAGA has a panmictic population, where all members have the ability to mate with any other member, hence no migration is required. Secondly, and more importantly, the DAGA differentiates between population solutions and population configuration, with the second evolutionary approach optimising how future generations are constructed through the performance analysis of configurations within previous generations. Typically, the island model will consist of a fixed population size at each island. However, the DAGA can be considered an island model implementation with a much larger number of islands where, at each island, the population fluctuates depending on the performance of solutions generated. Alba and Tomassini (2002) have written a good review of parallel evolutionary algorithms, which includes a good section on the island model and how it has been previously implemented.

Because the DAGA uses the original GA as its core, similar performance is expected. However, it was also expected that the adaptive elements of DAGA will allow the algorithm to perform better over a wider range of problem. To some extent this is corroborated by previous research into co-evolution, where free lunches can be obtained by a competitive evolutionary approach that enables, through greater success, one particular algorithm to dominate the optimisation process. Our DAGA can be viewed in this light, and as seen in the results section, proved to be the better overall solution over a range of problems.

4.4.2 Method of comparison

To evaluate and compare the performance of the four evolutionary algorithms, their optimisation potential across a series of adaptation targets was analysed. The target of the adaptation runs were two sets of real neural behaviours collected *in-vivo* and *in-vitro* from vasopressin releasing neurons found within the hypothalamus, see Section 2.1. The model adapted was a modified version of a excitability-based model developed by (Durie, 2007). The differences involve the addition of a third exponential decay that represents long term (in the order of seconds) post firing membrane excitability, and is required to properly represent the observed initial peak of activity these neurons produce upon burst initiation.

The model structure, shown below in Figure 4-4, is driven by two independent Poisson distributed noise sources that represent the excitatory and inhibitory Pre-Synaptic Potential (PSPs) inputs to the cell, which produce a step increase or decrease on the cell membrane potential.

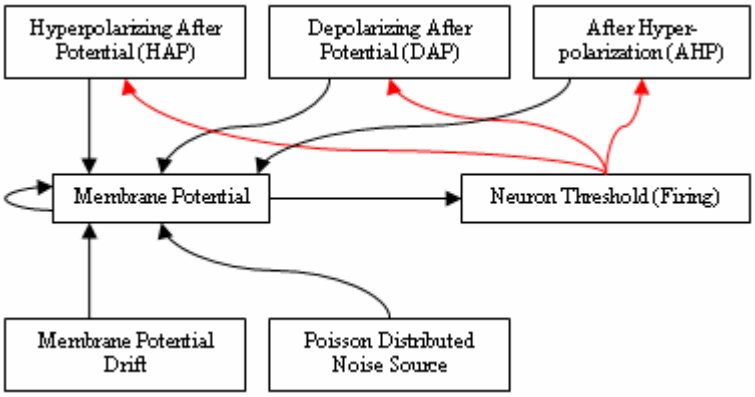


Figure 4-4 Structure of the Excitability-based neural model used to test the rapid model evaluation method. This model builds on Durie's (2007) work by including an AHP variable. Phasic behaviour is represented by a simple state machine.

Post firing membrane activity is directly modulated by three exponential decays that represent charge flow through the cell's ion channels. These are the Hyperpolarising After Potential (HAP), Depolarising After Potential (DAP), and AfterHyperPolarization (AHP). Each decay is controlled by two parameters, a half-life and a step amount. When the cell fires, the step amount parameter is added to the corresponding control variable, which is under constant decay controlled by the half-life parameter. The three

membrane activity variables are then summed together along with the membrane potential, which is modulated by the PSPs and the membrane's own decay parameter, to produce a value that is compared with the cell's firing threshold; if the threshold is met then the cell fires.

To represent the phasic firing pattern of vasopressin releasing neurons, a simple two state state-machine is implemented that controls the level the membrane potential decays to, creating rest and plateau potentials. In the plateau state, the cell has a greatly increased chance of firing, and is predominantly controlled by the HAP, DAP and AHP. In the rest state, the Poisson distributed noise sources are the main cause of firing. This plateau is a real function of the cell, and has been recorded using patch clamps, and was initially thought to be created and sustained by DAP summation. However, Durie's initial model, which relied on the DAP, could not create a sustainable plateau. Because of this, Durie implemented a black box approach, which is designed to model the behaviour, but not provide any explanation of how the cell accomplishes this feat. Chapter 6 details an investigation, using the rapid evaluation method developed in this chapter, to provide a more biological solution to this problem. It is theorised in Chapter 6 that voltage and activity dependant ion channels are responsible for modulating the phasic behaviour. The last mechanism in this model is a function called Drift. This is an exponentially decaying value that creates increased excitation during silent periods, and acts as a safeguard against stagnation of the phasic behaviour. In subsequent models this was omitted, as the adaptation method can always find areas of the parameter space that produce stable phasic behaviour.

4.4.3 Measures of Fitness

To analyse the fitness of a trial solution, a simulation run is performed followed by an analysis of that run. This analysis produces a number of descriptive single value measures, such as mean, and variance of specific aspects of the solution's behaviour, such as interspike interval. The measures from trail solutions are compared to those derived from the target behaviour to determine how different they are. This produces a set of fitness values in the form of "percent deviation from the target", each corresponding to a different single value measure. For instance, a give solution's mean could deviate from the target by 30% (fitness value of 30). These measures are

combined using a weighted combinatorial approach to create a single value that represents the overall fitness of the solution in terms of its percentage deviation from the target behaviour; hence a perfect fit will have a fitness value of 0 percent deviation from the target.

A simple combinatorial fitness approach has been adopted to combine the following measures of fitness. Even though Pareto non-dominance was not implemented for this proof of concept, it has a lot to offer in terms of the user making fewer decisions. However, because of the large number of measures of fitness, which equates a large dimensionality in the solution space, a hybrid of combinatorial and Pareto approaches may be required to reduce this dimensionality. This would be done by grouping fitness's into two or three subsets that can be used in a Pareto friendly manner.

Inter Spike Interval (ISI) and Hazard

Two ISI histograms were constructed for each simulation, with the first being constructed from the initial four seconds of each burst, and the second from the remaining action potentials. This was done to highlight and better define the behaviour of the initial peak of activity, as well as the rest of the burst. The analysis of ISI and Hazard were almost identical to the techniques discussed in Chapter 2. The averaged mode, mean, variance, skewness and kurtosis were used to classify the shape of the ISI histograms.

Burst and Space Characterisation

Vasopressin cells exhibit a distinct phasic firing pattern. To adapt to these behaviours, a method of quantifying the qualities observed was required. The is typical done by identifying the beginning and ending of bursts(Sabatier, 2004, Wakerley, 1978), thereby recording the mean and variance of the burst and silence periods. This initial technique was based upon the method used in the electrophysiology laboratory, where the start of a burst is identified by three consecutive rate bins that display activity. Similarly, the start of a silence is identified by three consecutive rate bins with no activity.

Initial trial simulation runs showed that this method was not reliable enough for guiding the EA, as this metric could be spoofed by noisy cells that were not displaying phasic

patterning. A floating cut-off was instead implemented that constantly re-adjusted itself based on mean firing rate within a burst. If a solution exhibited activity greater than 75% of the mean firing rate, then it was considered to have entered a burst. The end of a burst was likewise detected when the average rate dropped below 25% of the mean firing rate. For some particularly noisy target cells the lower cut-off needed to be reduced to 10%. Although this method can suffer from the same noisy conditions, giving a false phasic response, it does so in a way that produces poor fitness (failing gracefully), and is therefore better suited to this application. Both methods produced reliable performance when analysing phasic cells, except with extremely noisy bursts. For the rapid model analysis used in Chapter 6 this method was discarded for a more elegant and computationally efficient process that better meshes with the GPU architecture.

Noise Characterisation

As this project progressed, it became increasingly apparent that the shape of the noise seen in both phasic and non-phasic cells needed to be characterised. For this comparative study, the noise envelope within the burst was quantified by creating a histogram that represented the firing rate within a burst. Although crude, this process is amenable to being implemented within the GPU, and provides a measure of the magnitude of the noise. Chapter 6 presents a more in depth approach to noise characterisation that also provides information regarding the resonance of the noise (how quickly the rate fluctuated).

4.4.4 Results

Each algorithm was used to adapt the control parameters of the vasopressin model to produce behaviours similar to twenty seven different neural recordings, fifteen from an in-vivo environment and twelve from in-vitro. Each algorithm was restricted to having a population size of 4096, which corresponds to the maximum number of simulations, and hence maximum computational throughput that the GPU architecture can provide. Additionally, the number of generations was fixed to 21 (initial population plus 20 iterations), which equates to approximately 15 minutes of runtime with a simulation length of one and a half hours. Figure 4-5 shows the four algorithms adapting to the

same biological target. Each point within the figures represents an evaluated position within the parameter space, where the bands within the X axis correspond to generation, and the Y axis corresponds to the fitness as a relative deviation from the target (lower is better). The spread of fitness within each generation illustrates how explorative an algorithm is, where exploitative algorithms search more heavily within the areas of best fitness. Hence the majority of new positions have similar or improved fitness.

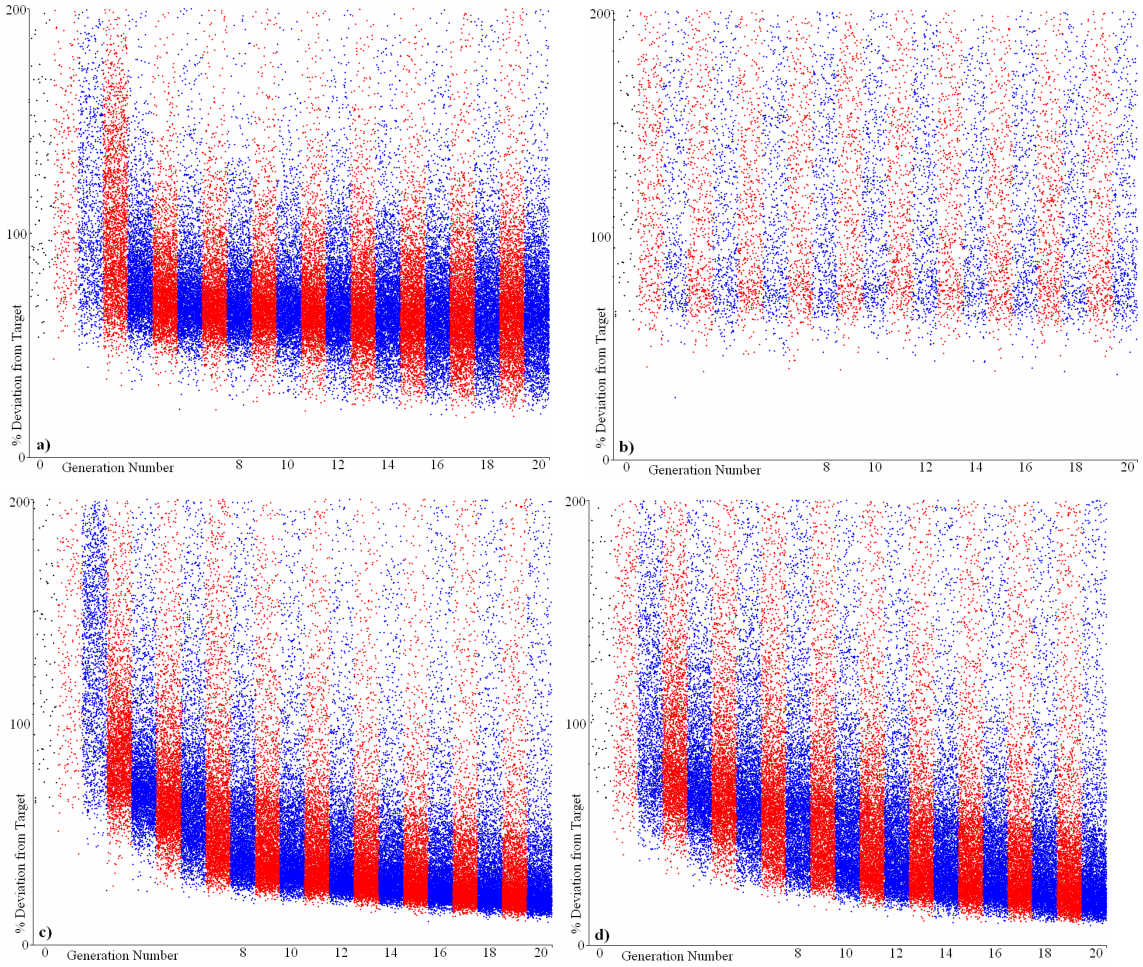


Figure 4-5 Algorithmic adaptation over time, (a) Estimation of Distribution Algorithm (EDA), (b) Particle Swarm Optimization (PSO), (c) Genetic Algorithm (GA), (d) Distributed Adaptive Genetic algorithm (DAGA). Each point represents a single trial solution. The X axis corresponds to the generation number with successive generations alternating between red and blue. The initial population is on the left hand side in black. The Y axis represents goodness of fit as a percentage deviation from the target behaviour; hence a perfect fitness would have a fitness score of 0 % deviation from the target.

Figure 4-6 presents the relative deviations from the target behaviour, attained when applying each of the optimisation algorithms against the two sets of recorded cell

behaviours. The GA and DAGA significantly outperform both the EDA and PSO, with the DAGA providing the greatest performance for 14 out of the 15 *in-vivo* targets. However, the DAGA does not provide any significant improvement over the traditional GA when working with the *in-vitro* data (the DAGA demonstrates fitter values for 7 adaptations compared to 4 for the GA).

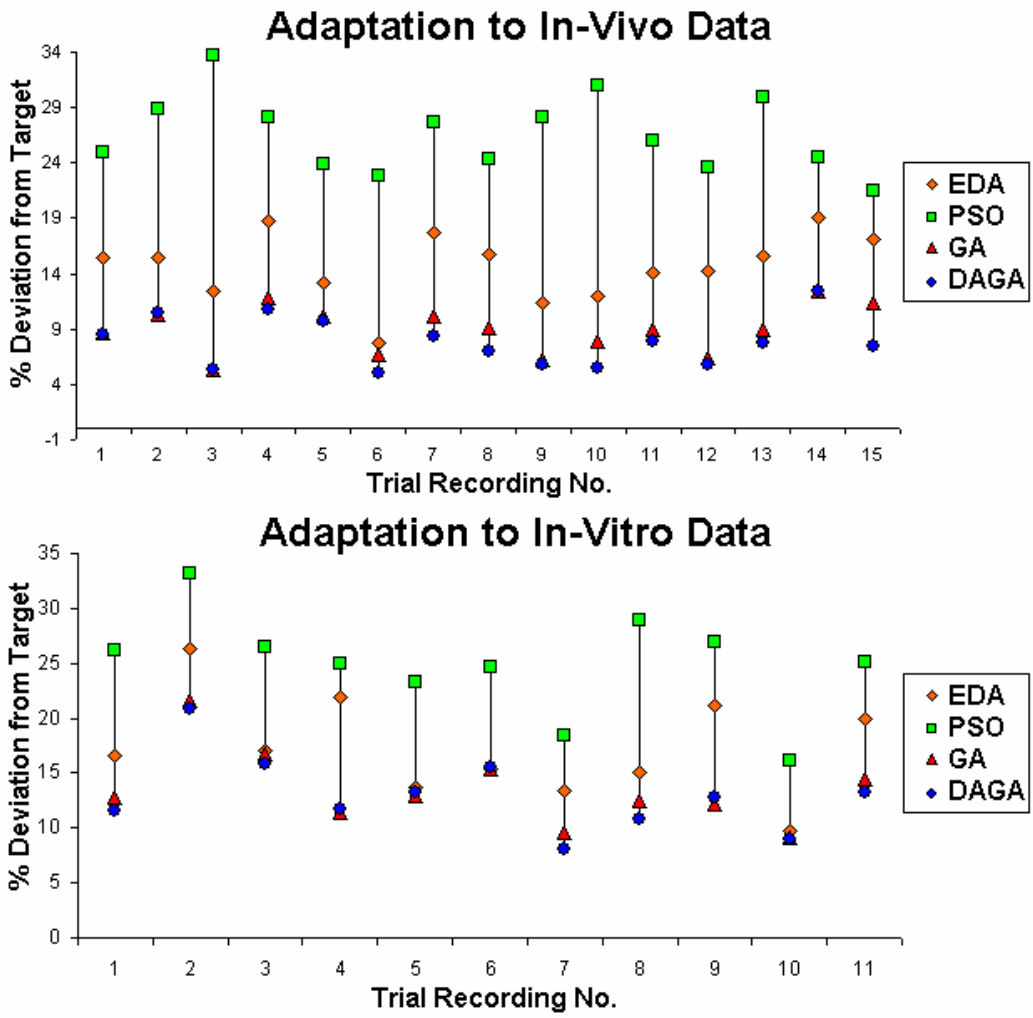


Figure 4-6 A comparison of algorithmic performance over a range of optimisation targets. Each point corresponds to the fitness value of the best solution found (the solution with the smallest deviation from the target behaviour) when adapting to each target whilst utilising one of the four evolutionary algorithms.

In Figure 4-7 a more detailed depiction of a solution from one of the previous adaptation runs, shows how close, behaviourally, the model can be optimized to fit the recorded biological behaviour. Each section of the diagram corresponds to one of the measures of fitness, blue represents the target, and red the adapted simulation. Figure

4-7(a) is the distribution of the two Inter-Spike-Interval (ISI) histograms (the left most blue and red histograms correspond to the initial four seconds of each burst, and the right most histograms capturing the ISI of the remainder of the burst behaviour) , Figure 4-7(b) and Figure 4-7(c) show how closely the simulation matches the targets burst and space distributions, Figure 4-7(d) shows the distribution of spike rate, and Figure 4-7(e) shows a comparison of the rate recordings. A perfect fit to the recorded data would show the red graphs (model instance) perfectly overlapping the blue graphs (the target).

It is important to note that no model is perfect, and so this fact combined with the probabilistic nature of the model and neural recording mean that, because of probabilistic noise, it is near impossible to recreate an exact replica of the neural recording through simulation. However, the measures of fitness are designed to integrate the length of a neural recording into a finite non time dependant measure, and hence this noise will have a lesser effect on longer recordings. A recording of infinite length would not be affected by probabilistic noise and so would be a pure representation of cell behaviour. Visually this reduction in noise is represented by the smoothness of the histograms. When the quantity of recorded data is small, the measures produce histograms heavily influenced by noise; hence a perfect fit can only be obtained by over fitting to mimic this noise. Furthermore, analysing a small quantity of data can be dangerous when the target is a largely probabilistic, not deterministic process. This is because this increases the likelihood that the recorded data represents outlying and not average neuron behaviour.

Figure 4-7 shows that this configuration of the model can reproduce many of the behaviours associated with this particular vasopressin releasing neuron, namely the burst and space distribution, intra-burst rate, and the ISI that relates to the first four seconds of each burst (leftmost histogram). However, the second, rightmost, ISI histogram shows that, in this configuration, the model cannot replicate the correct ISI behaviour for the remainder of the burst.

The model consistently failed to reproduce all the fitness criteria, leading to the conclusion that either the model was flawed or that the optimisation method was not robust enough. To remove doubt from the adaptation process, a trial behaviour was created from the model through simulation, and then used as an adaptation target. This

was done to assess if the adaptation process has the ability to find a behaviour that is known to exist within the model's parameter space, as opposed to an arbitrary biological behaviour, which the model may not even be able to produce. This test was successful, lending weight to the adaptation process, and inferring that the model structure is at fault.

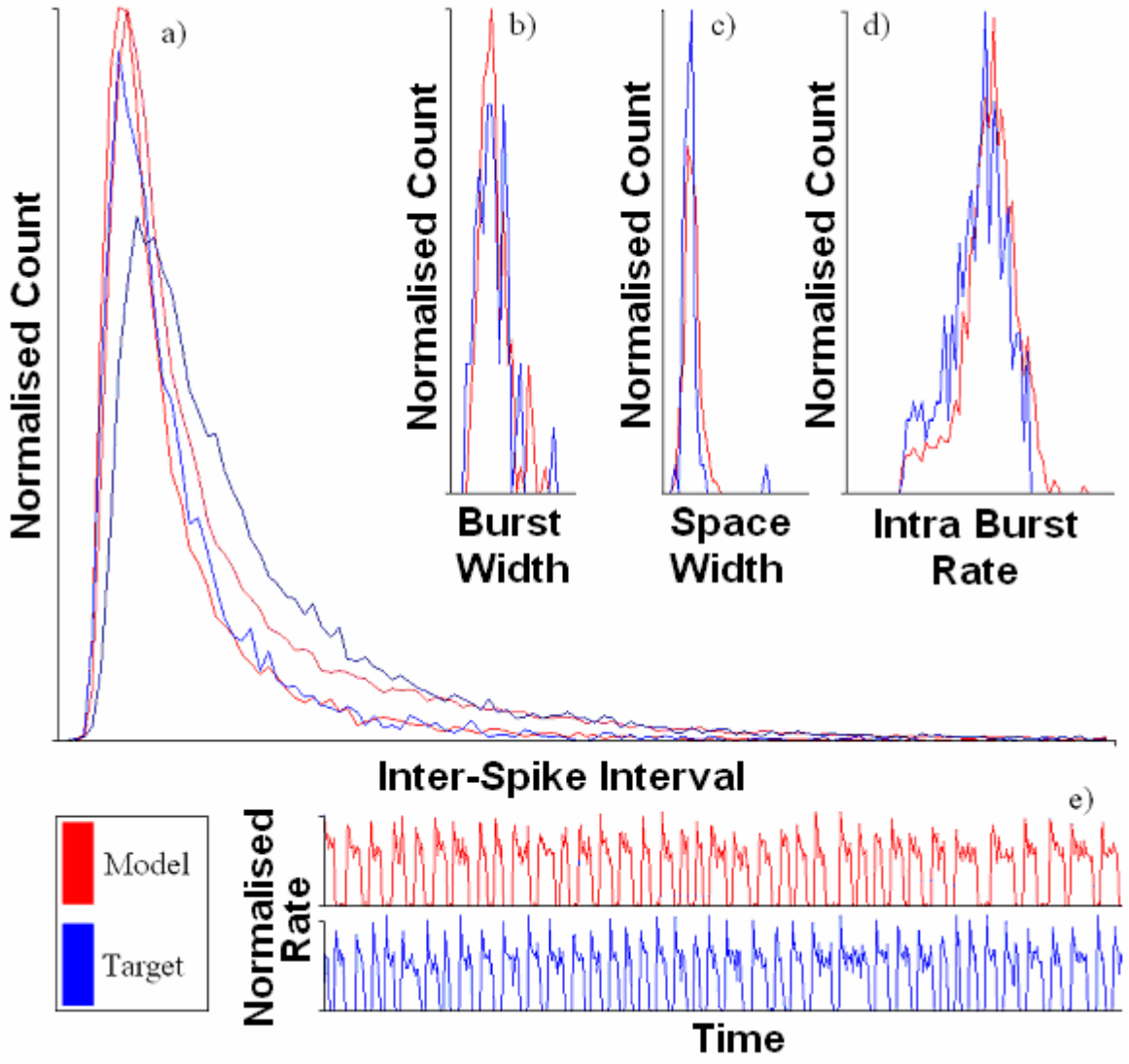


Figure 4-7 The point of best fitness found using the DAGA to fit the model to a specific target. This image displays, in more detail, the various measures of fitness used to guide each algorithm. (a) Inter Spike Interval histogram (b) Distribution of Burst Length, (c) Distribution of Space Length, (d) Distribution of Spike Rate, (e) Rate over Time; both traces do not have to match in time, but should look as if they came from the same cell. For a, b, c, and d, the perfect fitness would be represented by the red line (adapted model) perfectly overlaying the blue line (biological target). In all histograms the Y-axis is normalised to account for the difference between recorded and generated data lengths.

The ability of the adaptation process to mimic both *in-vivo* and *in-vitro* data indicates that the model is mostly correct, in terms of descriptive adequacy, but that there are still some inconsistencies to resolve. This is to be expected, as one of the most important behaviours, the phasic response, is controlled by an un-biological and overly simplistic state machine. In future iterations of the model (Chapter 6), this state machine mechanism was replaced with a more biologically inspired alternative that produced much better results; a bifurcation that represents voltage and activity dependent ion channels.

4.4.5 Discussion

Consistently, a large proportion of each initial population produced un-analysable data. Data was considered un-analysable if the simulation produced too few (less than 100 spike events over a 45 minute period) or too many (greater than 540,000, which corresponds to 200 spikes/s) spike events. This suggests that much of the parameter space was invalid, even when each control parameter was constrained heavily within a biologically plausible range. As a result, any algorithm that did not immediately respond to this problem suffered in performance. The PSO is one such algorithm, but in this case it also performed badly because it was not a suitable search method to be used alongside hardware acceleration that imposes the restriction of large population sizes. In this circumstance, a PSO will offer progressively better locations in different areas of the parameter space. This causes general "indecisiveness", from iteration to iteration, of the global best fit that excites the agents within the PSO to even greater velocities and prevents convergence within the allotted number of generation iterations. This reinforces the view that PSOs are best suited to applications that have small population sizes but a large number of available iterations. Because GPU hardware has been utilised, the optimisation environment enforces precisely the opposite characteristics, hence the PSO wastes the extra power provided by the GPU. It may be possible to improve the performance of the PSO by using a combination of a less informative swarm topology, like the K=2 topology mentioned early, and a reduced excitability (smaller ψ_1 and ψ_2). However, this more constrained topology will also decrease its convergence ability, and so even though the trade-off is likely to be favourable, it will not converge within the allotted number of generations.

The EDA has a tendency to focus on promising areas of the parameter space too soon, which causes it to perform badly. This "greedy" behaviour, even when reduced by globally increasing the kernel size, searches and finds many other favourable points within those areas and then "forgets" about exploring the rest of the parameter space, essentially losing the partial solutions that these outliers represent. This can be seen in Figure 4-5 by the rapid adaptation of the population to an average deviation from the target of 75% by the 4th generation, followed by the algorithms stagnation. Because of the large kernel size, over time the algorithm slowly spreads and is able to find better areas to search. However, once found the EDA finds it difficult to exploit these improved areas, and by the end of the adaptation, the average fitness has yet to improve significantly. To counter the initial problem of the "greedy" behaviour, a form of diversity preservation could be implemented similarly to the approaches used with multi-objective optimisation. This diversity would then help to preserve the more partial solutions, essentially limiting the need to re-discover them during the algorithms later iterations. A side effect of the EDA's aggressive nature is its ability to act as an indicator of the modality of the parameter space. Figure 4-8 shows the progress of a single parameter across an adaptation run, and clearly shows two final modes along with three others that have been previously discarded. Many of the model's control parameters produced similar behaviours, which is a good indication that this parameter space is highly multi-modal.

The GA outperforms the EDA and PSO. This is due to the algorithms' greater suitability to the adaptation environment, as the GA, through a combination of selection scheme and cross-over mechanism, splits the population into subsets with varying degrees of exploration and exploitation. The DAGA improves upon this regime by dividing the base GA into individual elements that have the potential for alternate algorithmic configuration. These elements are monitored such that a large portion of the computational power can be adapted to the more favourable areas within the configuration space, making this a co-evolutionary algorithm. Although only two algorithm parameters were adapted, and the mechanism for this adaptation is simplistic, this approach showed a general increase in performance across the majority of cell adaptations compared to the GA, which itself showed great improvement over the remaining algorithms.

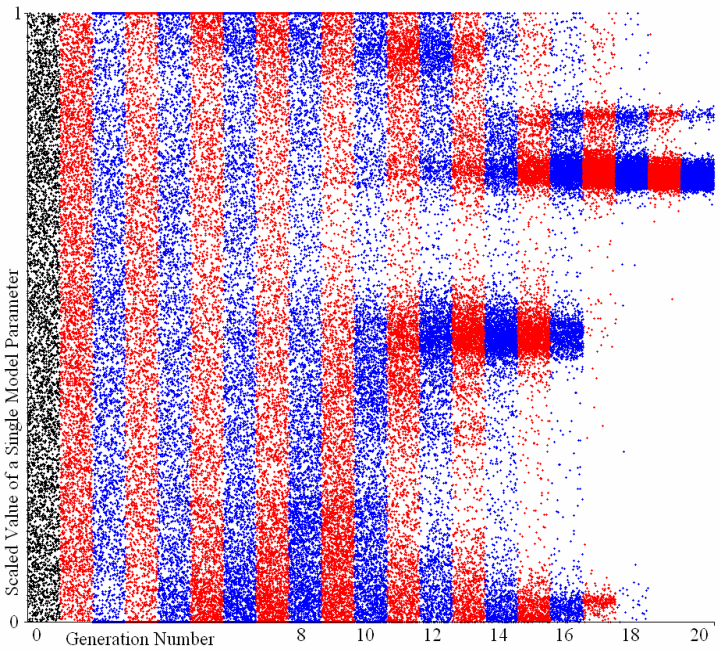


Figure 4-8 Algorithmic adaptation of a single parameter using the EDA. The clusters of points show how different modes are found, explored and then discarded until the fittest is found. Each point represents the parameter value of a single solution at different generations (initial population in black, all other generations in alternating red and blue). There are the same number of solutions present at each generation, hence the latter generations show convergence to areas of high fitness.

The *in-vivo* data was easier to adapt to than *in-vitro* data. The difference in overall performance of all the algorithms, when presented with *in-vivo* and *in-vitro* data, can be attributed to incorrect implementation of the model. The model was designed around behaviours recorded from *in-vivo* recordings. *In-vitro* recordings tend to produce a wider range of behaviours, which can be attributed to the fact that external stimulus, including regulatory behaviours of chemical gradients and PSPs from cells outside the brain slice not being present. Hence the model was not designed with these characteristics in mind, and had difficulty reproducing their resulting behaviours. However, it should be notes that a good model of the system should still be able to represent these behaviours as they come from the same class of neuron, but in these cases, the resulting parameter sets that produce this sort of behaviour would be outliers. In this case it may be that the parameter space has been over-constrained, and therefore it is impossible to create solutions with better fitness. This presents another avenue where an adaptive GA may flourish by altering the parameter constraints during the adaptation process (Eiben, 1999).

The GPU allowed for an experimental fitness analysis rate of approximately 40 seconds to a minute per generation, though this can easily be reduced by decreasing the simulation length. This results in a model being analysable once every 15 minutes, instead of the one and a half days that would normally be required. The typical maximum length of a neural laboratory experiment within electrophysiology is six hours; so whilst data is being collected from the biological sample, many iterations of a model design can be simultaneously tested.

4.5 Summary

This chapter introduced the proposed method for rapid model evaluation. This method involves the combination of evolutionary algorithms (EAs) and graphics processing units (GPUs) to rapidly adapt a trial model's control parameters, to produce behaviour similar to the biological system the model is attempting explain.

The reason for developing a rapid model evaluation method, which centres around two concepts, was also discussed. The first concept, enabling rapid iterative model design itself, seeks to bring modelling into the laboratory and make it accessible to experimentalists who do not have knowledge of computer science. The aim is to empower experimentalists to create and test their own hypotheses without the aid of a computer scientist, and hence remove the possibility for miscommunication, and reduce the long design time that such collaborations usually entail. Examples of how this platform can be used are covered in Chapter 5, where a simple excitability-based model is used to explain the cause of a range of VMH cell behaviours, and Chapter 6, where a hypothesis is tested that seeks to explain the more complex vasopressin cell behaviour. Secondly, this rapid model adaptation method can enable new forms of experimentation by providing guidance during laboratory experimentation. Some examples of this are covered in the future work section in Chapter 7, where cell classification and stimulus causality are discussed.

Following the introduction to EAs (Section 3.4), two forms of multi-objective fitness evaluation, combinatorial and Pareto, were discussed. All parameter estimation reported in this thesis utilised a simple combinatorial fitness analysis method, but it is stressed that future work should attempt to incorporate a Pareto non-dominance

approach. This is because Pareto non-dominance would provide a simpler fitness function control interface, which may prove beneficial for users who have difficulty in utilising fitness weights to manipulate adaptation behaviour.

This chapter has implemented four archetypal evolutionary algorithms in order to:

- assess the feasibility of adapting control parameters of neural models with this level of complexity,
- determine if, and how much, the adaptation environment effected algorithmic performance,
- determine the optimal class of evolutionary algorithm to be used within the adaptation environment to adapt control parameters of neural models.

The algorithms themselves were not heavily optimised, and the results showed that the adaptation environment can heavily impact upon their performance. The adaptation environment was constrained by the GPU architecture, which promotes large population sizes (up to 4096), and a small number of generations (typically less than 40) due to time limitations. The combination of EA and GPU hardware acceleration may initially seem well suited to one another, as the GPU architecture offers massive parallelism, and EAs are known to be "embarrassingly" parallel algorithms. However, upon investigation, we have found that due to the rapid advancement of parallel architectures, the resources that are being provided now outstrip the requirements of the majority of optimisation tasks that utilise EAs.

Within this environment it became apparent that the algorithm that will perform best is one that better utilises this large excess of population size. The "No Free Lunch" theorem (Wolpert, 1997), when applied to optimisation and search, states that all algorithms will have the same average performance across the set of all problems. This means that to perform well on one type of problem, an algorithm must sacrifice performance in another area. Co-evolution has shown that, through the selection of a champion algorithm from a set of peer algorithms, "free lunches" can be obtained through the champion algorithm dominating the majority of the adaptation process.

The results of the comparative study dictated the adaptation method used for the remainder of our parameter estimation tasks carried out in Chapters 5 and 6. The algorithm that presented the greatest performance was the Distributed Adaptive Genetic Algorithm (DAGA), which provided near-equal or greater performance across a range of *in-vivo* adaptations, when compared to the next best algorithm, the GA. The DAGA, which can be considered a form of co-evolutionary strategy, is dynamic, in that it can re-train towards a different area of the configuration space as the optimal ratio of exploration to exploitation changes. The PSO performed badly due to its preference for serial computational environments, and the EDA seemed to struggle with the problem space.

Future work should entail a more in-depth study of co-evolutionary strategies in an effort to maximise the use of large populations to gain “free lunches”. This is important for our application, as the range and complexity of the parameter and solution spaces must be considered extremely wide, with the user directly control these through the model design process and calibration of the fitness analysis methods.

Chapter 5 Excitability-based models as a candidate for modular model design

As stated in the introduction to this thesis, there are three requirements that need to be met before rapid iterative model design can be implemented within a laboratory environment. These are: a user friendly, electrophysiologist-oriented graphical user interface (outwith the scope of this thesis), a rapid model evaluation method (Chapter 4), and a modular model framework. This chapter, and to a degree Chapter 6, explores the use of excitability-based models as a framework for modular model design.

The core concept of a modular model is that it is made of component parts or sub-models, and so can be treated in an object-oriented way, where each sub-model consists of data (configuration) and functions that produce behaviour. This can be approached in many ways, and so the class of model presented here is meant as an example of the concept, and not the pinnacle of the design principle. Indeed, an alternative to the excitability-based model, the conductance based model, which was presented in Chapter 3, can also be broken down and treated as if it were made out of sub-models, in this case with each representing a different ion channel. The detail of the model can then be increased by substituting collections of ion channels in place of the consolidated single ion channels in the basic version.

Modular model design is not new, as all neural model design frameworks use this type of method to various degrees. NEURON, uses the Section, along with other similar sub models, which are designed to hide the complexity of the underlying mathematical equations, whilst GENESIS includes a large library of functionality which is specifically tailored to this type of design. Finally, although NEST itself does not provide modules for sub-neuron mechanisms (because it specialises with network design, not individual neuron behaviour), it does construct networks in an object oriented way. The strength of an object-oriented design method is its intuitiveness, where the structure of a model is more easily understood in terms of a series of

interconnected modules, each of which represent specific functionality, than a collection of simultaneous equations.

For the purposes of rapid iterative model design, an object-oriented method is a requirement. This is because any model developed must produce consistent behaviour when run across a large range of platforms. In the case of this work, these platforms include;

- Software simulation on a desktop machine,
- Massively paralleled simulation on a graphics processing unit (GPU).

And if interaction with the biology is a requirement;

- Real-time simulation on external hardware such as an FPGA or PIC.

This consistency can only be realised by creating standardised functional blocks (objects) that are optimised for each platform, providing greater performance, whilst still producing identical sub behaviours. In this way, higher levels of abstraction do not need to account for the simulation platform, as the medium itself decodes the model structure into computational code just prior to simulation.

Even with an object oriented approach, consistency between mediums may be difficult to obtain, especially when dealing with floating point arithmetic, where minor changes in the implementation of a hardware standard can cause deviations in behaviour through the summation of rounding errors. The work presented in this thesis suffered from this effect, as the GPU hardware utilised to accelerate the evaluation process had only single precision floating point capabilities. This manifested itself as deviations between the longer term behaviours of cells, such as burst and silence periods of the vasopressin releasing neurons. However, inter-spike-interval, or similar forms of analysis, were not affected because they represent behavioural analysis over much shorter timeframes, where rounding errors do not have time to accumulate.

This chapter explores the potential of excitability-based models to be used as the core of an object oriented rapid iterative model design process. As stated in chapter 3, excitability-based models, being a subset of the spike response model (SRM), can bring

several benefits to a modelling method that is to be used primarily by laboratory based electrophysiologists. The main advantage is their ability to encapsulate the summation of post action-potential membrane excitability (spike response) in an intuitive manner. With this class of model, excitability is a function of incoming synaptic events, summed with a series positive and negative going exponential decays of varying half-lives. The exponential decays can be mapped directly to fluctuations in membrane potential after each spike. Therefore it should be possible to compare spike responses derived from this type of model with membrane recordings collected *in-vitro* by patch clamping or sharp microelectrode whole cell recording.

Excitability-based models consolidate fluctuations in membrane potential into a small number of sub behaviours. This limits the level of detail models of this type can represent, as they are unable to represent the mechanisms that underpin this behaviour. Because of this, excitability-based models are limited to describing the cause of cellular or network, and not sub-cellular, mechanisms.

Section 3.1 introduced several criteria for good model design, namely, *plausibility*, *explanatory adequacy*, *interpretability*, *descriptive adequacy*, s*implicity*, and g*eneralisability*. Excitability-based models (Section 3.1.8) meet the first three criteria, as they are designed to directly reflect aspects of the biology, using the well understood mechanisms of the basic spike response model (Section 3.1.6). However, the *descriptive adequacy*, *simplicity*, and *generalisability* of the model are still in question.

Goodness of Fit can be used to determine d*escriptive adequacy*, but does not take a model's *simplicity* into account, and therefore cannot by itself be used to determine if a model correctly represents the underlying biology. This chapter explores the use of a correlation based method to quantify the level of complexity between the parameter and solution spaces. The less *simple* a model is, the greater its ability to reproduce any arbitrary set of observed behaviours, but the resulting distribution of control parameters that reproduce these behaviours tends to become more chaotic as the size and complexity of the parameter space, which correlates to the models complexity, increases; hence this measure is used here to analytically quantify a model's s*implicity*.

Generalisability is usually tested by one of two methods, but both require the use of observed data that was not used as part of the model fitting task. The first involves generating new behaviours from the model, and then compare the distribution of these behaviours with a subset of the observed data that was not used during the model fitting task. This method is suitable for problems with large quantities of observed behaviours, but it is less suitable for models based on small data sets, where the user will wish to use all their observed data to identify faults with the model. The second method involves the creation and evaluation of predictions. This is better suited to the in-lab design practices that are encouraged by this thesis because the experimentalist, who would be using the model to make the predictions, will also understand the limitations of experimental procedures. This knowledge puts them in better stead to test a model's *generalisability*, and will hopefully allow the creation and evaluation of predictions to be performed within the time frame of a single laboratory session.

In this chapter both d*escriptive adequacy* and *simplicity* are tested by utilising the rapid evaluation method developed in Chapter 4 to adapt the basic excitability-based model to four sets of neural spike time data collected from the VMH (Section 2.1). These four sets are extremely varied consisting of behaviours that have been previously classified as, broad, doublet, random and regular firing neurons (Sabatier and Leng, 2008). It should be noted that the original study divided the collected data into nine groups, not four. Some of these different behavioural groups are referenced in Section 5.5 as the ability of the model to extrapolate the cause of different behaviours is tested. *Generalisability* is not tested analytically, but predictions are made regarding the relationship between the different classes of cell behaviour by exploring the shape of the spike responses.

This chapter is divided into four sections, with the first providing a full description of the model, both formally and structurally. This is followed by the results of the adaptation process, which show how close the model can be adapted to produce the range of behaviours shown in the four data sets, and ultimately test the model's d*escriptive adequacy*. The third section investigates the *simplicity* and robustness of using the changes in cell spike response as an explanation of cell behaviour. This involves a complex analysis of the differences between the cell behaviours in terms of both inter-spike interval distribution and spike response, looking for correlation between

the two. If the implementation of spike response is robust then cell pairs which present similar ISI histograms should also have similar spike responses, and hence provide evidence to the robustness and *simplicity* of the model. The last section explores the cause of various behaviours in terms of their spike responses, which ultimately allows the extrapolation of new behaviours to discover the associations between the behavioural groups. These associations can then be tested within the laboratory through analysis of cell firing patterns pre and post the application of ion channel blockers, which will confirm the model's *generalisability*, though this is not done for this thesis, and so the predictions made are unsupported.

5.1 The Core Excitability-based Model

Consisting of two sections, the model is based upon the CSRM (Section 3.1.6). The first section encompasses a Gaussian noise source that represents incoming post synaptic potentials, with the second being a sum of exponential decays that corresponds to post action potential changes in membrane excitability. For ease of writing, two functions that are used throughout the formal description are summarised below. These are the Heaviside function and a function that converts from half-life to a multiplier based on the time slice width used by the simulation. These functions are important as the majority of the model consists of decaying variables that undergo step increases. It should be noted that the formal descriptions presented in this thesis are all of the form of a series of simultaneous discrete differential functions. This representation was chosen instead of the customary continuous form because they better depict the actual model implementation. A continuous representation of the basic excitability-based model can be found in Appendix A.

Equation 5-1 - Heaviside Function

$$H[n]=\begin{cases}0, n<0\\1, n>0\end{cases}$$

Equation 5-2 Half-Life to Multiplier Conversion Function

$$\tau[x]=1-e^{\frac{\ln(2)\Delta t}{x}}$$

The Gaussian Noise is generated using the Box Muller function fed by a Merssian Twister uniform random number generator. A set of random numbers are created before the simulation and stored in an array. A simpler uniform random number

generator is then used during the simulation to fetch from the array at random, and the resulting value is multiplied by the *Noise Amplitude* control parameter. The Gaussian noise is directly combined with the membrane excitability, which is under constant decay (*Membrane Half-Life*) towards the *Rest Excitability* level. The formal description, along with a functional block diagram, is shown in Figure 5-1.

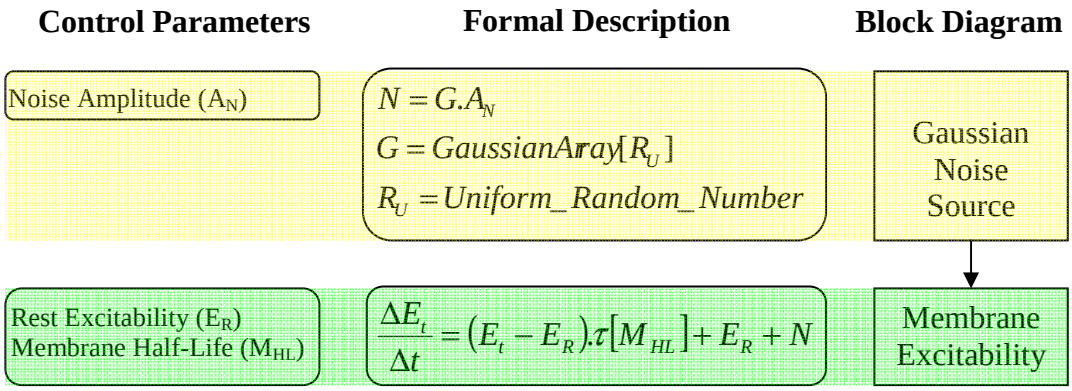


Figure 5-1 Description of the synaptic noise integration on a decaying membrane. Variables: E_t → Cell excitability at time step t; N → Gaussian Noise; Δt → Simulation time step size.

The spike response is represented as the sum of three exponentially decaying variables, all of which have a step increase or decrease in the event of the cell producing an action potential. These are designed to mimic the effects of the HAP, DAP, and AHP, over many action potentials, and in effect are a representation of the cell's history. It was theorised that when adapting to cells that do not exhibit a DAP or AHP, the relevant control parameters would tend close to zero. The formal description, along with a function block diagram, is shown in Figure 5-2.

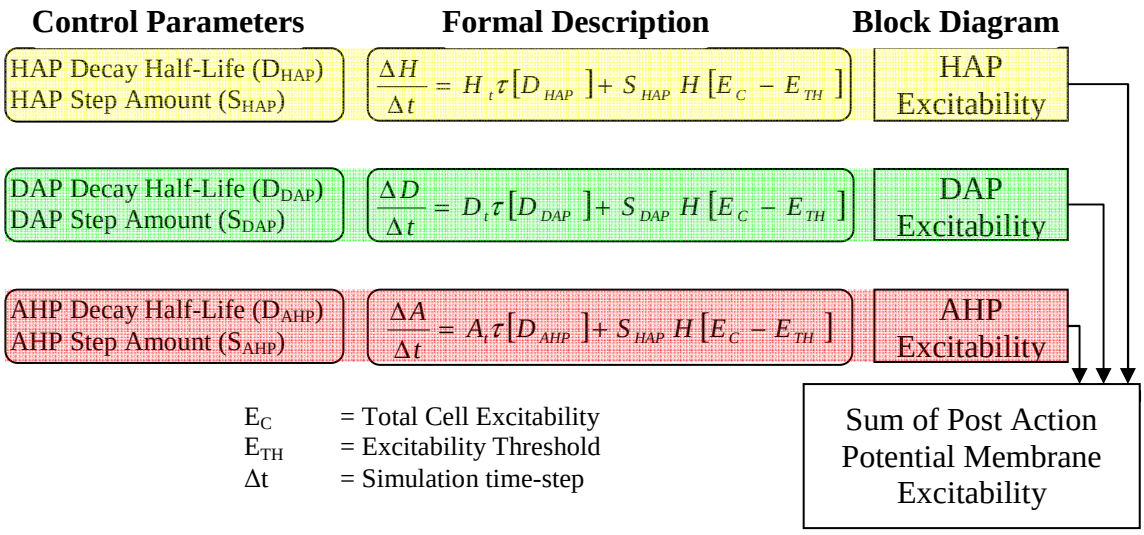


Figure 5-2 Description of the cumulative spike response. Three exponential decays are summed together to represent the model's spike response. Variables: H → HAP magnitude; D → DAP magnitude; A → AHP magnitude.

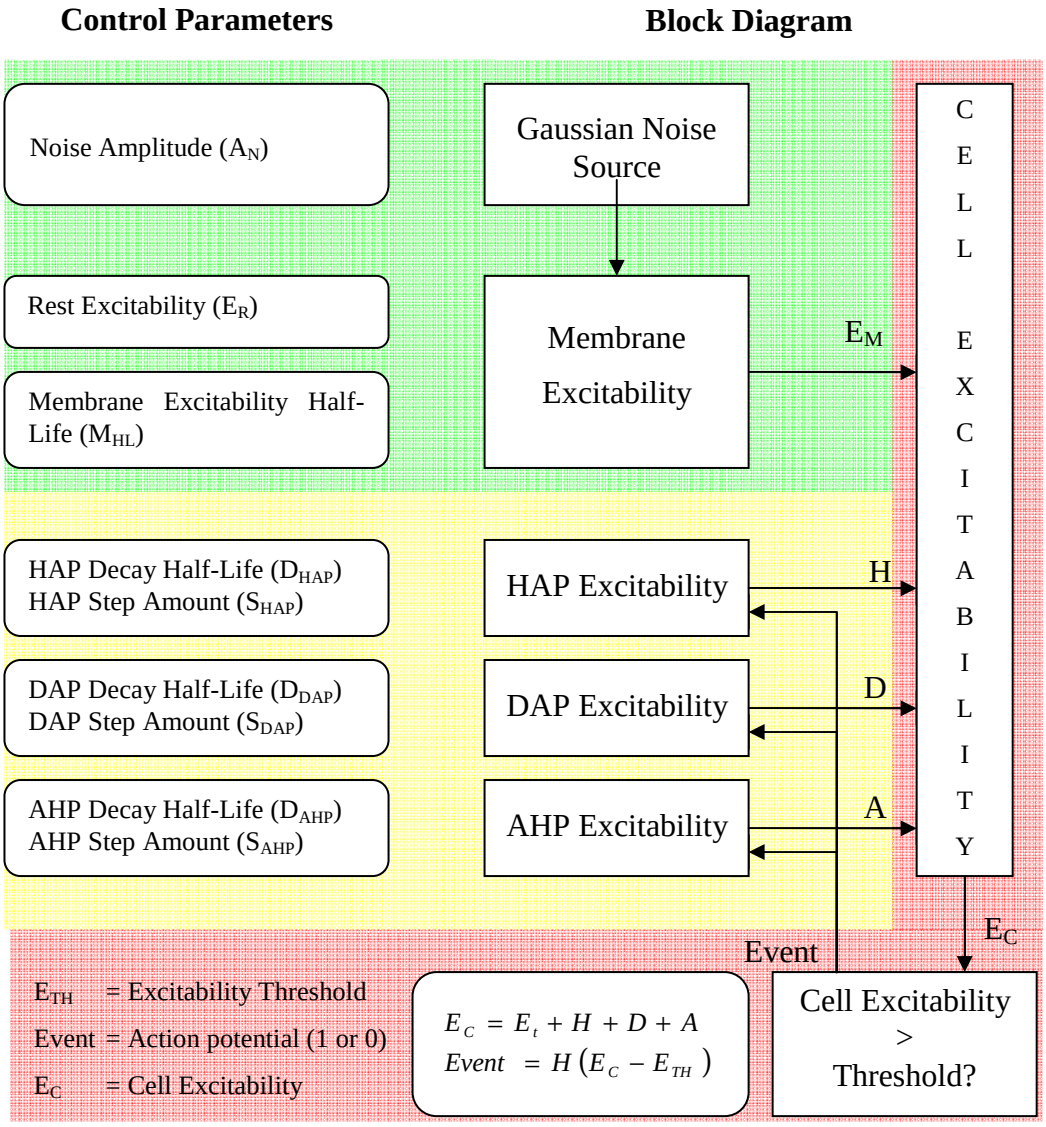


Figure 5-3 Overview of the basic excitability based model. The model is split into three parts which are depicted by the green, yellow, and red sections of the diagram. Green: integration of random synaptic input on the membrane. Yellow: cumulative spike response. Red: integration of both the spike response and synaptic input to determine if membrane threshold has been crossed.

To determine whether a new action potential is created, the membrane excitability, which includes the Gaussian noise, is combined with the total post action potential membrane excitability (HAP + DAP + AHP). If the combination of these effects exceeds the membrane threshold (which is set to the value of 0), then a spike occurs. A

functional overview of the model is shown in Figure 5-3, along with a formal description of how both sections are combined and compared with the threshold to determine if an action potential is created.

The model consists of a total of nine control parameters. However, the "Membrane Excitability Half-Life" control parameter is fixed at a half-life of 0.01s, which corresponds to a realistic half-life of incoming post–synaptic events, and the "Rest Excitability" parameter is also fixed at -10. This results in a seven dimensional parameter space defined entirely by the combination of the amplitude of incoming post synaptic events and the spike response. As shown by the results, this extremely minimalist model can represent a large range of behaviours, whilst still utilising biologically relevant control parameters.

5.2 Method of Analysis

The control parameters were adapted to the behaviours of 72 cell recordings, collected *in-vivo* from the VMH, using the rapid evaluation method developed in Chapter 4. These cell recordings, which are a subset of a larger study of 272 cells (Sabatier and Leng, 2008), have been previously analysed and divided into four groups according to their different firing patterns. To guide the evolutionary process, several measures of fitness were utilised that quantise how close a solution is to mimicking the behaviour of a target cell. The following section describes these measures and their use.

Because these are continuous firing cells, for characterisation purposes only short term behaviours are of interest, hence the majority of the measures of fitness are derived from the inter-spike-interval (ISI) and instantaneous acceleration (IA) histograms (described below). The averaged mode, mean, and coefficient of variance of the ISI histogram are used to guide the algorithm towards approximately fit areas, as they produce a dramatic decrease in fitness when the solution deviates from the optimal, though they are not as good at fine tuning the solution set to the shape of the histogram once in these fit areas. For this reason, two other measures that focus on optimising specific sections of the ISI are also included. These are the RMS Noise Envelope, and the RMS Selected Envelope.

The noise envelope performs a comparison between the first N bins of the target and solution ISI histogram, calculating the RMS error. The selected envelope performs the same RMS error calculation, but the section of the ISI that is analysed can be changed. For this work, the nose envelope was used to optimise the bins up to the mode, and the selected envelope to focus on the remaining bins that contain data. In all cases, the magnitude of the ISI histograms are normalised, as both the size of the solution, and that of the target data sets vary. The hazard of the ISI is also derived and compared as it tends to highlight detail in broader distributions.

Because of the range of behaviours that are observed, specifically the spread of mean firing rates, a non-standard ISI histogram is utilised. In this work, and in Chapter 6, the histogram consisted of 500 bins, starting with a width of 1ms, and increasing every 20 bins. This was done for two reasons. Firstly, there was a need to capture a wide range of firing rates with a generic process. The non-standard histogram allows the capture of ISIs of up to 6.25 seconds. Additionally, this process helps to counter noise, which can disrupt further analysis, caused by lack of data at the tail end of the ISI histogram. The second reason is linked to cells that exhibit doublet activity.

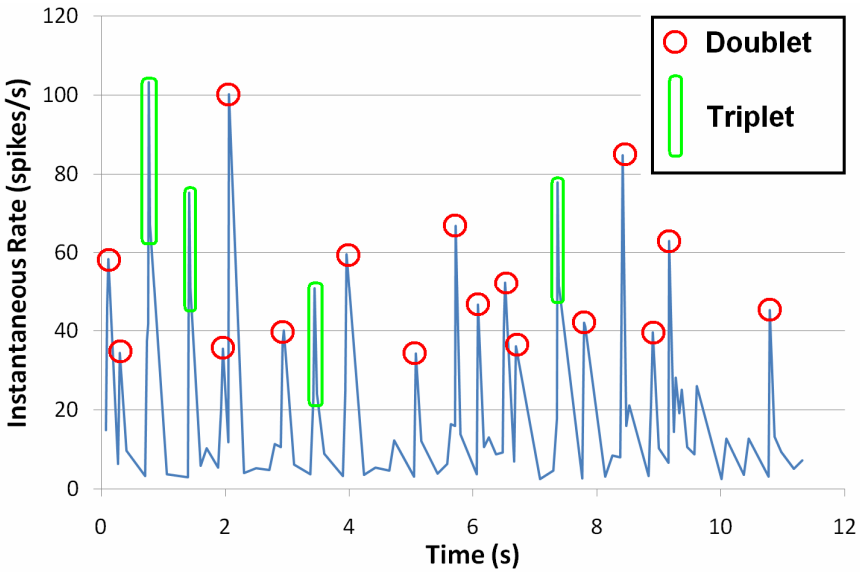


Figure 5-4 Instantaneous Rate of a Doublet recording collected by (Sabatier and Leng, 2008). Doublets can be identified by a single peak; Triplets and other multiplets consist of several sequential events with high instantaneous rates

When characterising Doublet cells, the analyst is interested in both the short ISI behaviour, which represents the actual doublets, as well as the longer term ISIs, that show the range of relaxation times between doublets. In general, this method of histogram construction allows analysis of a wide variety of firing activities without requiring prior information regarding the shape and mean firing rate of the cell. It should be noted that, because non-standard ISI histogram distributions have been used, the derived hazard functions also appear different from those that would be created from standard ISI histograms. Even so, this is still a good measure of fitness for the purposes of this work, as it continues to highlight different aspects of a cell's firing pattern. Many cells found within the VMH are classified as doublet cells (because they exhibit fast re-activation followed by extended periods of silence, as shown in Figure 5-4). Using standard ISI and Hazard histogram techniques, it is impossible to differentiate a cell that has a burst-like behaviour (periods of intense activity followed by sparse sporadic firing) from the doublet firing observed in these VMH cells. An extreme example of this is shown in Figure 5-5 where the inter-spike intervals of a doublet cell recording were re-ordered into groups, and spike times created from this newly ordered set. Both the ISI (a) and hazard distribution (b) are identical, yet the two traces show completely different behaviours when their instantaneous rate (c) is examined.

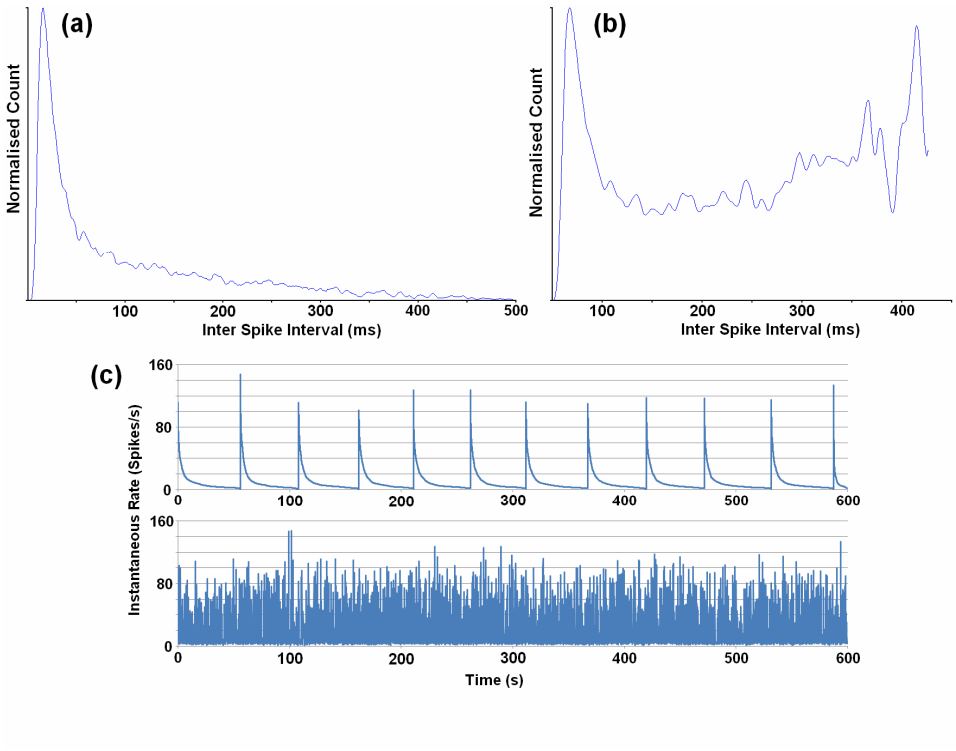


Figure 5-5 A doublet trace compared with a reordered version of itself, where the interspike intervals were ordered from smallest to largest in batches of 55 seconds. (a) Inter-spike Interval of both traces. (b) Hazard distribution of both traces. (c) Instantaneous rate, Top: Re-ordered set, Bottom: Original set recorded by (Sabatier and Leng, 2008). Note that interspike interval cannot differentiate between the original and re-ordered sets, and therefore does not contain any information regarding local spike ordering.

This problem can be solved by examining the instantaneous acceleration - the difference in ISI between subsequent pairs of spikes. This provides information about local spike ordering, and allows differentiation between doublet and bursting behaviour. The bursting behaviour has much less variance of instantaneous acceleration than doublet firing, as shown in Figure 5-6. This is due to the large change in ISI caused by doublet activity. It is interesting to note that this analysis also provides insight into the ratio of doublet to triplet and other multiplet activity. Doublets will skew the distribution to the right by promoting larger accelerations, and multiplets, being very regular, will do the opposite.

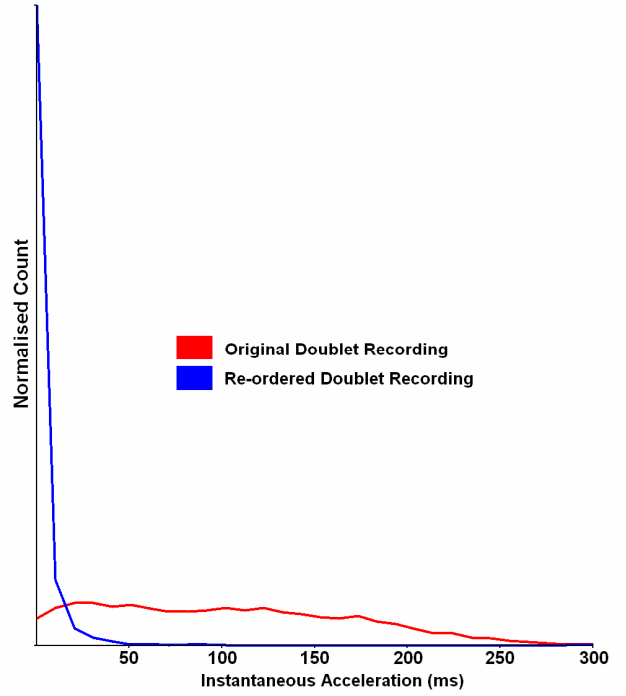


Figure 5-6 Instantaneous Acceleration of the two previous traces in Figure 5-5. This graph clearly shows the ability of the instantaneous acceleration measure of fitness to differentiate between different types of local spike ordering; specifically bursting and doublet behaviours.

Each adaptation comprised of either 20 or 30 generations of the evolutionary algorithm. The varying number of generations reflected the difficulty the algorithm encountered when performing the adaptation, and in some cases, the adaptation was performed multiple times with different fitness weights to emphasise specific characteristics of the cell behaviour. This has the effect of guiding the evolutionary algorithm towards a different point on the Pareto optimal front.

5.3 Results - Descriptive Adequacy

To provide an indication as to the effectiveness of the model to reproduce cell behaviours, this section presents two solutions from each of the four pre-defined cell groups that represent the best and the worst cases of adaptation. Each solution presented consists of the inter-spike-interval (ISI) and instantaneous acceleration (IA) histograms as well as a section of the resulting rate data. In all figures, blue traces represent the biological behaviour and the red the model solution with the greatest fitness.

In general, the poorer adaptations were caused by a reduced target dataset, where the lack of data created noisy target ISI and IA histograms. Because the simulations are always long enough to produce smooth ISI and IA distributions, its behaviour tended towards the average of the noise. However, this caused the measures of fitness that use the RMS error between histograms to lower the average fitness of the EA population. This in turn caused the mechanisms that focus the EA, such as a reduction in mutation amount, to not function fully. A good example of this is the worst fit to the Doublet cell shown in Figure 5-10. Even so, the adaptation process still managed to steer the model towards regions of the model parameter space that at worst produced similar behaviour, and so it can be concluded that the model is indeed able to reproduce these four types of cell behaviour, but in future, automatic mechanisms should be put in place to protect the evolutionary approach from noisy histograms caused by lack of data.

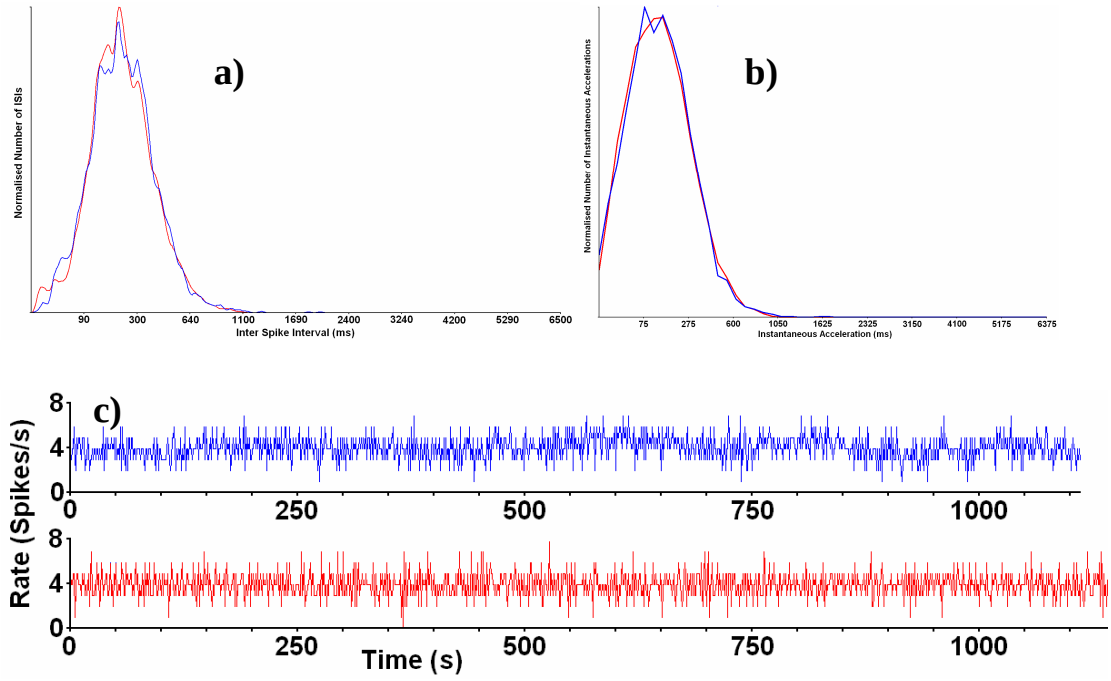


Figure 5-7 Best fit to Broad VMH cell; a) ISI histogram; b) Instantaneous Acceleration; c) Rate trace

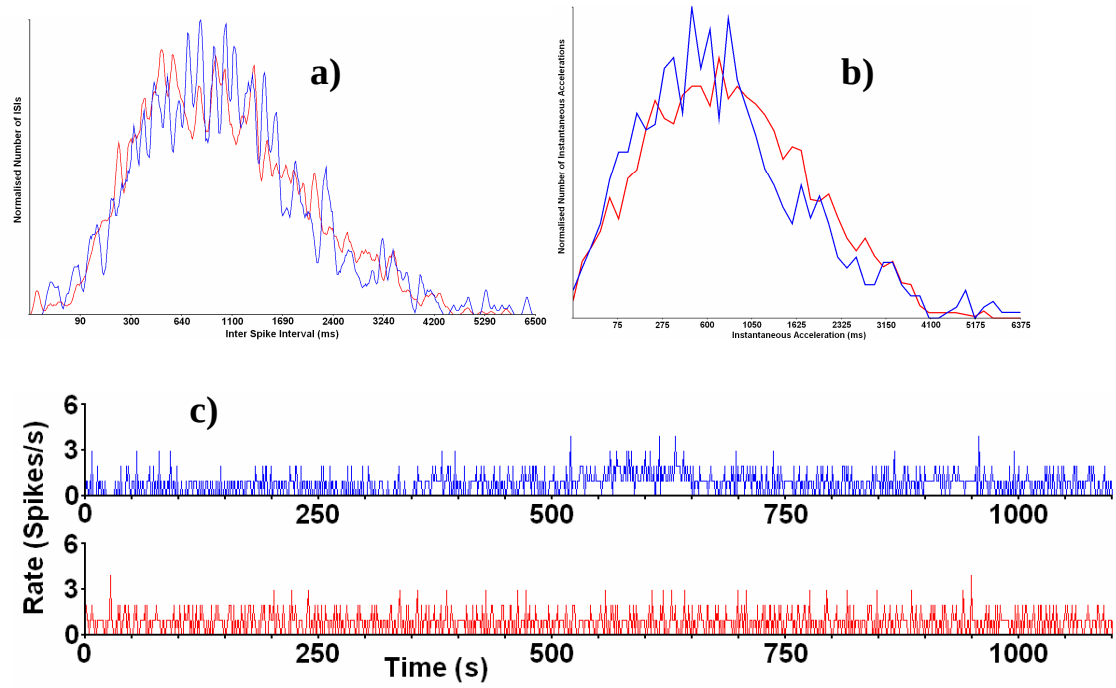


Figure 5-8 Worst Fit to Broad VMH cell; a) ISI histogram; b) Instantaneous Acceleration; c) Rate trace

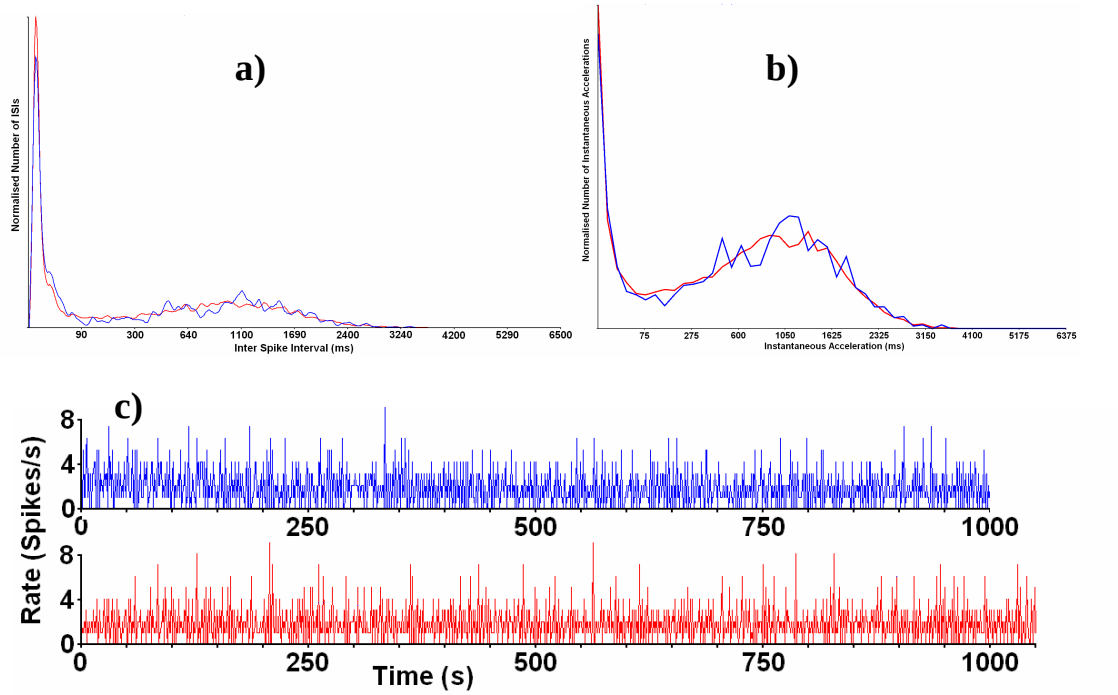


Figure 5-9 Best fit to Doublet VMH cell; a) ISI histogram; b) Instantaneous Acceleration; c) Rate trace

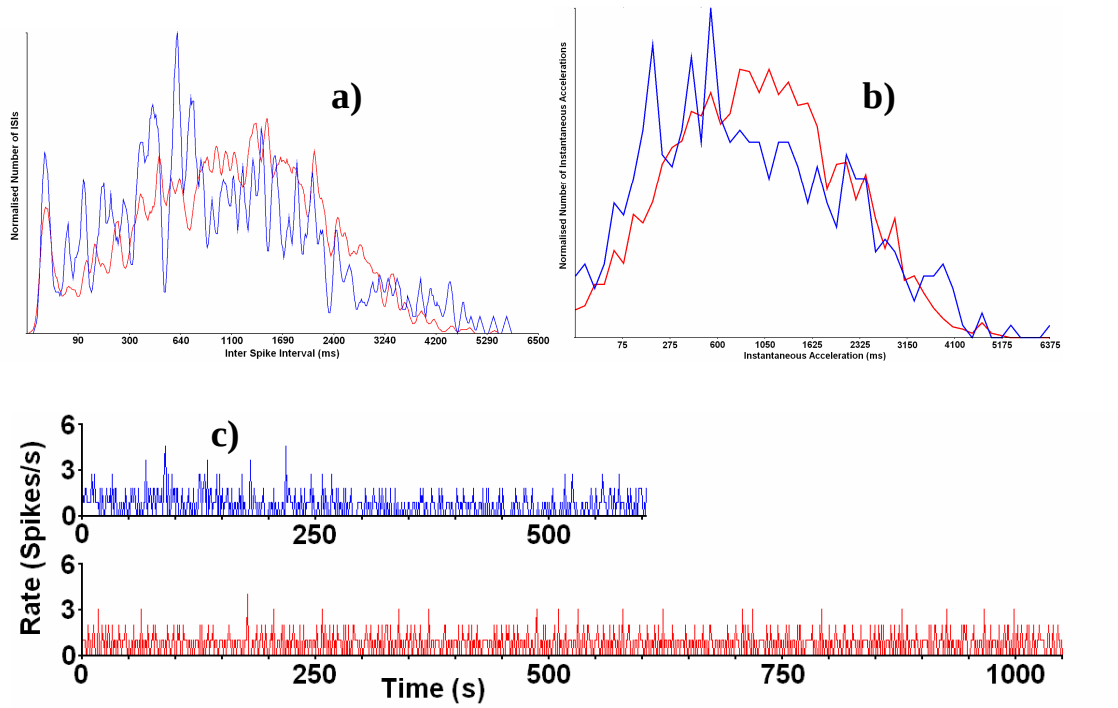


Figure 5-10 Worst fit to Doublet VMH cell; a) ISI histogram; b) Instantaneous Acceleration; c) Rate trace

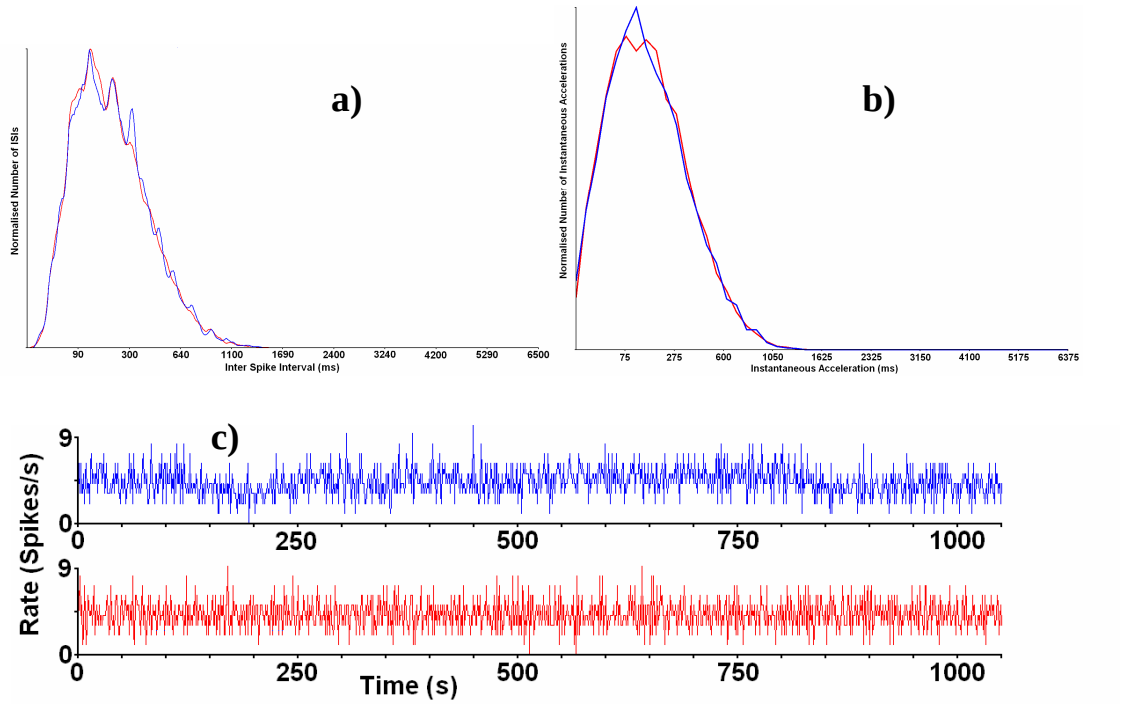


Figure 5-11 Best fit to Random cell; a) ISI histogram; b) Instantaneous Acceleration; c) Rate trace

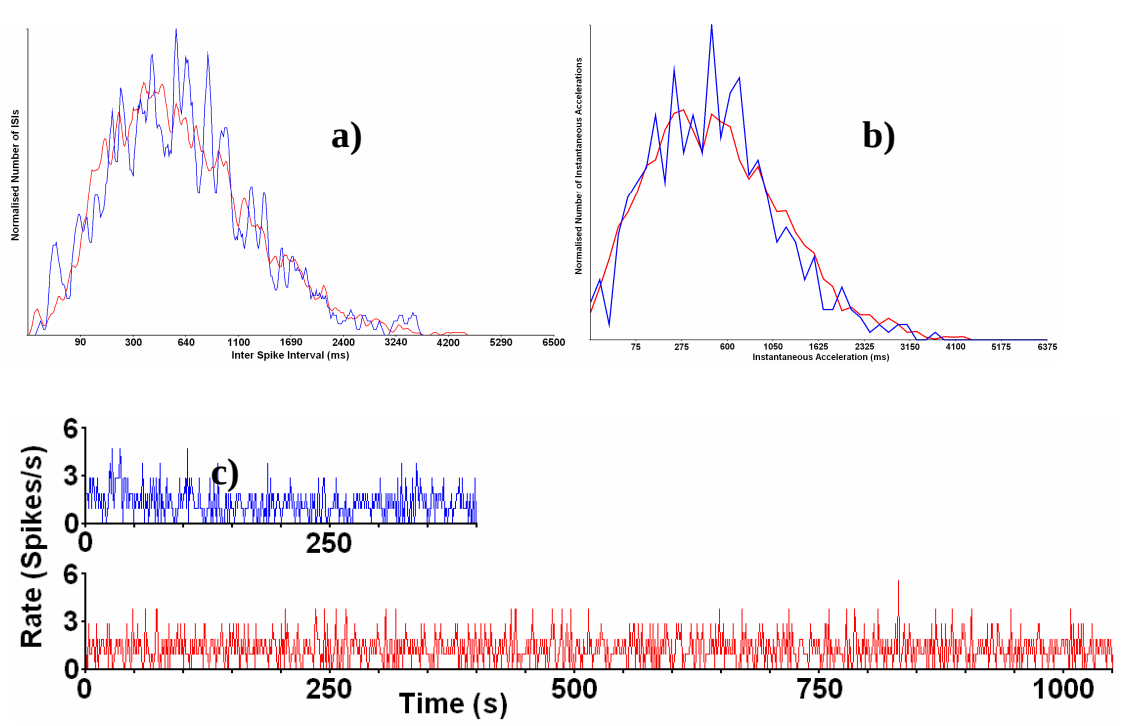


Figure 5-12 Worst fit to Random cell; a) ISI histogram; b) Instantaneous Acceleration; c) Rate trace

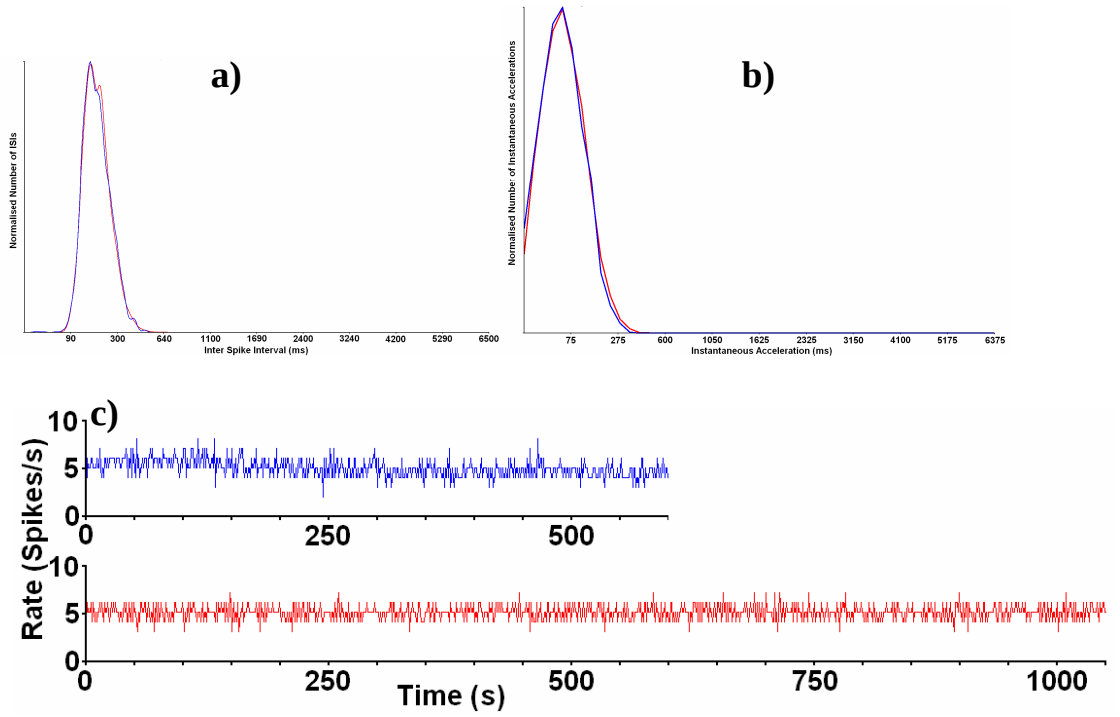


Figure 5-13 Best fit to Regular VMH cell; a) ISI histogram; b) Instantaneous Acceleration; c) Rate trace

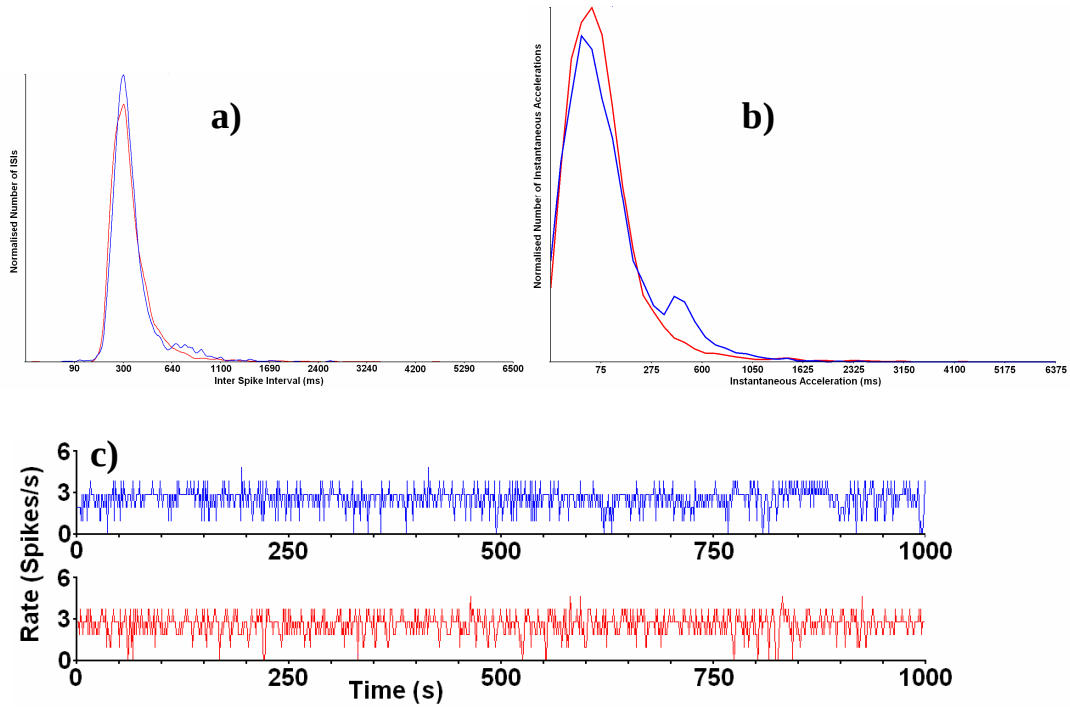


Figure 5-14 Worst fit to Regular VMH cell; a) ISI histogram; b) Instantaneous Acceleration; c) Rate trace

5.4 Results – Robustness and Simplicity

The previous section concluded that the model can reproduce the wide range of cell behaviours represented in the four sets of cell recordings collected from the VMH. However, the robustness and *simplicity* of the model still needs to be tested. Where *simplicity* has already been defined in Section 3.1, robustness is defined as a model's ability to maintain broad behaviours over a range of continuous parameter configurations. A robust model is able to associate individual areas of the model's parameter space with specific behaviours or classes of cell. However, this may not be the case, as the control parameters may have little or no correlation to resulting behaviour as a result of excess complexity, thereby providing little or no robustness. This is undesirable for a model, as ultimately the model will be used to make predictions regarding the function of a cell based on the distribution of control parameters that produce behaviour similar to the biology. Additionally, a model that is chaotic, where small deviations within the parameter space can produce large changes in behaviour, is a bad candidate for seeding a network model, as the distribution of behaviours would be unpredictable.

To test the robustness of this model, and thereby gauge its s*implicity*, correlation between the excitability patterns derived from the solutions of the model and the behaviours of the cells must be found. This is done by comparing similarities between all the target cell ISI histograms against similarities between the excitability patterns of the model solutions. Before the comparison can be carried out, the raw ISI data needs to be pre-processed, and the excitability patterns of the solutions constructed.

All the cell ISI histograms are normalised by their mean firing rate. This is done for two reasons. Firstly, the work undertaken by (Sabatier and Leng, 2008) suggests that the identified behavioural subsets are firing rate independent, and so normalisation allows us to compare the underlying shape of the ISI histograms when disassociated from their rate of firing. Secondly, as shown in the previous section, lack of data, which is usually associated with a decreased mean firing rate, causes the ISIs to be spread thinly over a large number of bins, thereby producing noisy ISI histograms. This effect is limited by normalising the bin widths of the ISI histograms. A complete comparison of each cell ISI histogram against all others is then performed by treating each

histogram as a multidimensional space, where each bin represents a single dimension. The Euclidean distance between each pair of cells as then calculated.

To construct the spike response, the top ten fittest model solutions for each adaptation are taken. The excitability patterns of each of these solutions are the calculated, where the half-life of each spike response element is normalised by the mean firing rate of the model solution. An example is shown below in Figure 5-15.

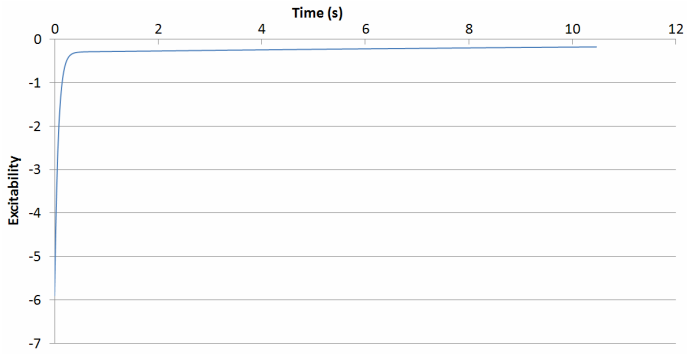


Figure 5-15 Excitability pattern of a Random VMH cell. This includes the summed effects of the HAP, DAP, and AHP.

To better examine both long and short term affects of the summed exponential decays an exponentially increasing X axis is applied. To focus on the area around 0 excitability, where the shape of the excitability pattern has the greatest effect, the excitability is scaled by the natural log of the graph when above 1, and the negative of the natural log of the graph when below -1, or more formally with Equation 5-3. This results in Figure 5-16.

$$E_S = \begin{cases} \ln(E), E > 1 \\ E, 1 > E > -1 \\ -\ln(-E), E < -1 \end{cases}$$

Equation 5-3

The resulting graphs from the top ten solutions of each adaptation are then averaged. In a manner similar to the ISI histogram comparison, the excitability graphs for each cell is treated as a point within multi-dimensional space, and the Euclidean distances between all graphs are calculated.

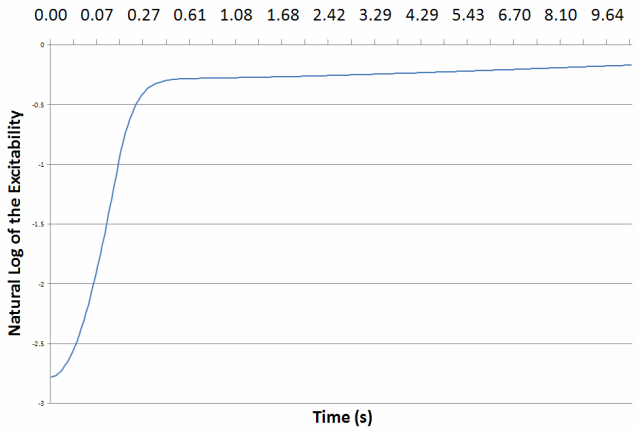


Figure 5-16 Skewed, but more informative version of Figure 5-15. Both the X-axis and Y-axis have been scaled to accentuate the effects of the all three exponentials at different timescales.

We now have two sets of fully compared cell data, one from the ISIs that are taken directly from the cell recordings, and the other derived from the control parameters of the model that represent the spike responses. For excitability to be used as a way to explore cell behaviour there needs to be some correlation between these two sets, in so far as cell pairs that are close together in one set should also be close together in the other and visa-versa. The following scatter graph plots the Euclidean distances of both methods for each cell pair against one another.

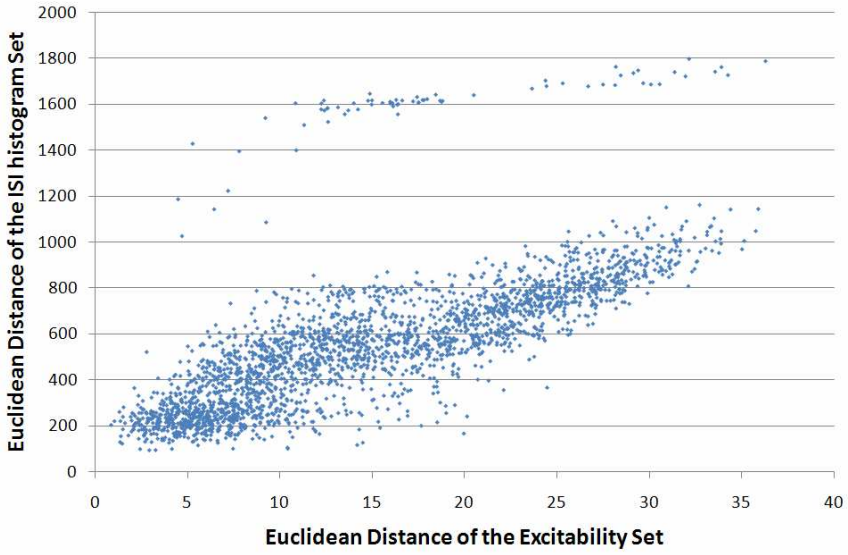


Figure 5-17 Scatter graph of Euclidean distances between all cell pairs. The Euclidean distance between pairs of cells were calculated via two methods, one that analysed the raw spike event data, and the other that analysed the spike response curve derived from the model after adaptation. Correlation between these two measures suggests a robust, and hence not overly complex model.

The majority of the cell pairs fall in a line giving an indication of correlation, but there is a small subset that is significantly removed from the main group. This inconsistency is caused by a behavioural outlier from the Doublet set, which interestingly was the best fit for the doublet cells, see Figure 5-9. The extreme Euclidean distances are, in this case, caused by the scaling of the ISI histograms. Because the amount of data is variable, all ISI histograms have their magnitude normalised such that the sum total of all the bins are equal. In the case of this cell, the first couple of bins are very large, followed by the remainder of the bins which are extremely low. As a result of the normalisation process, the larger bins get even larger, causing huge Euclidean distances between this and all the other ISI histograms. To properly analyse the correlation between the two sets, all cell pairs that included this outlier were removed. This resulted in Figure 5-18.

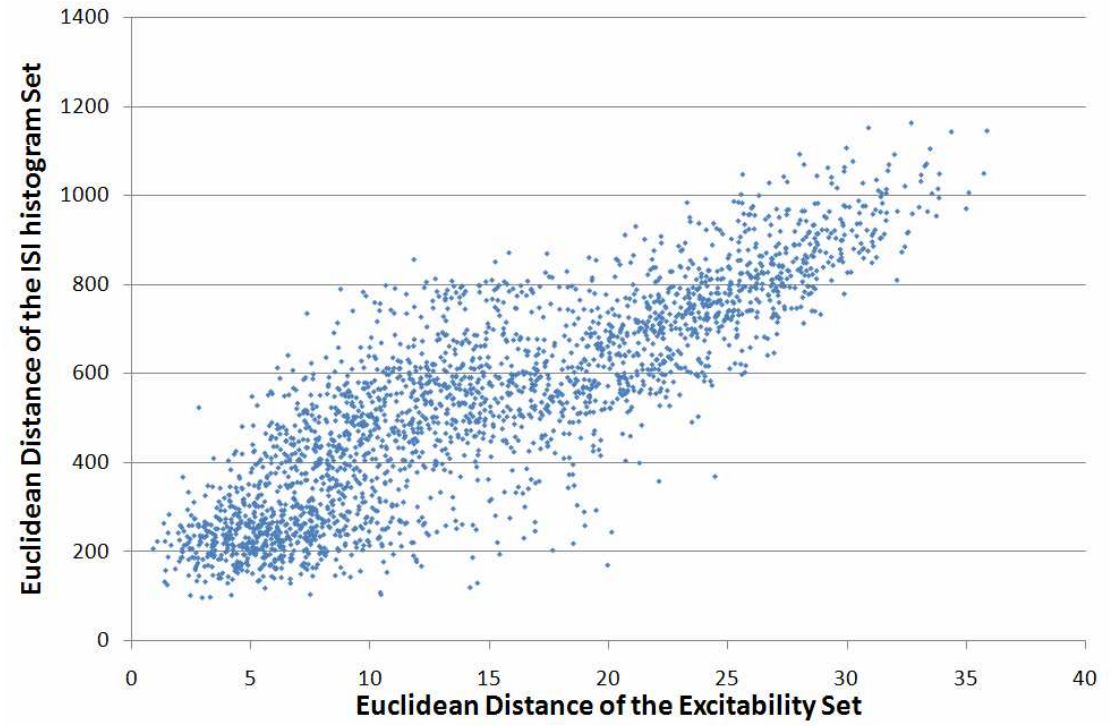


Figure 5-18 Scatter graph of Euclidean distances between all cell pairs except the Doublet outlier. The two sets are correlated, which suggests model robustness and simplicity.

The Pearson product-moment correlation coefficient was then calculated for the resulting distribution, and was found to be 0.86, which suggests a strong correlation between both sets.

A strong correlation suggests that spike response can be used to explore cell behaviour and that the model, which is based upon this concept, is robust insofar as similar excitability patterns produce similar ISI behaviours, which indicates simplicity.

5.5 Results – Extrapolation

The following sections examine the spike responses of the four pre-defined groups in an effort to identify the cause of specific cell behaviours. The spike response for this model consists of a mixture of three summed exponential decays, representing the HAP, DAP, and AHP of the target neuron. The model spike responses that are shown are those used in Section 5.4 to calculate robustness and simplicity, and are hence skewed along both the X and Y axis to better accentuate the effects of the all three exponentials at different timescales and magnitudes.

Each of the following sections corresponds to a group of neurons pre-classified by their behaviour by (Sabatier and Leng, 2008). The main graph in each section presents all the normalised and averaged excitability patterns from this group. It is important to note that although the majority of spike responses in the following graphs show a trend, there are a few odd patterns that do not entirely fit. This is for several reasons, firstly the cells were pre-defined into these groups, and some of them can be considered behavioural outliers, secondly, in a few cases, the adaptation method performed poorly, usually due to lack of data, and therefore did not entirely capture the cell's behaviour. Therefore the excitability pattern may not be an accurate depiction of that class of cell.

This section introduces the concept of sustained excitability, which is the cause of spike adaptation (Benda and Herz, 2003). Sustained excitability represents the base level of excitability experienced by a cell caused by the accumulation of the spike responses when the cell fires at its mean firing rate. This measure is the most accurate for Regular cells, as the variance about the mean is reduced, but it is also interesting to analyse when exploring the other cell classes. The majority of cells exhibit a negative, or inhibitory, sustained excitability that is caused primarily by the summation the AHP components. The exception to this observation is a subset of the regular firing cells. Sustained excitability essentially raises or lowers the spike response relative to the

threshold, and hence can greatly affect the firing pattern by determining how close the spike response is to the threshold.

It is important to note that trends discussed in this section have not been tested experimentally, but do pose some interesting questions regarding the structure of the cells within the VMH. During this section it is shown that several of the behaviours from (Sabatier and Leng, 2008) that were not adapted to during this chapter could still be generated through the strengthening and weakening of various subsections of the model's spike response. This process essentially moved the model from one region of the parameter space to another, suggesting that these neurons may have a generic but flexible neural structure.

5.5.1 Doublet Cells

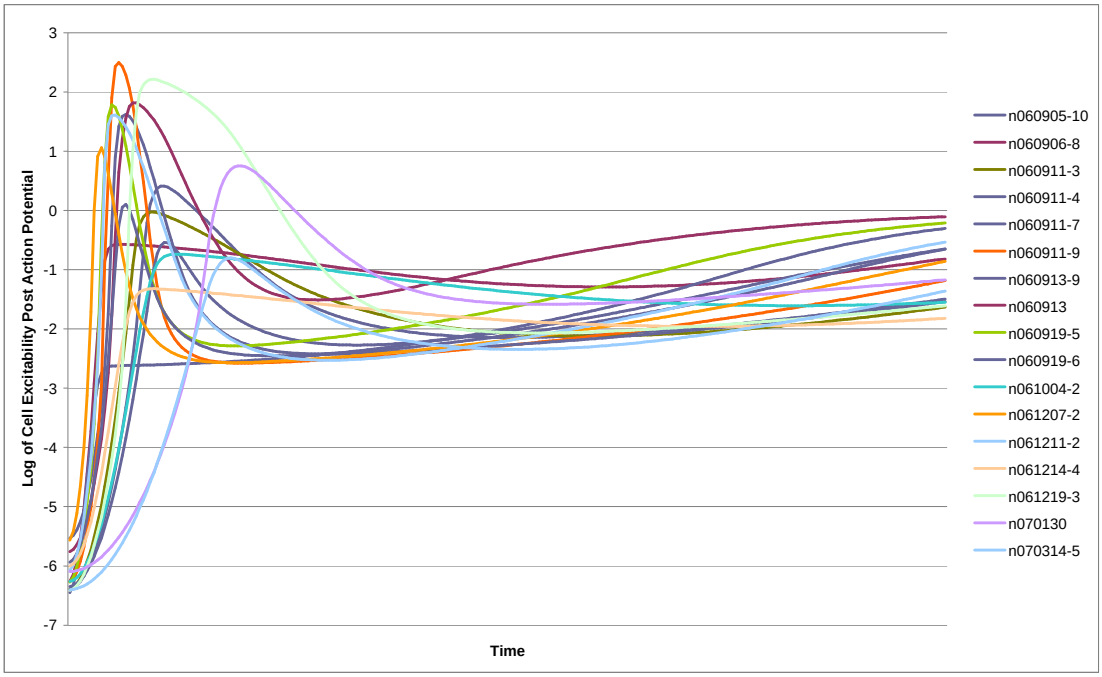


Figure 5-19 Seventeen doublet cell spike responses. Each spike response is the average of the 10 fittest solutions found through parameter estimation. The spike responses suggest that doublet behaviour is created by a large but short HAP followed by a fast DAP and an AHP to suppress further cell reactivation.

Doublet cells seem to be driven by a short but large HAP followed by a fast DAP and long suppressing AHP, to create a greatly increased chance of cell firing immediately

after an action potential. The AHP is responsible for then suppressing this excitability, and determines the ratio of doublet to multiplet activity.

When the DAP is suppressed in a version of the model that has been pre-adapted to a Doublet cell, the doublet behaviour ceases, and the cell acts more akin to a Random cell, Figure 5-21. If the AHP is greatly reduced in a similar way, then the cell exhibits bursting behaviour, or long multiplets, as it takes the summation of many more excitability patterns to suppress the increased chance of firing provided by the DAP summation, Figure 5-22.

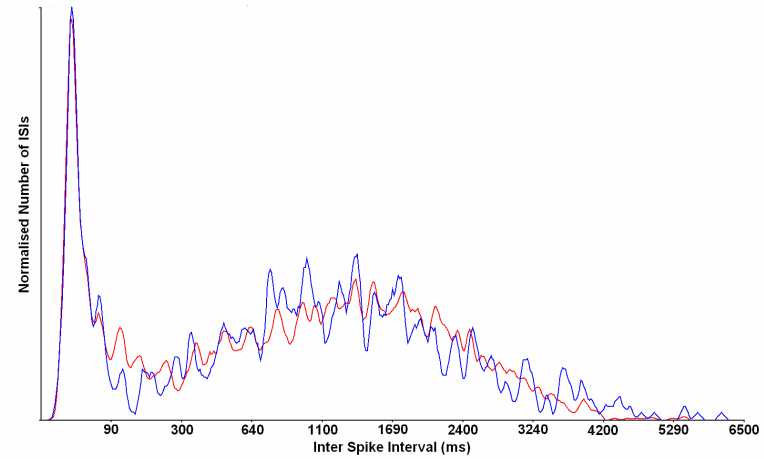


Figure 5-20 Best fit to a Doublet cell. This image is used to differentiate the effects of changing the magnitude and/or half-life of aspects of the model spike response in Figure 5-21 and Figure 5-22.

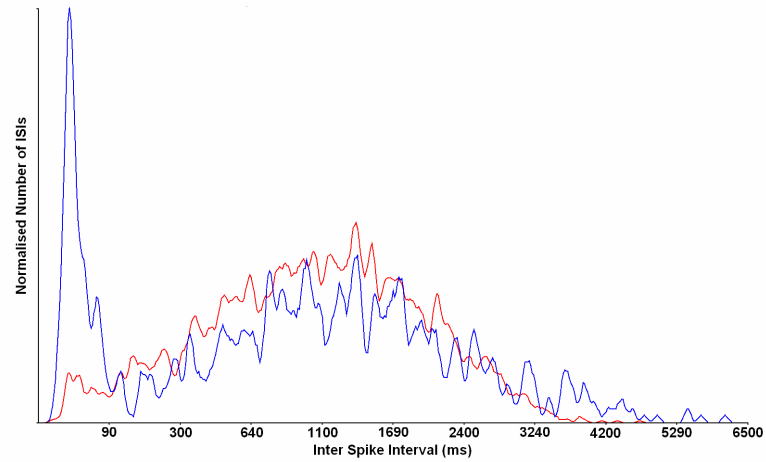


Figure 5-21 Best fit to the Doublet cell in Figure 5-20 with the DAP magnitude then set to 0. This ISI histogram shows that a DAP is the main (but not only) requirement of doublet behaviour. It is interesting to note that even though the DAP has been removed, the beginnings of doublet activity is still apparent. This is due to the extremely small half-life of the HAP.

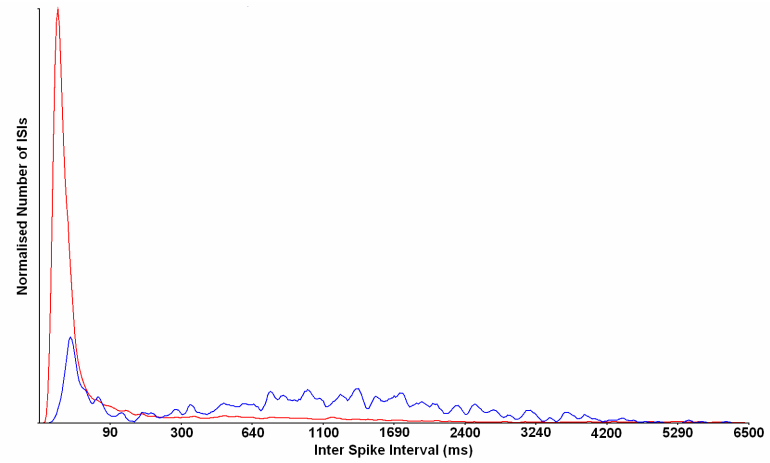


Figure 5-22 Best fit to the Doublet cell in Figure 5-20 with the AHP magnitude greatly reduced. It was not possible to set the magnitude to 0, as the cell fired to quickly to be analysed. The model now fires in bursts of approximately 40 action potentials. This shows that the AHP is responsible for controlling the number of cell reactivations.

5.5.2 Regular Cells

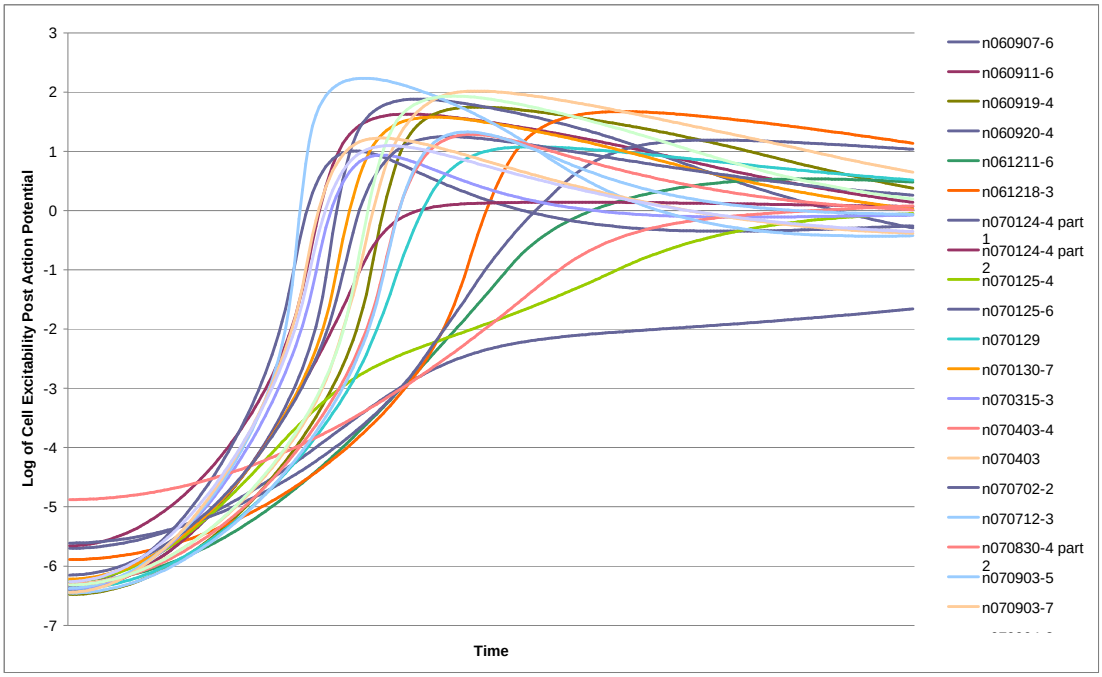


Figure 5-23 Twenty regular cell spike responses. Each spike response is the average of the 10 fittest solutions found through parameter estimation. The spike responses suggest that regular firing is created through a spike response that has both a HAP and DAP with long half-lives relative to the resulting mean firing rate. This chart is not long enough to show the presence of an AHP, which has an extremely long half-life compared to the mean firing rate, and exists for nearly all cells.

The spike responses of regular cells show a trend for a large HAP and DAP, in both magnitude and half-life relative to the mean firing rate. In nearly all cases, the HAP and DAP are accompanied by a long AHP (too long to be shown in the above graph). This acts as a limiter on the firing rate, and when suppressed allows the summation of a positive excitability plateau that causes a large increase in firing rate along with a reduction in ISI variance, which is only limited by the relaxation period enforced by the HAP. When in this state, the spike response is essentially elevated, such that the cell will fire well before the peak of the spike response. Because the gradient of the excitability pattern at this point is steep, the variance of the ISI is reduced.

It can be concluded that the regular cells can be sub-divided into two functional groups. The first are those that, although regular, are still heavily influenced by the input noise. This is because the cell exhibits a negative sustained excitability and is therefore self suppressing. The second group are those which show a positive sustained excitability. These cells are less susceptible to the effect of the input noise, where in extreme cases no noise is required to sustain firing because the positive excitability is self-sustaining. None of the above cells show this extreme behaviour, but some exhibit significant positive excitability.

5.5.3 Random Cells

Random cells are characterised by a short and small HAP followed by a long AHP with little or no DAP. This simple arrangement places a small limit on maximum firing rate, but leaves the remainder of the ISI distribution to be determined solely by the sustained negative excitability caused by the summation of the AHP, and the Gaussian noise source that represents the cell's synaptic input.

A reduction of the HAP half-life or magnitude results in the model producing near doublet activity[6], thereby showing a strong link between these two cell types, Figure 5-26. Increasing the HAP half-life produces an ISI distribution with a hazard that

[6] After discussions with N. Sabatier, the cell class was determined to be "Doublet-Broad" not "Doublet", which is one of the other five cell types classified in (Sabatier and Leng, 2008).

shows a longer rise time, relative to the mean firing rate, towards the plateau, which results in a distribution more akin to a Broad cell[7], Figure 5-28. If the AHP magnitude is suppressed, the models mean firing rate increases while the rise time of the Hazard does not, Figure 5-27. This increase in firing rate compared to Hazard rise time results in the model exhibiting behaviour that leans towards Broad activity. The Hazard rise time now constitutes for a third of the Hazard function, but its shape remains largely unchanged.

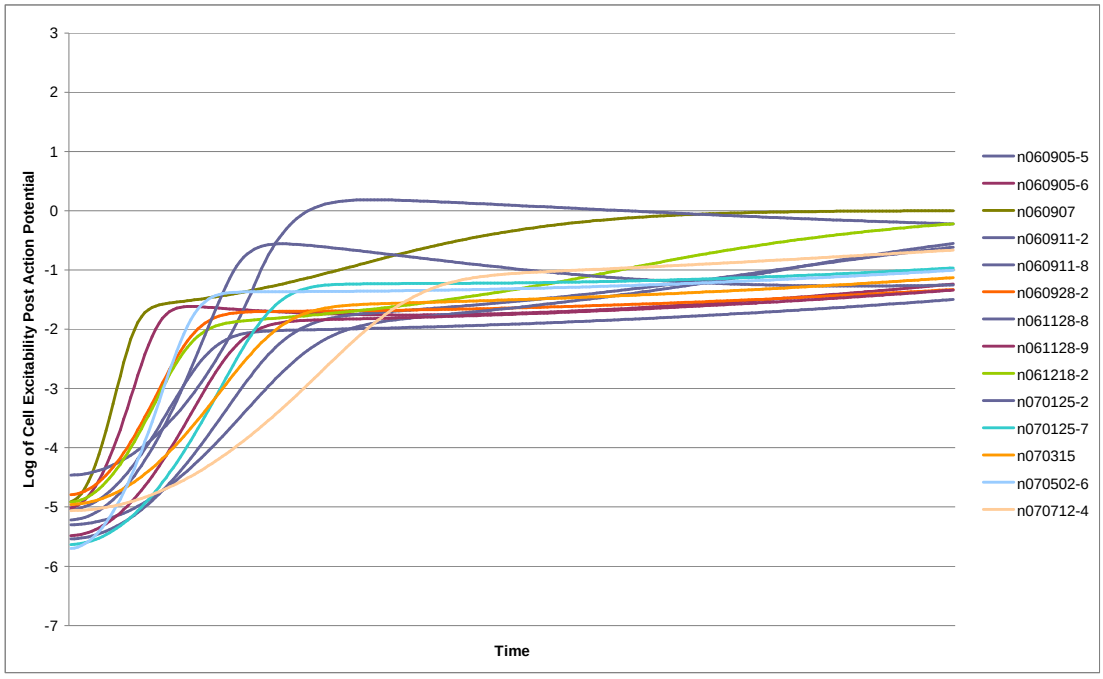


Figure 5-24 Fourteen random cell spike responses. Each spike response is the average of the 10 fittest solutions found through parameter estimation. The spike responses suggest that random cell behaviours is a product of a small HAP with short half-life, and an AHP with a long half-life.

[7] Again N. Sabatier identified the resulting behaviour to be that of a long tailed type 2 cell, another of the other five cell types.

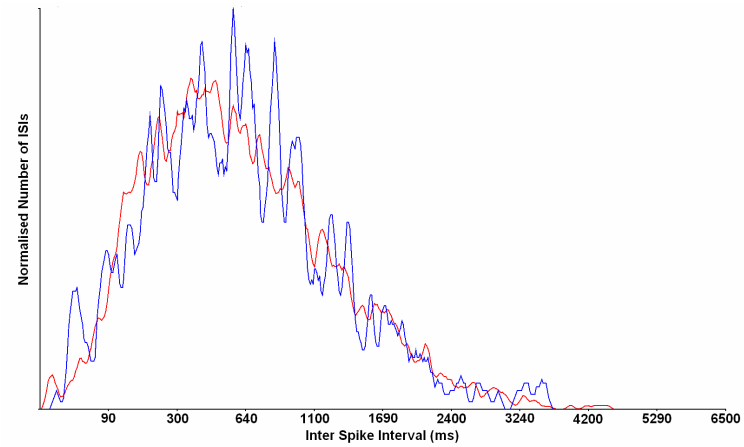


Figure 5-25 Best fit to a Random Cell. This image is used to differentiate the effects of changing the magnitude and/or half-life of aspects of the model spike response in Figure 5-26.

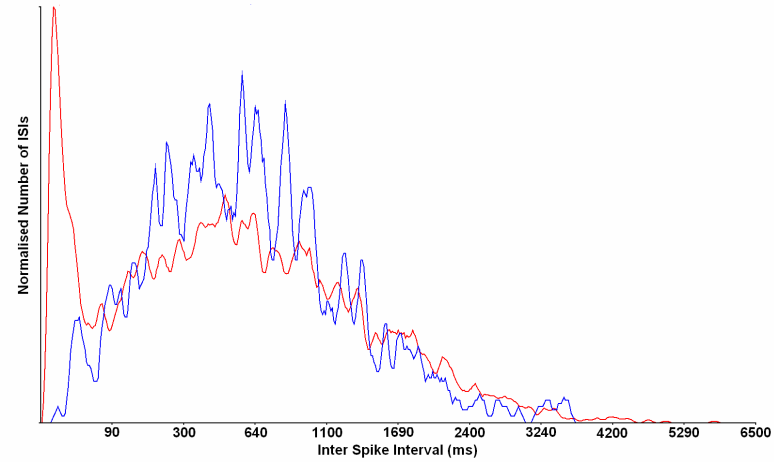


Figure 5-26 Best fit to the "Random" cell in Figure 5-25 with the HAP half-life greatly reduced. The resulting "Doublet" like behaviour is in fact more closely related to "Doublet-Broad" cells classified in (Sabatier and Leng, 2008).

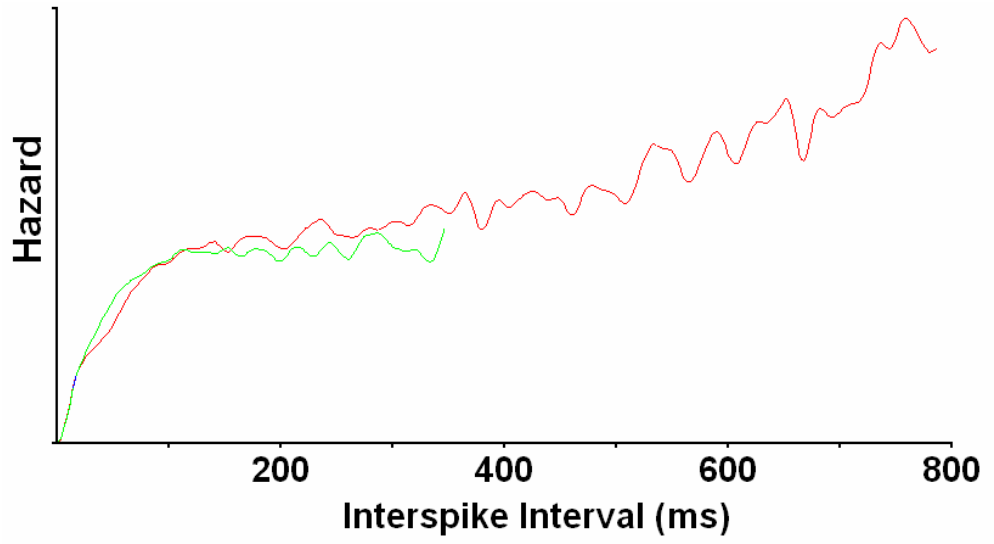


Figure 5-27 (Red) Hazard of best fit to a "Random" cell alongside (Green) the same configuration except with a suppressed AHP. This second Hazard has been scaled to show the similarity between Hazard rise times. This cell is much closer in behaviour to "Broad" cells.

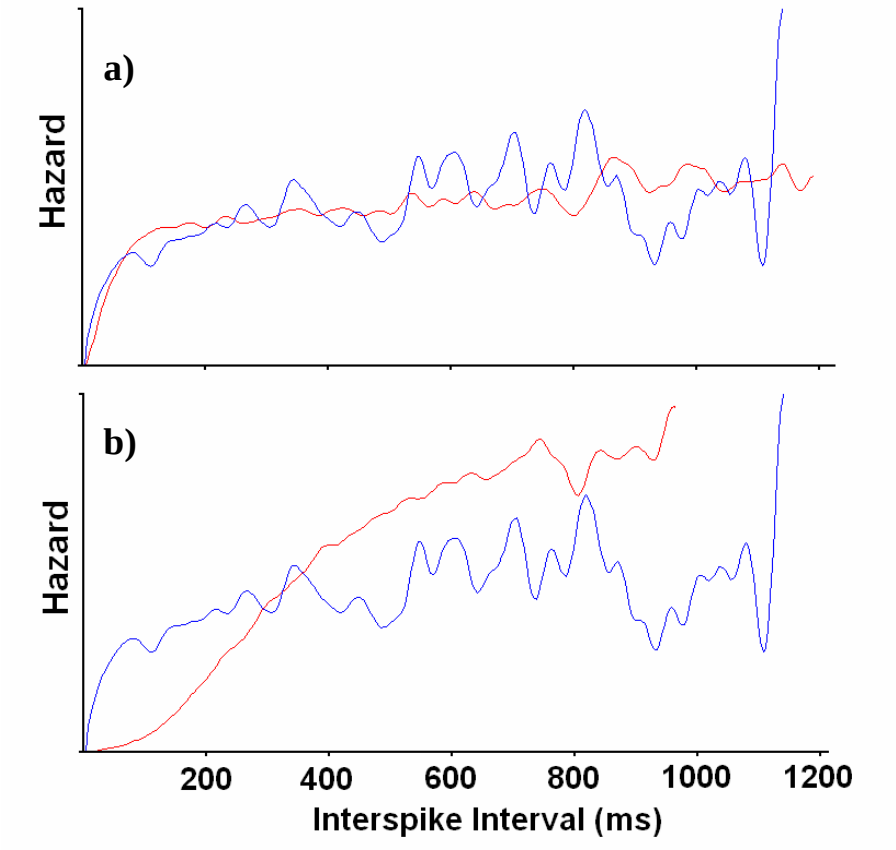


Figure 5-28 a) Hazard of the best fit to a Random cell; b) Hazard of the best fit to the Random cell in a) except with the HAP half-life greatly increased. This produces the broader "long tail type 2" behaviour defined in (Sabatier and Leng, 2008).

5.5.4 Broad Cells

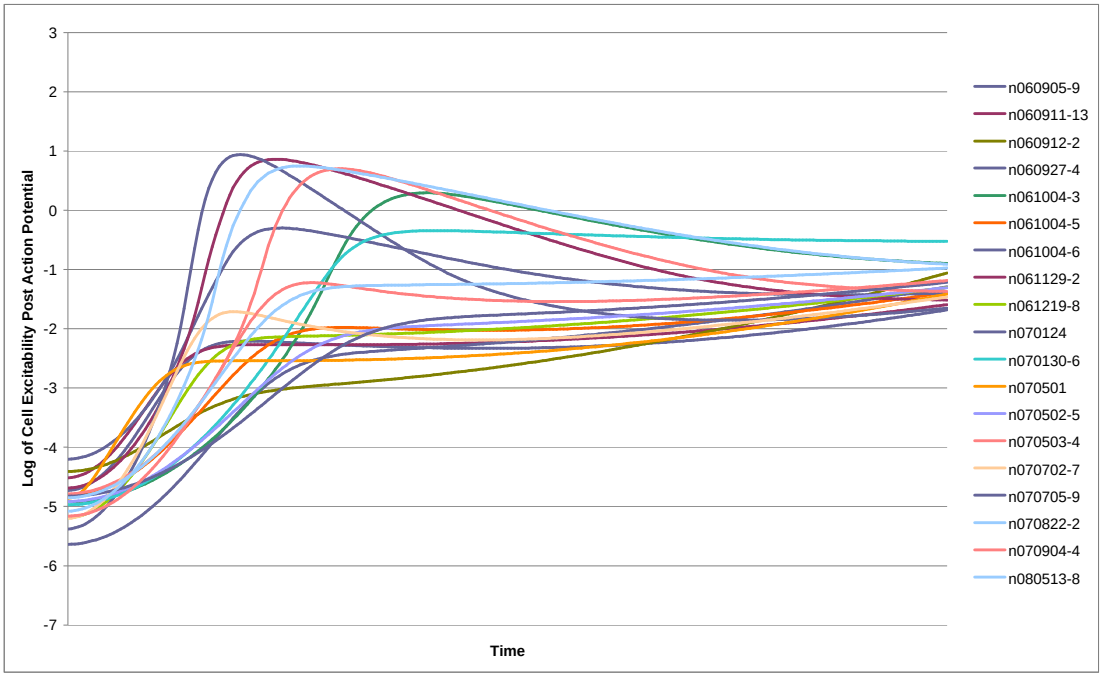


Figure 5-29 Nineteen broad cell spike responses. Each spike response is the average of the 10 fittest solutions found through parameter estimation. It is difficult to deduce the cause of broad cell behaviour from this graph due to the range of spike responses that are thought to be caused by adapting to non-stationary data.

Broad cells are similar to Random cells. As shown in the examination of Random cells, Broad cell behaviour can be created through increasing the half-life of the HAP. The excitability patterns of the Broad cell set are complicated by the fact that many of the cell recordings include drifting, or near phasic cell behaviours, as shown in the rate traces in Figure 5-30.

These traces cause an increase in the width of the ISI histogram peak that slows the rise of each cells Hazard. Because the model is not capable of creating these long term fluctuations, the ISI histogram of these cells require the addition of a DAP component to recreate this pattern as a constant behaviour. Ultimately, traces that include drifting behaviours should be filtered to enforce a picture of homeostasis to adapt to. However, for many of the traces this would reduce the amount of available data to unacceptable levels, hence these cells were instead removed. The resulting distribution of spike responses provided a much clearer picture.

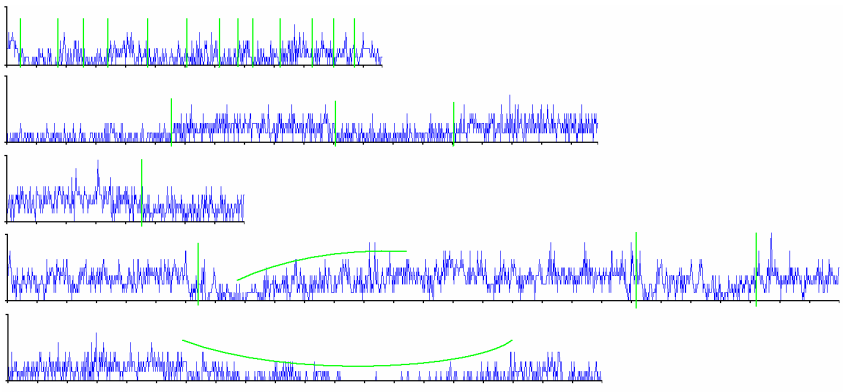


Figure 5-30 Rate traces of cells with drifting (non-homeostatic) behaviours. Green vertical lines indicate quick behavioural changes; Sweeping horizontal lines shows gradual behavioural changes.

There are two main differences between the Random cells shown in Figure 5-24 and the broad cells below. The HAP has a slightly longer half-live, whilst the AHP's diminishes, which corroborates with the results of the parameter manipulation of the Random cells.

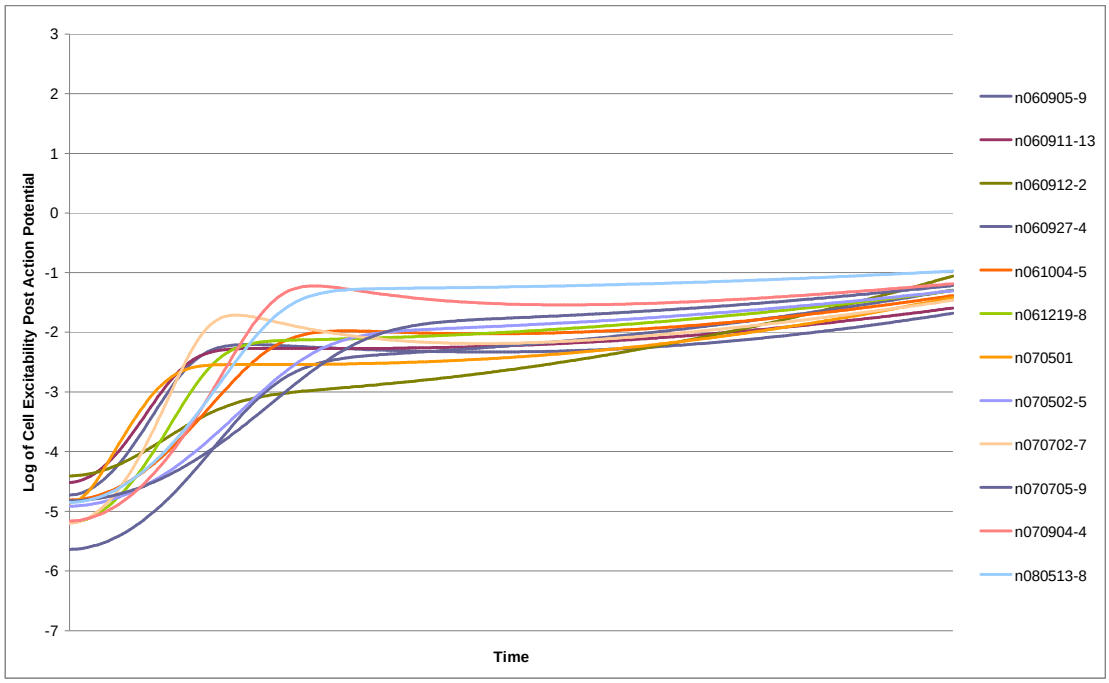


Figure 5-31 32 Twelve broad cell spike responses. Each spike response is the average of the 10 fittest solutions found through parameter estimation to broad cells with stationary behaviours. The spike responses suggest that broad and random cells are very similar in that they both require a relatively small HAP with a short half-life. However, the difference between the two is characterised by the general reduction in half-life of the AHP component. This reinforces the hypothesis drawn from the experimental results presented in Figure 5-27.

5.6 Summary

This chapter introduced as a candidate for a class of model that can be used modularly for rapid iterative model design, a basic excitability-based model. The model was introduced, both formally and structurally, and analysed in order to determine its viability. The model's *Plausibility*, *Explanatory Adequacy*, and *Interpretability* are presumed (Section 3.1), as the model's subcomponents are representations of measurable cell characteristics, and the design is based heavily on the well-established spike-response model (Section 3.1.6). The analysis instead focused on d*escriptive adequacy*, and robustness, which was used as an analytical method to determine if the model has viable *simplicity*. The final section explored the model's ability to extrapolate new behaviours, and hence make hypotheses regarding the causes of cell behaviour.

This analysis found that the model is extremely flexible, as it is able to reproduce behaviour similar to 72 VMH cells, which have been pre-defined into four behavioural subsets: Broad, Doublet, Regular and Random. The conclusion from this test is that the model has good *descriptive adequacy*. To test robustness, and hence s*implicity*, a correlation between the spike responses, derived from model control parameters, and the ISI histograms of each cell were performed. This test examines the null hypothesis that the shape of the spike response has no bearing on the resulting behaviour, with multiple vastly different excitability patterns being able to reproduce similar behaviours. If this is the case then the model can be presumed to be overly complex. Correlation was found between the similarity between cells when analysed through both the ISI histograms and spike response, thereby falsifying the null hypothesis and providing evidence for the model's robustness and *simplicity*.

The final section of this chapter investigated the shape of the excitability patterns in relation to target cell behaviour. Explanation is provided for all four groups, with the Doublet, Random and Regular cells showing distinct spike responses from one another. From the spike response of the model, it was hypothesised that Doublet behaviour is caused by a small HAP combined with a rapid DAP, with a long AHP controlling the ratio of doublet to multiplet activity. Random behaviour is caused by a the combination of a fast HAP, which stops the immediate cell reactivation, followed by a long AHP that

summates to create a negative sustained excitability, which controls the mean firing rate of the cell. Regular cell behaviour is caused by a slower HAP than the Random cells combined with a DAP. In nearly all cases, this DAP is balanced by an AHP that limits the creation of a large positive sustained excitability. It is this AHP that controls the mean firing rate, as well as the variance of the ISI, by dictating the effect the noise source (incoming action potentials) has upon the cell. A large sustained excitability essentially elevates the spike response, such that the only factor limiting immediate reactivation is the HAP.

The Broad cells exhibit very similar excitability patterns to the Random cells. Many of the broad cells showed long term drifting behaviour, which could be caused by changes in the magnitude of incoming synaptic noise. The model is unable to alter the magnitude of the noise during the simulation, so other parameters were instead altered to compensate, resulting in the formation of a DAP in some cases. This problem, combined with the natural similarity between Broad and Regular cells when normalised by their mean firing rate, meant that significant differences between the two sets were difficult to isolate. However, experimentation suggests that the difference between Broad and Random cells is caused by a general increase in HAP and decrease in AHP half-life.

In terms of the model being a candidate for rapid iterative model design, the basic version has shown itself to be capable of reproducing a wide range of cell behaviours through the inclusion of only three exponential decays and a Gaussian noise source. Because of the modular design of the model it could be easily modified to provide more or less functionality. This is done in the following chapter, where this model is extended in an effort to test a hypothesis regarding the long term phasic firing patterns of vasopressin releasing neurons.

The examination of spike response is a useful tool for the analysis of cell behaviour, in part because it is a more intuitive and ultimately more visually appealing way of depicting the differences in cell behaviour. However, this information does not represent the magnitude of the noise source, or the effect the mean firing rate has upon the overall cell excitability when combined with the spike response. Future work should look to investigate these aspects, as well as isolating new functional blocks that

can be added or substituted into the model to allow analysis of a wider range of cells, under many experimental protocols, some of which are discussed in the future work section in Chapter 7.

Chapter 6 Case Study: Development of a Model of the Vasopressin Releasing Neuron

This chapter presents an overview of the development of a computational model describing the behaviour of the vasopressin releasing neurons found within the Supraoptic nucleus of the Hypothalamus (Section 2.1). This model was developed for three reasons. The first was to study the iterative model design process using the rapid evaluation method presented in Chapter 4. The second was to test a hypothesis regarding the phasic nature of the vasopressin releasing neurons. The third was to test a hypothesis about the ease with which the basic excitability-based model can be extended to produce a wider range of cell behaviours.

The model evaluation method presented in Chapter 4 functions by thoroughly searching the model's parameter space for specific behaviours. As a result of this, a simple model structure can be used as a starting point for the iterative model design process, as the adaptation process can be relied upon to find the required behaviours. This promotes a minimalistic approach to modelling, where extra functionality, and hence complexity, is only added if the correct behaviour cannot be found. Ultimately, as the model fitting is being performed by the user, it is still possible for an overly-complex model to be produced, especially if the user is zealous in their pursuit of the perfect fit. However, with a suitable understanding of the effects of noise and the importance of *simplicity*, using this framework should allow a user with minimal modelling expertise to utilise modelling to explore their hypotheses.

This chapter is split into seven sections, followed by a summary. Section 6.1 details the iterative design method, describing the structure of the rapid evaluation method and the makeup of the fitness function used to guide the evolutionary algorithm. Section 6.2 provides a behavioural overview of the vasopressin releasing neuron. Sections 6.3 to 6.6 present, in order, the various stages of the design process that make up this trial and error modelling approach. At the end of this design process a fully functioning computational model of the vasopressin releasing neuron, which is both

computationally and structurally simple whilst being representative of the underlying biology, is presented (Section 6.7). This model is able to recreate a wide range of pre-recorded vasopressin neural behaviours, which supports the original hypothesis about the cause of phasic behaviour. Furthermore, through the exploration of the final model's control parameters, it was found that the model can reliably reproduce the three archetypal behaviours of vasopressin releasing neurons (continuous; sparse sporadic; and phasic) by the modulation of a single parameter. Additionally, it was found that three parameters, which all contribute to the behaviour of a single biologically plausible model variable, can be used to completely describe the phasic behaviour of the model, which appears to be largely independent from the short term ISI behaviour.

6.1 The Rapid Iterative Design Method

This chapter explores the trial and error-based iterative design philosophy. While a graphical user interface to facilitate the construction of the neural model would be ideal, this was outside the prevue of this work, and so all model changes were handled manually, and as such took substantially longer to implement. As a result, each modification of the model was carefully scrutinised before implementation. The ultimate aim is to enable "rapid" iterative model design because it removes the need for this scrutiny, where model changes may be implemented on a hunch, because the cost model evaluation is low. While our rapid model analysis method is fast in comparison to the single-CPU approach, the model and corresponding measures of fitness are complex, forming a highly multimodal, multidimensional parameter space that is difficult to explore. Hence, the majority of the time was spent adapting the model to the various biological targets, to isolate behaviours the model had difficulty replicating.

6.1.1 Rapid Evaluation

Each adaptation comprised of 30 to 60 generations of the evolutionary algorithm. The varying number of generations reflected the difficulty the algorithm encountered when performing the adaptation. In most cases, the adaptation was performed multiple times with different fitness weights, to emphasise specific characteristics of the cell behaviour. In this way, specific aspects of the model were tested, by attempting to better mimic individual cell sub-behaviours. It may seem strange that both *in-vivo* and

in-vitro adaptation targets were used to evaluate the same model, but this was done because it is presumed that both sets of data were generated by the same biological structure.

However, cells recorded *in-vitro* tended to exhibit strange behaviours that can only be recreated by outlying parameter sets, and sometimes not at all. This seems counter intuitive at first, as *in-vitro* is considered to be a more controlled environment, and hence less prone to producing data sets which present outlying behaviours. However, it is important to note that vasopressin releasing neurons are part of a complex feedback mechanism, and hence their removal from this environment will remove the majority of the feedback pressures that guide each neuron's phasic behaviour (chemical gradients and the shape of the random synaptic noise distribution).

This makes the adaptation to *in-vitro* behaviours doubly difficult, as not only are these behaviours less characterisable, but the parameter space being searched must also be much larger to include areas that may produce the more exotic *in-vitro* behaviours. However, it is often these strange behaviours that highlight the limitations in the model. This is discussed in more detail in following sections.

6.1.2 Fitness Evaluation

Several measures of fitness were utilised to guide the evolutionary process, allowing quantisation of the similarities between both the long and short term behaviours of a target and a solution. The majority of these measures were based on the standard electrophysiological practices detailed in Section 2.3, and used in Chapters 4 and 5.

The measures of fitness used to analyse short term behaviours are all derived from the inter-spike-interval (ISI) histogram, using the methods described in Section 5.2, producing the averaged mode, mean, coefficient of variance, and the RMS Noise and Selected Envelopes of the ISI and Hazard histograms

To characterise and quantise the fitness of longer term behaviours, such as the phasic firing pattern and the noise within the burst, a combination of seven measures of fitness was used, of which two were only required for the in-vitro adaptations. At the core of all these measures is the rate data, which counts the number of events per second.

During software simulation, the rate data is stored as one long array, with its size defined by the length of the simulation. However, memory constraints within the GPU architecture prevent the data being collected in this way, and instead, pre-processing must be performed during simulation to condense the data. As the same analysis procedures must be performed on both software and GPU, to allow a fair comparison between target and simulation, the original array will not be used as part of the fitness evaluation, though it is required to produce visualisations of the rate data.

The first measures that can be calculated through accumulation, are the mean and standard deviation of the period of the bursting behaviour. Next, the average noise throughout the bursting periods of the simulation is found by averaging a four bin envelope across the length of the simulation, ignoring silence periods. Then, because the initiation time of each burst is known, as it was used to find the period of the bursting behaviour, all events in the 1st through 500th second of each burst can be summed together to get the average bursting behaviour. The RMS error of the initial 30 bins and the entire histogram are then calculated. This method clearly defines the initial peak of activity. The rate at which the average burst activity falls provides insight into the mean and variance of burst lengths, with a slow fall from a plateau designating a high variance, as shown in Figure 6-1.

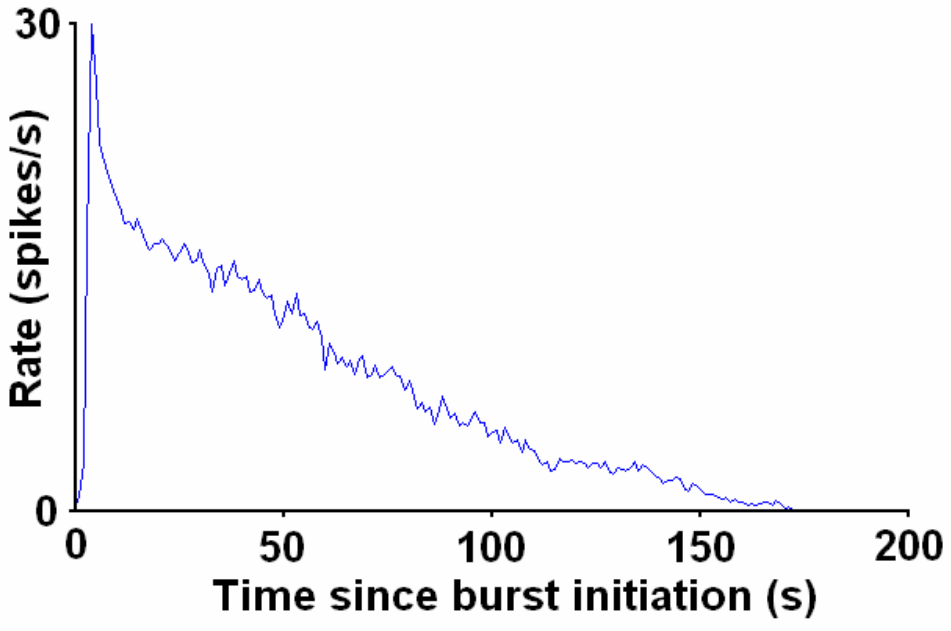


Figure 6-1 Average bursting behaviour of a vasopressin releasing neuron.

The final set of measures of fitness are designed to better quantify the "flavour" of the noise seen within the bursts. When inspected at a resolution of 1s, the rate data reveals repeated patterns in the cycle time of the noise, essentially wavelets. This behaviour was initially characterised by Fourier analysis, but because the system is heavily noise-driven, this did not perform well. Instead, an approach of searching for wavelets (from a high rate to a low rate and back up again) was used, and which creates a histogram calculated from their peak to peak times. The mean, mode, and RMS error of the histogram in comparison to the target were then derived. Finally, it was found that the algorithm required further guidance when trying to fit behaviours with a high silence period variance (*in-vitro* data), and thus the mean and variance of the silence length was calculated similarly to the bursting period. The cause of this difficulty was a fault in the model, which was corrected during its fourth iteration; in other circumstances, the further guidance might not have been required.

6.2 The vasopressin releasing neuron and our Pre-recorded Data sets

Vasopressin cells, which are described in detail in Section 2.1.1, are primarily responsible for maintaining blood osmolarity by releasing varying quantities of vasopressin into the blood stream. This is a vital part of the endocrine system, which signals the kidneys to retain water. Although the spiking behaviour of the neurons is well documented and characterised, the underlying system that promotes this phasic behaviour is not. This section reports the testing of a hypothesis that provides an elegant explanation of the underlying mechanism.

The rapid model analysis method introduced in Chapter 4 is used to develop the model, and hence test the hypothesis. This method requires a set of biological behaviours that are used as adaptation targets, which in this case were recorded both *in-vivo* and *in-vitro*, from vasopressin releasing neurons from the Supraoptic nucleus of the Hypothalamus, as described in Section 2.2.3. Some examples of the *in-vivo* pre-recorded data are shown below in Figure 6-2.

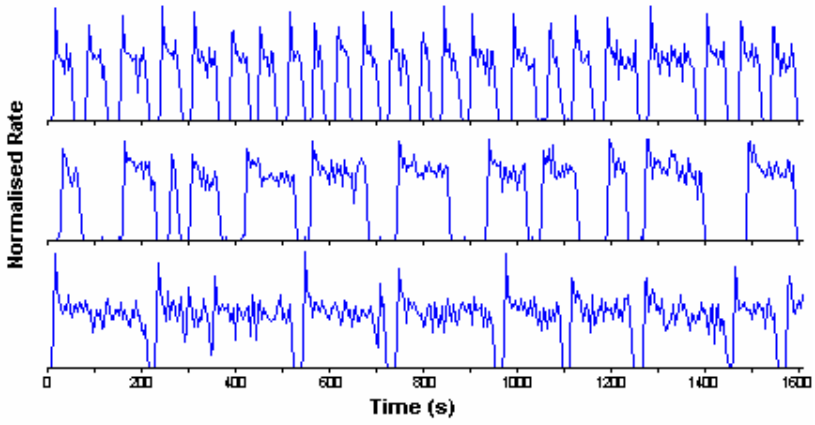


Figure 6-2 Pre-recorded *in-vivo* timestamp data formatted into four second rate bins

In general, the *in-vitro* data tended to be more sporadic, producing larger and more varied silent periods, and when in a burst, exhibited less noise. In many cases, the model was unable to find a good match to the *in-vitro* recordings due to lack of characterisable behaviours caused by short recordings.

6.2.1 A Brief Introduction to Phasic Firing

The phasic behaviour of vasopressin releasing neurons has been well documented (Brown and Bourque, 2006), both *in-vivo* (Arnauld et al., 1975, Bourque and Renaud, 1991, Brimble and Dyball, 1977) and *in-vitro* (Kirkpatrick and Bourque, 1996, Greffrath et al., 1998), but to date, the majority of modelling work has concentrated on the explanation of *in-vitro* data utilising both single- (Roper et al., 2004) and multi- (Komendantov et al., 2007) compartmental Hodgkin-Huxley-like (Section 3.1.1) models. Through this line of modelling, which has been corroborated by experimentation, the phasic behaviour of *in-vitro* vasopressin cells has been attributed to the creation and destruction of a plateau potential that, for *in-vitro* cells, is self-sustaining. When *in-vitro*, this plateau is thought to be created through the summation of post-action potential DAPs, which eventually results in a self-sustaining plateau potential that exceeds spike threshold (Andrew and Dudek, 1983). The plateau has been shown to be sensitive to the k-opioid peptide Dynorphin, which is co-released with vasopressin, slowly inhibiting the DAP over the course of a burst, causing the eventual collapse of the plateau that results in the end of the burst (Brown and Bourque, 2004).

While for *in-vitro* recordings this plateau is above the threshold and is therefore self-sustaining, this is not the case for cells recorded *in-vivo*, where the plateau does not reach the threshold, and therefore requires spontaneous EPSPs to fire (Brown et al., 2004). This difference greatly affects the distribution of recorded action potentials collected by the different methods. Those collected *in-vitro* are greatly influenced by the intrinsic properties of the cell, and hence are largely deterministic, whilst those collected *in-vivo* are stochastic, where the random synaptic inputs have a greater affect (Sabatier et al., 2004).

But how does this difference affect the theory of plateau creation and destruction? The fact that there is a plateau is undeniable, as it has been measured experimentally, but its cause, at least in-vivo, has been brought into question as Durie, (2007) was unable to create a model that maintained a plateau through DAP summation alone, and ultimately resorted to a black box approach that utilised a simple state-machine to describe the phasic mechanism. This ultimately brings the current theory of burst termination into question because, if DAP summation is not enough to maintain a plateau, then it is also unlikely to be the sole reason for burst termination. That dynorphin regulates phasic activity by acting as a slow inhibiter is indisputable, as it has been proven experimentally, but ***how*** this happens is still unclear.

Sabatier and Leng have experimentally tested for the presence of a slowly increasing, inhibitory, activity dependant cellular mechanism, *in-vivo*, dubbed the "slow inhibition," that slowly increases the probability that the cell will fall out of its heightened state. To test this theory, a phasic cell was periodically stimulated during its burst period. They found that if a cell was near the end of a burst, the stimulation would have a significant chance of causing the cell to fall silent after the corresponding burst of activity. This reinforces the slow inhibition hypothesis, as a burst of activity would also increase the magnitude of the slow inhibition variable, and thereby increase the chance of falling from the heightened state. Durie used the concept of slow inhibition as the input to her state-machine-based black box model, resulting in realistic secretion behaviour (Durie, 2007). Chapter 4 of this thesis used a modified version of Durie's model as the test case for a comparative algorithmic analysis. This modified version, which included an AHP component, was able to produce realistic neuron firing patterns,

though the model neurons were still identifiable from the recorded data due to the steepness of their burst initiation and termination.

One of the goals of this chapter is to build upon Durie's work, and test a hypothesis regarding a more biologically plausible explanation of phasic behaviour. This hypothesis is still based around the existence of a "slow inhibition" that ultimately is the main contribution factor to burst termination, but differs from traditional *in-vitro* models by not presuming that the plateau potential is constructed through DAP summation alone. Instead, it is hypothesised that the phasic behaviour recorded *in-vivo* is caused by activity and voltage dependant ion channels, which in essence create a bifurcation on the membrane. Additionally, although the intrinsic properties of the cell, such as the HAP, DAP and AHP, will greatly effect the firing pattern, and may even contribute to burst initiation and termination, they are not solely responsible for the phasic behaviour.

6.3 Model Iteration 1 – 1D Bifurcation

This chapter seeks to explain the underlying mechanism that controls the phasic behaviour of vasopressin releasing neurons in a biologically plausible way. It is hypothesised that the phasic behaviour is controlled by voltage dependant ion channels, which are in turn suppressed by sustained activity.

As with Chapter 5, the Heaviside function, and a function that converts from half-life to a multiplier based on time are, used repeatedly. They are repeated here for convenience. These functions are important because, much like the basic excitability-based model, the majority of this vasopressin model consists of decaying variables that undergo step increases. The formal descriptions within this chapter are written with discrete instead of continuous notation because this better reflects the behaviour of the simulation. However, a continuous representation is also supplied in Appendix B.

Equation 6-1 Heavyside Function

$$H[n] = \begin{cases} 0, n < 0 \\ 1, n > 0 \end{cases}$$

Equation 6-2 Half-Life to Multiplier Conversion Function

$$\tau[x] = 1 - e^{\frac{-\ln(2)\Delta t}{x}}$$

In keeping with the parsimonious trend throughout this thesis, the basic excitability-based model presented in Chapter 5 is utilised as the core of the new model (this is also used, in part, as the core for Durie's model). In fact, the only difference between the basic excitability-based model presented in Chapter 5, and the first iteration presented in this section, is the replacement of the basic membrane decay mechanism for a voltage and activity dependant one, which is essentially a one dimensional bifurcation.

From a modelling point of view, this is the initial step of a top down approach, where as simple a system as possible is implemented, knowing that it will probably not work exactly as expected and will need to be fixed. In this instance, a basic and proven model is adopted and combined with a simple bifurcation function in an effort to provide the basic mechanism for the desired functionality. This is the proof of concept stage, and so all that is required is an initial working prototype.

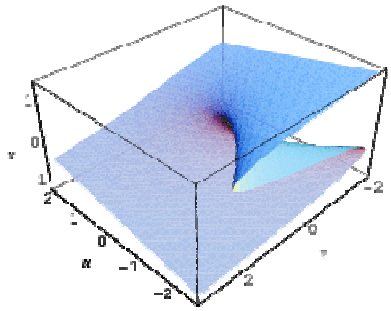


Figure 6-3 Cusp Catastrophe; $F(x,u,v) = x^4 + ux^2 + vx$

The bifurcation is the derivative of the Cusp Catastrophe function shown in Figure 6-3, where '*u*' is fixed, taking a slice of the derivative as shown in Figure 6-4a and defined by the following equation.

$$y = 4x^3 + 2ux + v$$ **Equation 6-3**

This curve, when applied so that x = excitability and y = the change in excitability with time, creates a system where any excitability will always drift to one of two points designated the rest or plateau excitabilities (Figure 6-4b). When in the plateau state the cell fires continuously, driven by the incoming noise, however, the chance of the noise causing firing when in the rest state is dramatically reduced

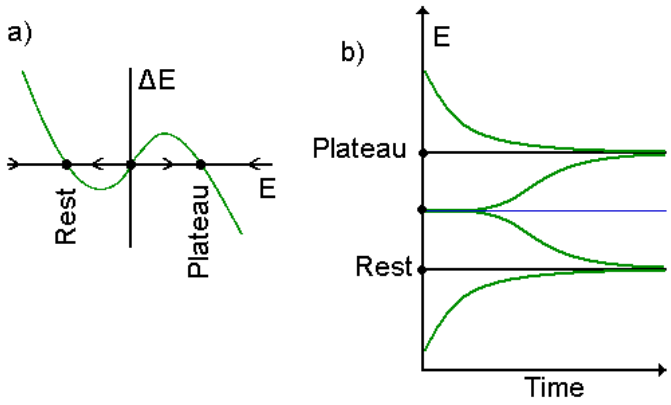


Figure 6-4 a) Derivative of the Cusp catastrophe function with a negative '*u*' and '*v*' set to 0. This is a 1D – Bifurcation with the change in excitability (ΔE) as a function of excitability (E); b) Resulting change in excitability (E) over time

The position of these states relative to 0 is controlled by the '*u*' parameter, and so this relationship can be reversed enabling us to define '*u*' from the distance between the rest and plateau excitabilities (E_{RP}) using the following equation.

$$u = -\frac{E_{RP}^2}{2}$$ **Equation 6-4**

Because the intention is to directly integrate the noise source with the bifurcation function, the gradient of the curve, which represents the decay rate of the membrane potential, must be scaled to bring it within a biologically plausible range of values.

$$S = \frac{\tau[M_{HL}]}{E_{RP}^2}$$ **Equation 6-5**

The amount the function is scaled by (S) is a derived parameter defined by; the length of time each simulation iteration represents (Δt); the distance between the rest and plateau excitabilities (E_{RP}); and the desired half-life of the function as it passes through 0 (M_{HL}), as shown in Figure 6-5, and described formally in equation 3.

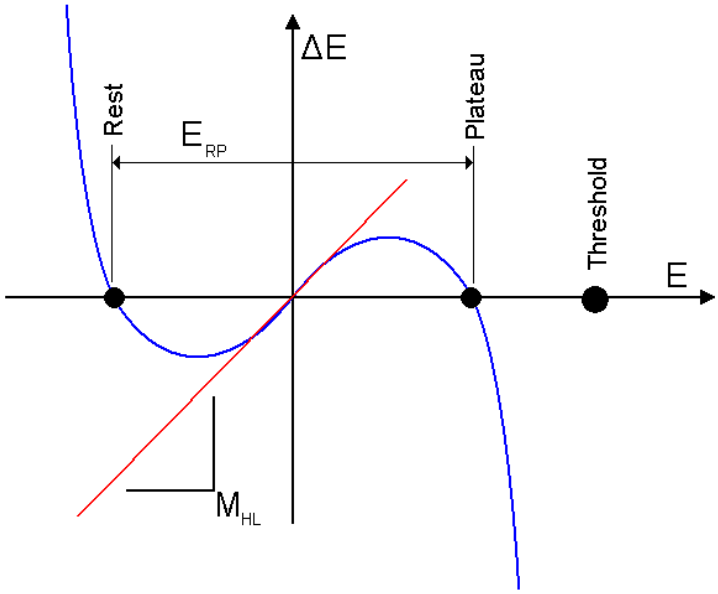


Figure 6-5 The function that replaces the simple decay in the basic excitability-based model; The red line indicates where the desired half-life of the membrane decay will correspond to. This half-life is realised by scaling the entire function with S.

The transition between states is facilitated through a modulated offset of the bifurcation (v in Equation 6-3) that is designated the "slow inhibition" variable. This variable is controlled by two parameters, a decay rate, and a step size which is applied whenever the cell creates an action potential. This causes the slow inhibition to build up over the course of a burst, and in turn depress the bifurcation; this is shown in Figure 6-6.

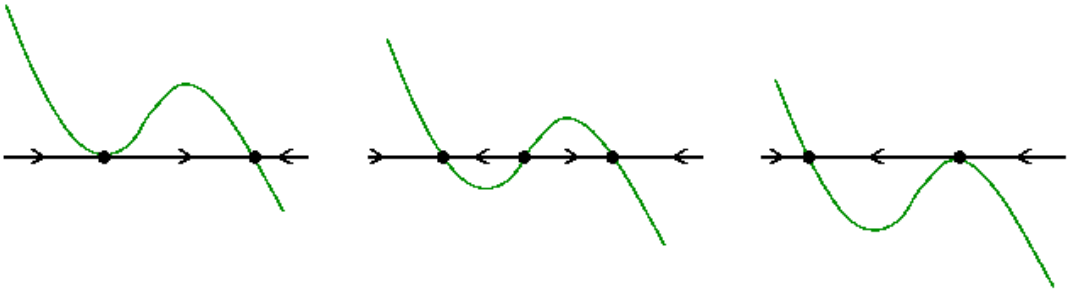


Figure 6-6 1D bifurcation with suppression caused by a slow inhibition variable. Left: bifurcation function with initial offset; centre → right: bifurcation suppressed by the slow inhibition.

As the cusp catastrophe function is depressed there is an increase in the probability that the noise will jump into the negative region of the function. This behaviour means that there can be great variation in cell excitability at the beginning of a burst, but it is highly

unlikely that it will stop the burst. As the burst progresses, there will be an increase in the severity of the response if the cell does stop bursting, and to a degree, the length of time the cell maintains a burst will contribute to the length of its silence period.

One last derived parameter is required to control the bifurcation, the starting point or offset. For all but the last iteration of the model, the starting point, i.e. slow inhibition = 0, is set at the state showed on the left of Figure 6-6 with the magnitude depicted in Figure 6-5. This offset is calculated relative to the distance between rest and plateau (E_{RP}), see Equation 6-6, and so must be scaled with the rest of the algorithm by S.

$$E_{Offset} = -3^{-\frac{3}{2}} E_{RP}^3 \qquad \textbf{Equation 6-6}$$

When combined with the remainder of the excitability-based model (noise source and post firing excitability pattern), the resulting model is structured as shown in Figure 6-7, or more formally below.

Noise Source Representing Synaptic Inputs

$$N = G.A_N$$
$$G = GaussianArray[R_U]$$
$$R_U = Uniform\_Random\_Number$$

Membrane potential bifurcation plus activity dependant slow inhibition (I)

$$\frac{\Delta E_M}{\Delta t} = -S\left(4E_M^3 + 2u.E_M + E_{Offset} + I\right) + N$$

$$\frac{\Delta I}{\Delta t} = I.\tau\left[D_{Inhib}\right] + S_{Inhib}\, H\left[E_C - E_{TH}\right]$$

Post action potential membrane excitability, HAP (H), DAP (D), and AHP (A).

$$\frac{\Delta H}{\Delta t} = H.\tau\left[D_{HAP}\right] + S_{HAP}\, H\left[E_C - E_{TH}\right]$$

$$\frac{\Delta D}{\Delta t} = D.\tau\left[D_{DAP}\right] + S_{DAP}\, H\left[E_C - E_{TH}\right]$$

$$\frac{\Delta A}{\Delta t} = A.\tau\left[D_{AHP}\right] + S_{AHP}\, H\left[E_C - E_{TH}\right]$$

Cell firing mechanism

$$E_C = E_M + H + D + A$$
$$SpikeEvent = H\left[E_c - E_{TH}\right]$$

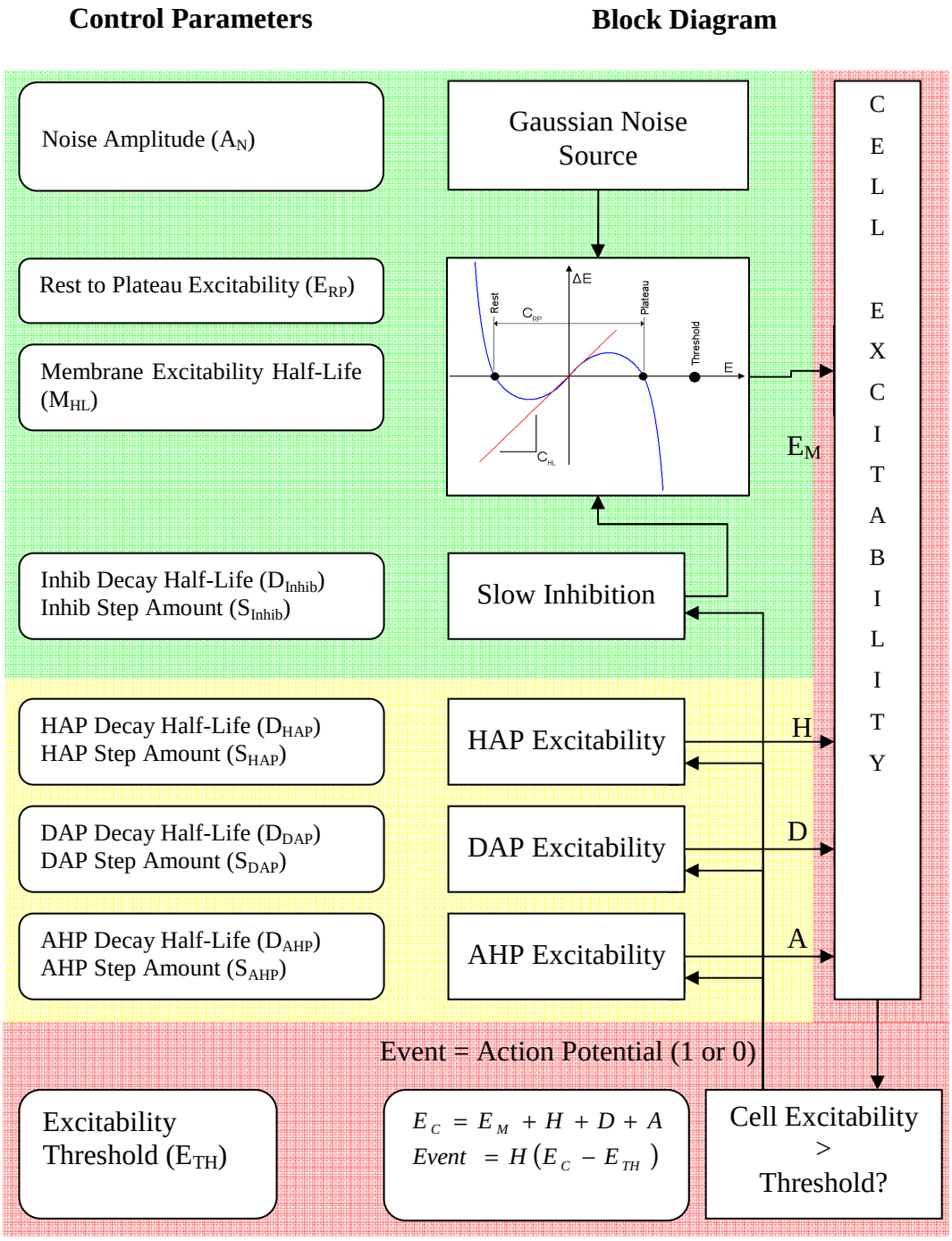


Figure 6-7 Structural overview of the first iteration of the vasopressin neural model

This model structure was easily adapted to produce phasic behaviour. However, it was found that realistic burst initiation and termination had to be sacrificed to produce the correct short term spike response. Although the correct shape was obtained, a peak of activity falling to a plateau (Figure 6-8), the rise and fall times at the start and end of

each burst were too abrupt, even when adapting to cells that took several seconds to start or finish a burst. Additionally, there was very little variability in the space widths.

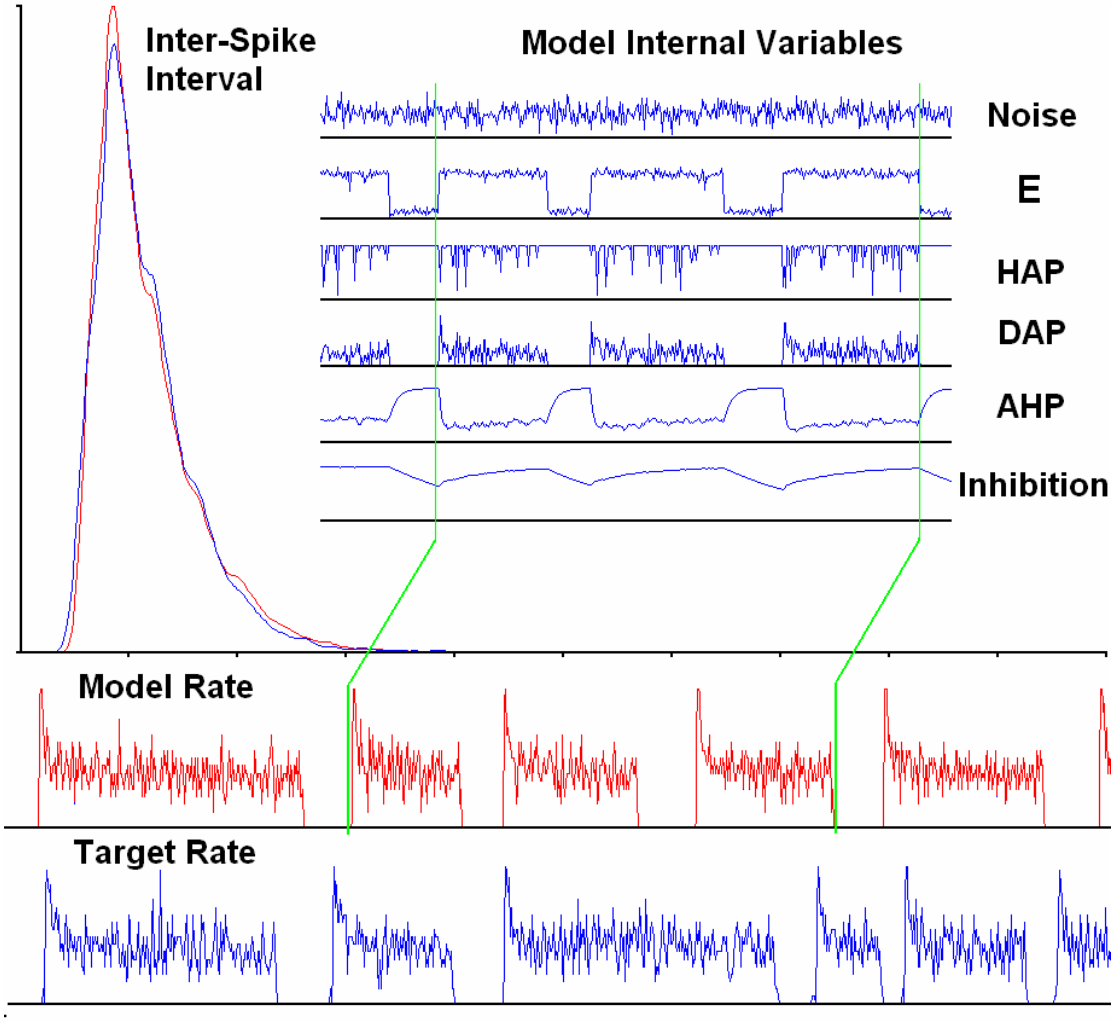


Figure 6-8 Vasopressin model v1 adapted to a vasopressin releasing neuron recorded *in-vivo*. The green lines show the correlation between internal variables of the model, and the output response. Note that the rise and fall of the model excitability is extremely rapid in comparison with the real recoded data. Additionally, although this is a good fit to the data, there is much less variation in silence duration, as well as the shape of the initial peak of activity.

When analysed, the internal variables of the model highlighted two interesting issues. Firstly, the abrupt burst initiation and termination was caused by the decay rate of the bifurcation as it crosses the origin, which is set by Equation 6-5. Because incoming noise was integrated directly with the membrane dynamic, the membrane decay rate was set such that it produced a realistic decay of the synaptic inputs (Noise source). For the basic excitability-based model this corresponded to a half-life of 10ms, which as

shown in Chapter 5, produced good results, and so the same value was adopted for this task. However, in this model this value was performing two tasks. The first was to decay the incoming noise, whilst the second was to control the transition between the two states of the bifurcation. With such a small half-life, the transition is always going to be sharp (~100ms rise time), and will produce nearly the same initiation behaviour at the start of each burst, as the noise source will have less of an impact, producing a more deterministic process. This also means that there is very little variability in the maximum rate seen at the beginning of each burst.

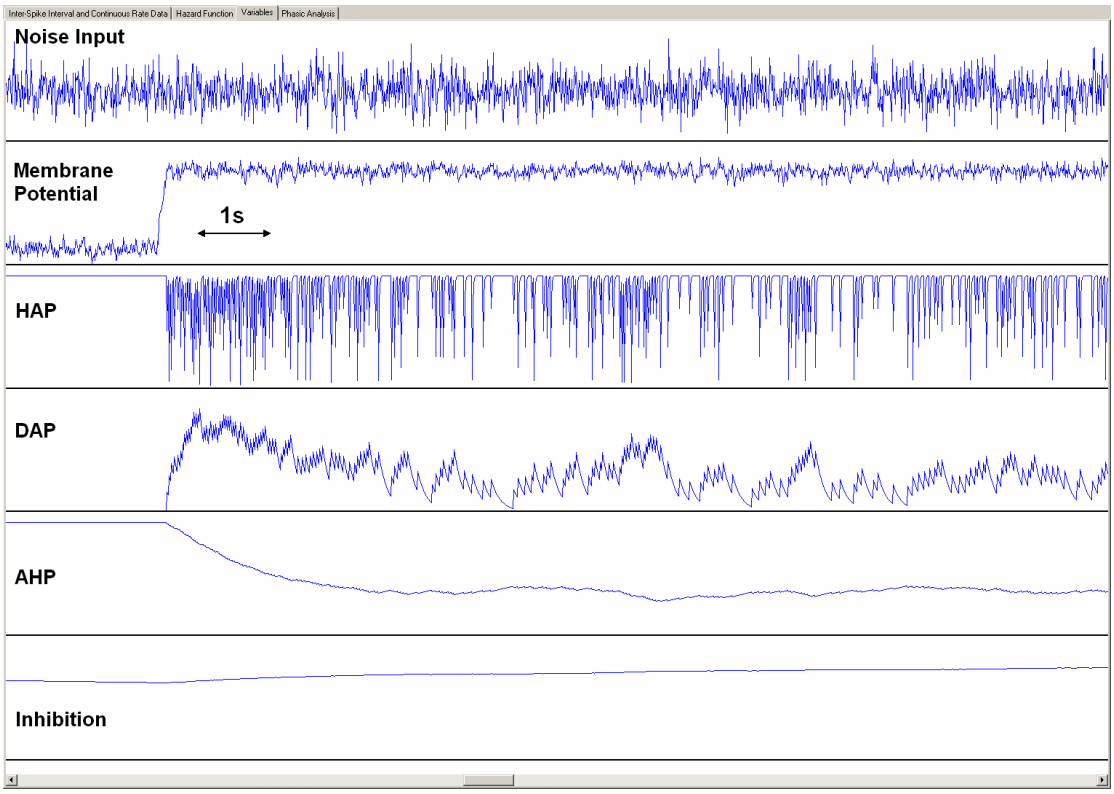


Figure 6-9 Internal variables of the first iteration of the model when adapted to the target depicted in Figure 6-8. Note the extremely rapid rise time of the membrane potential, as well as the spike adaptation caused by the DAP and AHP at the beginning of the burst.

The second issue was also linked with the noise source. During the optimisation process, the range of possible noise magnitudes had to be dramatically increased, with the fittest solutions tending to have very large noise magnitudes. It was found that the shape of the bifurcation was the cause of this requirement. Because the bifurcation is derived from a cubic equation, the response below the Rest and above the Plateau is also cubic in nature. This is problematic, as an exponential is linear in this domain,

where the amount that the excitability drops due to decay is relative to the current excitability, not the excitability[3], as shown in Figure 6-10.

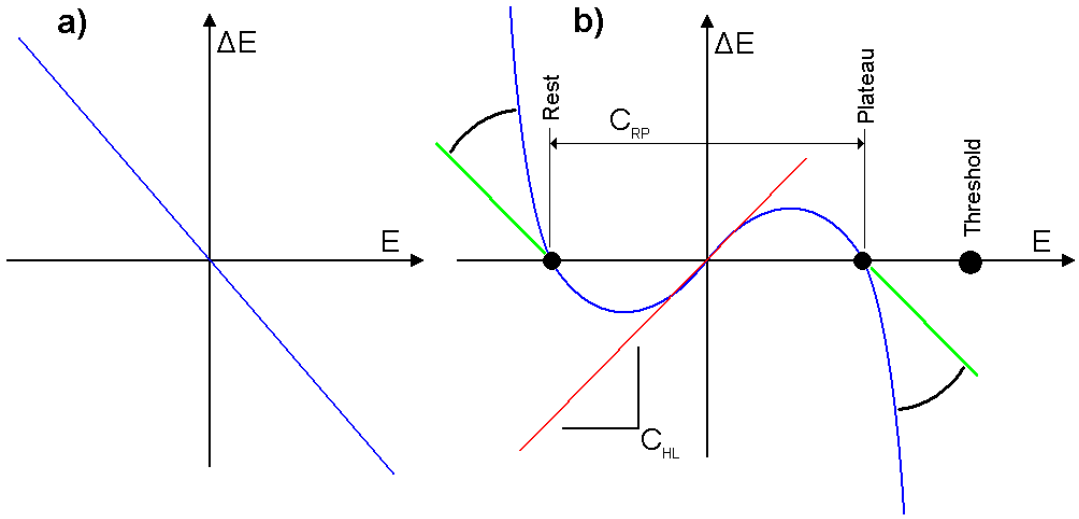


Figure 6-10 Membrane decay dynamic of the a) basic excitability-based model presented in Chapter 6. Note the linear increase ΔE with E. b) original bifurcation model. Note that on either side of the stable excitabilities ΔE has a quadratic association with E, where it should be linear.

Whilst this initial model was flawed, it did successfully produce vasopressin releasing neuron like behaviour, and could be easily adapted to produce almost a facsimile of any cell's short term behaviour (ISI histogram), though this is not unexpected as the basic excitability-based model has already been shown to be extremely flexible, being able to produce a wide range of continuous firing patterns. It was immediately apparent that the basic excitability-based model is extremely easy to extend because of its simple and modular nature. Just as easily as the modified membrane mechanism, the noise source could have been replaced with a more complex system, or additional excitability characteristics could have been added.

6.4 Model Iteration 2 – Compound function, and Noise Source Separation

The second iteration of the model answered the two issues raised in the previous section, namely the non-exponential decay of the bifurcation above the plateau and below the rest excitability, and the uncharacteristic rapid rise and fall in firing rate upon burst initiation and termination. The first of these issues is simple to address, as it is

possible to simply substitute a compound function for the original bifurcation, as shown in Figure 6-11.

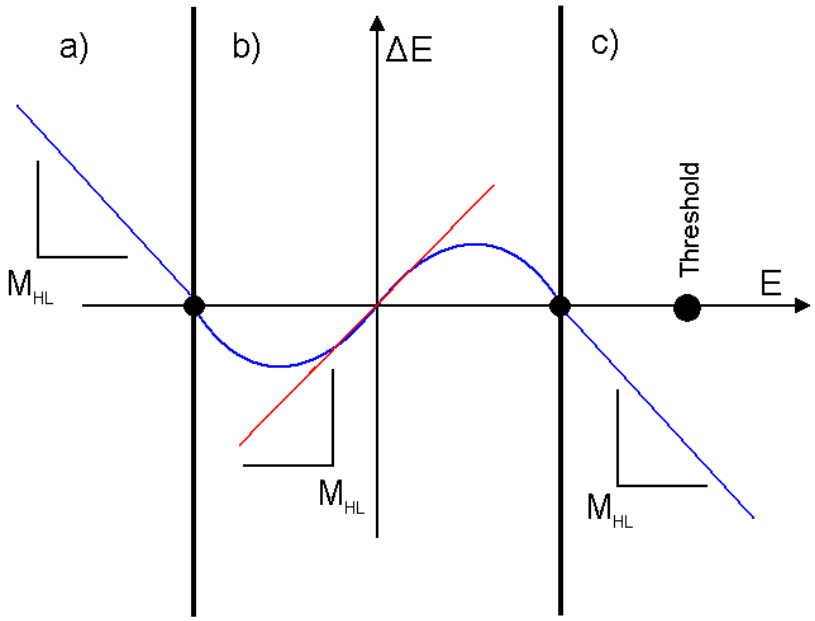


Figure 6-11 Compound function that replaces the original bifurcation; a) simple exponential that decays towards the rest state; b) bifurcation constructed in the same way as the original; c) simple exponential that decays towards the plateau state

Or more formally:

$$\frac{\Delta E}{\Delta t} = S.\left(E_{Offset} - I\right) + \begin{cases} -\left(E - \frac{E_{RP}}{2}\right)\tau\left[M_{HL}\right], & E \geq \frac{E_{RP}}{2} \\ -S.\left(4E^3 + 2uE\right), & \frac{E_{RP}}{2} > E > \frac{-E_{RP}}{2} \\ -\left(E + \frac{E_{RP}}{2}\right)\tau\left[M_{HL}\right], & E \leq \frac{-E_{RP}}{2} \end{cases}$$

Equation 6-7

This modified function required a much smaller noise magnitude in comparison to the difference between the plateau and the threshold excitability, as the force that gravitates the current state of the compound bifurcation function to either the rest or plateau states (sections a) and c) in the Figure 6-11) is now linear, not cubic.

The other issue that needed to be resolved was the problem relating to the state transition time. The first attempt involved using two different decay rates for the different parts of the compound function, as shown in Figure 6-12. However, this meant that the forces that maintain the bifurcation were uneven, with those between the Rest and Plateau excitabilities being much smaller than those outside. This meant that the Gaussian noise input would be suppressed unevenly; skewing the noise such that it would always favour a state transition, creating a phasic behaviour that was extremely uneven.

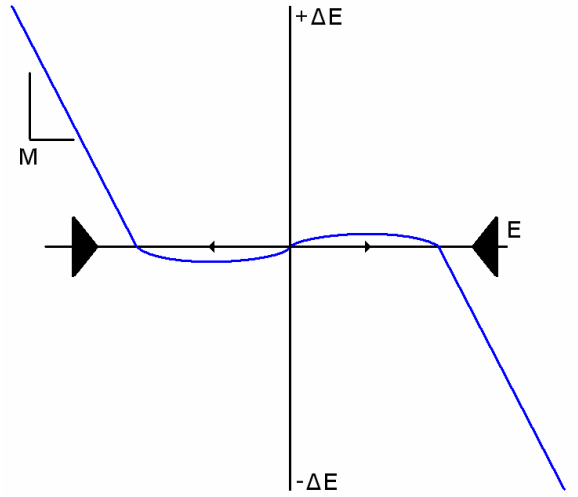


Figure 6-12 Compound bifurcation with differing decay rates. Size of arrows show the strengths of the algorithmic pressure at different parts of the algorithm.

Ultimately, the best way to solve the transition time problem was to separate the Gaussian noise, which represents incoming synaptic events, from the bifurcation. This would allow the noise to have a half-life separate from the bifurcation half-life, facilitating the control of both. This involves storing an extra variable to keep track of the current value off the summed and decayed noise. This value is then referred to when calculating the change in membrane excitability, but never fully integrated with it.

6.5 Model Iteration 3 – Full integration of membrane excitability

During the implementation of the noise separation it was realised that it should be possible to incorporate the spike response in the same way, not fully integrating it, but

letting it influence the overall change in excitability. The overall post influence excitability E_{inf} would then be used in place of E in the compound function, where:

$$E_{\inf} = N + H + D + A \qquad \textbf{Equation 6-8}$$

This seemed practical initially until upon simulation it was found that the inclusion of the HAP, which is a largely negative, caused the system to oscillate from below the Rest to above the Threshold with every increasing magnitude.

Removal of the HAP from the equation would only create a similar situation but in reverse, because the HAP is required to suppress the effects of the DAP for a short period of time after each action potential. Without this force, the resulting influence from the excitability pattern would seem more positive than it actually is, forcing the compound bifurcation function to respond with its own suppressing force, lowering the firing rate.

As a solution, a combination of the HAP, DAP and AHP, was used, where the full sum of the three is only applied to the E_{inf} when the sum of the DAP (+ve) and the AHP (-ve) is greater than the HAP (-ve), or more formally when the Gaussian noise is included.

$$V = D + A$$
$$E_{\inf} = H[V + H].(V + H) + H[-V].V + N$$

Or in Pseudo code:

```
V = D + A;
if  (V + H) > 0  then
        E_inf = V + H + N;
else if  V < 0  then
        E_inf = V + N;
else
        E_inf = N;
```

It should be noted that this is an arbitrary solution that represents the hypothesis that spike response affects long term phasic behaviour through the build-up of resonant behaviour. The solution does not aim to mimic any specific biological mechanism, and is only present to tie the spike response to the compound bifurcation function. This implementation presumes that the effects of the HAP have little or no direct effect on the phasic response. The advantage of including the spike response with the bifurcation function is that it allows the more complicated shorter timeframe firing pattern history to affect burst initiation and termination, in addition to the simple long term activity dependant “slow inhibition”. One of the key behaviours that is allowed by the inclusion of these variables, is the “resonate-and-stop” behaviour discussed in the following section. After the inclusion of the spike response with the influence variable, the model showed the ability to be adapted to produce near identical behaviour to a wide range of target behaviours.

Figure 6-13 is the best solution found when adapting this latest iteration of the model to one of the in-vivo cell behaviours. It shows all the measures of fitness that were used to guide the adaptation process, but ultimately a direct comparison of the rate data and ISI histograms are all that are required to see the success of the adaptation, as well as highlight aspects that the model did not entirely capture correctly.

The following sections focus on different aspects of the model, highlighting specific areas where the model performed well or badly, and suggesting reasons for its success and failure. At an algorithmic level, the functions of fitness described in Section 6.1.2 quantify behaviour as functions of ISI, Hazard, Burst profile etc, however, here behaviour is subdivided a sub-sections of the more visually appealing Rate traces. These subsections are designated: Burst initiation and termination; Rate modulation during bursts; Rate modulation outside of bursts; Burst and silence periods.

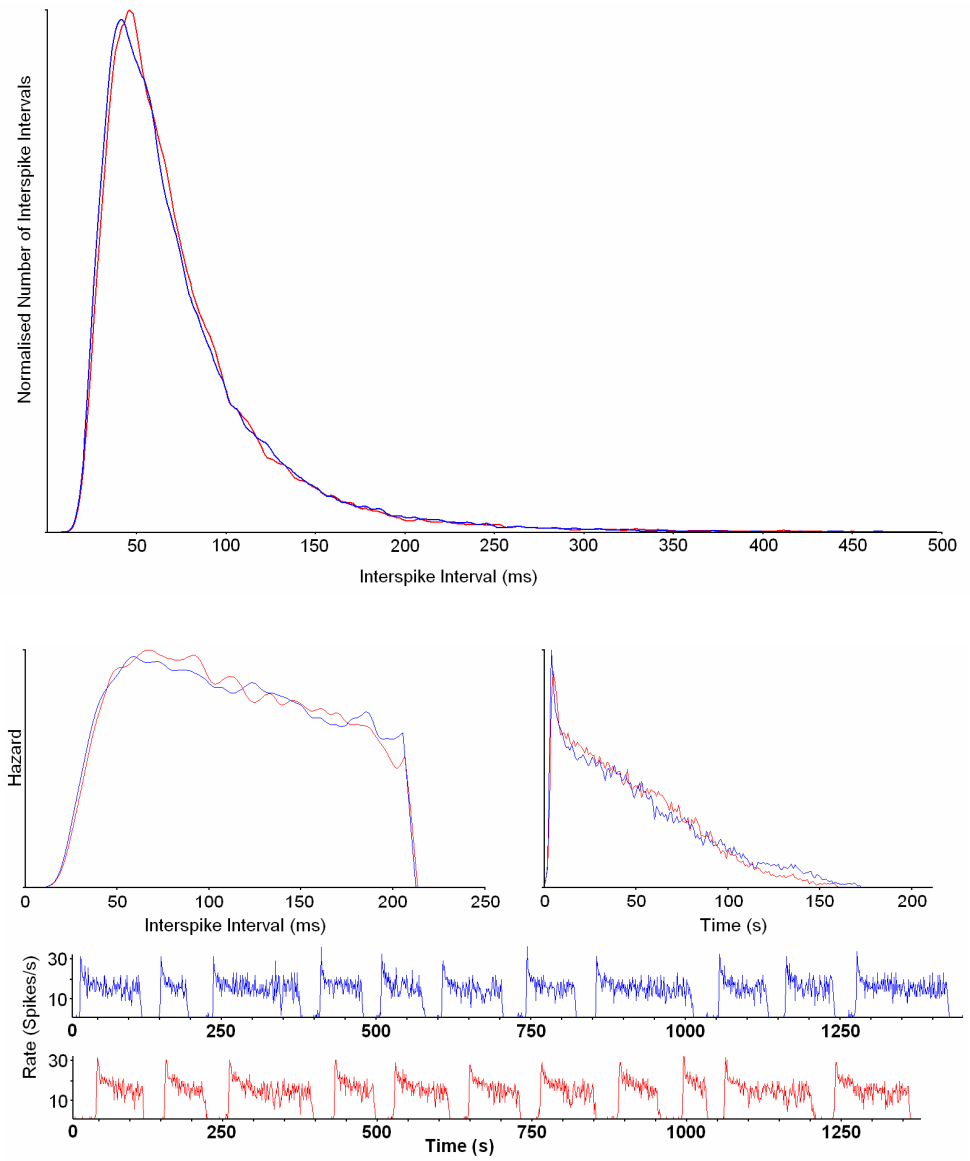
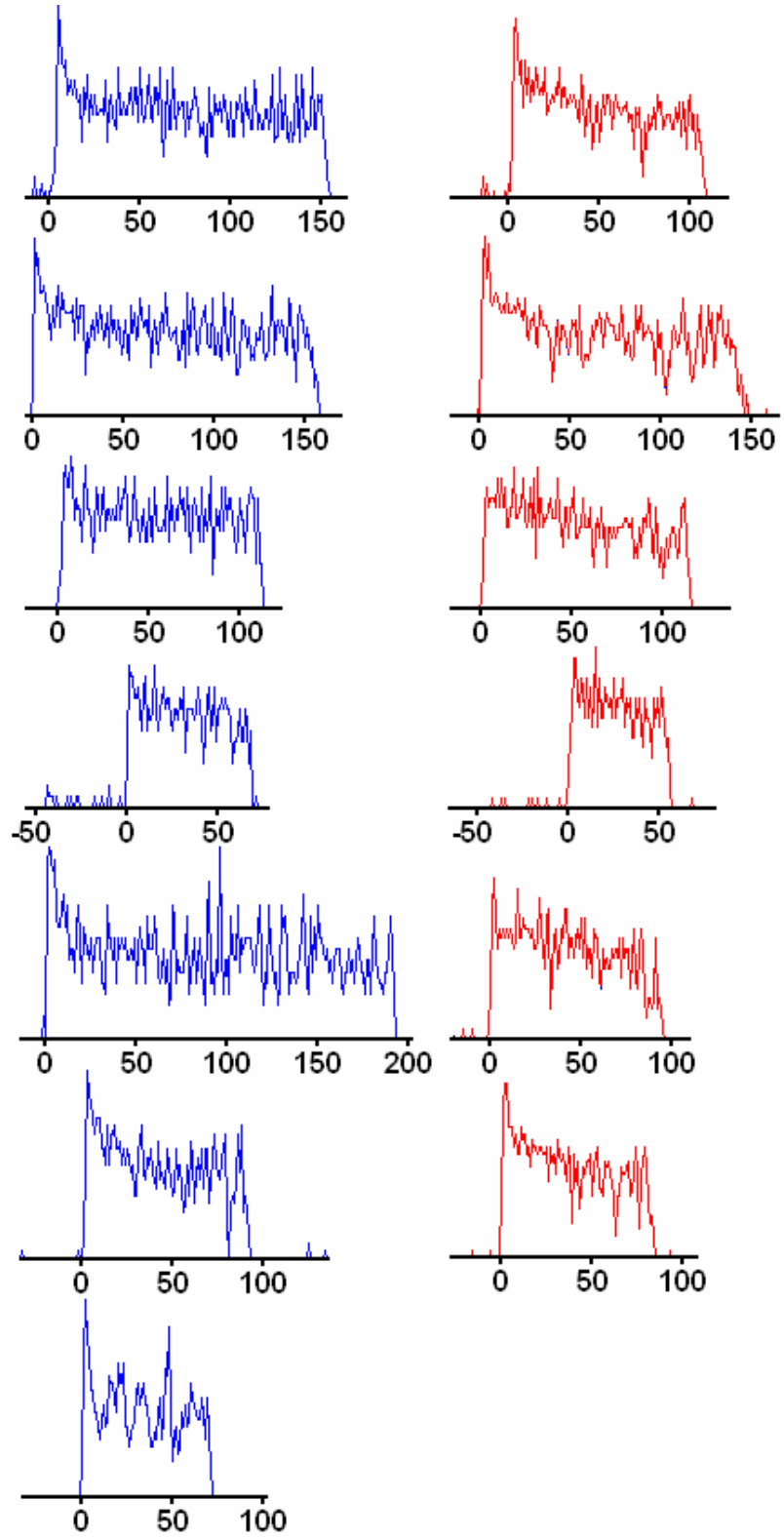


Figure 6-13 A good fit to an in-vivo cell recording. Blue → target; Red → model; A) Inter-Spike Interval (ISI) ; B) Hazard function; C) Summed synchronised rate traces; D) Rate Traces.

6.5.1 Burst initiation and termination

Mimicking burst initiation and termination is an important part of proving the hypothesis, as this will determine if the bifurcation, and hence the theory of an activity dependant bi-stable oscillator, is viable. Of particular interest are the rise time to the peak of activity at the beginning of the burst, and the range of burst termination behaviours. Figure 6-14 shows the wide range of burst initiation and termination times produced by analysis of the biological data, alongside corresponding behaviours produced by the model.

Analysis of the distribution of rise times and the corresponding changes in control parameters suggests that the primary cause of the variation is the bifurcation half-life. This value usually ranges between 0.2s and 0.6s, with shorter half-lives corresponding to more abrupt rise and fall times. Longer half-lives have the chance of producing unusual termination behaviours, where a burst may have oscillatory behaviour that pushes the bi-stable mechanism over a threshold, such that it falls from the plateau, decaying slowly towards the rest state. Oscillation has a more abrupt effect with shorter half-lives, bringing swift termination. While the half-life controls how the model initiates and terminates a burst, termination is caused by a combination of the “Noise Amplitude”, “Slow Inhibition” and spike response control parameters.



Noise before a sharp burst initiation with an initial peak of activity.

No noise before a sharp burst initiation with an initial peak of activity. Slow burst termination.

No noise before a burst initiation with reduced initial peak of activity. Fast termination due to burst of activity.

Noise before a burst initiation with reduced initial peak of activity. Fast termination.

Burst termination after a burst of activity.

After burst initiation, magnitude of noise steadily increases until fluctuations cause burst termination.

The immediate suppression of the firing rate after burst initialisation could not be replicated.

Figure 6-14 Various burst initiation and termination behaviours and correspondingly similar behaviour from the model

An increase in the "Noise Amplitude" parameter relative to the E_{RP} increases the probability of both burst initiation and termination, whereas the slow inhibition has a purely inhibitory effect. Additionally, the dynamics of the noise within the burst has a large influence on burst termination. This is covered in the next section. The last aspect of interest regarding burst initiation and termination is the size, both in activity and time, of the initial peak activity. Although the bifurcation plays a part in the rise time of this initial peak, it is primarily controlled by the relative sizes and half-lives of the DAP and AHP control parameters. This peak of activity represents the time taken for the AHP to accumulate and suppress the DAP's effect, after which the longer term behaviours become the dominant factor in determining mean firing rate. This is shown in Figure 6-15.

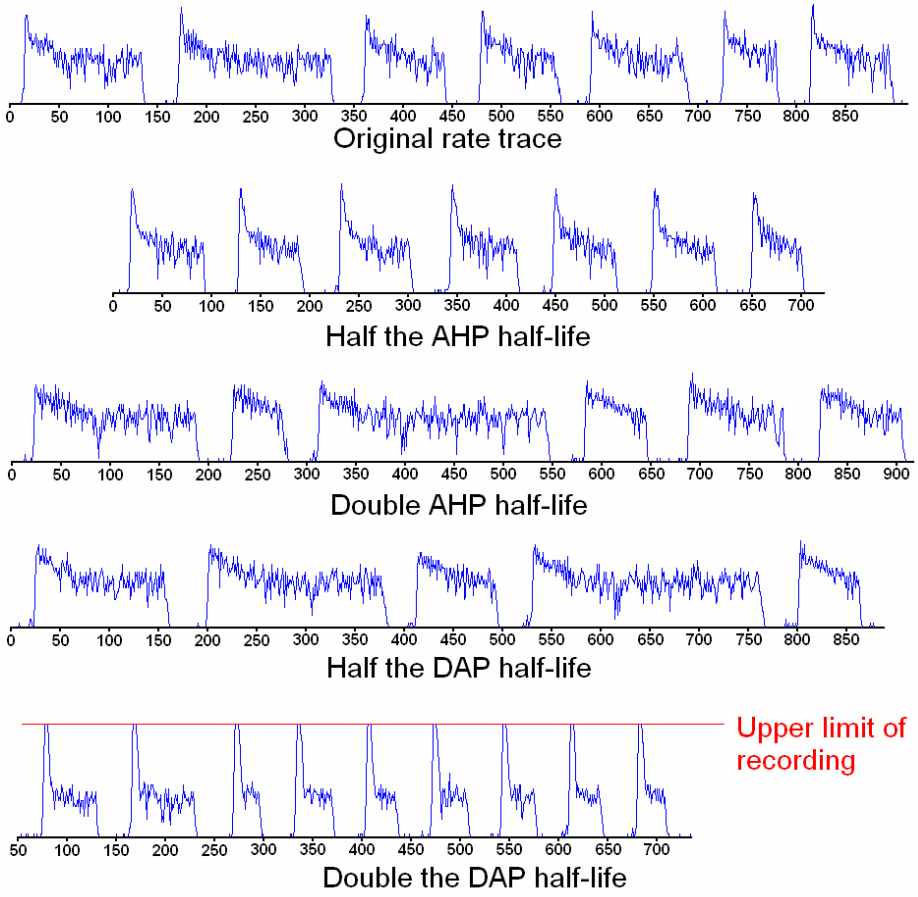


Figure 6-15 Shows how the initial peak of activity changes with the DAP and AHP magnitudes. Halving the half-life of the AHP increases the initial peak of activity, whereas doubling the AHP suppresses it (but the slow decay is still present). Halving the half-life of the DAP suppresses the initial peak of activity in an identical way to doubling the AHP. However, doubling the DAP dramatically increases the initial peak of activity so that its true peak could not be displayed.

6.5.2 Noise Characteristics within the burst

After the initial stages of model construction, it became apparent that firing characteristics within bursts have a "texture", which is represented by the average peak-to-peak magnitude within the burst (the noise envelope measure of fitness) as well as how rapidly the rate fluctuates (the resonance measure of fitness). Both measures are introduced in Section 6.1.2. Figure 1-16 shows several different instances of noise collected from the biology that illustrate the range of textures that vasopressin cells can produce.

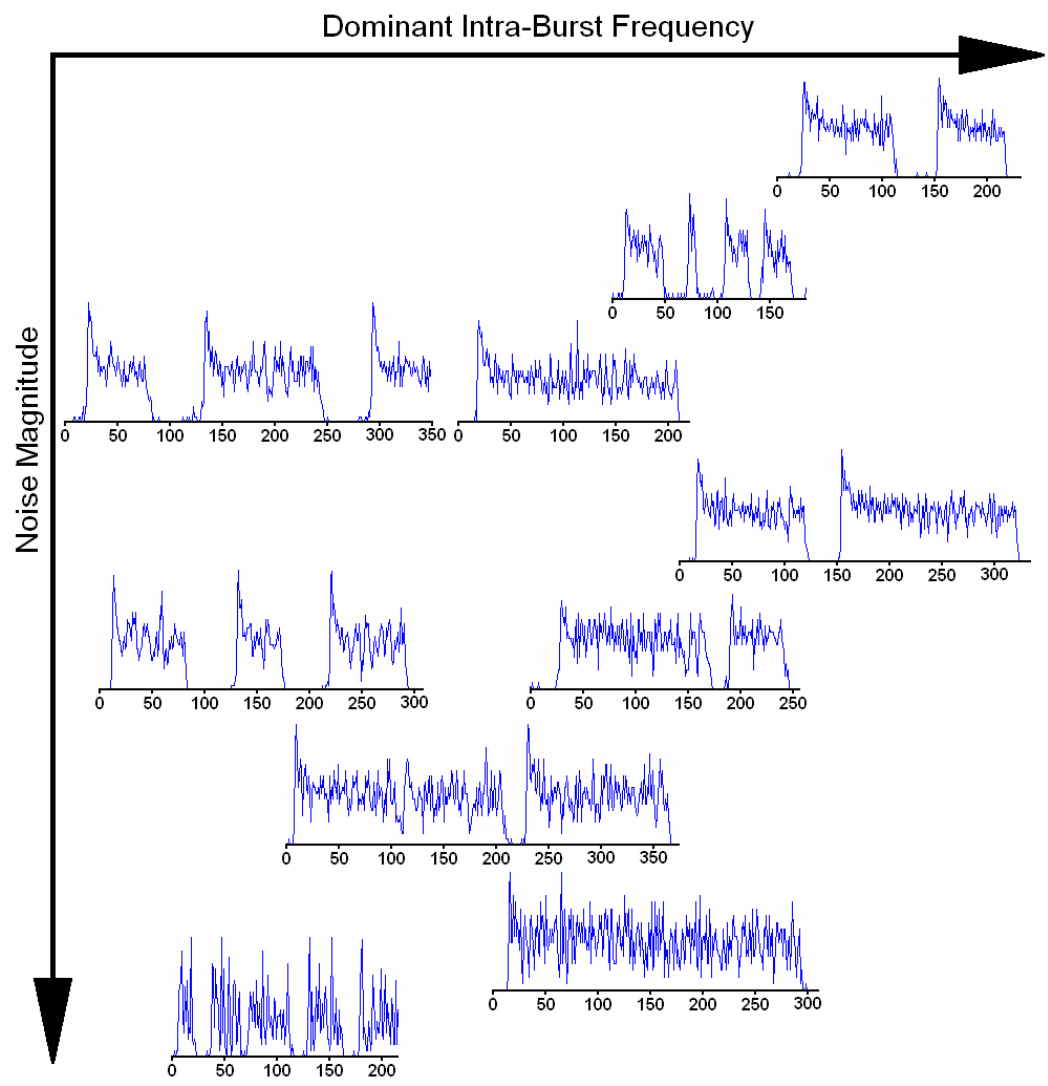


Figure 6-16 Range of noise characteristics loosely ordered by magnitude (increasing top to bottom) and frequency (increasing left to right). Even though only short sections of data are shown, the resonance characteristic is a consistent aspect of each cell's homeostatic behaviour.

Initially it was suspected that this resonating behaviour was controlled by interactions between the DAP and AHP, similar but on a smaller scale than the initial peak of activity. However, after the summed post action potential membrane excitability (spike response) was integrated with the bifurcation, it was found that this interaction was only partially responsible, and the cusp half-life, energised by the noise source, is also a regulator of the resonance. When near the plateau state, the cusp catastrophe function applies pressure to drive the base membrane excitability towards the plateau state, be that at a higher or lower level of excitability, and the cusp catastrophe half-life controls the size of this pressure. Trial simulations showed that when the noise was eliminated, the resonant behaviour could still be obtained, but in a much cleaner form, showing that the noise is a secondary factor when determining the quality of the resonant behaviour. However, if either the DAP or AHP are removed then the resonance disappears, see Figure 6-17.

In general it was found that longer bifurcation half-lives lengthened and magnified the mean resonant response. This can be explained by the cycle of feed back pressures from the DAP, AHP, and bifurcation. Repeated firing causes the DAP to summate to a plateau that results in faster firing. The bifurcation along with the AHP, under a pressure to maintain the overall excitability at the plateau, will decrease the base excitability to compensate, but the speed in which this is done is determined by their half-lives. The DAP plateau is eventually countered by the combined effects of the inhibitory AHP and the bifurcation, resulting in its collapse. When this happens, the overall excitability is then subjected to both the negative effects of the summed AHP and the bifurcation. The bifurcation moves to counter the negative excitability by moving the base excitability to a level higher than the plateau. The AHP will then decay to the point where action potentials, and hence DAP summation can occur, beginning the cycle anew.

If the half-life of the bifurcation is increased, it will respond more slowly to the offsets in excitability caused by the AHP and DAP. Specifically, a longer half-life allows for the DAP to promote the elevated state of excitability for longer, resulting in a large inhibitory force by the AHP post DAP collapse. An exploration of the parameter space suggests that it is this interaction that controls the resonant behaviour. However, for

much of the parameter space the resonant behaviour is not present, because it is not the absolute, but rather the relative, values of these three variables that are important.

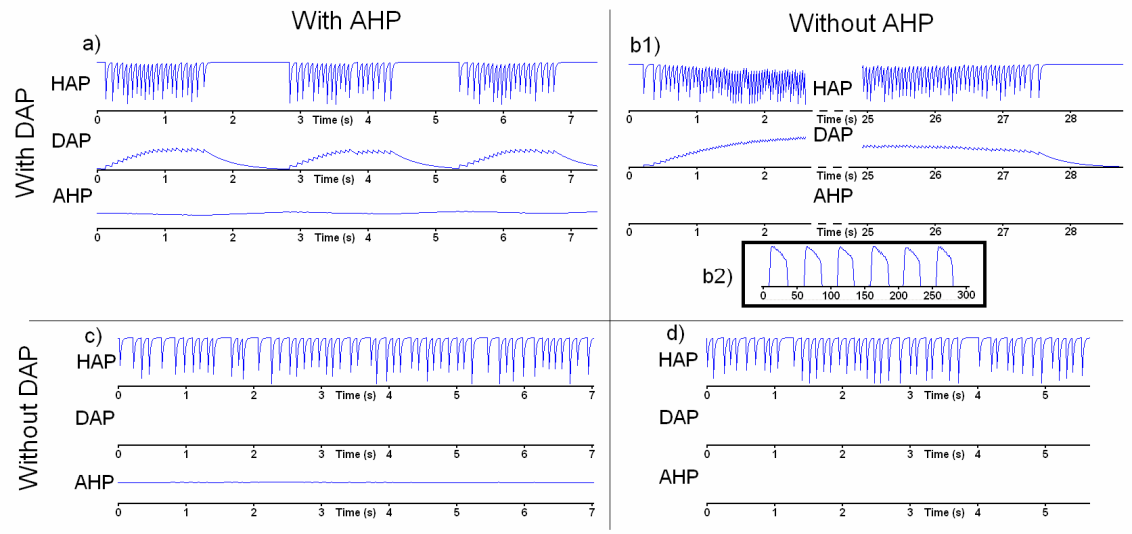


Figure 6-17 a) Traces of the HAP, DAP and AHP from a simulation with very little noise input. b1) simulation a) when the AHP is removed has no short term resonance, instead presenting bursts greater than 20s in length; b2) rate trace showing the bursting behaviour of b1); c) simulation a) when the DAP is removed has no resonance instead being a random process; d) simulation a) with no DAP or AHP, has no resonance similar to c).

It was realised that the resonance is most likely partially responsible, alongside the slow inhibition, for burst termination. The inhibition brings the cell into a behavioural state that makes burst termination likely, but it is often the resonant behaviour that pushes it over the edge. This can been seen in some cells, where at the beginning of the burst there is very little resonance, or the resonance has a small magnitude, but as the burst nears termination, this resonant component becomes larger until the burst stops. This is shown in Figure 6-18.

This behaviour can be explained by taking the effects of the slow inhibition into account. As this increases, the rate of change in excitability, seen either side of the plateau, lessens as the bifurcation is depressed, mimicking a slower half-life and creating a reduced force toward the plateau. The greater the inhibition the more likely random perturbations on the membrane, caused by synaptic events, will lower the membrane excitability past the peak of the bifurcation it a region where there is a greatly reduced force towards the plateau. Burst termination in this case would be a

result of the DAP collapsing and causing the overall excitability to fall into the negative region of the bifurcation, pulling the base excitability back towards the rest state.

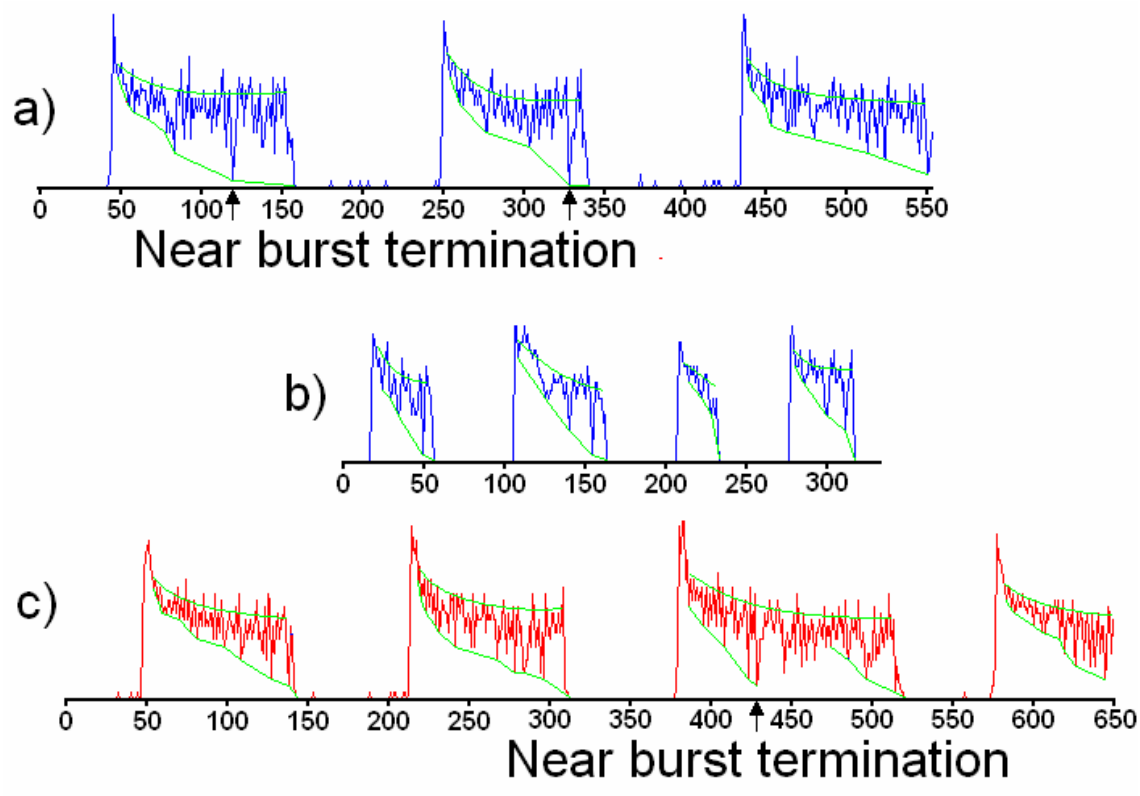


Figure 6-18 Resonate and stop behaviour; The green lines emphasise the increasing resonant behaviour as the burst progresses.

Another interesting quality that emerges from the model is its ability to mimic not only the initial peak of activity, but also the immediate following region, which sometimes shows a slow decrease in average firing rate towards a plateau of firing rate. The cause of this behaviour is a combination of the shape of the bifurcation, and the parameters that control the slow inhibition. As the slow inhibition depresses the bifurcation, the plateau level, which the function decays towards, drifts. This reduces the mean firing rate as the distance between the plateau and threshold increases, resulting in a slow decrease in firing rate towards a plateau of activity. In some cells the burst terminates before this plateau is reached.

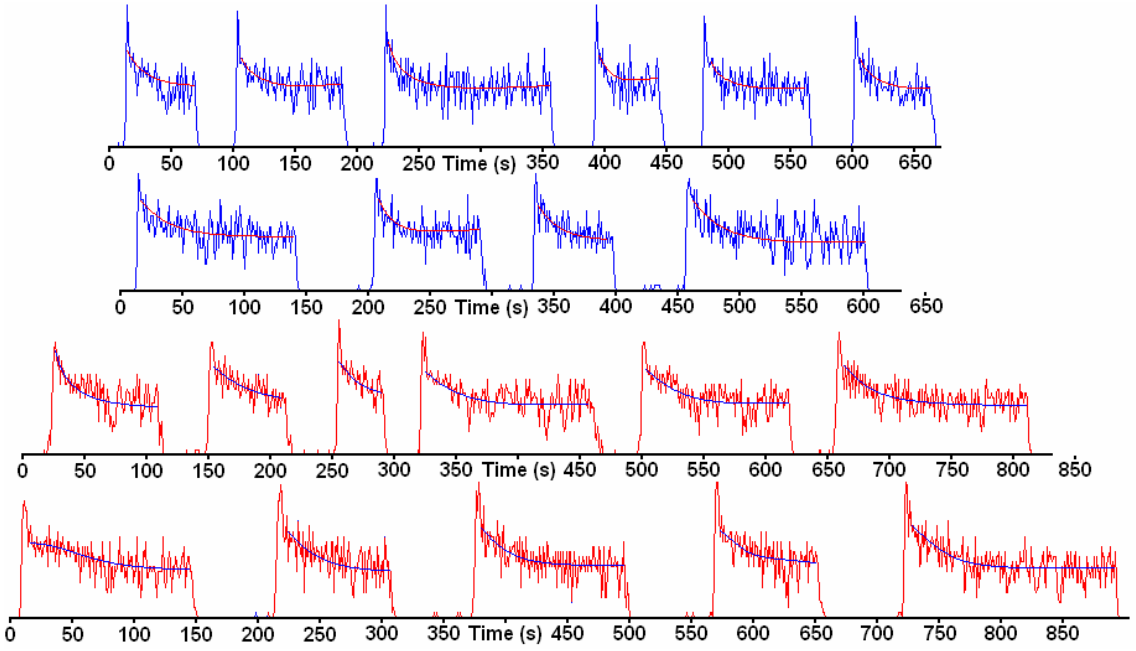


Figure 6-19 Slow decay of the burst firing rate over the length of the burst to a plateau of activity

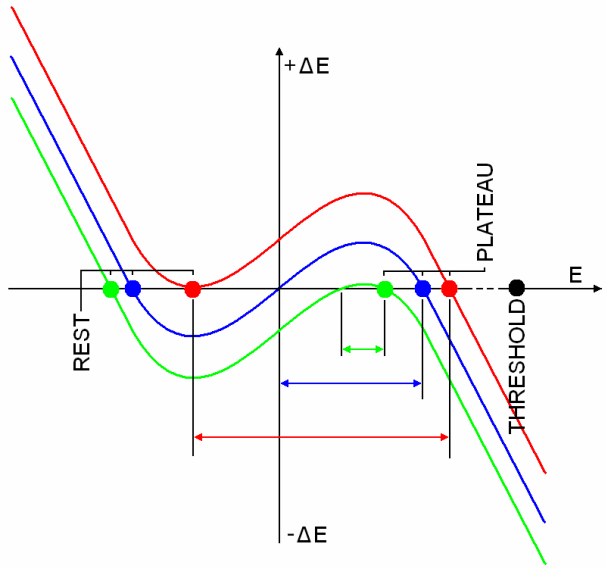
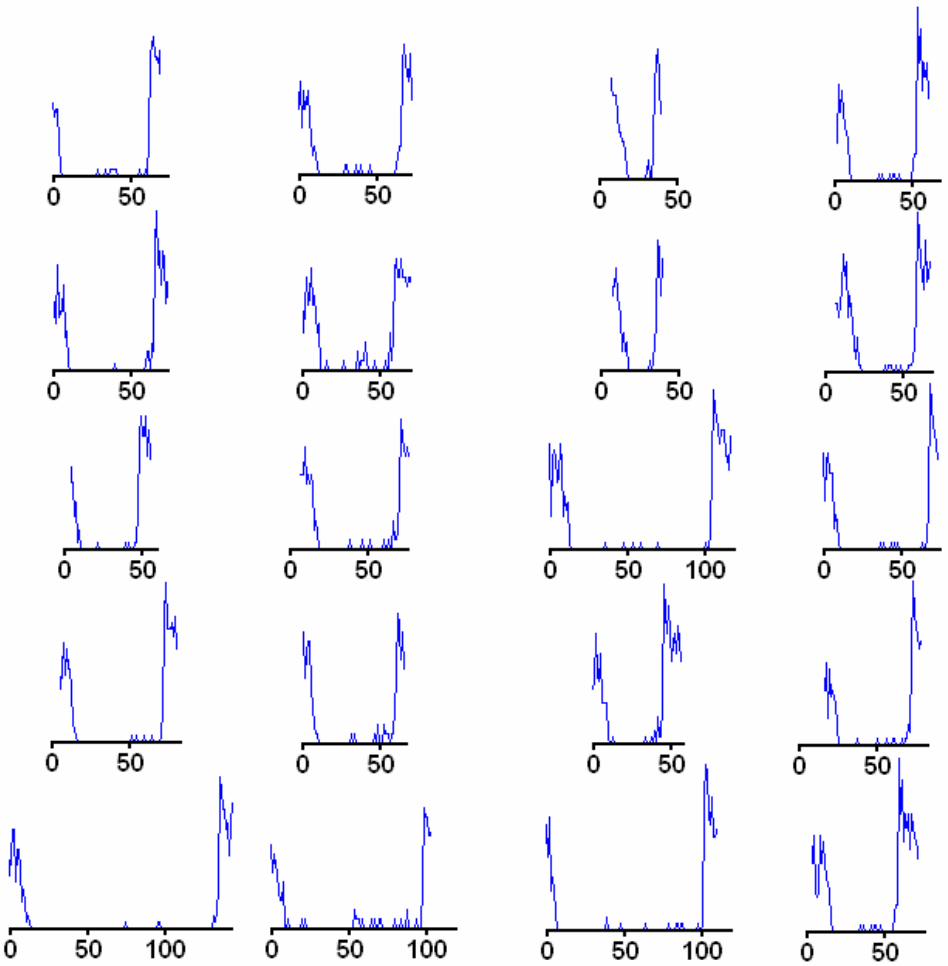


Figure 6-20 Suppression of the bifurcation by the slow inhibition variable, from Red to Blue to Green. The dots indicate the shift in the location of the Plateau and Rest excitabilities with this suppression, and the length of the arrows represent the size of the noise required to cause a change in bifurcation state from Plateau to Rest.

6.5.3 Noise between bursts

Most of the cell recordings show spurious action potentials between bursts, Figure 6-21. In most cases the probability of these events increase as the cell nears the end of its silence period. It is important therefore to be able to mimic this behaviour in the model. Interestingly this behaviour emerges from the combination of the model mechanisms with no real encouragement from the measures of fitness. This behaviour is controlled in the model by a combination of several parameters. Firstly, the magnitude of the noise is controlled by the distance between the rest and the threshold, and the magnitude of the Gaussian noise source that represents the incoming synaptic events. These determine the base likelihood of generating an action potential without any other influence. The base likelihood is then modulated by the bifurcation, in a manner similar to previously mentioned rate drift that occurs during a burst, with the only difference being that this acts in reverse and during the silence periods, slowly increasing the chance of firing by the elevation of the rest state. This is shown in Figure 6-20.



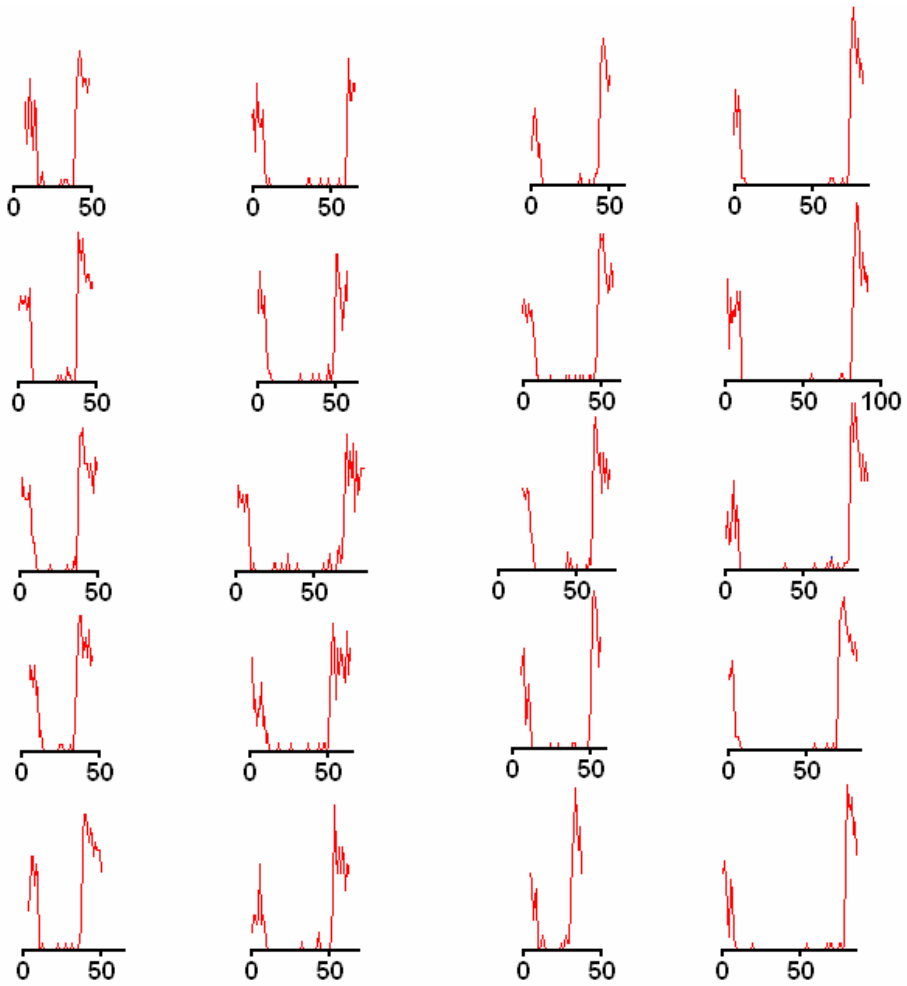


Figure 6-21 Examples of noise between burst from real (Blue) and model cells (Red)

6.5.4 Burst and Space lengths

The burst and space lengths in the model are controlled primarily by the slow inhibition control parameters. However, the noise levels, as well as the parameters that contribute to the resonant behaviour, also play a part in determining the probability of starting and terminating a burst, and therefore their respective durations. The model is able produce the correct mean and variance of the burst period, but suffers when trying to mimic the variance of silence period.

One of the problems with the model is its regularity when compared to the biology. This is partly due to each model having fixed parameters for a given simulation, whereas those aspects of the real biology that the parameters represent are constantly changing. This regularity is most easily seen with the lack of silence length variation.

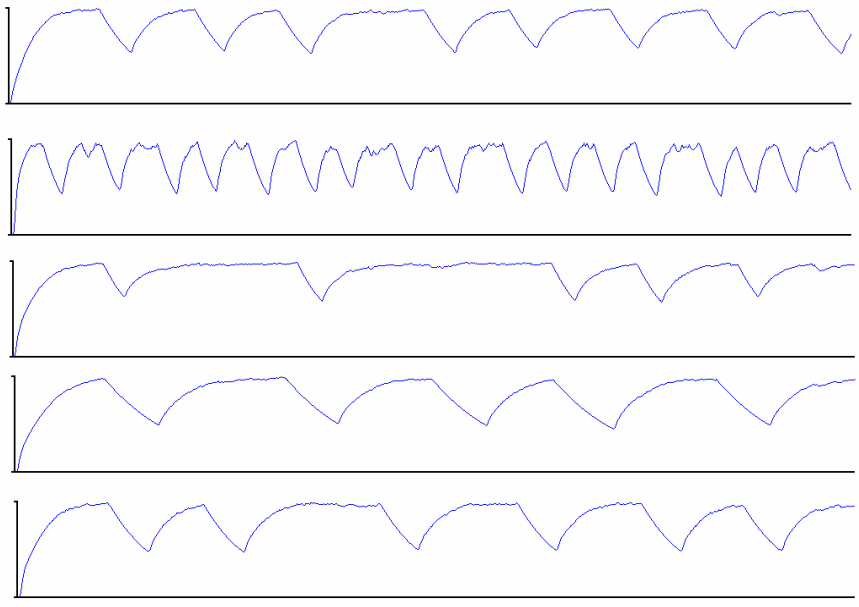


Figure 6-22 Modulation of the slow inhibition variable over time. Note that in all cases the slow inhibition oscillates away from zero, and hence its decay that results from a silence period is almost linear, and highly deterministic in nature. Also note that in all but the second trace, the slow inhibition reaches a plateau early in the burst, and hence accounts for the much greater variability of falling from the plateau state.

This particular regularity is caused by the slow inhibition variable overly dominating the noise source whilst decaying, because the inhibition variable is made to oscillate away from its rest position. When in this state, the decay of the inhibition after burst termination is almost linear, Figure 1-22. This heightened state is enforced as an unintentional effect of the "bifurcation offset" derived parameter. The offset acts as a drift mechanism making sure that the model will become elevated back to the plateau state in a timely fashion, and it is this drift mechanism that is the cause of the lack of silence variation.

6.6 Model Iteration 4 – Offset Variation

To achieve greater variance, the noise source must have greater influence on the probability of state transition. To do this, the inhibition variable must be allowed to decay more fully by lessening the effect of the drift mechanism. This can be done by applying a new control parameter that dictates the percentage of the original offset that is applied. Figure 6-23 shows how the model's behaviour is modified by varying this

parameter. By reducing the amount of initial offset, the inhibition variable can now be allowed to fully decay, and hence the Gaussian noise has the greatest influence upon the initiation of a burst.

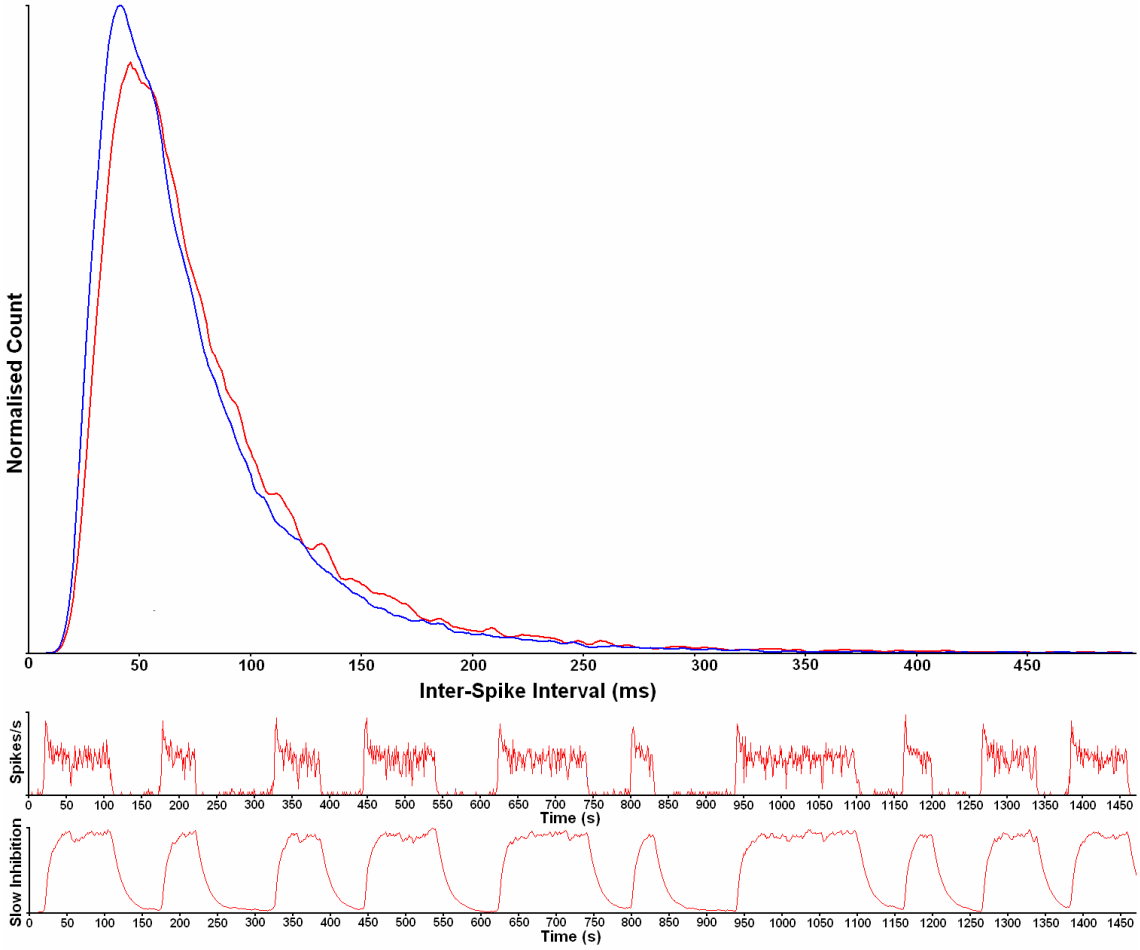


Figure 6-23 Shows how by decreasing the initial offset of the slow inhibition, which suppresses the bifurcation, the inhibition variable is allowed to fully decay and hence the entire burst initiation process is much more random.

6.6.1 One Parameter – Three behaviours

The offset parameter, which essentially represents pre-charging the inhibition variable, can be used to produce all three of the archetypal vasopressin release neuron behaviour; Continuous; Phasic; and Sparse sporadic, as shown in Figure 6-24.

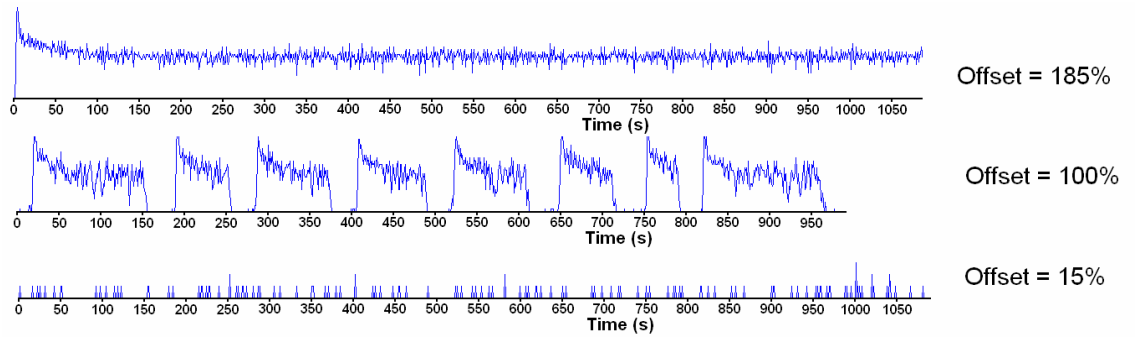


Figure 6-24 The model can produce three different behaviours by simply modulating the initial offset parameter from 15% through to 185%

Interestingly, the modulation of the initial offset parameter does not significantly effect the short term behaviour of the model (ISI and hazard) while firing either continuously or phasicly. Additionally, the initial offset controls the amount of slow inhibition that is required to reach a plateau of activity, and it is this build-up that controls the slow depression of rate immediately after the initial peak of activity.

6.6.2 Three Parameters – Phasic Control

With a combination of the initial offset, slow inhibition half-life and slow inhibition step amount, it is possible to fully control the phasic behaviour of the model producing any combination of burst and silence mean and variance. Figure 6-25 is a good example of the model producing highly realistic neuronal behaviour.

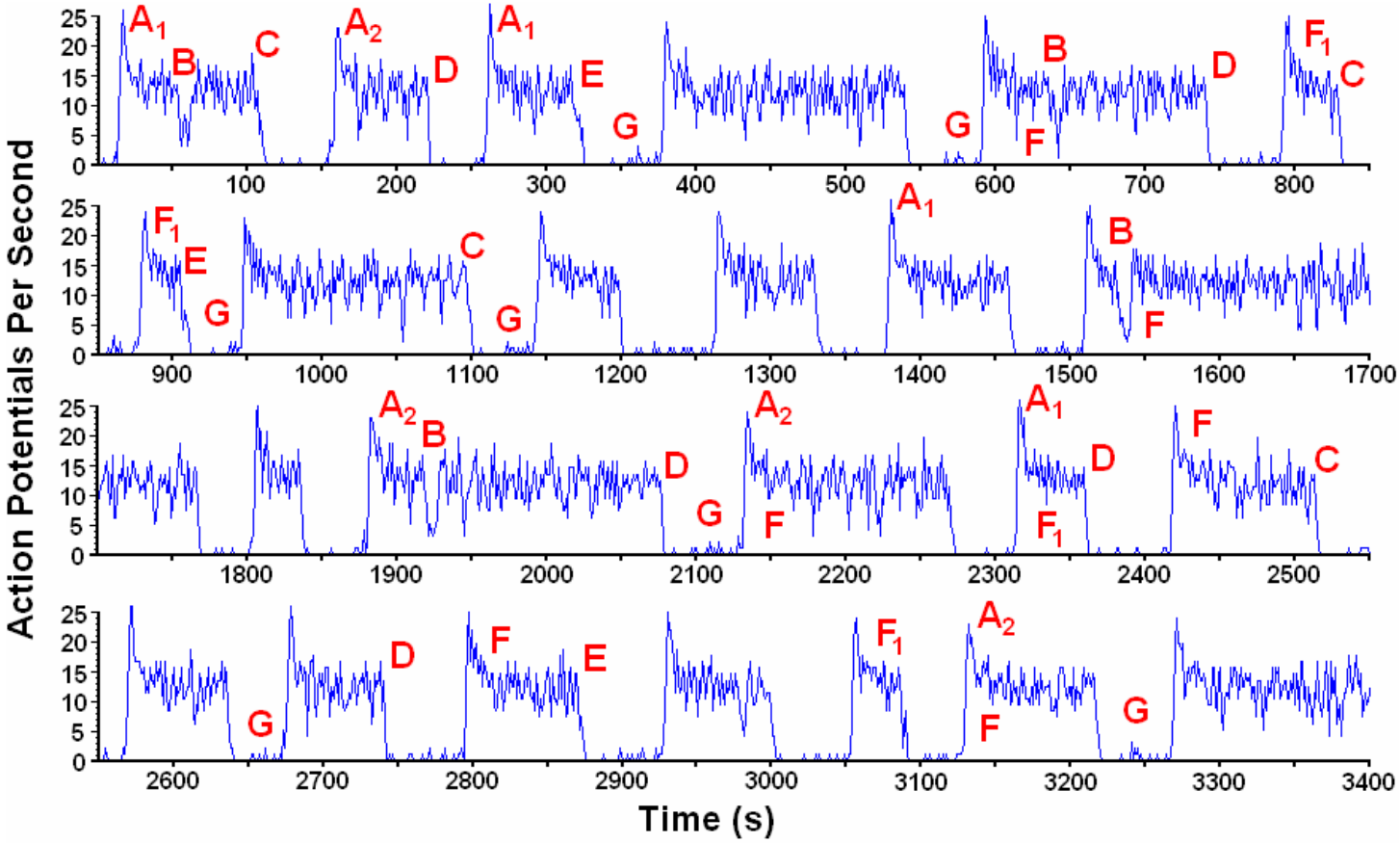


Figure 6-25 A highly realistic, 1s per bin, rate trace from the final model. This model instance shows all of the discussed characteristics of a real vasopressin releasing neuron. A) Varying degrees of initial peak of activity based upon the rate of activity increase (A_1 – Swift, A_2 – Slow). B) Large amounts of short term activity nearly stopping a cell, with it restarting with a little peak of activity if still near the start of the burst. C) Large amounts of short term activity stopping a cell. D) Swift burst termination. E) Slow burst termination. F) Decay of a burst to a plateau of activity, which is sometimes not reached due to burst termination (F_1). G) Spurious cell activations within the silence period that generally increase in number is the cell approaches burst initiation.

6.7 Final Model

The final model consists of 14 control parameters spread over four interconnected functional blocks; the Synaptic Input; Membrane bifurcation function, including the slow inhibition; Summation of the post action potential membrane excitability; and firing mechanism. This final model can be formally described by the following equations divided under the headings of the four functional blocks:

Synaptic Inputs

$$N = N.\tau[N_{HL}] + G[R].N_A$$

G[] = array of Gaussian distributed random numbers

R = discrete uniform random number normalised to the size of the Gaussian array.

Summation of post action potential membrane excitability, HAP (H), DAP (D), and AHP (A)

$$\frac{\Delta H}{\Delta t} = H.\tau[D_{HAP}] + S_{HAP} H[E_C - E_{TH}]$$

$$\frac{\Delta D}{\Delta t} = D.\tau[D_{DAP}] + S_{DAP} H[E_C - E_{TH}]$$

$$\frac{\Delta A}{\Delta t} = A.\tau[D_{AHP}] + S_{AHP} H[E_C - E_{TH}]$$

Cell firing mechanism

$$E_C = E_M + H + D + A + N$$
$$SpikeEvent = H[E_c - E_{TH}]$$

Membrane Bifurcation function

$$\frac{\Delta E_M}{\Delta t} = S.\left(E_{Offset} - I\right) + \begin{cases} -\left(E_{\inf} - \frac{E_{RP}}{2}\right).\tau\left[B_{HL}\right] & , \quad E_{\inf} > \frac{E_{RP}}{2} \\ -S.\left(4E_{\inf}^3 + 2uE_{\inf}\right) & , \quad \frac{E_{RP}}{2} > E_{\inf} > -\frac{E_{RP}}{2} \\ -\left(E_{\inf} + \frac{E_{RP}}{2}\right).\tau\left[B_{HL}\right] & , \quad -\frac{E_{RP}}{2} > E_{\inf} \end{cases}$$

Where the base excitability with influence from noise, HAP, DAP, and AHP (E_{inf}), activity dependant slow inhibition (I), initial offset (E_{Offset}), and the bifurcation parameter U and scale (S) are is defined by:

$$V = D + A$$

$$E_{\inf} = E_M + H\left[V + H\right].(V + H) + H\left[-V\right].V + N$$

$$\frac{\Delta I}{\Delta t} = I.\tau\left[D_{Inhib}\right] + S_{Inhib}\, H\left[E_C - E_{TH}\right]$$

$$E_{Offset} = -3^{-\frac{3}{2}} E_{RP}^{3} E_{\%}$$

$$u = \frac{E_{RP}^{2}}{2}$$

$$S = \frac{\tau\left[B_{HL}\right]}{E_{RP}^{2}}$$

The following table lists the model's control parameters and briefly describes their effect on the model behaviour. It is important to note that because of the level of inter-parameter interaction, all parameters have some effect on every aspect of the model's behaviour, but as a general rule, the bifurcation and inhibition control parameters determine the longer term phasic behaviour whilst the spike response control parameters control the shorter term behaviours that can be visualised through the ISI and Hazard histograms.

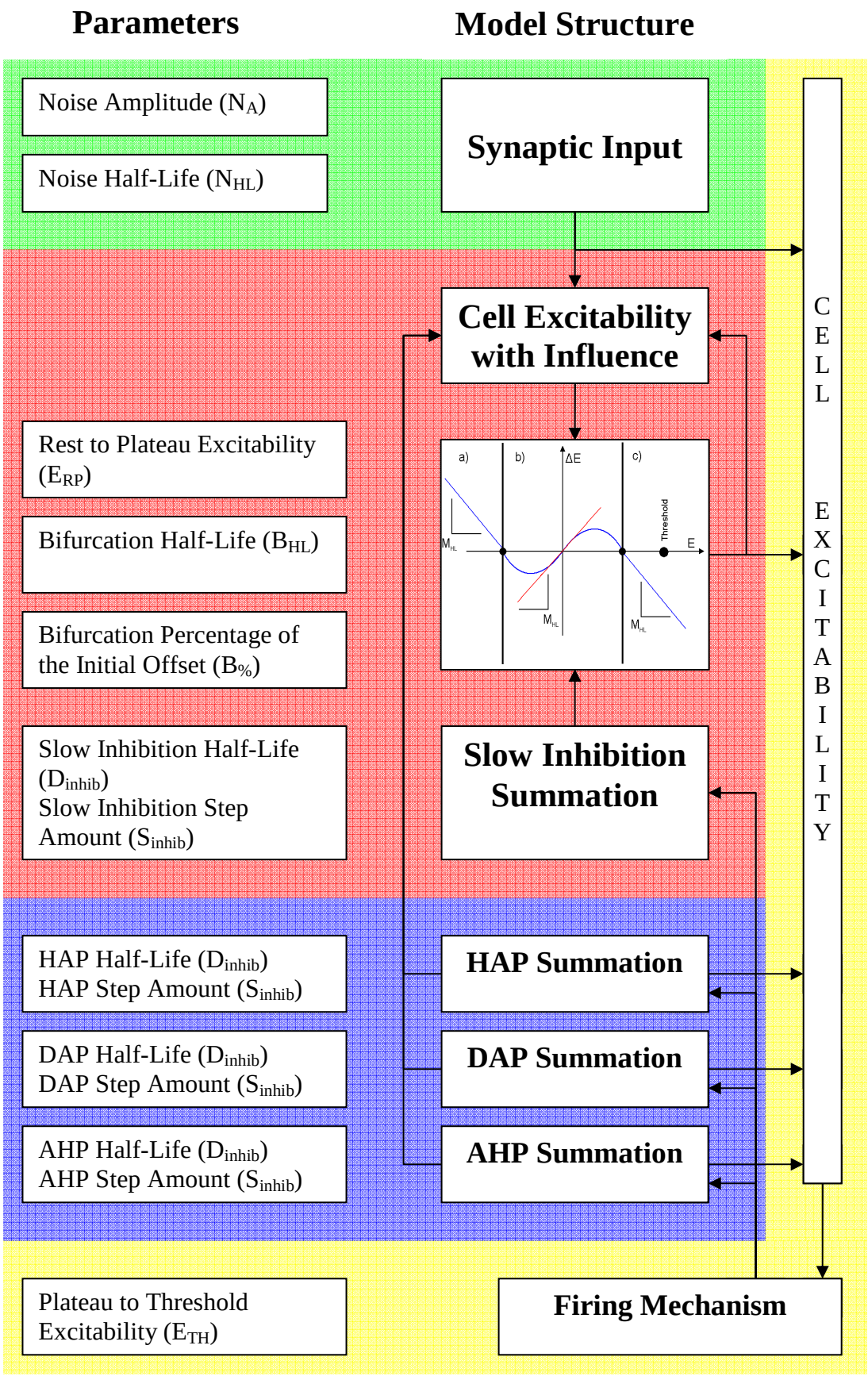


Figure 6-26 Structure of the final model

Parameter	Symbol	Behavioural Associations
Noise Amplitude	N_A	The magnitude of the synaptic input is determined entirely by the combination of N_A and N_{HL}. The magnitude of the synaptic input relative to E_{TH} is largely responsible for the intra burst mean firing rate. The size of the synaptic input relative to E_{TH}+E_{RP} determines the firing rate during silence periods. N_{HL} is set to 10ms for all our simulations.
Noise Half-Life	N_{HL}	
Rest to Plateau Excitability	E_{RP}	The bifurcation state transition time is determined by a combination of E_{RP} and B_{HL}. This parameter can be made a constant it is the relative difference in magnitude between N_A, E_{RP} and E_{TH} that is important.
Bifurcation Half-Life	B_{HL}	
Bifurcation Percentage of Initial Offset	$B_{\%}$	The initial offset parameter controls the overall state of the neuron; Continuous; Phasic; or sparse sporadic.
Slow Inhibition Half-Life	D_{inhib}	When combined with the initial offset parameter $B_{\%}$, the Slow Inhibition control parameters can fully control the Phasic behaviour, including burst and silence period mean and variance. These parameters also combine with bifurcation parameters to determine the decay in average rate across the length of a burst.
Slow Inhibition Step Amount	S_{inhib}	
HAP Half-Life	D_{HAP}	The HAP, DAP, and AHP primarily determine short term behaviour that can be visualised through the ISI and Hazard histograms. Additionally, the combination of the DAP and AHP controls the magnitude and width of the initial peak of activity during burst initiation.
HAP Step	S_{HAP}	
DAP Half-Life	D_{DAP}	
DAP Step	S_{DAP}	
AHP Half-Life	D_{AHP}	
AHP Step	S_{AHP}	
Plateau to Threshold Excitability	E_{TH}	Combined with the Noise control parameters, N_A and N_{HL}, this parameter determines the intra burst firing rate.

6.8 Summary

During this chapter, a minimalist model of the vasopressin releasing neurons found in the hypothalamus was developed. This model, which was built using a top-down approach, consists of fourteen control parameters, many of which are held constant, and is able to be adapted to produce behaviours very similar to real vasopressin neurons recorded both *in-vivo* and *in-vitro*. To perform this adaptation, many aspects of vasopressin cell behaviours were analysed to form a function of fitness which guides the adaptation algorithm. Through the analysis of the performance of the algorithm we can state that the model is able to reproduce:

- The short term firing patterns, characterised by the Inter-Spike Interval histogram and Hazard function.
- The shape of burst initiation and termination, including resonate-and-stop behaviours, the initial peak of activity, as well as how quickly the rate falls away at the termination of a burst.
- The distribution of spurious action potentials between bursts, including the increased frequency of these events relative to the proximity of burst initiation.
- The Noise characteristics within the bursts analysed by our resonance measure of fitness. This includes the reduction of mean firing rate after the initial peak of activity towards a plateau, which is not always present.
- The Bursting behaviour, which includes the mean and variance of burst and silence periods.

The adaptation algorithm found it difficult to fit to some of the in-vitro recordings, which suggests that either; the adaptation algorithm is insufficiently explorative; our model, and hence our understanding, is missing a vital mechanism; or the parameters required to fit this behaviour are outside our pre-determined range of biologically plausible values.

The development and analysis of this minimalist model, and its ability to reproduce the complex behaviours of vasopressin neurons in such detail, suggests that a bifurcation-

driven membrane is a good representation of the biology. Furthermore, the structure and function of this model suggest that the overall behaviour of the model can be controlled by the modulation of a single control variable, the initial offset, which could represent a chemical store within the cell. It should therefore be possible to create a network model from many instances of this single cell model. This network model should be able to respond to changes in required population activity by the modulation of this single control parameter, which will increase the mean firing rate of the overall population, whilst distributing population workload more evenly.

6.8.1 Iterative Design Method

Though the use of the rapid evaluation methodology presented in Chapter 4, a model to test a hypothesis regarding the nature of a class of neuron was successfully developed. It should be noted that although the minimalist model developed consisted of only fourteen control parameters (two of which are fixed), there is extensive interaction between those parameters, creating a highly multi-model parameter space. This multi-modality is much more complex than the basic excitability-based model presented in Chapter 5; so much so that the adaptation algorithm required much more guidance, in the form of an iterative search, where the parameter space was restricted after each adaptation run until a suitable solution was found. This presents a new problem as, although a model was successfully created, the adaptation algorithm required too much coaxing to produce an acceptable solution within this complex environment. Ideally, the adaptation algorithm should function without extensive fine tuning, thereby allowing a user to interact with the tool without understanding the fitness function or underlying parameter interactions in much detail. This outcome suggests that more research will be required, to create evolutionary algorithms that can handle these complex parameter spaces more easily. Even so, this case study shows that the use of adaptation methods as a way to test models is sound, and in the case of this model, required. This is due to the level of interaction between the control parameters, which means that it would take extensive mathematical study to understand, and hence constrain the parameter space to produce only realistic behaviours, as much of the parameter space is un-analysable. Ultimately, the wish is for understanding to come from the modelling process, not the other way round. Without a method of this nature, it would be extremely difficult for an experimental electrophysiologist to determine if a

model could produce a desired behaviour. Confirmation of a model's veracity can proves or disproves their hypothesis, thereby improving their understanding of the target biology.

Chapter 7 Summary, Conclusions and Future Work

7.1 Summary

This thesis has presented a proof of concept study for the better integration of the electrophysiological and modelling fields within neuroscience. Members of these two sub-disciplines regularly collaborate, but due to differing resource requirements and largely incompatible spheres of knowledge, cooperation is often impeded by miscommunication and delays. To reduce the model design time, and hence provide a platform for more efficient experimental analysis, a rapid iterative model design method has been proposed; this document investigated various aspects of this method, concluding in a case study.

Chapter 1 introduced the three requirements for rapid iterative model design. These include the development of a rapid model evaluation method (Chapter 4), and the investigation of an easily extensible modular neural model (Chapters 5 and 6). The final requirement, the development of a user-friendly graphical user interface designed specifically for experimental electrophysiologists, was outside the scope of this work.

7.1.1 Rapid Model Evaluation

The rapid model evaluation method is based upon Goodness of Fit (GoF), where a trial model's control parameters are adapted in an effort to recreate observed neural patterns, and thereby test a model's *descriptive adequacy*. If the model can replicate many observed behaviours from the same class of neuron, then the model, and hence the hypothesis that it represents, can be considered good. However, as discussed in Section 3.1, GoF alone should not be used to determine the strength of a model, as it does not take into account a model's s*implicity*; an overly complex model will suffer from over-fitting. Chapters 5 and 6 discuss this problem in relation to experimental neuroscience, where observed data can be scarce, as well as discussing the benefits of a top down iterative design approach. This involves first producing as simple a model as possible, and extra complexity is iteratively added to "patch" the model when the basic version cannot reproduce specific behaviours. This does not prevent the user from creating an

overly complex model, but it does present a design philosophy that favours simpler solutions over complex ones.

One of the core contributions of this thesis is the creation of a generic approach to rapidly evaluate a model's ability to reproduce specific observed behaviours whilst within the laboratory environment (Chapter 4). The two main requirements (Section 4.1) for such a method are; an adaptation algorithm that is does not make any presumptions about the shape of the parameter space, does not require *a-priori* information to guide its search, and is robust to complex parameter interactions; and a method to accelerate the chosen algorithm such that it can be performed in near real-time within the laboratory. A review of current modelling and simulation platforms is presented in Section 3.2, which summarises the current trends in the field and concludes that no modelling environment exists that is designed specifically to cater for laboratory-based neuroscientists.

Evolutionary Algorithms (EAs), which were reviewed in Section 3.3, were chosen to fulfil the generic parameter estimation task of evaluating a model's goodness of fit. EAs are extremely robust to complex and/or unknown parameter spaces, and have already been used by several modelling platforms for parameter estimation (Section 3.2), hence there was precedent for their use within the proposed model evaluation method. However, EAs are also extremely computationally expensive, requiring many fitness evaluations during the optimisation process. When fitting a neural model to observed data, each fitness evaluation involves a simulation run followed by an analysis of that run. The quality of the fitness evaluation function is extremely important for the proposed method, more so than for typical parameter estimation tasks, as not only does it guide the EA, but also acts as the main interface to the user, who is optimising the structure of the model. Fitness evaluation was reviewed in Section 3.4, and discussed with a particular focus on usability by experimental electrophysiologists in Section 4.3.

Hardware acceleration was required to combat the extreme computational expense of Evolutionary Algorithms. Section 4.2 reviewed four forms of hardware acceleration, namely, application specific integrated ciruits (ASICs), field programmable gate arrays (FPGAs), server clusters, and graphics processing units (GPUs). This section concluded that of the four forms of hardware acceleration, GPUs are the best suited to

accelerate the proposed model evaluation, due to their low cost and high computational throughput. Due to the competitive nature of the GPU industry over the last 10 years, and with processor specialists Intel now entering the competition with ATI and NVIDIA, it is thought that the aggressive trend in power growth is likely to continue.

Utilisation of GPU hardware acceleration, known as General-Purpose computation on Graphics Processing Units (GP-GPU), places limitations on both the EA and the types of fitness calculation that can be performed. The GPU's fixed amount of parallelism determines the population size of the EA (Section 4.4), and limited memory resources enforce pre-processing of simulation results onboard the GPU (Section 4.3). Section 4.4 presents a comparative algorithmic analysis of three archetypal evolutionary approaches; Particle Swarm Optimisation (Section 3.3.2), Estimation of Distribution Algorithms (Section 3.3.3), and Genetic Algorithms (Section 3.3.1). This was done to determine which class of evolutionary algorithm would perform best whilst searching parameter spaces of a similar difficulty to that of excitability-based models (Section 3.1.8), under the constraints set by the GPU architecture. The results favoured Genetic Algorithms (GAs).

To determine whether a poor utilisation of available population size can cause a reduction in performance, a fourth algorithm, a Distributed Adaptive Genetic Algorithm (DAGA – Section 4.4.1.4) was designed and tested. The DAGA consists of two evolutionary algorithms, the second optimising the configuration of the first, which is a facsimile of the already utilised standard archetypal GA. The DAGA attempts to capitalise on the "No Free Lunch" theorem for parameter search and optimisation, where thousands of different GA configurations compete to take control of the population. Hence successful configuration will dominate the search method, improving the efficiency of the entire algorithm, and gaining "free lunches". Compared to the GA, the DAGA provided similar or greater performance across a range of adaptation targets, suggesting that an algorithm that seeks to take better advantage of large populations is better suited to being accelerated by a GPU. Furthermore, this also suggests that utilisation of co-evolutionary approaches may provide a way to better exploit the massive parallelism that future hardware acceleration architectures will provide. This is especially important for the GPU industry where the parallel

computational resources available will soon far outstrip the requirements of the majority of EAs.

An analysis of the performance of all four EAs resulted in the conclusion that the choice of hardware acceleration greatly affects the performance of an evolutionary algorithm; algorithmic performance can be drastically reduced when it is ill suited to the adaptation environment imposed by the hardware acceleration.

7.1.2 Object Oriented Model Design

Chapter 4 provided the engine behind the rapid iterative design philosophy by allowing models to be evaluated within the laboratory alongside biological experimentation, and ultimately proved the grounding hypothesis of this thesis. However, such an engine is immaterial without models to evaluate, which is where object oriented model design comes into play.

Object oriented model design has been used by modelling packages such as GENESIS for many years (Chapter 3.2). It is a subset of modular design, where a model is divided into functional blocks, or classes, where the behaviour of a class is defined by its structure (class functions), and control parameters (class variables). This allows a complex model to be represented more simply as a set of interdependent, simultaneously computed modules. Additionally, object oriented model design promotes code reuse, where old classes can be combined to make incremental increases in functionality with a minimum of effort (inheritance). The use of an object oriented model design method is of particular importance to this thesis's rapid iterative design philosophy, for two reasons. Firstly, the idea of modular model design is intuitive, which is a requirement for use by non-computer scientists. Secondly, any model developed must produce consistent behaviour when run across a wide range of computational platforms, and object oriented design allows, through the use of Polymorphism and Interface (two further aspects of object orientation), a layer of abstraction between model design and module implementation. This allows localised implementations of modules on specific hardware (Chapter 5), which for this application will include GPUs, CPUs, and possibly custom digital devices like PICs,

FPGAs or even ASICs, which might be used to interface with laboratory equipment, integrating modelling with experimental protocols.

Section 3.1 reviewed a range of single cell neural models, which stem from two archetypical types, the Hodgkin-Huxley (HH) single compartmental conductance based model (Section 3.1.3), and the integrate-and-fire (IF) model (Section 3.1.5). Because this thesis focuses on a top down modelling approach, models based upon the IF model have greater appeal, due to their *simplicity*. Spike response models (SRM), which are descendents of the IF model, offer a good compromise between the *simplicity* of the Izhikevich model (Section 3.1.4) and the more biologically structured full conductance-based models such as the HH. The basic excitability-based model (Section 3.1.8), which is a refinement of the cumulative spike response model with noise (Section 3.1.7), is an ideal candidate to form the basis of an object oriented design method, because it is representative of the behaviour of the underlying biological subsystems whilst being computationally simple. However, the real advantage of this type of model is its ability to represent the majority of the control parameters in a manner intuitive to electrophysiologists (6 out of 7 parameters), through the spike response curve (Section 5.5).

Chapter 5 presented an investigation into the viability of using the basic excitability-based model to form the basis of an object oriented model design method. The model was formally introduced (Section 5.1), and its d*escriptive adequacy* tested (Section 5.3), by utilising rapid parameter estimation (Chapter 4) to reproduce the behaviours of four sets of neural data, which were collected from the ventral medial hypothalamic (VMH) region of the brain (Section 2.1.2). The results of this analysis led to the conclusion that the basic excitability-based model can recreate the behaviours of a wide range of continuous firing neurons, thereby showing good *descriptive adequacy*.

An analytical correlation-based method was used in Section 5.4 to determine the robustness of the model, and hence gain a measure of its s*implicity*. In the context of model construction, this thesis defines robustness as a measure of the simplicity of the association between behaviour and parameter space, i.e do similar parameter sets produce similar behaviours? The results of this measure led to the conclusion that the

basic excitability-based model is robust, and can be considered simple insofar as similar configurations produce similar behaviours.

However, the nonlinear nature of the measure meant that one of the samples, which presented outlying behaviour, had to be excluded from the correlation coefficient calculation. This suggests that, while the underlying concept of s*implicity* measurement through analysis of correlation is sound, an alternative, self-constraining data preparation method would be required to reduce the effect of extreme outliers.

Section 5.4 investigated the role of the spike response pattern in shaping cell behaviour. The individuals from the four spike response sets produced by the parameter estimation process were each normalised by the mean firing rate. The resulting sets were then analysed, to determine the cause of the differences in the four broad forms of firing pattern behaviour. Of the four sets, three showed distinctive differences in their spike response, which suggests that each class of neuron has its own broad type of spike response, that can be described by the sum of three exponential decays, corresponding to the HAP, DAP and AHP.

Experimentation on the model showed that it is possible to move between behaviours by altering single parameters. This was done by starting with a parameter set that produced a behaviour derived from a member of one of the four sets, and then altering the half-life or magnitude of that particular aspect of the spike response. In several cases, it was found that the new behaviours, although similar to one of the existing four sets, was in fact from one of the other 5 sets previously derived by Sabatier and Leng, (2008) but not used as part of this study. This provides further evidence of the robustness and generalisability of the basic excitability-based model, suggesting that broad areas within its parameter space can be attributed to broad classes of behaviour, each defined by a combination of HAP, DAP, and AHP magnitudes and decays.

7.1.3 A Case Study of the Iterative Design Process

While Chapter 5 investigated the suitability of excitability-based models as a candidate for modular model design, Chapter 6 explored in more depth the ease with which this class of model can be extended. A case study of the iterative design process was performed, testing a hypothesis regarding the function of vasopressin neurons. In this

chapter, the basic excitability-based model was extended by replacing the membrane decay mechanism with a bifurcation designed to loosely represent voltage and activity dependant ion channels. Over the course of the chapter, the initial model was iteratively improved, using the rapid model evaluation method to gauge performance. The final iteration can reproduce a range of sub-behaviours, seen in both *in-vivo* and *in-vitro* data: including ISI and hazard histograms, noise within and without the burst, burst and silence mean and variances, and several more subtle behaviours, such as the recently-identified long AHP. Additionally, the model can replicate the three archetypal vasopressin behaviours through the modulation of a single control parameter. Complete control of the model's phasic pattern can be attained through the modulation of three control parameters, which in turn control the activity of a single model variable. The results of this iterative design process led to the conclusion that an activity dependent mechanism, which functions by creating a sub threshold bifurcation on the membrane, is likely to be largely responsible for the phasic behaviour observed *in-vivo*.

The outcome of Chapter 6 suggests that it is unlikely that DAP inhibition through the application of co-released Dynorphin is responsible for the sub threshold in-vivo membrane plateau that is the cause of phasic behaviour. This is due to the fact that:

- Previous investigators (Durie, 2007) could not model a stable sub-threshold self-sustaining membrane plateau through the summation of DAPs alone.

- The model presented in Chapter 6 does not rely on the modulation of the DAP for burst initiation or termination.

- The model presented in Chapter 6 has extremely good D*escriptive Adequacy*, being able to replicate a large range of behaviours.

However, that Dynorphin inhibits the DAP and controls bursting activity is undeniable, as this has been directly recorded from the biology. Hence the work presented in Chapter 6 suggests that an alternative hypothesis where, at least *in-vivo*, the DAP is not entirely responsible for creating and sustaining the sub-threshold plateau. The hypothesis therefore suggests that the effect of Dynorphin on the DAP is a secondary effect, as it also inhibits other aspects of the cell that have direct control of the sub-threshold membrane plateau. This explanation does not conflict with experimental

findings, as Dynorphin is still considered to control phasic behaviour, but through a different medium.

It is important when combining hypothesis derived from both modelling and experiment to not let either source dominate the other. Both are formulated opinions and should be compared based upon the strength of their arguments. Usually this means that experimentally derived hypothesis are stronger, as they are based upon experimental data, which should be considered sacrosanct. However, it should be noted that experimental data represents a subset of the recorded process, which is an incomplete picture; hence hypothesis drawn from this data can be in error. A model can be created based on hypothesis derived from experimentation, and if this model cannot attain sufficient descriptive adequacy, then it disproves the hypothesis. This is not to say that the experimentation is wrong; it cannot be, as experimental data is drawn directly from biological behaviour. This only indicates that the hypothesis is false.

Chapter 6 combines modelling and experimentally based hypothesis, and determines that a previous hypothesis is false based upon previous mathematical implementations of it. An alternative hypothesis is then put forward that suggests the presence of a new cellular mechanism, a slow bifurcation current. This hypothesis is not in conflict with previous experimental data, and when encapsulated within a model, was shown to provide extremely good descriptive adequacy. However, this does not by itself provide enough evidence for this hypothesis to be accepted, as this would require experimentation to identify this new cellular mechanism.

This model has shown itself to be extremely flexible, both in terms of extensibility, and as a candidate for iterative design, as it is able to be extended to reproduce complex firing patterns, whilst still maintaining an underlying level of simplicity. From this, it is concluded that excitability-based models can be easily extended to produce a large range of both short term (Chapter 5) and long term (Chapter 6) behaviours, and are therefore suitable candidates for rapid iterative model design.

The case study presented in Chapter 6 demonstrated that, although no measure of a model's s*implicity* has been taken into account, the concept of using parameter estimation to evaluate models is sound, and in fact required for more complex neural

models. This is because the structure of the measures of fitness used to guide the EA presents more information to the user than just how successful a model is at reproducing a behaviour; thus the user has an idea of the model's level of complexity, and through the measures of fitness, whether that level of complexity is suitable. Additionally, it is important to realise that the alternative, heavily mathematical approach, where optimal parameter sets are derived, would be difficult for even a trained mathematician to solve (as well as computationally intractable for more complex problems), let alone the non-mathematical specialists who are the target of this work. This leads to the final conclusion that rapid iterative model design, by non-computer scientists within the laboratory environment is currently possible, and that the technology required to implement such a method, in terms of hardware acceleration, adaptation algorithms, graphical user interface, and object oriented model design, are largely available off-the-shelf.

7.2 Conclusions

This thesis has investigated the better integration of the electrophysiological and modelling aspects of neuroscience, by taking the initial steps towards enabling modelling to be performed by electrophysiologists, with little or no mathematical, and no computer science, expertise, alongside experimentation within the laboratory environment. It is hoped that this work will eventually be extended to produce a development environment allowing rapid iterative model design.

Throughout this work, conclusions were drawn regarding the development and utilisation of the tools needed to perform rapid iterative model design. Here, these conclusions are collated and divided into four sections, corresponding to the main sections of the thesis; a review of the literature; the development of a tool for rapid model analysis; the investigation of excitability-based models as a candidate to form the basis of an object oriented model design process; and finally the further testing of the excitability-based model's extensibility, which was achieved through the analysis of a case study investigating the behaviour of vasopressin releasing neurons.

7.2.1 Conclusions from the literature

- There is currently no modelling environment specifically designed to cater to laboratory based neuroscientists. Such an environment would need to be able to hide the complexities of computational modelling (coding and mathematics), whilst still allowing the neuroscientist to rapidly construct and evaluate models, within the laboratory environment.

- On the basis of the literature regarding the four forms of hardware acceleration reviewed, GPUs were thought to be best suited to accelerate evolutionary algorithms within the laboratory environment, due to the suitability of EAs, which are classed as "embarrassingly parallel" tasks, to the massively parallel nature of GPUs, in addition to the low cost, ease of integration into laboratories, ease of upgradeability, and total resource control that they offer.

7.2.2 Conclusions from Chapter 4, the Development of a New Rapid Model Evaluation Method

In Chapter 4, a comparative algorithm analysis was performed on three archetypal evolutionary algorithms, along with a fourth algorithm specially constructed to test whether better utilisation of the population produced a significant performance improvement (Section 4.4).

- The results of the comparative algorithmic analysis showed that, out of the three archetypal evolutionary algorithms, Genetic Algorithms (GAs, Section 3.3.1) were better suited to optimising the neural models presented in this thesis, whilst under the constraints set by the GPU architecture, than either Particle Swarm Optimisers (PSOs, Section 3.3.2) or Estimation of Distribution Algorithms (EDAs, Section 3.3.3).

- The results of the study showed that the choice of hardware acceleration greatly affects the performance of an evolutionary algorithm, where the performance can be drastically reduced when the algorithm is ill suited to the adaptation environment imposed by the hardware acceleration. An example of this is the poor performance of the PSO in the comparative algorithmic analysis.

- The analysis of the DAGA (Section 4.4.1.4) suggested that utilisation of co-evolutionary approaches provides a way to better exploit the massive parallelism provided by GPUs, as they can better exploit the resources available to them.

In answer to the overarching hypothesis presented in the introduction: *that a rapid model evaluation tool, based upon parameter estimation, utilising evolutionary algorithms, and implemented using graphics card hardware acceleration, can provide a modelling framework that is flexible, cost effective, and suitable for a laboratory environment.*

- During chapter 4, an algorithmic acceleration of 150 times that of a single CPU approach was attained. (It should be noted that this increase was model specific, and the acceleration achieved with the more complex model and fitness evaluations used in chapter 6 was reduced. However, even with reduced performance, accelerations of at least two orders of magnitude were attained) It can therefore be concluded that GPU-based hardware acceleration is highly suited to performing embarrassingly parallel fitness evaluations, which form the bulk of any evolutionary algorithm used to optimise models evaluated through simulation.

- Chapter 4 successfully demonstrated the ability to perform rapid parameter estimation utilising off-the-shelf GPU-based hardware acceleration, which can be easily installed within an electrophysiology laboratory. The DAGA algorithm combined with GPU-hardware acceleration was used throughout Chapters 5 and 6 to evaluate, explore, and develop neural models.

7.2.3 Conclusions from Chapter 5, Excitability-based Models as a Candidate for Modular Model Design

Chapter 5 investigated the suitability of excitability-based models to form the basis of an iterative design method. During this chapter, the basic model was used to investigate several sets of pre-recorded neural data collected from the VMH. This data had already been pre-classified into 9 subsets, four of which were used in this study. Three tests were performed on the model to verify its *descriptive adequacy* (using the rapid model analysis method developed in Chapter 4), *simplicity*, and *generalisability* (these three

terms are explained at the beginning of chapter 4), thereby testing the first particular hypothesis presented in the introduction: *Excitability-based models are modular in nature and, whilst simple in structure, can be made to mimic the behaviours of a wide range of cell types though parameter adaptation*.

- The rapid model analysis performed in Section 5.3 showed that the basic excitability-based model can recreate the behaviours of a wide range of continuous firing neurons, thereby demonstrating good d*escriptive adequacy*.

- Through the use of a correlation-based method of analysis, the basic excitability-based model is shown to be robust and have good *simplicity*, insofar as similar configurations produce similar behaviours.

- The model's s*implicity* is further corroborated by the distinctive differences shown in the spike response of three of the four pre-defined classes of neuron; and that modifications of these spike responses produced behaviours of other classes of neurons, including the five classes from the original study that were not used. This not only provides evidence of the model's *generalisability*, but also led to the conclusion that each class has its own broad type of spike response, where large areas within the model parameter space correspond to broad classes of behaviour, which are defined by the existence and combination of their HAP, DAP, and AHP components.

7.2.4 Conclusions from Chapter 6, Case Study: Development of a Model of the Vasopressin Releasing Neuron

Chapter 6 furthered the investigation of the basic excitability-based model by performing a case study to test its extensibility, thereby testing the second particular hypothesis presented in the introduction. This case study involved the development of a neural model by iterative design, where the neuron of interest was the vasopressin releasing neuron found within the hypothalamus. This neuron emits a distinctive phasic pattern of activity, and it is this that the modelling process seeks to explain. At the beginning of the chapter, it was hypothesised that the behaviour is caused by activity and voltage dependant ion channels. The conclusions drawn from the testing of this hypothesis are:

- A study of the literature suggested that the DAP is not solely responsible for the sub threshold *in-vivo* membrane plateau that causes phasic behaviour. Thus, the effect of Dynorphin on the DAP can also be regarded as less significant than other factors. However, this is not to say that Dynorphin has no effect on the cellular mechanisms, as Dynorphin has been shown to control bursting behaviour. Therefore, the logical conclusion is that there is an additional mechanism affected by co-released Dynorphin, which has control over the phasic nature of the cell.

- The structure and descriptive adequacy of the final model suggests that an activity dependent mechanism, consisting of a sub threshold bifurcation on the membrane, is largely responsible for the observed *in-vivo* phasic behaviour.

Conclusions drawn from the outcome of the case study:

- It has been shown that excitability-based models can be easily extended to produce a large range of both short term (Chapter 5) and long term (Chapter 6) behaviours, and therefore they are a suitable candidate for rapid iterative design.

- It has been shown that rapid iterative model design, by non-computer scientists within a laboratory environment, is currently possible, and that the technology required to implement such a method, in terms of hardware acceleration, adaptation algorithms, graphical user interface, and object oriented model design, are largely available off-the-shelf.

7.3 Future Work

This thesis has presented the initial steps of a somewhat open-ended project, to develop a method for rapid iterative design of neural models with the laboratory environment. Because of this, the possibilities for future work are numerous and broad. The following sections focus on the three main categories of future work: topics that relate to the rapid iterative design method, topics that relate to its use, and finally, topics that, although not directly linked to the main driving force of the thesis, came to light during the electrophysiological aspects of the work.

7.3.1 Rapid iterative model design method

- The implementation of an intuitive, electrophysiologist-targeted, graphical user interface (GUI) is the most important aspect of any future work to be carried out. This cannot be emphasised enough, as the GUI will be the main factor that determines whether the target audience will choose to adopt the tool.

- The final version of the rapid adaptation method had difficulty searching the more complex vasopressin model for desired behaviours. Because the entire rapid iterative design philosophy is based upon the robustness of the adaptation algorithm, more work is required to improve this process. It should be noted that, in the time between the first implementation of the GPU hardware acceleration and the time of writing, the computational throughput supplied by GPUs has increased by nearly an order of magnitude, and thus a portion of this increase in power could be used to perform a more thorough search of the parameter space.

- The conclusions from Chapter 4 suggest that a co-evolutionary approach to optimisation may offer a way to better utilise the ever increasing resources supplied by the GPU architecture, and thus this is one potential avenue of research. Additionally, as the degree of parallelism steadily increases, in an effort to combat the limitations of minimum feature size in fabrication, resource utilisation is likely to become relevant to a wide range of applications.

- To aid creation of a robust interface to the adaptation algorithm, that provides greater information regarding causality of behaviour, true Pareto non-dominance fitness analysis method should be implemented, rather than the current combinatorial approach. A true Pareto approach would bring many benefits, including a simpler control interface and a more explorative search method, as well as providing the ability to explore the ties between the solution and parameter spaces though, the exploration of the non-dominated solution set.

- To enable modular model design, it is imperative that there be a pool of readily available pre-designed modules, and hence one of the major aspects of any

future work would be to isolate and implement these modules within an object oriented hierarchy, similar to that of the SNNAP framework (Chapter 3.1).

7.3.2 Applications

One of the main advantages of a rapid parameter estimation method, available to experimentalists within the laboratory environment, is that it allows new forms of experimentation, which can provide direct insight and feedback to the experimentalist within the timeframe of an experiment. The following sections briefly explain some test concepts that could be performed, if a trusted parameter estimation method was implemented within the laboratory.

One of the eventual aims of this work is to be able to construct, evaluate and run a model alongside a biological sample, allowing stimuli to be applied to both model and biology simultaneously. The resultant behaviour would be monitored, and deviations in behaviour would highlight potential problems with the model. The ability to rapidly adapt to not only the general averaged behaviour of a cell, but also to its internal state, presents a significant challenge. This is especially problematic when the model has non-deterministic elements, such as noise sources.

Before and after stimulus causality tests

A trusted model can be utilised to gain insight into the underlying effects a stimulus has on the biology. This is done by first adapting the model to biology in homeostasis, in order to estimate its parameters (which correspond to physical properties of the biology). Next, a stimulus (electrical or chemical) that alters the behaviour of the biology to produce a new homeostatic behaviour is applied. The model is then re-adapted to the new behaviour from the old behaviour, where the change in parameter values will provide insight into the effect the stimulus had on the biology.

Classification

A trusted model could be used to quickly determine a cell's type as it is being recorded. The advantage of this is that is does not rely on human judgment to make a decision, but can still be guided by it, through the training of the classification algorithm. For a

model where the resulting behaviour of the parameter space is contiguous, such as the basic excitability-based model (Chapter 5), classification could be performed through the parameter estimation task.

Exploring sub systems

All the models presented in this thesis share a common feature, in that they use a noise source to represent incoming synaptic events. While all the models discussed use uniform Gaussian distribution, a more elaborate representation of noise could be used to investigate synaptic input change, in a manner similar to the stimulus causality test described above. The majority of control parameters would be fixed at their initially adapted values, with only those controlling the noise distribution allowed to change. This is a generic method, which could also be used to investigate other aspects of a cell.

7.3.3 Modelling

To identify the range of behaviours that can be produced by the model, and how they relate to previously observed behaviours, a thorough search of the parameter space of a reduced version of the excitability-based model could be performed. This would be done in a manner similar to Prinz et al. (2004), exploring the range of behaviours produced by altering the shape of the triple exponential spike response.

One of the primary goals of producing a simple model is the push towards a network-based architecture. This is particularly relevant to the vasopressin releasing neurons, as they present several interesting population dynamics designed to protect the population from overactivity, whilst maintaining the correct proportion of vasopressin within the body. Understanding of these dynamics would not only provide vital information about the endocrine system, but may also have applications in other disciplines which require load balancing and self preservation mechanisms.

Appendix A Basic Excitability-based Model

Discrete, rather than continuous, differential equations have been used to describe model implementations throughout this thesis because structurally, they are closer to how a model is implemented for simulation. However, for the purposes of a universal description, the behaviour of the basic excitability-based model (explored in Chapter 5) is now presented in continuous form.

The basic model consists of two main sections; The noise source, with its integration with the membrane potential and firing mechanism, and the spike response, which is split into three component parts, the HAP, DAP and AHP. During the formal description, control parameters are highlighted with **(CP)**, with the remaining symbols corresponding to internal variables.

A.1 Noise Source

$$\frac{dN}{dt} = A_N . \frac{dG_U}{dt}$$

N - Incoming Synaptic Noise
G_U - Uniform Gaussian Random number
A_N - Noise Amplitude **(CP)**

A.2 Noise Integration

$$\frac{dE_M}{dt} = \frac{\left(E_R - E_M\right)}{\tau_M} + E_R + N$$

E_M - Membrane Excitability
E_R - Membrane Rest Excitability **(CP)**
τ_M - Membrane Time Constant **(CP)**
N - Incoming Synaptic Noise

A.3 Firing Mechanism

A spike is produced at time $t = t_S$, S = 1,2,…, , if $E_T(t_S) = E_{TH}$, where $E_T(t_S)$ is the total cell excitability, including the effects of the HAP, DAP, and AHP, membrane decay and incoming synaptic events, and E_{TH} is the excitability threshold.

$$E_T = E_M + H + D + A$$

E_T - Total Cell Excitability
H - HAP Excitability
D - DAP Excitability
A - AHP Excitability

A.4 Spike Response

The spike response is split into three sections, each corresponding to the HAP, DAP or AHP. Even though they have completely different timescales, the controlling differential equations have exactly the same form. The Dirac delta function is used to represent the impulse response of the cell firing at times of t_s, where S = 1,2,…,.

A.4.1 HAP

$$\frac{dH}{dt} = -\frac{H}{\tau_{HAP}} + S_{HAP}\,\delta(t_s)$$

H - HAP Excitability
τ_{HAP} - HAP Decay Constant **(CP)**
S_{HAP} - HAP Step Amount **(CP)**

A.4.2 DAP

$$\frac{dD}{dt} = -\frac{D}{\tau_{DAP}} + S_{DAP}\,\delta(t_s)$$

D - DAP Excitability
τ_{DAP} - DAP Decay Constant **(CP)**
S_{DAP} - DAP Step Amount **(CP)**

A.4.3 AHP

$$\frac{dA}{dt} = -\frac{A}{\tau_{AHP}} + S_{AHP}\,\delta(t_s)$$

A - AHP Excitability
τ_{AHP} - AHP Decay Constant **(CP)**
S_{AHP} - AHP Step Amount **(CP)**

Appendix B Vasopressin Releasing Neural Model

Discrete, rather than continuous, differential equations have been used to describe model implementations throughout this thesis because structurally, they are closer to how a model is implemented for simulation. However, for the purposes of a universal description, the behaviour of the final version of vasopressin releasing neural model (explored in Chapter 6) is now presented in continuous form.

The vasopressin releasing neural model consists of four main sections; The noise source; the spike response, which is split into three component parts, the HAP, DAP and AHP; the membrane potential bifurcation mechanism that controls the model's phasic behaviour, and integrates the effects of the HAP, DAP, AHP, and noise; and the firing mechanism. During the formal description, control parameters are highlighted with **(CP)**, derived parameters are highlighted with **(DP)**, with the remaining symbols corresponding to internal variables.

B.1 Noise Source

$$\frac{dN}{dt} = -\frac{N}{\tau_N} + A_N . \frac{dG_U}{dt}$$

N - Incoming Synaptic Noise
G_U - Uniform Gaussian Random number
A_N - Noise Amplitude **(CP)**
τ_N - Noise Time Constant **(CP)**

B.2 Spike Response

The spike response is split into three sections, each corresponding to the HAP, DAP or AHP. Even though they have completely different timescales, the controlling differential equations have exactly the same form. The Dirac delta function is used to represent the impulse response of the cell firing at times of t_s, where S = 1,2,…,.

B.2.1 HAP

$$\frac{dH}{dt} = -\frac{H}{\tau_{HAP}} + S_{HAP}\,\delta(t_s)$$

H - HAP Excitability
τ_{HAP} - HAP Decay Constant **(CP)**
S_{HAP} - HAP Step Amount **(CP)**

B.2.2 DAP

$$\frac{dD}{dt} = -\frac{D}{\tau_{DAP}} + S_{DAP}\,\delta(t_s)$$

D - DAP Excitability
τ_{DAP} - DAP Decay Constant **(CP)**
S_{DAP} - DAP Step Amount **(CP)**

B.2.3 AHP

$$\frac{dA}{dt} = -\frac{A}{\tau_{AHP}} + S_{AHP}\,\delta(t_s)$$

A - AHP Excitability
τ_{AHP} - AHP Decay Constant **(CP)**
S_{AHP} - AHP Step Amount **(CP)**

B.3 Firing Mechanism

A spike is produced at time $t = t_S$, S = 1,2,…, , if $E_T(t_S) = E_{TH}$, where $E_T(t_S)$ is the total cell excitability, including the effects of the HAP, DAP, and AHP, membrane decay and incoming synaptic events, and E_{TH} is the excitability threshold.

$$E_T = E_M + H + D + A$$

E_T - Total Cell Excitability
H - HAP Excitability
D - DAP Excitability
A - AHP Excitability

B.4 Membrane Potential Bifurcation Mechanism

The dynamics of the membrane potential are complex, being influenced by various long and short term effects (Noise, HAP, DAP, AHP, and the Slow Inhibition), that increase or decrease the membrane excitability. To calculate the change in the membrane potential, the value of this excitability, including all the influences, must be calculated.

$$\frac{dE_{\inf}}{dt} = \frac{dE_M}{dt} + N + \begin{cases} \frac{dH}{dt} + \frac{dD}{dt} + \frac{dA}{dt} & : \quad H + D + A > 0 \\ \frac{dD}{dt} + \frac{dA}{dt} + 0 & : \quad D + A < 0 \\ 0 & : \quad otherwise \end{cases}$$

$E_{\inf}$ - Membrane Excitability with influence
E_M - Membrane Excitability
N - Incoming Synaptic Noise
H - HAP Excitability
D - DAP Excitability
A - AHP Excitability

The membrane potential with influence is then used to determine the change in base membrane excitability.

$$\frac{dE_M}{dt} = S.\left(E_{Offset} - I\right) + \begin{cases} -\frac{\left(E_{\inf} - \frac{E_{RP}}{2}\right)}{\tau_M} & : \quad E_{\inf} > \frac{E_{RP}}{2} \\ -S.\left(4E_{\inf}^3 + 2uE_{\inf}\right) & : \quad \frac{E_{RP}}{2} > E_{\inf} > -\frac{E_{RP}}{2} \\ -\frac{\left(E_{\inf} + \frac{E_{RP}}{2}\right)}{\tau_M} & : \quad -\frac{E_{RP}}{2} > E_{\inf} \end{cases}$$

E_M - Membrane Excitability
S - Bifurcation Scale **(DP)**
E_{Offset} - Initial Offset **(DP)**
I - Slow Inhibition
$E_{\inf}$ - Membrane Excitability with influence
E_{RP} - Rest to Plateau Excitability **(CP)**

τ_M - Base membrane decay constant **(CP)**
u - Derived parameter that controls size of the bifurcation **(DP)**

The above equation utilises the slow inhibition, which is an activity dependant decaying variable, much like the HAP, DAP, and AHP.

$$\frac{dI}{dt} = -\frac{I}{\tau_{Inhib}} + S_{Inhib}\,\delta(t_s)$$

I - Slow Inhibition
τ_{Inhib} - Slow Inhibition Decay Constant **(CP)**
S_{Inhib} - Slow Inhibition Step Amount **(CP)**

In addition, three derived parameters are required to control the bifurcation. The first sets the initial offset of the bifurcation.

$$E_{Offset} = -3^{-\frac{3}{2}} E_{RP}^3 E_{\%}$$

E_{Offset} - Initial Offset **(DP)**
E_{RP} - Rest to Plateau Excitability **(CP)**
$E_{\%}$ - % of base offset used as the initial offset **(CP)**

The second derived parameter controls the size of the bifurcation.

$$u = \frac{E_{RP}^2}{2}$$

u - Derived parameter that controls size of the bifurcation **(DP)**
E_{RP} - Rest to Plateau Excitability **(CP)**

The last controls the rate of decay between the rest and plateau potentials by scaling the bifurcation function.

$$S = \frac{\tau_M}{E_{RP}^2}$$

S - Bifurcation Scale **(DP)**
τ_M - Base membrane decay constant **(CP)**
E_{RP} - Rest to Plateau Excitability **(CP)**

Appendix C Initial FPGA based Real-Time Implementation of the Vasopressin Releasing Neuron

The initial aim of this work was to create a tool that would allow for a model to be simulated in real-time alongside an *in-vivo* laboratory session recording from a vasopressin releasing neuron. The presumed next step would be to apply a stimulus to both the model and the biology at the same time, as see if and how their behaviours diverged. For this particular experiment, the stimulus would be an excitatory pulse that would cause a vasopressin releasing neuron to fire rapidly for a couple of seconds. The hypothesis was that a slow activity dependant inhibitory process controls the bursting behaviour seen in these neurons, and hence by stimulating the cell, it would excite this process and, rather counter-intuitively, terminate the burst if the burst was already of sufficient length.

For this tool to function, two criteria must be met:

1. A software driven, real-time simulation platform must be provided, which is not dependent on a PC for timing information.

2. A parameter adaptation method to minimise the difference between the behaviour of a model and that of the neuron under test. This includes tuning the phase of the model, such that burst and silent periods co-inside with the biology.

The FPGA-based real-time implementation of the vasopressin releasing neuron was the initial attempt at meeting the first criteria.

C.1 Initial Parallel Method

The first incarnation of the simulation platform was a direct implementation of the vasopressin releasing neural model initially developed by Durie, (2007). It consisted of four exponential decay units, one 64-bit and three 32-bit, which calculated the slow inhibition and the three exponential decays that form the excitability pattern. Each exponential decay unit was designed to perform Equation C1, and consisted of a fixed

point multiply and accumulate unit that calculates the next value of the exponential decay based upon its current value, and if a step amount is required.

$$E_{t+1} = E_t . D_{DAP} + \begin{cases} S_{DAP} & : NeuronFire \\ 0 & : NotFiring \end{cases}$$ **Equation C1**

E - Exponential Value
D_{DAP} - Decay Constant. This is a number between 0 and 1 (usually very close to 1)
S_{DAP} - Step Amount.

Because the first incarnation was implemented within an FPGA, each iteration of every exponential decay was calculated in parallel. Alongside these processes was a state machine with two states that calculated if the membrane potential decayed towards the rest or plateau value. Finally, the state of the neuron was calculated (firing or not firing), which was used to determine if step increases were applied to the exponential decays. The time-step of the simulation was 100µs.

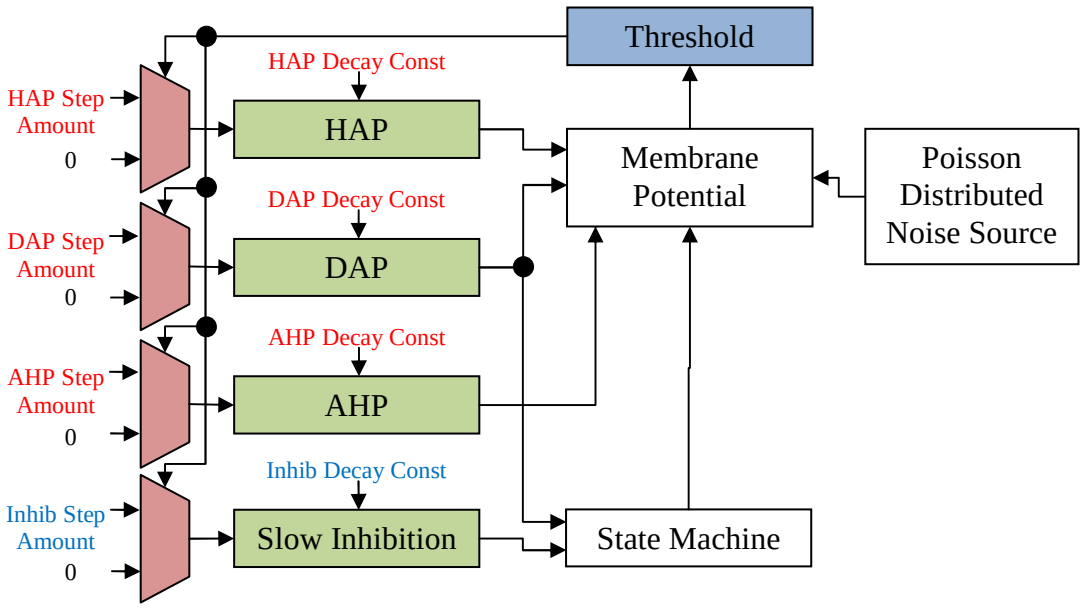


Figure 27 Block diagram of the initial model implementation. Red boxes are multiplexors; Green boxes are multiply and accumulate units, where the decay constants are one of the inputs to the internal multiplier (the other is the internal variable under decay). The blue box is a comparator that determines if the threshold of the model neuron has been met. All red text are 32-bit fixed point parameters, and blue test are 64-bit fixed point parameters. All parameters are stored in dual port block memory, accessible by the model and the RS232 interface, which is not shown.

The entire model was wrapped in an interface that allowed, at runtime via RS232, parameters to be re-configured and variables monitored and reset if wished. Software was written to record the state of each variable, and the timing of output events. Figure C-1 presents a block diagram of the model implementation.

C.2 Custom Processor Method

The previous implementation was developed to gain support from electrophysiologists by demonstrating the ability to provide an interface to modelling within the laboratory. Once the aims of the project were more substantial, and a GPU solution was being used to solve the rapid parameter fitting task, this implementation was discarded for a more general solution that utilised a simple custom processor, and could therefore be used to implement a range of similar models. This is important because, the rapid parameter fitting task was being used to generate new models that would eventually be required to run alongside experimentation in real-time.

The processor had instructions designed specifically to efficiently simulate neural models consisting of exponential decays and step functions. Unlike the previous incarnation, this implementation utilised a floating point multiplier and floating point adder/subtractor, both 32-bit, bringing it more in-line with the GPU architecture, although the floating point standards used within the FPGA and GPU still differed slightly. Model control parameters and variables were placed in block ram, as was the model program. An Opal Kelly board was used as the target device, as it includes an interface that allows direct access to user designs from high level software, such as $C^{\#}$, through pre-generated modules. See the Opal Kelly, (2009) website for more details. Finally, unlike the previous version, the implementation was designed to have a time-step of 1ms, the same as the software and GPU simulations. Figure C-2 presents the structure of the processor core.

C.3 Outcome

Both versions of the real-time simulator succeeded at their intended purpose. The first showed how easily a model could be implemented within the laboratory, and the second completed the loop between recording biological data, and mimicking it though the

creation and near real-time adaptation of a model and its parameters to re-produce the recorded behaviour.

The second custom processor based platform was also a proof of concept for the utilisation of custom instruction sets to streamline the description of neural models. Any future work will require a real-time model simulator to interface with biological experimentation. By using custom instruction sets to describe a model, the divide between model structures simulated on different forms of hardware, and at different levels of abstraction becomes smaller, as instructions can be created that are more akin to objects. This can only be done because the simulation is time step based, and therefore repeats the same instructions at each iteration of the model. Because of this, a section of the repeated code can be seen more of an object, having fixed methods that utilise or control its own internal parameters and variables.

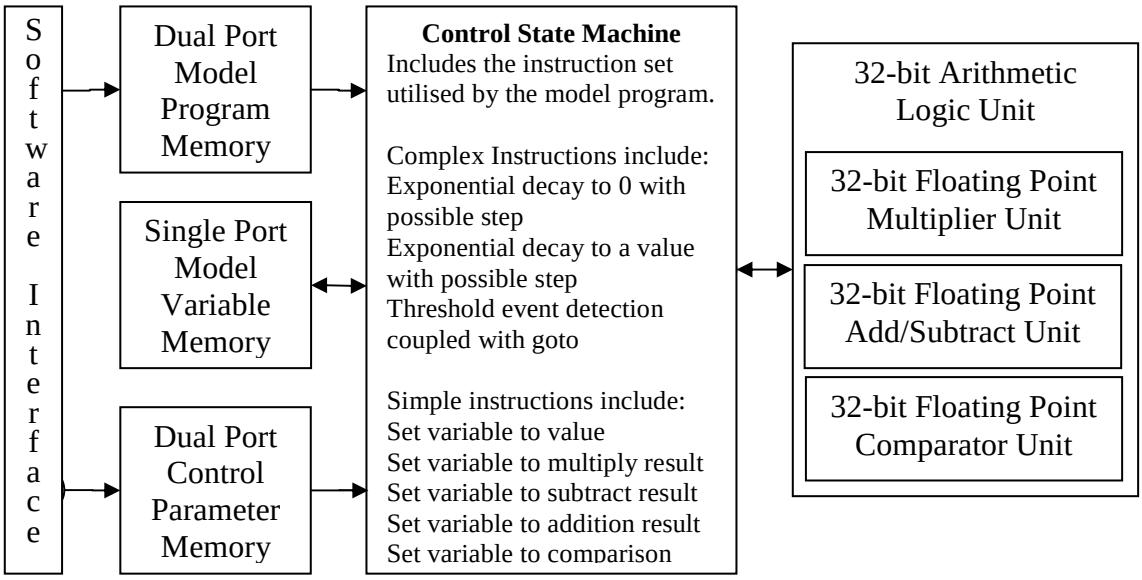


Figure 28 Block diagram of the custom processor approach. The 32-bit arithmetic logic unit was created through the combination of several Xilinx pre-generated IP cores. Because of the difference in the complexity of different arithmetic operations, the latency from presenting inputs to getting the result ranged from 4 clock cycles for multiplication, to one for addition/subtraction. Both the model program memory and the parameter memory are dual port, although they can only be written to via an Opel Kelly IP core, through a USB interface, by the controlling software. A control state machine arbitrates the data flow to implement a series of complex and simple operations. The complex operations can be created by using simple operations, however, complex operations will be more efficient as they are designed to coincide memory fetches with arithmetic logic unit latency.

Appendix D A Brief Introduction to GPU Constraints

Current day GPUs consist of a several of SIMD (Single Instruction Multiple Data) processors running in parallel. Although the number and size of these processors vary between architectures, there are similarities among all GPU frameworks. GPUs dedicate much more of their chip area to processors, and less to data path control, as shown in Figure D-1. This makes them very good at performing massively parallel highly arithmetic tasks. The following sections discuss the programming constraints set by this architecture, with a specific focus on the nVidia G80 chipset that is at the core of the 8800GTX GPU, which is used in this thesis. Much of the information in this appendix can be found in (NVIDIA, 2008a), which should be read before attempting to utilise nVidia's GPU programming environment (CUDA).

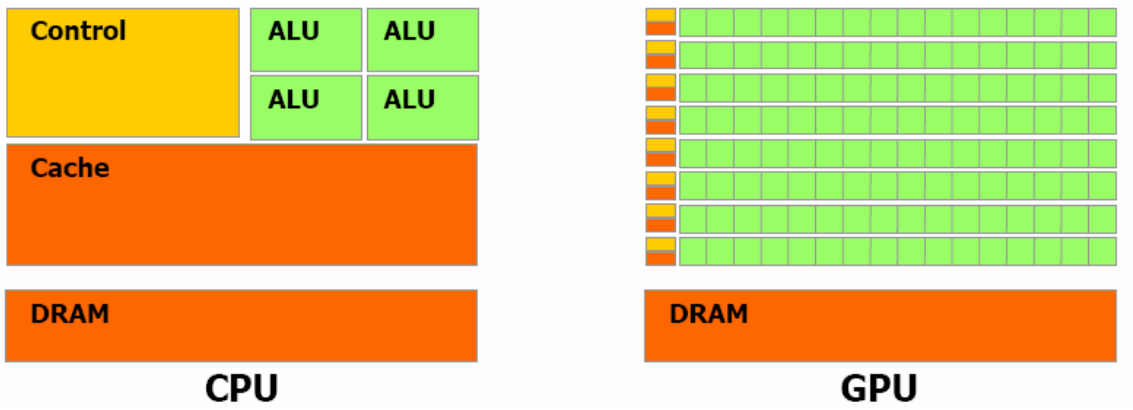


Figure 29-1 Distribution of transistors within a typical CPU vs a GPU. GPUs consist of much less control logic but many more times the number of arithmetic logic units. This makes them bad at performing more complex data management operations such as out of order execution, but much better at raw arithmetic computation. Image re-printed from (NVIDIA, 2008a).

D.1 GPU Memory Architecture

GPUs have a complex memory architecture designed specifically to streamline data access. This data usually represents images or textures, but for this work can also hold other data, such as model control parameters, and variables. The G80 memory architecture is split into several sections; register memory; shared memory; constant memory; texture memory; and global memory; see Figure D-2.

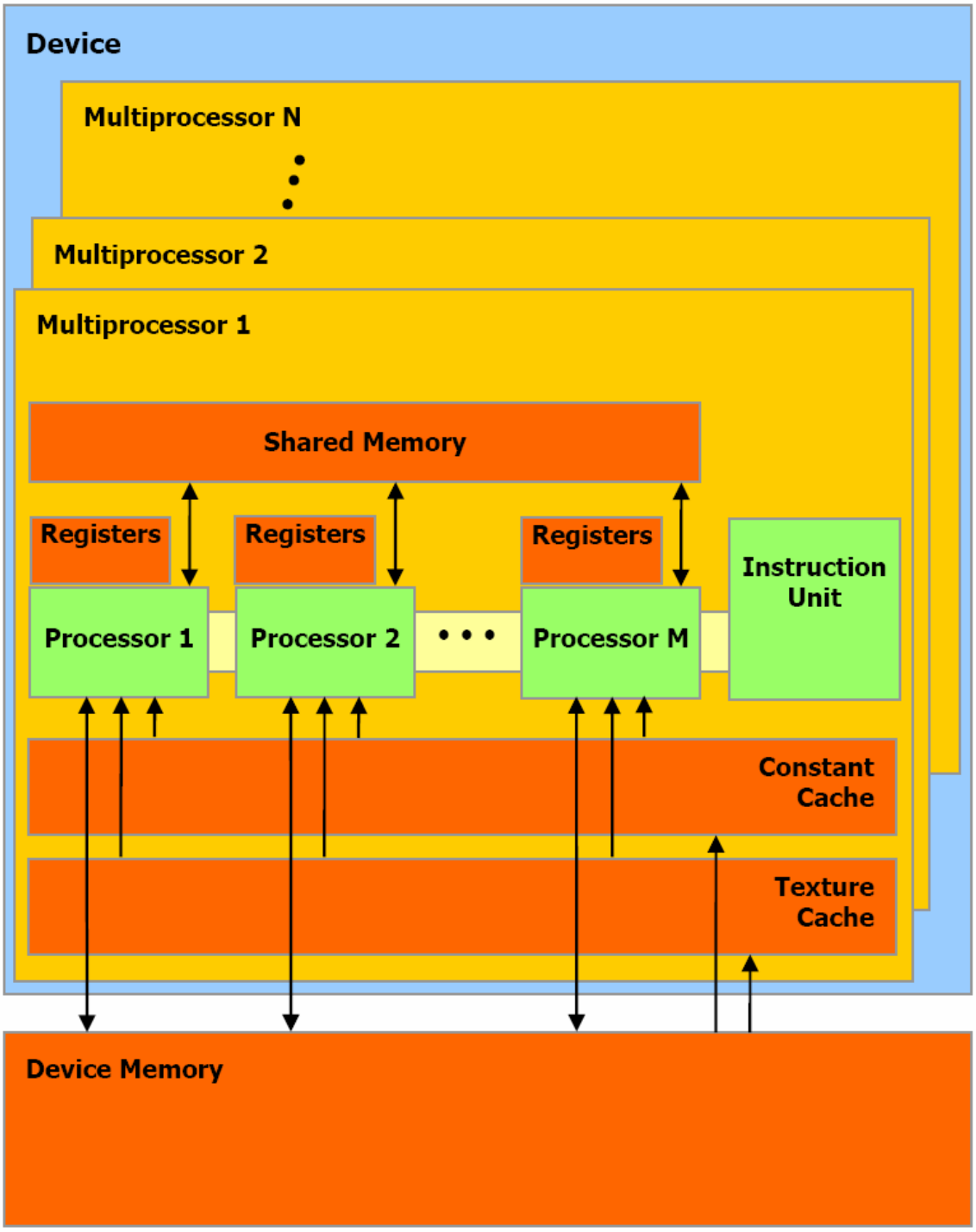


Figure 30 Conceptual block diagram of the memory architecture. For the G80 core, each SIMD multiprocessor (of which there are 16) consists of 8 32-bit floating point processors. Each processor has its own set of 32-bit register memory, which is used to store data between sequential operations. Each multiprocessor includes; 16kB of shared memory, accessible by each processer within the SIMD core; two read only caches of memory, constant and Texture caches, which are used to pre-fetch data from device memory (off-chip); and each SIMD core has access to the global 768MB pool of off-chip GDDR3 memory. Image re-printed from (NVIDIA, 2008a).

Access to register and local memory is extremely fast, with a latency of 1 clock cycle, hence when programming for the GPU it is important to optimise the use of local memory. Constant and Texture memory both have a latency of 1 clock cycle, but only if the location within the constant or texture memory is deterministic. If this is the case then the data can be pre-fetched from the external device to the cache before it is required, if this is not the case, then the latency is much higher (over a thousand times slower), as the data must be fetched directly from the off-chip memory.

This represents one of the main problems with the use of GPUs for simulating neural experiments, as the methods of data collections (such as the creation of ISI histograms) are non-deterministic and hence poorly suited for implementation within a GPU framework. However, the actual model is highly deterministic, with control parameters and variables called at known times. Hence, conversely, model implementation is highly suited to a GPU framework.

D.2 Processors, Threads, and CUDA

NVidia GPUs are programmed through the CUDA (Compute Unified Device Architecture) framework. CUDA provides one or more layers of abstraction from the GPU hardware via the CUDA drivers, the CUDA Runtime environment, and CUDA Libraries. There are currently two libraries, CUFFT and CUBLAS, which provide efficient solutions to common computational problems regarding fast Fourier transforms and matrix manipulation respectively.

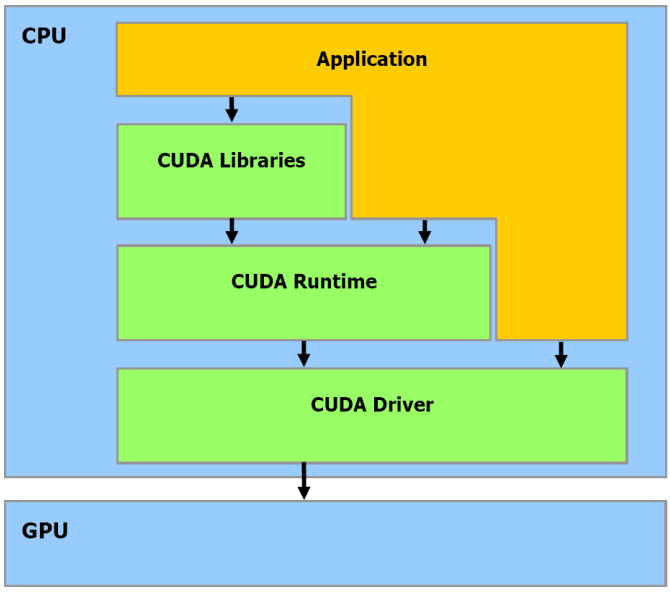


Figure 313 Conceptual block diagram of the various layers of abstraction that make up the CUDA framework. Image re-printed from (NVIDIA, 2008a).

This work utilises the CUDA runtime environment, which provides a C++ interface to the CUDA driver. This interface includes extensions to the C language that allow the description of SIMD programs. A typical CUDA program includes two sections. The first sets up the CUDA environment on the host processor, and involves the following steps.

1. Find and initialise the GPU device and make it CUDA ready
2. Allocate and load/create host (the PC side) input memory for input data
3. Allocate device (the GPU side) memory for input data
4. Allocate host output memory for output data
5. Allocate device memory for output data
6. Load input data from host memory to device memory
7. Run one or more kernels
8. Transfer output data from device back to the host
9. Do something interesting with the output data

The second section describes what is to be run within the GPU, and consists of one or more kernels. A kernel is a set of instructions that is run on an SIMD multiprocessor, and hence each arithmetic logic unit (ALU) within it. Across the GPU, a kernel can be run a number of times in parallel based upon the resources a single instance of a kernel utilises. Multiple kernel instances (threads) can often be run on a single ALU, where the local memory is used to store the state of each thread. This is often done with only a minor increase in processing time, as the bottleneck in many applications is not the arithmetic computation, but fetching data from the external memory, hence the extra computation overlaps and hides memory latency.

Branching statements, such as ‘if’ and ‘switch’ are poorly suited to the GPU environment because the same instructions are simultaneously run across each ALU within an SIMD multiprocessor core. Therefore, if even one thread decides to go down one branch then each ALU must perform the whole branch, even if it does not actually apply the instructions within the branch to any data. When programming for this type of environment, it is often more efficient to turn a conditional statement into a mathematical one by multiplying a result by 1.0 or 0.0, as the architecture is much better suited to resolve complex mathematical computation than simple conditional

statements (the G80 core can perform two floating point operations ‘flops’ per clock cycle).

D.3 Summary

Writing a program that is to be run on a GPU is conceptually different from traditional programming. With a CPU, the hardware is flexible, and designed to accommodate a variety of computational styles. However, when writing a program to solve a problem with a GPU, often the first objective is to reorient the problem so that it is better suited to the GPU architecture. Some problems, usually those that are highly non-deterministic in their use of memory, or rely on sophisticated data control flow (lots of branching statements), are just not suitable to be accelerated by GPUs. Others, such as N-body simulation, which involves the same series of operation performed once for each pair of particles (an $O(n^2)$ problem), is extremely suitable.

The use of GPUs to accelerate thousands of concurrent neural simulation lies between the two extremes, but still providing significant increase in performance. By itself, the acceleration of simulation is highly suited, but unfortunately, data collection methods that allows external processes to weigh the outcome of a simulation are usually non-deterministic, and therefore the GPU implementation suffers from excessive memory access latencies.

D.3.1 Three golden rules of GPU programming

- Make sure that your problem suits the GPU environment
- Try to use local memory as much as possible, and if forced to use alternative memory, use constant or texture memory, both of which have on chip caches so as to be able to pre-fetch data.
- Avoid the use of branching statements such as ‘if’ and ‘switch’. If at all possible substitute mathematical operations instead of branching statements, as the GPU is optimised to perform these.

An example of the CUDA host processor and kernel code for the simulation of the vasopressin releasing neuron presented in chapter 6 is provided at the end of this appendix.

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM.cu 1

```
/*
 * Copyright 1993-2007 NVIDIA Corporation.  All rights reserved.
 *
 * NOTICE TO USER:
 *
 * This source code is subject to NVIDIA ownership rights under U.S. and
 * international Copyright laws.  Users and possessors of this source code
 * are hereby granted a nonexclusive, royalty-free license to use this code
 * in individual and commercial software.
 *
 * NVIDIA MAKES NO REPRESENTATION ABOUT THE SUITABILITY OF THIS SOURCE
 * CODE FOR ANY PURPOSE.  IT IS PROVIDED "AS IS" WITHOUT EXPRESS OR
 * IMPLIED WARRANTY OF ANY KIND.  NVIDIA DISCLAIMS ALL WARRANTIES WITH
 * REGARD TO THIS SOURCE CODE, INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.
 * IN NO EVENT SHALL NVIDIA BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL,
 * OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
 * OF USE, DATA OR PROFITS,  WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE
 * OR OTHER TORTIOUS ACTION,  ARISING OUT OF OR IN CONNECTION WITH THE USE
 * OR PERFORMANCE OF THIS SOURCE CODE.
 *
 * U.S. Government End Users.   This source code is a "commercial item" as
 * that term is defined at  48 C.F.R. 2.101 (OCT 1995), consisting  of
 * "commercial computer  software"  and "commercial computer software
 * documentation" as such terms are  used in 48 C.F.R. 12.212 (SEPT 1995)
 * and is provided to the U.S. Government only as a commercial end item.
 * Consistent with 48 C.F.R.12.212 and 48 C.F.R. 227.7202-1 through
 * 227.7202-4 (JUNE 1995), all U.S. Government End Users acquire the
 * source code with only those rights set forth herein.
 *
 * Any use of this source code in individual and commercial software must
 * include, in the user documentation and internal comments to the code,
 * the above Disclaimer and U.S. Government End Users Notice.
 */

/* Template project which demonstrates the basics on how to setup a project
* example application.
* Host code.
*/
#include <windows.h>

// includes, system
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

// includes, project
#include <cutil.h>

// includes, kernels
#include <VasopressinNoSM_kernel.cu>

////////////////////////////////////////////////////////////////////////////////
// declaration, forward
extern "C" __declspec(dllexport) void runCudaVasopressinNoSM    (   float Simlength,
    int CutOffTop,
    int BinSpacing,
    int ParamNo,
    float* ParamArray,
    short* ISIArray,
    short* PhasicArray,
    float* PhasicMeanArray,
    float* PhasicStdDevArray,
    float* PhasicBurstNoise,
    float* SilenceMeanArray,
```

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM.cu 2

```
    float* SilenceVarArray,
    float* RandomNoise,
    float* Resonance
                                                                    );
////////////////////////////////////////////////////////////////////////////////
// Program main
////////////////////////////////////////////////////////////////////////////////
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    return TRUE;
}

////////////////////////////////////////////////////////////////////////////////
//Run VasopressinNoSM model
////////////////////////////////////////////////////////////////////////////////
extern "C" void
runCudaVasopressinNoSM( float Simlength,
                        int CutOffTop,
                        int BinSpacing,
                        int ParamNo,
                        float* ParamArray,
                        short* ISIArray,
                        short* PhasicArray,
                        float* PhasicMeanArray,
                        float* PhasicStdDevArray,
                        float* PhasicBurstNoise,
                        float* SilenceMeanArray,
                        float* SilenceVarArray,
                        float* RandomNoise,
                        float* Resonance
                      )
{
    ////////////////////////////////////////////////////////////////////////////
    //Count the number of vidio graphics devices cards
    int deviceCount;
    CUDA_SAFE_CALL_NO_SYNC(cudaGetDeviceCount(&deviceCount));
    if (deviceCount == 0) {
        fprintf(stderr, "There is no device.\n");
        return;
    }

    ////////////////////////////////////////////////////////////////////////////
    //For each device examine its properties and if it is a valid device select
    //it for opperation.  After examining this code, i have added the
    // "&& (deviceProp.minor == 0)" to the if statment to make it select only
    // the 8800 card.  This is a bodge and there must be a better way of doing
    //this.
    int dev;
    for (dev = 0; dev < deviceCount; ++dev) {
        cudaDeviceProp deviceProp;
        CUDA_SAFE_CALL_NO_SYNC(cudaGetDeviceProperties(&deviceProp, dev));
        if ((deviceProp.major >= 1) && (deviceProp.minor == 0))

           break;
    }
    ////////////////////////////////////////////////////////////////////////////
    //If there are no suitable cards, Exit with a suitable error, otherwise
    //select the first suitable device.
    if (dev == deviceCount) {
        fprintf(stderr, "There is no device supporting CUDA.\n");
        printf("EXIT\n");
        return;
    }
```

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM.cu 3

```
    else    CUDA_SAFE_CALL(cudaSetDevice(dev));


    //Data is already provided so go straight to the CUDA mem creation

    unsigned int num_threads        = 128;
    int i_Block_No                  = 32;
    int TotalSims                   = num_threads * i_Block_No;
    int ParamPerExp                 = ParamNo;
    int NoiseSampleLength           = 4096;

    ////////////////////////////////////////////////////////////////////////////
    //Gaussian Noise Construction                                             //
    ////////////////////////////////////////////////////////////////////////////
    // set texture parameters for Noise
    Noise.addressMode[0]    = cudaAddressModeClamp;
    Noise.addressMode[1]    = cudaAddressModeClamp;
    Noise.filterMode        = cudaFilterModePoint;
    Noise.normalized        = false;    // access with normalized texture coordinates

    ////////////////////////////////////////////////////////////////////////////
    // allocate array and copy Noise Sample data
    cudaArray* cu_iRandomNoiseSample = NULL;
    CUDA_SAFE_CALL( cudaMallocArray( &cu_iRandomNoiseSample, &Noise.channelDesc,
    NoiseSampleLength, 1 ));
    CUDA_SAFE_CALL( cudaMemcpyToArray( cu_iRandomNoiseSample, 0,0, RandomNoise,
    NoiseSampleLength*sizeof(float), cudaMemcpyHostToDevice));

    ////////////////////////////////////////////////////////////////////////////
    // Bind the array to the texture
    CUDA_SAFE_CALL( cudaBindTextureToArray( Noise, cu_iRandomNoiseSample, Noise.
    channelDesc));


    // allocate device memory for result
    short* d_oISI;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_oISI,            sizeof(short)*TotalSims
    *500));
    CUDA_SAFE_CALL( cudaMemcpy( d_oISI,         ISIArray,           sizeof(short)*TotalSims
    *500,   cudaMemcpyHostToDevice));

    short* d_oPhasic;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_oPhasic,         sizeof(short)*TotalSims
    *500));
    CUDA_SAFE_CALL( cudaMemcpy( d_oPhasic,      PhasicArray,        sizeof(short)*TotalSims
    *500,   cudaMemcpyHostToDevice));

    float* d_oPhasicMean;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_oPhasicMean,     sizeof(float)*
    TotalSims));
    CUDA_SAFE_CALL( cudaMemcpy( d_oPhasicMean,  PhasicMeanArray,    sizeof(float)*TotalSims
    ,       cudaMemcpyHostToDevice));

    float* d_oPhasicStdDev;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_oPhasicStdDev,   sizeof(float)*
    TotalSims));
    CUDA_SAFE_CALL( cudaMemcpy( d_oPhasicStdDev,PhasicStdDevArray,  sizeof(float)*TotalSims
    ,       cudaMemcpyHostToDevice));

    float* d_oPhasicBurstNoise;
    CUDA_SAFE_CALL( cudaMalloc( (void**)                &d_oPhasicBurstNoise,   sizeof
    (float)*TotalSims));
    CUDA_SAFE_CALL( cudaMemcpy( d_oPhasicBurstNoise,    PhasicBurstNoise,       sizeof
    (float)*TotalSims,      cudaMemcpyHostToDevice));

    float* d_oSilenceMean;
```

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM.cu 4

```
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_oSilenceMean,    sizeof(float)*
    TotalSims));
    CUDA_SAFE_CALL( cudaMemcpy( d_oSilenceMean, SilenceMeanArray,   sizeof(float)*TotalSims
    ,       cudaMemcpyHostToDevice));

    float* d_oSilenceStdDev;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_oSilenceStdDev,  sizeof(float)*
    TotalSims));
    CUDA_SAFE_CALL( cudaMemcpy( d_oSilenceStdDev,SilenceVarArray,   sizeof(float)*TotalSims
    ,       cudaMemcpyHostToDevice));

    float* d_oResonance;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_oResonance,  sizeof(float)*16*
    TotalSims));
    CUDA_SAFE_CALL( cudaMemcpy( d_oResonance,   Resonance,  sizeof(float)*16*TotalSims,
        cudaMemcpyHostToDevice));

    ////////////////////////////////////////////////////////////////////////////
    //Edit the Control Parameters
    for(int E = 0; E < 4096; E++)
    {
        float D = ParamArray[E*ParamPerExp + 1];
        //Param Conversion
        //(0)  - Noise Amplitude        - No Change
        //(1)  - Rest -> Plateau        - Calculate (B)  B = D*D*0.5
        ParamArray[E*ParamPerExp + 1] = (float) -D*D*0.5;
        //(2)  - Rest -> Threshold      - Change to absolute Threshold (FT = (R->T) - (R->)
    *0.5)
        ParamArray[E*ParamPerExp + 2] = (float) ParamArray[E*ParamPerExp + 2] + D*0.5;
        //(3)  - Cusp Cat H/L           - Calculate Membrane Dynamics Scaler Value (S = [1-
    e(ln(2)/(1000*MPBHL))]/[D*D]) -- This gets used in two different ways now
        //ParamArray[E*ParamPerExp + 3] = (float) (1-exp(-0.693147180559453094172321 /
    (ParamArray[E*ParamPerExp + 3]*1000.0)))/(D*D);
        //(4)  - MemBase H/L
        ParamArray[E*ParamPerExp + 4] = (float) exp(-0.693147180559453094172321 /
    (ParamArray[E*ParamPerExp + 4]*1000.0));
        //(5)  - Inhibition Decay   - Change to multiplier
        ParamArray[E*ParamPerExp + 5] = (float) exp(-0.693147180559453094172321 /
    (ParamArray[E*ParamPerExp + 5]*1000.0));
        //(6)  - Inhibition Step        - No Change
        //(7)  - HAP Decay  - Change to multiplier
        ParamArray[E*ParamPerExp + 7] = (float) exp(-0.693147180559453094172321 /
    (ParamArray[E*ParamPerExp + 7]*1000.0));
        //(8)  - Inhibition Step        - No Change
        //(9)  - DAP Decay  - Change to multiplier
        ParamArray[E*ParamPerExp + 9] = (float) exp(-0.693147180559453094172321 /
    (ParamArray[E*ParamPerExp + 9]*1000.0));
        //(10)  - Inhibition Step       - No Change
        //(11) - AHP Decay  - Change to multiplier
        ParamArray[E*ParamPerExp + 11] = (float) exp(-0.693147180559453094172321 /
    (ParamArray[E*ParamPerExp + 11]*1000.0));
        //(12) - AHP Step               - No Change
    }

    float* d_iParams;
    CUDA_SAFE_CALL( cudaMalloc( (void**)    &d_iParams,     sizeof(float)*TotalSims*
    ParamPerExp));
    CUDA_SAFE_CALL( cudaMemcpy( d_iParams,  ParamArray,     sizeof(float)*TotalSims*
    ParamPerExp,    cudaMemcpyHostToDevice));


    //Setup inputs
    int* d_iExpNo;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_iExpNo,          sizeof(int)));
    CUDA_SAFE_CALL( cudaMemcpy( d_iExpNo,       &ParamPerExp,       sizeof(int),
    cudaMemcpyHostToDevice));
```

```
    float* d_iSimLength;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_iSimLength,      sizeof(float)));
    CUDA_SAFE_CALL( cudaMemcpy( d_iSimLength,   &Simlength,         sizeof(float),        ↙
    cudaMemcpyHostToDevice));

    int* d_iCutOffTop;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_iCutOffTop,      sizeof(int)));
    CUDA_SAFE_CALL( cudaMemcpy( d_iCutOffTop,   &CutOffTop,         sizeof(int),          ↙
    cudaMemcpyHostToDevice));

    int* d_iParamNo;
    CUDA_SAFE_CALL( cudaMalloc( (void**)        &d_iParamNo,        sizeof(int)));
    CUDA_SAFE_CALL( cudaMemcpy( d_iParamNo,     &ParamNo,           sizeof(int),          ↙
    cudaMemcpyHostToDevice));

    // setup execution parameters
    dim3  grid( i_Block_No, 1, 1);
    dim3  threads( num_threads, 1, 1);


    // execute the kernel
    VasopressinNoSMKernel<<< grid, threads >>>( d_iParams,
                                        d_iSimLength,
                                        d_iParamNo,
                                        d_iCutOffTop,
                                        d_oISI,
                                        d_iExpNo,
                                        d_oPhasic,
                                        d_oPhasicMean,
                                        d_oPhasicStdDev,
                                        d_oPhasicBurstNoise,
                                        d_oSilenceMean,
                                        d_oSilenceStdDev,
                                        d_oResonance
                                        );

    // check if kernel execution generated and error
    CUT_CHECK_ERROR("Kernel execution failed");

    // copy result from device to host
    CUDA_SAFE_CALL( cudaMemcpy( ISIArray,           d_oISI,             sizeof(short)* ↙
    TotalSims*500,  cudaMemcpyDeviceToHost) );
    CUDA_SAFE_CALL( cudaMemcpy( PhasicArray,        d_oPhasic,          sizeof(short)* ↙
    TotalSims*500,  cudaMemcpyDeviceToHost) );
    CUDA_SAFE_CALL( cudaMemcpy( PhasicMeanArray,    d_oPhasicMean,      sizeof(float)* ↙
    TotalSims,      cudaMemcpyDeviceToHost) );
    CUDA_SAFE_CALL( cudaMemcpy( PhasicStdDevArray,  d_oPhasicStdDev,    sizeof(float)* ↙
    TotalSims,      cudaMemcpyDeviceToHost) );
    CUDA_SAFE_CALL( cudaMemcpy( PhasicBurstNoise,   d_oPhasicBurstNoise, sizeof(float)* ↙
    TotalSims,      cudaMemcpyDeviceToHost) );
    CUDA_SAFE_CALL( cudaMemcpy( SilenceMeanArray,   d_oSilenceMean,     sizeof(float)* ↙
    TotalSims,      cudaMemcpyDeviceToHost) );
    CUDA_SAFE_CALL( cudaMemcpy( SilenceVarArray,    d_oSilenceStdDev,   sizeof(float)* ↙
    TotalSims,      cudaMemcpyDeviceToHost) );
    CUDA_SAFE_CALL( cudaMemcpy( Resonance,          d_oResonance,       sizeof(float)* ↙
    TotalSims*16,       cudaMemcpyDeviceToHost) );


    // cleanup memory
    CUDA_SAFE_CALL(cudaFree(d_iParams));
    CUDA_SAFE_CALL(cudaFree(d_iSimLength));
    CUDA_SAFE_CALL(cudaFree(d_iCutOffTop));
    CUDA_SAFE_CALL(cudaFree(d_iExpNo));
    CUDA_SAFE_CALL(cudaFree(d_oISI));
    CUDA_SAFE_CALL(cudaFree(d_oPhasic));
```

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM.cu 6

```
    CUDA_SAFE_CALL(cudaFree(d_oPhasicMean));
    CUDA_SAFE_CALL(cudaFree(d_oPhasicStdDev));
    CUDA_SAFE_CALL(cudaFree(d_oPhasicBurstNoise));
    CUDA_SAFE_CALL(cudaFree(d_oResonance));
    CUDA_SAFE_CALL(cudaFreeArray(cu_iRandomNoiseSample));

}
```

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM_kernel.cu 1

```
/*
 * Copyright 1993-2007 NVIDIA Corporation.  All rights reserved.
 *
 * NOTICE TO USER:
 *
 * This source code is subject to NVIDIA ownership rights under U.S. and
 * international Copyright laws.  Users and possessors of this source code
 * are hereby granted a nonexclusive, royalty-free license to use this code
 * in individual and commercial software.
 *
 * NVIDIA MAKES NO REPRESENTATION ABOUT THE SUITABILITY OF THIS SOURCE
 * CODE FOR ANY PURPOSE.  IT IS PROVIDED "AS IS" WITHOUT EXPRESS OR
 * IMPLIED WARRANTY OF ANY KIND.  NVIDIA DISCLAIMS ALL WARRANTIES WITH
 * REGARD TO THIS SOURCE CODE, INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.
 * IN NO EVENT SHALL NVIDIA BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL,
 * OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
 * OF USE, DATA OR PROFITS,  WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE
 * OR OTHER TORTIOUS ACTION,  ARISING OUT OF OR IN CONNECTION WITH THE USE
 * OR PERFORMANCE OF THIS SOURCE CODE.
 *
 * U.S. Government End Users.   This source code is a "commercial item" as
 * that term is defined at  48 C.F.R. 2.101 (OCT 1995), consisting  of
 * "commercial computer  software"  and "commercial computer software
 * documentation" as such terms are  used in 48 C.F.R. 12.212 (SEPT 1995)
 * and is provided to the U.S. Government only as a commercial end item.
 * Consistent with 48 C.F.R.12.212 and 48 C.F.R. 227.7202-1 through
 * 227.7202-4 (JUNE 1995), all U.S. Government End Users acquire the
 * source code with only those rights set forth herein.
 *
 * Any use of this source code in individual and commercial software must
 * include, in the user documentation and internal comments to the code,
 * the above Disclaimer and U.S. Government End Users Notice.
 */

/* Template project which demonstrates the basics on how to setup a project
 * example application.
 * Device code.
 */

#ifndef _TEMPLATE_KERNEL_H_
#define _TEMPLATE_KERNEL_H_

#include <stdio.h>

texture<float, 1, cudaReadModeElementType> Noise;


////////////////////////////////////////////////////////////////////////////////
//! Simple test kernel for device functionality
//! @param g_idata  input data in global memory
//! @param g_odata  output data in global memory
////////////////////////////////////////////////////////////////////////////////
__global__ void
VasopressinNoSMKernel(  float*  d_iParams,
                        float*  d_iSimLength,
                        int*    d_iParamNo,
                        int*    d_iCutOffTop,
                        short*  d_oISI,
                        int*    d_iExpNo,
                        short*  d_oPhasic,
                        float*  d_oPhasicMean,
                        float*  d_oPhasicStdDev,
                        float*  d_oPhasicBurstNoise,
                        float*  d_oSilenceMean,
                        float*  d_oSilenceStdDev,
```

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM_kernel.cu 2

```
                    float*  d_oResonance
                    )
{
    ////////////////////////////////////////////////////////////////////////////////
    //////////////
    // access thread id + No thread stuff.  This is used for addressing
    const int SimLength = (int) (*d_iSimLength * 1000.0f);
    const int CutOffTop = (int) (*d_iCutOffTop );
    const unsigned int tid = threadIdx.x;
    const unsigned int num_threads = blockDim.x;
    const unsigned int bid = blockIdx.x;

    ////////////////////////////////////////////////////////////////////////////////
    //////////////
    //LCG stuff
    unsigned int RAND = 0x12345678;

    ////////////////////////////////////////////////////////////////////////////////
    //////////////
    //Copy Parameters that are in perpetual use to local memory
    float NoiseAmp      =       d_iParams[0 + (tid + bid*num_threads)*(*d_iExpNo)];
    float B             =       d_iParams[1 + (tid + bid*num_threads)*(*d_iExpNo)];
    float Theshold      =       d_iParams[2 + (tid + bid*num_threads)*(*d_iExpNo)];
    float S             =       d_iParams[3 + (tid + bid*num_threads)*(*d_iExpNo)];
    float MemBaseDecay  =       d_iParams[4 + (tid + bid*num_threads)*(*d_iExpNo)];
    float Inhib_Decay   =       d_iParams[5 + (tid + bid*num_threads)*(*d_iExpNo)];
    float Inhib_Step    =       d_iParams[6 + (tid + bid*num_threads)*(*d_iExpNo)];
    float HAP_Decay     =       d_iParams[7 + (tid + bid*num_threads)*(*d_iExpNo)];
    float HAP_Step      =       d_iParams[8 + (tid + bid*num_threads)*(*d_iExpNo)];
    float DAP_Decay     =       d_iParams[9 + (tid + bid*num_threads)*(*d_iExpNo)];
    float DAP_Step      =       d_iParams[10 + (tid + bid*num_threads)*(*d_iExpNo)];
    float AHP_Decay     =       d_iParams[11 + (tid + bid*num_threads)*(*d_iExpNo)];
    float AHP_Step      =       d_iParams[12 + (tid + bid*num_threads)*(*d_iExpNo)];
    float CustpCutOff   =       1;

    if(*d_iParamNo > 13)
        CustpCutOff     =       (float) (d_iParams[13 + (tid + bid*num_threads)*(*
    d_iExpNo)]) / 100.0f;

    float D = sqrt(-B*2);
    //Derived Parameters
    float Inhibition_Catastrophe_OffSet = (2.44948974278317809819728407470 59/12.
    72792206135785543921519851788 7)*D*D*D*CustpCutOff;

    //Sort Out Cusp Decay
    float TailDecay = 1.0f - (float) exp(-0.693147180559945309417232 1 / (S*1000.0));
    S = (float) (1-exp(-0.693147180559945309417232 1 / (S*1000.0)))/(D*D);

    ////////////////////////////////////////////////////////////////////////////////
    //////////////
    //Create the Variables (HAP, DAP, AHP, LastDAP, Drift, Membrane Potential, Inhibition,
    Balance Point)
    float V_Noise           = 0;
    float V_Inhib_A         = 0;
    float V_HAP             = 0;
    float V_DAP             = 0;
    float V_AHP             = 0;
    float V_MemPot          = -D;
    float V_MemPotWNoise    = 0;
    float SectionLimit      = D/2;
    bool  InBurst           = false;
    float Event             = 0;

    //Hidden Vars
    int LastEvent   = 0;
```

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM_kernel.cu 3

```
    //Rate Recording
    int   RateTime      = 0;
    short PhasicCounter = 0;

    int NoiseSum        = 1;
    short InBurstTime   = 0;

    //Period Mean and StdDev calculation
    float SumOfPeriodLength         = 1;
    float SumOfPeriodLengthSquared  = 1;
    float PeriodNumber              = 1;

    float SumOfSilenceLength        = 1;
    float SumOfSilenceLengthSquared = 1;
    float SilenceStart              = 1;

    //Resonance
    char RythmArray[16];
    for(int i = 0; i < 16; i++) RythmArray[i] = 0;

    __syncthreads();

    //Model
    for(int Itt = 0; Itt < SimLength; Itt ++)
    {
        //Decays
        //AHP
        V_Inhib_A       = V_Inhib_A*Inhib_Decay + Inhib_Step*Event;

        //HAP
        V_HAP           = V_HAP*HAP_Decay + HAP_Step*Event;

        //DAP
        V_DAP           = V_DAP*DAP_Decay + DAP_Step*Event;

        //AHP
        V_AHP           = V_AHP*AHP_Decay + AHP_Step*Event;

        //First do the noise
        ////////////////////////////////////////////////////////
        //Lets try a Linear congruential generator
        //V(j+1) = (A*Vj + C)mod(M)
        //A = 16807, C = 0, M = 2147483647
        //This is about twice the speed of the linear feedback shift register
        //Use the top bits as the bottom bits are pretty crap
        //const unsigned int A = 16807;
        //const unsigned int M = 0x7FFFFFFF;

        //Get a random number for an address in memory
        RAND = (22695477*RAND + 1) & 0xFFFFFFFF;

        //Decay the current noise
        V_Noise *= MemBaseDecay;

        //Add the new noise to the current noise
        V_Noise += tex1Dfetch(Noise, (RAND >>20)) * NoiseAmp;

        //These lines allows us to keep the noise separate from the Cusp Catastrophe
        //First we construct the influence from the firing patterns
        //This is set to the AHP if the HAP pulls it below this value.
        float V_Inf = V_DAP + V_AHP;
        if(V_Inf > 0)
        {
            if(V_Inf > -V_HAP)  V_Inf += V_HAP;
            else                V_Inf = 0.0;
        }
```

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM_kernel.cu 4

```
        V_MemPotWNoise = V_Noise + V_MemPot + V_Inf;

        ////////////////////////////////////////////////////////
        //Find out what the change in the membrane will be:
        //
        //If < -D or > D use base decay with offset
        //Else use cusp catastrophe
        //-(4x^3 + 2xb + a)*S (Catastrophe theory)

        if(V_MemPotWNoise > SectionLimit)
        {
            V_MemPot    += -(V_MemPotWNoise - SectionLimit)*TailDecay;
        }
        else if(V_MemPotWNoise < -SectionLimit)
        {
            V_MemPot    += -(V_MemPotWNoise + SectionLimit)*TailDecay;
        }
        else
        {
            V_MemPot    += -(4*V_MemPotWNoise*V_MemPotWNoise*V_MemPotWNoise + 2*
    V_MemPotWNoise*B)*S;
        }

        V_MemPot        += (-V_Inhib_A + Inhibition_Catastrophe_OffSet)*S;


        //When it fires
        if( (V_MemPot + V_HAP + V_DAP + V_AHP + V_Noise) >  Theshold)
        {
            Event = 1;
            //If the model fires record and timestamp the event
            int Diff = Itt - LastEvent;

            if((Diff > 5) && (Diff < 500/*CutOffTop*/))
            {
                /*
                short T1 = (short) (Diff/20);
                short Section = (short) ((sqrt(8.0*T1+1)-1)*0.5);
                short StartBin = Section*20;
                short StartTime = Section*(Section+1)*10;
                short Bin = (short) ((Diff - StartTime)/(1+Section)) + StartBin;
                */

                d_oISI[Diff /*Bin*/ + tid*500 + bid*num_threads*500]++;
            }
            LastEvent = Itt;

            //Increment the number of events this second
            RythmArray[0]++;
        }
        else Event = 0;

        //Rate Binning
        if(RateTime < Itt)
        {
            //Check to see if this is a new Burst
            int TotalCounts = RythmArray[0]+ RythmArray[1] + RythmArray[2] + RythmArray[3];
            if((TotalCounts > 6) && (!InBurst) )
            {
                PeriodNumber++;
                SumOfPeriodLength += PhasicCounter;
                SumOfPeriodLengthSquared += PhasicCounter*PhasicCounter;

                float temp = PhasicCounter - SilenceStart;
                SumOfSilenceLength += temp;
```

c:\Work\CUDA\projects\VasopressinNoSM\VasopressinNoSM_kernel.cu 5

```
            SumOfSilenceLengthSquared += temp*temp;
            PhasicCounter = 0;
            InBurst = true;
        }
        else    PhasicCounter++;

        //Measure the distribution of Periods
        if((TotalCounts < 2) && (InBurst))
        {
            SilenceStart = PhasicCounter;
            InBurst = false;
        }

        if(PhasicCounter < 500)
            d_oPhasic[PhasicCounter + tid*500 + bid*num_threads*500] += RythmArray[3];


        //Noise Characteristics
        //Find the highest and lowest
        short highest = 0;
        short lowest = 10000;
        for(int r = 0; r < 4; r++)
        {
            if(RythmArray[r] > highest)     highest = RythmArray[r];
            if((RythmArray[r] < lowest) && (RythmArray[r] > 0))     lowest  =
RythmArray[r];
        }

        if(InBurst)
        {
            //Find the difference and average them
            NoiseSum += highest - lowest;

            //increment time
            InBurstTime++;
        }

        for(int r = 4; r < 16; r++)
        {
            if(RythmArray[r] > highest)     highest = RythmArray[r];
            if((RythmArray[r] < lowest) && (RythmArray[r] > 0))     lowest  =
RythmArray[r];
        }


        //Now do the resonance stuff
        //Calculate cut-off
        float CutOffTop = (highest - lowest) * 0.75 + lowest;
        float Middle    = (highest - lowest) * 0.5 + lowest;

        //Find the top and bottom distributions
        bool firsttop = true;
        bool GoneDown = false;
        int countertop = 0;
        for(int r = 0; r < 16; r++)
        {
            if(RythmArray[r] > CutOffTop)
            {
                if(!firsttop && GoneDown)
                {
                    d_oResonance[countertop + tid*16 + bid*num_threads*16]++;
                    countertop = 0;
                }
                else if(firsttop)
                {
                    countertop = 0;
```

```
                    }
                    else
                    {
                        countertop++;
                    }
                    GoneDown = false;
                    firsttop = false;
                }
                else
                {
                    if(RythmArray[r] < Middle) GoneDown = true;
                    countertop++;
                }

            }

            //Shift the array and set element 1 to 0
            for(int R = 15; R > 0; R--) RythmArray[R] = RythmArray[R-1];
            RythmArray[0] = 0;
            RateTime    += 1000;
        }
    }


    //__syncthreads();

    //Calculate the final Mean and StdDev values
    //Mean
    float temp = SumOfPeriodLength/PeriodNumber;
    float temp2 = SumOfSilenceLength/PeriodNumber;
    d_oPhasicMean[tid + bid*num_threads] = temp;
    d_oSilenceMean[tid + bid*num_threads] = temp2;


    //StdDev
    d_oPhasicStdDev[tid + bid*num_threads] = sqrt((SumOfPeriodLengthSquared/PeriodNumber) -↙
     temp*temp );
    d_oSilenceStdDev[tid + bid*num_threads] = sqrt((SumOfSilenceLengthSquared/PeriodNumber)↙
     - temp2*temp2 );

    //Noise Mean
    d_oPhasicBurstNoise[tid + bid*num_threads] = (float)NoiseSum/(float)InBurstTime;

}

#endif // #ifndef _TEMPLATE_KERNEL_H_
```

References

Abbott, L. F. (1999), "Lapicque's introduction of the integrate-and-fire model neuron (1907)." Brain Research Bulletin **50**(5): 303-304.

Ahn, C. W. (2004), "New Evolutionary Algorithm for Multiobjective Optimization." Gwangju Institute of Science and Technology (GIST).

Alander, J. T. (1999), "Population size, building blocks, fitness landscape and genetic algorithm search efficiency." Practical Handbook of GENETIC ALGORITHMS: Complex Coding Systems **3**: Chapter 13.

Alander, J. T. (1999), "Population size, building blocks, fitness landscape and genetic algorithm search efficiency." Practical Handbook of GENETIC ALGORITHMS: Complex Coding Systems **3**: Chapter 13.

Alba, E. & Tomassini, M. (2002), “Parallelism and Evolutionary Algorithms” IEEE Transactions on Evolutionary Computation **6**(5): 443-461.

AMD (2008), "ATI Developers website." http://www.ati.amd.com/developer/.

AMD (December 2008), "AMD Stream Computing: Users Guide."

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M. & Werthimer, D. (2002), "SETI@home: an experiment in public-resource computing." Communications of the ACM **45**(11): 56-61.

Andrew, R. D. & Dudek, F. E. (1983), "Burst discharge in mammalian neuroendocrine cells involves an intrinsic regenerative mechanism." Science **221**(4614): 1050-1052.

Andrew, R. D. & Dudek, F. E. (1984), "Intrinsic inhibition in magnocellular neuroendocrine cells of rat hypothalamus." The Journal of Physiology(353): 171–185.

Arnauld, E., Dufy, B. & Vincent, J. D. (1975), "Hypothalamic supraoptic neurones: rates and patterns of action potential firing during water deprivation in the unanaesthetized monkey." Brain Research **100**(2): 2315–25.

ATI (2008), "ATI HD 4870 data sheet."

Av-Ron, E., Byrne, M. J., Byrne, J. H. & Baxter, D. A. (2007), "SNNAP: A tool for Teaching Neuroscience." Brains, Minds and Media **3**: bmm1408.

Av-Ron, E., Cai, Y., Byrne, J. H. & Baxter, D. A. (2005), "Parallel SNNAP: from object-oriented design to multithreading." 22nd Annual Houston Conference on Biomedical Engineering Research.

Back, T. (1998), "On the Behavior of Evolutionary Algorithms in Dynamic Environments." The 1998 IEE International Conference on Computational Intelligence: 446-451.

Back, T., Hammel, U. & Schwefel, H. P. (1997), "Evolutionary computation: comments on the history and current state." IEEE Transactions on Evolutionary Computation **1**(1): 3-17.

Bamford, S. A., Murrey, A. F. & Willshaw, D. J. (2008), "Synaptic Rewiring for Topographic Map Formation." Internation Conference on Artificial Neural Networks (ICANN): 218-227.

Barker, K. J., Davis, K., Hoisie, A., Kerbyson, D. J., Lang, M., Pakin, S. & Sancho, J. C. (2008), "Entering the Petaflop Era: The Architecture and Performance of Roadrunner." SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing: 1-11, ISBN: 978-1-4244-2835-9.

Benda, J. & Herz, A. V. M. (2003), "A universal model for spike-frequency adaptation."

Neural Computation **15**(11): 2523-2564.

Bezanilla, F. & Armstrong, C. M. (1977), "Inactivation of the sodium channel. I. Sodium current experiments " The Journal of General Pysiology **70**: 549-566.

Blythe, D. (2006), "The Direct3D 10 System." International converence on Computer Graphics and Interactive Techniques **ACM SIGGRAPH**(ISBN: 1-59593-364-6): 724-734.

Bofill-i-Petit, A. & Murray, A. F. (2004), "Synchrony detection and amplification by silicon neurons with STDP synapses." IEEE transactions on Neural Networks **15**(5): 1296-1304.

Bosman, P. A. N. & Thierens, D. (2005), "The Naive MIDEA: A Baseline Multi-objective EA." Evolutionary Multi-Criterion Optimization Conference (EMO 2005)(Alternative Methods): 428-442.

Bourque, C. W. & Renaud, L. P. (1991), "Membrane properties of rat magnocellular neuroendocrine cells in vivo." Brain Research **540**(1-2): 349-352.

Bouteiller, J.-M. C. (2009), "EONS v2.0." http://www.synaptic-modeling.com/eons_v2.html: Accessed January 2009.

Bouteiller, J.-M. C., Yumei, Q., Ziane, M. B., Baudry, M. & Berger, T. W. (2006), "EONS: An Online Synaptic Modeling Platform." Engineering in Medicine and Biology Scociety: 4155-4158.

Bower, J. M. & Beeman, D. (1998), "The book of GENESIS: Exploring realistic neural models with the General Neural Simulation System." Springer **2nd Edition**: ISBN-10: 0387949380.

Bower, J. M. & Beeman, D. (2007), "GENESIS (Simulation Environment)." Scholarpedia **2**(3): 1383 http://www.scholarpedia.org/article/GENESIS_(simulation_environment).

Bower, J. M., Beeman, D. & Hucka, M. (2003), "The GENESIS Simulation System." The Handbook of Brain Theory and Neural Networks **Second Edition**: 475-478.

Bretschneider, F. & Weille, J. (2006), "Introduction to Electrophysiological Methods and Instrumentation." Elsevier London **1st Edition**: ISBN-10: 0123705886.

Brimble, M. J. & Dyball, R. E. (1977), "Characterization of the responses of oxytocin- and vasopressin- secreting neurones in the supraoptic nucleus to osmotic stimulation." Journal of Physiology **271**(1): 253–71.

Brown, C. H. & Bourque, C. W. (2004), "Autocrine feedback inhibition of plateau potentials terminates phasic bursts in magnocellular neurosecretory cells of the rat supraoptic nucleus." Journal of Physiology **557**(3): 949-960.

Brown, C. H. & Bourque, C. W. (2006), "Mechanisms of rhythmogenesis: insights from hypothalamic vasopressin neurons." Tends in Neuroscience **29**(2): 108-115.

Brown, C. H., Bull, P. M. & Bourque, C. W. (2004), "Phasic bursts in rat magnocellular neurosecretory cells are not intrinsically regenerative in vivo." European Journal of Neuroscience **19**(11): 2977-2983.

Brown, T. H., Fricke, R. A. & Perkel, D. H. (1981), "Passive electrical constants in three classes of hippocampal neurons." Journal of Neurophysiology **46**(4): 812-827.

Burjorjee, K. M. (2008), "The fundamental problem with the building block hypothesis" arXiv:0810.3356v1 [cs.NE]

Burkitt, A. N. (2006), "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input." Biological Cybernetics **95**(1): 1-19.

Cambridge Electronic Design, C. E. D. (2008), "Spike2." **http://www.ced.co.uk/**: Accessed December 2008.

Carlisle, A. & Dozier, G. (2001), "An off-the-shelf PSO." Proceedings of the workshop on particle swarm optimization.

Carnevale, N. T. & Hines, M. L. (2006), "The NEURON Book." Cambridge University Press(ISBN-10: 0521843219).

Casanova, H., Berman, F., Bartol, T., Gokcay, E., Sejnowski, T., Birnbaum, A., Dongarra, J., Miller, M., Ellisman, M., Faerman, M., Obertelli, G., Wolski, R., Pomerantz, S. & Stiles, J. R. (2004), "The Virtual Instrument: Support for Grid-Enabled MCell Simulations." International Journal of High Performance Computing **18**: 3-17.

Chuang, C. & Chen, Y. (2008), "On the Effectiveness of Distributions Estimated by Probabilistic Model Building." GECCO 08(ISBN:978-1-60558-130-9): 391-398.

Clerc, M., Kennedy, J. (2002), "The particle swarm – explosion, stability, and convergence in a multidimensional complex space." IEEE Transactions on Evolutionary Computation **6**(1): 58-73.

Costa, M. & Minisci, E. (2003), " MOPED: A Multi-objective Parzen-Based Estimation of Distribution Algorithm for Continuous Problems."

Costa, M., Minisci, E. & Pasero, E. (2003), "An Hybrid Neural/Genetic Approach to Continuous Multi-objective Optimization Problems." Lecture Notes in Computer Science - Neural Nets: 61-69.

Davies, A., Blakely, A. & Kidd, C. (2001), "Human Physiology." Churchill Livingstone **1st Edition**: ISBN-10: 0443045593.

Davis, P. G. & Barfield, R. J. (1979), "Activation of Feminine Sexual Beaviour in Castrated Male Rats by Intrahypothalamic Implants of Estradiol Benzoate." Neuroendocrinology **28**(4): 228-233.

Dayan, P. & Abbort, L. F. (2001), "Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems." MIT Press: ISBN-10: 0262541858.

de Kamps, M., Baier, V., Drever, J., Dietz, M., Mosenlechner, L. & van der Velde, F. (2008), "The state of MIIND." Neural Networks **21**(8): 1164-1181.

De Schutter, E. (2008), "Why are Computational Neuroscience and Systems Bbiology So Separate?" Public Library of Science (PLoS) Computational Biology **4**(5): e1000078.

Deb, K. (2001), "Multi-Objective Optimization using Evolutionary Algorithms." John Wiley & Sons(ISBN 0-471-87339).

Deb, K., Pratap, A. Agarwal, S., Meyarivan, T. (2002), "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." IEEE Transactions on Evolutionary Computation **6**(2): 182-196.

Deco, G., Jirsa, V. K., Robinson, P. A., Breakspear, M. & Friston, K. (2008), "The Dynamic Brain: From SPiking Neurons to Neural Masses and Cortical Fields." Public Library of Science (PLoS) Computational Biology **4**(8): e1000092.

Delorme, A. & Thorpe, S. (2003), "SpikeNET: An Event-driven Simulation Package for Modeling Large Networks of Spiking Neurons." Network: Computation in Neural Systems **14**: 613-627.

Delorme, A., Gautrais, J., VanRullen, R. & Thorpe, S. J. (1999), "SpikeNET: A simulator for modeling large networks of integrate and fire neurons." Neurocomputing **26-27**: 989-996.

Diosan, L. & Oltean, M. (2007), "Observing the swarm behaviour during its evolutionary design." GECCO 07(ISBN:978-1-59593-698-1).

Dix, A., Finlay, J., Abowd, G. D. & Beale, R. (2003), "Human Computer Interaction." Prentice Hall **Third Editon**(ISBN-10: 0130461091).

Dominques, P., Marques, P. & Silva, L. (2005), "Resource usage of Windows computer Laboratories." International Converence Workshops on Parallel Processing(ISBN: 0-7695-2381-1): 469-476.

Durie, R. (2007), "A Population Model of Vasopressin Secretion." Institute for Adaptive and Neural Computation, School of Informatics, University of Edinburgh.

Eberhart, R. C. & Kennedy, J. (1995), "A new optimizer using particle swarm theory." In proceedings of the sixth international symposium on micro machine and human science: 39-43.

Eiben, A. E. & Smith, J. E. (2008), "Introduction to Evolutionary Computing." Springer: ISBN-10: 3540401841.

Eiben, A. E., Hinterding, R., Michalewicz, Z. (1999), "Parameter control in evolutionary algorithms." IEEE transactions on Evolutionary Computation **3**(2): 124-141.

Eng, E. (1996), "Qt GUI Toolkit: Porting graphics to multiple platforms using a GUI toolkit." Specialised Systems Consultants, Inc **31**(2).

Feng, J. (2003), "Computational Neuroscience: A Comprehensive Approach." Chapman and Hall **1st Edition**: ISBN-10: 1584883623.

Finkel, A. S. & Redman, S. J. (1983), "A shielded microelectrode suitable for single-electrode voltage clamping of neurones in the CNS." Journal of Neuroscience Methods **9**(1): 23-239.

FitzHugh, R. (1961), "Impulses and physiological states in theoretical models of nerve membrane." Biophysical Journal **1**: 445-466.

Fogel, D. B. (2000), "What is evolutionary computation." IEEE Spectrum **37**(2): 26, 28-32.

Fonseca, C. M., Knowles, J., Thiele, L. & Zitzler, E. (2005), "A tutorial on the performance assessment of stochastic multiobjective optimizers." Evolutionary Multi-Criterion Optimization Conference (EMO 2005).

Fonsecay, C. M. & Flemingz, P. J. (1993), "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization." Genetic Algorithms: Proceedings of the Fifth International Conference: 416-423.

French, L. & Pavlidis, P. (2007), "Informatics in Neuroscience." Briefings in Bioinformatics: doi:10.1093/bib/bbm047.

Geit, W. V., Schutter, E. & Achard, P. (2008), "Automated neuron model optimization techniques: a review." Biological Cybernetics **99**(4-5): 241-251.

Gerstner, W. & Kistler, W. K. (2002), "Spiking Neuron Models - single neurons, populations, pasticity." Cambridge University Press: ISBN 0521890799.

Gerstner, W. & van Hemmen, J. L. (1996), "What matters in neuronal locking." Neural Computation **8**: 1653-1676.

Gerstner, W. (2000), "Population Dynamics of Spiking Neurons: Fast Transients, Asynchronous States, and Locking." Neural Computation **12**(1): 43-89.

Gerstner, W. (2008), "Spike-response model." Scholarpedia **3**(12): 1343 revision #55029.

Gerstner, W., Ritz, J. & van Hemmen, J. L. (1993), "Why Spikes? Hebbian Learning ad Retrieval of time-resolved excitation patterns." Biological Cybernetics **69**: 503-515.

Gewaltig, M. O. & Diesmann, M. (2007), "NEST (Neural Simulation Tool)." Scholarpedia **2**(4): 1430.

Goddard, N. H., Hucka, M., Howell, F., Cornelis, H., Shankar, K. & Beeman, D. (2001), "Towards NeuroML: Model Description Methods for Collsaborative Modelling in Neuroscience." Philosophical Transactions of The Royal Society of Biological Sciences **356**(1412): 1209-1228.

Goldberg, D. (1985), "Optimal initial population size for binary-coded genetic algorithms." The Clearinghouse for Genetic Algorithms: Department of Engineering Mechanics - University of Alabama **TCGA Report No. 85001**.

Goldberg, D. (2002), "Genetic Algorithms: The Design of Innovation." Springer **First Edition**(ISBN:-10: 1402070985).

Goldberg, D., Sastry, K. (2009), "Genetic Algorithms: The Design of Innovation." Springer **Second Edition**(ISBN:-10: 0387353747).

Golshan, K. (2007), "Physical Design Essentials." Springer **First Edition**(ISBN-10: 0387366423).

Graham, L. J. (2004), "The Surf-Hippo Neuron Simulation System, v3.5a." http://www.neurophys.biomedicale.univ-paris5.fr/□graham/surf-hippo.html: Accessed January 2009.

Greffrath, W., Martin, E., Reuss, S. & Boehmer, G. (1998), "Components of after-hyperpolarization in magnocellular neurones of the rat supraoptic nucleus in vitro, J. Physiol. ." Journal of Physiology **513**(2): 493-506.

Grosan, C. & Abraham, A. (2007), "Hybrid Evolutionary Algorithms: Methodologies, Architectures, and Reviews." Studies in Computational Intelligence: Hybrid Evolutionary Algorithms **75**: 1-17.

Gurkiewicz, M. & Korngreen, A. (2007), "A Numerical Approach to Ion Channel Modelling using Whole-Cell Voltage-Clamp Recordings and a Genetic Algorithm." PLoS Computational biology **3**(8): e169.

Gurney, A. M. (2000), "Electrophysiological recording methods used in vascular biology." Journal of Pharmacological and Toxicological Methods **44**(2): 409-420.

Hamill, O. P., Marty, A., Neher, E., Sakmann, B. & Sigworth, F. J. (1981), "Improved patch-clamp techniques for high-resolution current recording from cells and cell-free membrane patches." Pflugers Archiv European Journal of Physiology **391**(2): 85-100.

Hansen, M. H. & Yu, B. (2001), "Model selection and the principle of minimum description length." Journal of the American Statistical Association **96**: 746–774.

Herz, A. V. M., Gollisch, T., Machens, C. K. & Jaeger, D. (2006), "Modeling Single-Neuron Dynamics and Computations: A Balance of Detail and Abstraction." Science **314**(5796): 80-85.

Hines, M. I. & Carnevale, N. T. (2008), "Translating network models to parallel hardware in NEURON." Journal of Neuroscience Methods **169**(2): 425-455.

Hines, M. I. (1984), "Efficient computation of branched nerve equations." International Journal of Biomedical Computing **15**(1): 69-76.

Hines, M. I. (1989), "A program for simulation of nerve equations with branching geometries." International Journal of Biomedical Computing **24**(1): 55-68.

Hines, M. L. & Carnevale, N. T. (1997), "The NEURON simulation environment." Neural Computation **9**(6): 1179-1209.

Hines, M. L. & Carnevale, N. T. (2005), "Recent Developments in NEURON." Brains, Minds and Media **1**: 1861-1680.

Hines, M. L., Morse, T. M. & Carnevale, N. T. (2008), "Model Structure Analysis in NEURON: towards interoperability among neural simulators." Methods in Molecular Biology **401**: ISBN: 978-1-58829-720-4.

Hodgkin, A. & Huxley, A. (1952), "A quantitive description of membrane current and its application to conduction and excitation in nerve." Journal of Physioloy **117**(4): 500-544.

Holland, J. H. (1975), "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence." The MIT Press(ISBN-10:0262581116).

Holmes, W. R., Ambros-Ingerson, J. & Grover, L. M. (2006), "Fitting experimental data to models that use morphological data from public databases." Journal of Computational Neuroscience **20**(3): 359-365.

Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P. & Kummer, U. (2006), "COPASI, a COmplex PAthway SImulator." Bioinformatics **22**(24): 3067-3074.

Horn, R. & Marty, A. (1988), "Muscarinic activation of ionic currents measured by a new whole-cell recording method." Journal of General Physiology **92**(2): 145-159.

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C. & Kitano, H. (2003), "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models." Bioinformatics **19**(4): 524-531.

Hucka, M., Shankar, K., Beeman, D. & Bower, J. M. (2002), "The Modeler's Workspace: Making model-based studies of the nervous system more accessible." Chapter 5 in Computational Neuroanatomy: Principles and Methods **First Edition**(ISBN-10: 158829000X).

Izhikevich, E. M. (2003), "Simple model of spiking neurons." IEEE transactions on Neural Networks **14**(6): 1569-1572.

Izhikevich, E. M. (2006), "Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting." The MIT press(ISBN-10: 0262090430).

Jacobs, A. M. & Grainger, J. (1994), "Models of visual word recognition–sampling the state of the art." Journal of Experimental Psychology: Human Perception and Performance **29**(6): 1311-1334.

Janson, S. & Middendorf, M. (2005), "A Hierarchical particale swarm optimizer and its adaptive variant." IEEE Transations on System Man and Cybernetics, Part B **35**(6): 1272-1282.

Janssen, P. & Shadlen, M. N. (2005), "A representation of the hazard rate of elapsed time in macaque area LIP." Nature Neuroscience **8**: 234 - 241

Jaszkiewicz, A. (2002), "Genetic local search for multi-objective combinatorial optimization." European Journal of Operational Research **137**(1): 50-71.

Jolivet, R., Lewis, T. J. & Gerstner, W. (2003), "The Spike Response Model: A Framework to Predict Neuronal Spike Trains." Artificial Neural Networks and Neural Information Processing - ICANN/ICONIP 2003 **2714**: 173.

Jolivet, R., Rauch, A., Luscher, H. R. & Gerstner, W. (2006), "Predicting spike timing of neocortical pyramidal neurons by simple threshold models." Journal of Computational NEuroscience **21**(1): 35-49.

Jong, K. A. (2002), "Evolutionary Computation: A Unified Approach." MIT Press(ISBN-10: 0262041944).

Kennedy, J. & Eberhart, R. (1995), "Particle Swarm Optimisation." Proceedings of the IEEE international Conference on Neural Networks **4**: 1942-1948.

Khare, V., Yao, X. & Deb, K. (2003), "Performance Scaling of Multi-objective Evolutionary Algorithms." Evolutionary Multi-Criterion Optimization. Second International Conference, Lecture Notes in Computer Science **2532**: 376-390.

Kicingera, R., Arciszewskia, T. & Jong, K. D. (2005), "Evolutionary computation and structural design: A survey of the state-of-the-art." Computers & Structures **83**(23-24): 1943-1978.

Kim, M., Hiroyasu, T., Miki, M., Watanabe, S. (2004), "SPEA2+: Improving the Performance of the Strength Pareto Evolutionary Algorithm 2." Parallel Problem Solving from Nature - PPSN VIII **3242**: 742-751.

King, B. M. (2006), "The rise, fall, and resurrection of the ventromdeial hypothalamus in the regulation of feeding behaviour and body weight." Physiology & Behaviour **87**(2): 221-244.

Kirkpatrick, K. & Bourque, C. W. (1996), "Activity dependence and functional role of the apamin-sensitive K+ current in rat supraoptic neurones in vitro,." Journal of Physiology **494**(2): 389-398.

Kistler, W. M., Gerstner, W. & van Hemmen, J. L. (1997), "Reduction of Hodgkin-Huxley equations to a threshold model." Neural Computation **9**: 1069-1100.

Koch, C. & I., S. (1998), "Methods in neuronal modeling: from ions to networks." MIT Press **2nd Edition**: ISBN-10: 0262112310.

Komendantov, A. O., Trayanova, N. A. & Tasker, J. G. (2007), "Somato-dendritic mechanisms underlying the electrophysiological properties of hypothalamic magnocellular neuroendocrine cells: A multicompartmental model study." Journal of Computational Neuroscience **23**: 143–168.

Kornreich, B. G. (2007), "The patch clamp technique: Principles and technical considerations." Journal of Veterinary Cardiology **9**(1): 25-37.

Kumar, S. P. & Feidler, J. C. (2003), "BioSPICE: A Computational Infrastructure for Integrative Biology." OMICS: A Journal of Integrative Biology **7**(3): 225-225.

Kuon, I. & Rose, J. (2006), "Measuring the Gap Between FPGAs and ASICs." Internation Syposium on Field Programmable Gate Arraya(ISBN: 1-59593-292-5): 21-30.

Larranaga, P., Etxeberria, R., Lorano, J. A. & Pena, J. M. (2000), "Combinatorial Optimization by Learning and Simulation of Bayesian Networks." Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence: 343-352.

Larranga, P. & Lozano, J. A. (2002), "Estimation of Distribution Algorithms: A new tool for evolutionary computation." Kluwer Academic Publishers, Boston **First Edition**(ISBN-10: 0792374665).

Lazaridis, E., Drakakis, E. M. & Barahona, B. (2007), "Full analogue electronic realisation of the Hodgkin-Huxley neuronal dynamics in weak-inversion CMOS." Engineering in Medicine and Biology Scociety: 1200-1203.

Lewicki, M. S. (1998), "A review of methods for spike sorting: the detection and classification of neural action potentials." Network: Computation in Neural Systems **9**(4): R53-R78.

Liang, J. J. & Suganthan, P. N. (2005), "Dynamic multiswarm particle swarm optimizer (DMS-PSO)." In Proceedings of the IEEE swarm intelligence symposium (SIS)(ISBN: 0-7803-8916-6): 124-129.

Lloyd, C. M., Halstead, M. D. B. & Nielsen, P. F. (2004), "CellML: its future, present and past." Progressi Biophysics and Molecular Biology **85**(2-3): 433-450.

Luce, R. D. (1986), "Response Times: Their Role in Inferring Elementary Mental Organization." Oxford Univ. Press, New York: ISBN-10: 0195036425.

Lytton, W. W. & Stewart, M. (2005), "A rule-based firing model for neural networks." International Journal of Bioelectromagnetism **7**(1): 47-50.

Martí, L., García, J., Berlanga, A. & Molina, J. M. (2008), "Introducing MONEDA: Scalable Multiobjective Optimization with a NEural Estimation of Distribution Algorithm." GECCO 08(ISBN:978-1-60558-130-9).

McClellan, K. M., Parker, K. L. & Tobet, S. (2006), "Development of the ventromedial nucleus of the hypothalamus." Frontiers in Neuroendocrinology **27**(2): 193-209.

Mendes, P. (1993), "GEPASI: a software package for modelling the dynamics, steady states and control of biochemical and other systems." Bioinformatics **9**(5): 563-571.

Mendes, R. (2004), "Population topologies and their influence in particle swarm performance." PhD thesis Departmento de Informatica, Escola de genharia, Universidade do Minho.

Mendes, R., Kennedy, J., Neves, J. (2003), "Watch thy neighbour or how the swarm can learn from its environment." In proceedings of the IEEE swarm intelligence symposium (SIS)(ISBN: 0-7803-7914-4): 88-94.

Meunier, C. & Segev, I. (2002), "Playing the Devil's Advocate: is the Hodgkin-Huxley model useful?" Trends in Neuroscience **25**(11): 558-563.

Migliore, M., Cannia, C., Lytton, W. W., Markram, H. & Hines, M. I. (2006), "Parallel netowrk simulations with NEURON." Journal of Computational Neuroscience **21**(2): 119-129.

Moore, G. E. (1965), "Cramming more components onto integrated circuits." Electronics Magazine **38**(8).

Morrison, A., Mehring, C., Geisel, T., Aertsen, A. & Diesmann, M. (2005), "Advancing the boundaries of high connectivity network simulation with distributed computing." Neural Computation **17**(8): 1776–1801.

Nagumo, J., Arimoto, S. & Yoshizawa, S. (1962), "An Active Pulse Transmission Line Simulating Nerve Axon." Proceedings of the Institute of Radio Engineers **50**(10): 2061-2070.

NVIDIA (2008a), "NVIDIA CUDA Compute Unified Device Architecture Programming Guide." Version 2.

NVIDIA (2008b), "NVIDIA Developers website." http://developer.nvidia.com/.

NVIDIA (November 2006), "Technical Brief: NVIDIA GeForce 8800 GPU Architecture Overview." NVIDIA Corporation(TB-02787-001_v01).

Opal Kelly (2009), "Opal Kelly website." http://www.opalkelly.com

O'Reilly, R. C. & Munakata, Y. (2000), "Computatonal Explorations in Cognitive Neuroscience: Understanding the Mind by Simulationg the Brain." MIT Press: ISBN-10: 0262650541.

Paninski, L., Pillow, J. W. & Simoncelli, E. P. (2004), "Maximum Likelihood Estimation of a Stochasitic Integrate-and-Fire Neural Encoding Model." Neural Computation(16): 2533-2561.

Parabon (2008), "Parabon Computation website." http://www.parabon.com.

Particle Swarm Central - Programs (accessed 2009), "http://www.particleswarm.info/Programs.html."

Particle Swarm Central (accessed 2009), "Particle Swarm Central Web Site." http://www.particleswarm.info.

Patterson, R. D., " J Acoust Soc Japan(E) 21 (4), 183-190 (pdf) (2000), "Auditory images: How complex sounds are represented in the auditory system." Journal - Acoustical Society of Japan (English Edition) **21**(4): 183-190.

Patterson, R. D., Allerhand, M. H. & Giguere, C. (1995), "Time-domain modelling of peripheral auditory processing: A modular architecture and a software platform." Journal of the Acoustical Society of America **98**: 1890-1894.

Patterson, R. D., Robinson, K., Holdsworth, J., McKeown, D., Zhang, C. & M., A. (1992), "Complex sounds and auditory images." Proceedings of the 9h International Symposium on Hearing: Auditory physiology and perception 429-446.

Peterson, B. E., Healy, M. D., Nadkarni, P. M., Miller, P. L. & Shepherd, G. M. (1996), "ModelDB: an environment for running and storing computational models and their results applied to neuroscience." Journal of the American Medical Informatics Association **33**(6): 389-398.

Pfaff, D. & Keiner, M. (1973), "Atlas of estradiol-concentrating cells in the central nervous system of the female rat." The Journal of Computational Neuroscience **151**(2): 121-158.

Pitt, M. A. & Myung, I. J. (2002), "When a good fit can be bad." Trends in Cognitive Sciences **6**(10): 421-425.

Pitt, M. A., Myung, I. J. & Zhang, S. (2002), "Toward a method of selecting among computational models of cognition." Psychological Review **109**(3): 472-91.

Poli, R. (2007), "Analysis of the Publications on the Applications of Particle Swarm Optimisation." Journal of Artificial Evolution and Applications.

Poli, R., Kennedy, J. & Blackwell, T. (2007), "Particle Swarm Optimisation: An Overview." Swarm Intelligence **1**(1): 33-57.

Poulain, D. A., Brown, D. & Wakerley, J. B. (1988), "Statistical analysis of patterns of electrical activity in vasopressin and oxytocin-secreting neurones." Pulsatility in neuroendocrine systems, CRC Press: 119–154.

Powell, W. B. (2007), "Approximate Dynamic Programming: Solving the Curses of Dimensionality." WilwyBlackwell **Chapter 1: The challenges of Dynamic Programming**(ISBN-10 0470171553): 1-16.

Powell, W. B. (2007), "Approximate Dynamic Programming: The challenges of Dynamic Programming." WilwyBlackwell(ISBN-10 0470171553): 1-16.

Prinz, A. A. (2008), "Computational Exploration of Neuron and Neural Network Models in Neurobiology." Methods in Molecular Biology **401**: ISBN: 978-1-58829-720-4

Prinz, A. A., Bucher, D. & Marder, E. (2004), "Similar network activity from disparate circuit parameters." Nature Neuroscience **7**(12): 1345-1352.

Rae, J., Cooper, K., Gates, P. & Watsky, M. (1991), "Low access resistance perforated patch recordings using amphotericin B." Journal of Neuroscience Methods **37**(1): 15-26.

Roberts, S. & Pashler, H. (2000), "How persuasive is a good fit? A comment on theory testing." Psychological Review **107**: 358-367.

Robertson, G. L. (2001), "Antidiuretic hormone. Normal and disordered function." Endocrinology & Metabolism Clinics of North America **30**(3): 671–94.

Rodriguez-Andina, J. J., Moure, M. J. & Valdes, M. D. (2007), "Features, Design Tools, and Application Domains of FPGAs." IEEE Transactions on Industrial Electronics **54**(4): 1810-1823.

Roper, P., Callaway, J. & Armstrong, W. (2004), "Burst Initiation and Termination in Phasic Vasopressin Cells of the Rat Supraoptic Nucleus: A Combined

Mathematical, Electrical, and Calcium Fluorescence Study." Journal of Neuroscience **24**(20): 4818-4831.

Roper, P., Callaway, J., Shevchenko, T., Teruyama, R. & Armstrong, W. (2004), "AHP's, HAP's and DAP's: How Potassium Currents Regulate the Excitability of Rat Supraoptic Neurones." Journal of Computational Neuroscience **15**(3): 367-389.

Sabatier, N. & Leng, G. (2008), "Spontaneous discharge characteristic of neurons in the ventromedial nucleus of the rat hypothalamus in vivo." European Journal of Neuroscience **28**(4): 693-706.

Sabatier, N., Brown, C. H., Ludwig, M. & Leng, G. (2004), "Phasic spike patterning in rat Supraoptic neurones in vivo and in vitro." The Journal of Physiology **558**(1): 161-180.

Sackmann, B. & Neher, E. (1976), "Single-channel currents recorded from membrane of denervated frog muscle fibres." Nature **260**(5554): 799-802.

Saighi, S., Bornat, Y., Tomas, J. & Renaud, S. (2006), "Neuromimetic ICs and System for Parameters Extraction in Biological Neuron Models." Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS): 4207-4210.

Schaller, R. R. (1997), "Moore's Law: past, present and future." IEEE Spectrum **34**(6): 52-59.

Schutter, E. & Cannon, R. C. (2000), "Computational Neuroscience: Realistic Modeling for Experimentalists (Methods & new frontiers in neuoscience)." CRC Press **First Edition**(ISBN-10 0849320682).

Schwefel, H. P. (2000), "Advantages (and disadvantages) of evolutionary computation over other approaches." Evolutionary Computation **1**: 20-22.

Segev, I., Fleshman, J. W. & Burke, R. E. (1989), "Methods in Neuronal Modeling: Compartmental model oc complex neurons." MIT Press: 63096.

Shackleford, B., Snider, G., Carter, R. J., Okushi, E., Yasuda, M., Seo, K. & Yasuura, H. (2001), " A High-Performance, Pipelined, FPGA-Based Genetic Algorithm Machine." Genetic Programming and Evolvable Machines **2**(1): 33-60.

Shi, Y. & Eberhart, R. C. (1999), "Empirical Study of Particle Swarm Optimization." Proceedings of the 1999 Congress on Evolutionary Computation **3**.

Silver, R. A., Boahen, K., Grillner, S., Kopell, N. & Olsen, K. L. (2007), "Neurotech for NEuroscience: Unifying Concepts, Organizing Principles, and Emergin Tools." The Journal of Neuroscience **27**(44): 11807-11819.

Skinner, F. K. (2006), "Conductance-based models." Scholarpedia **1**(11): 1408 : revision #36853.

Song, Z., Levin, B. E., McArdle, J. J., Bakhos, N. & Routh, V. H. (2001), "Convergence of Pre- and Postsynaptic influences on glucosensing neurons in the ventromedial hypothalamic nucleus." Diabetes **50**(12): 2673-2681.

Steuber, V., Mittmann, W., Hoebeek, F. E., Silver, R. A., De Zeeuw, C. I., Hausser, M. & De Schutter, E. (2007), "Cerebellar LTD and pattern recognition by Purkinje cells." Neuron **54**(1): 121-136.

Stiles, J. R., Van Helden, D., Bartol, T. M., Salpeter, E. E. & Salpeter, M. M. (1996), "Miniature endplate current rise times <100 μs from improved dual recordings can be modeled with passive acetylcholine diffusion from a synaptic vesicle." Proceedings of the National Academy of Sciences of the United States of America **93**(12): 5747-5752.

Suganthan, P. N. (1999), "Particle swarm optimiser with neighbourhood operator." In proceedings of the IEEE congress on evolutionary computation (CEC) **3**(ISBN: 0-7803-5536-9): 1958-1962.

Toumazau, C., Georgiou, J. & Drakakis, E. M. (1998), "Current-mode analogue circuit representation of Hodgkin and Huxley neuron equations." Electronic Letters **34**(14): 1376-1377.

Trelea, I. C. (2003), "The Particle swarm optimization algorithm: Convergence analysis and parameter selection." Information processing letters **85**(6): 317-325.

Van Horn, J. D. & Ball, C. A. (2008), "Domain-Specific Data Sharing in Neuroscience: What Do We Have to Learn from Each Other?" Neuroinformatics **6**(2): 117-121.

van Schaik, A. (2001), "Building blocks for electronic spiking neural networks." Neural Networks **14**(6-7): 617-628.

Wakerley, J. B., Poulain, D. A., Brown, D. (1978), "Comparison of firing patterns in oxytocin- and vasopressin-releasing neurones during progressive dehydration." Brain Research **148**(2): 425-440.

Wolpert, D. H., Macready, W. G. (1997), "No free lunch theorems for optimization." IEEE Transactions on Evolutionary Computation **1**(1): 67-82.

Wong, M. L., Wong, T. T. & Fok, K. L. (2005), "Parallel evolutionary algorithms on graphics processing unit." The IEEE Congress on Evolutionary Computation **3**(ISBN: 0-7803-9363-5): 2286-2293.

Wright, A. H. (1991), "Genetic Algorithms for Real Parameter Optimization." Foundations of Genetic Algorithms v.1: 205-220 : ISBN-10: 1558601708.

Xilinx (2008a), "Virtex-5 Family Overview." v4.4.

Xilinx (2008b), "Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs."

Zhang, Q., Zhou, A. & Y., J. (2008), "RM-MEDA: A Regularity Model-Bassed Multiobjective Estimation of Distribution Algorithm." IEEE Transactions on Evolutionary Computation **12**(1): 41-63.

Zitzler, E., Laumanns, M. & Thiele, L. (2001), "SPEA2: Improving the Strength Pareto Evolutionary Algorithm." TIK Report 103.

Ziv, I., Baxter, D. A. & Byrne, J. H. (1994), "Simulator for Neural Networks and Action Potentials: Description and Application." Journal of Neurophysiology **71**(1): 294-308.