THE UNIVERSITY *of* EDINBURGH

THE INITIATION AND MAINTENANCE OF SWIMMING IN HATCHLING *XENOPUS LAEVIS* TADPOLES

MICHAEL HULL

Doctor of Philosophy

Institute for Adaptive and Neural Computation
School of Informatics
University of Edinburgh
March 2013

ABSTRACT

Effective movement is central to survival and it is essential for all animals to react in response to changes around them. In many animals the rhythmic signals that drive locomotion are generated intrinsically by small networks of neurons in the nervous system which can be switched on and off. In this thesis I use a very simple animal, in which the behaviours and neuronal networks have been well characterised experimentally, to explore the salient features of such networks. Two days after hatching, tadpoles of the frog *Xenopus laevis* respond to a brief touch to the head by starting to swim. The swimming rhythm is driven by a small population of electrically coupled brainstem neurons (called dINs) on each side of the tadpole. These neurons also receive synaptic input following head skin stimulation. I build biophysical computational models of these neurons based on experimental data in order to address questions about the effects of electrical coupling, synaptic feedback excitation and initiation pathways. My aim is better understanding of how swimming activity is initiated and sustained in the tadpole.

I find that the electrical coupling between the dINs causes their firing properties to be modulated. This allows two experimental observations to be reconciled: that a dIN only fires a single action potential in response to step current injections but the population fires like pacemakers during swimming. I build on this hypothesis and show that long-lasting, excitatory feedback within the population of dINs allows rhythmic pacemaker activity to be sustained in one side of the nervous system. This activity can be switched on and off at short latency in response to biologically realistic synaptic input. I further investigate models of synaptic input from a defined swim initiation pathway and show that electrical coupling causes a population of dINs to be recruited to fire either as a group or not at all. This allows the animal to convert continuously varying sensory stimuli into a discrete decision. Finally I find that it is difficult to reliably start swimming-like activity in the tadpole model using simple, short-latency, symmetrical initiation pathways but that by using more

complex, asymmetrical, neuronal-pathways to each side of the body, consistent with experimental observations, the initiation of swimming is more robust. Throughout this work, I make testable predictions about the population of brainstem neurons and also describe where more experimental data is needed. In order to manage the parameters and simulations, I present prototype libraries to build and manage these biophysical model networks.

LAY SUMMARY

Many animal processes are repetitive actions, such as chewing, breathing and walking. Some rhythms, such as the heartbeat, are continuous, but others, such as locomotion, need to be switched on and off. Once a rhythm is started, it is sustained by networks of nerve cells which produce output signals to drive muscles. A simple example of such a network produces swimming in young frog tadpoles. The tadpole begins to swim in response to a brief touch, swimming can last several seconds and stops when the tadpole bumps into something. The swimming rhythm can be generated by a small population of neurons which can be switched 'on' and 'off' by input from short sensory pathways. Two symmetrical populations of 30 nerve cells, one on each side of the brain, play a central role in initiating and sustaining these rhythms. These particular nerve cells have distinctive features: they are electrically connected and this is essential for reliable swimming; they excite each other to sustain their rhythmic activity and they receive direct excitation in response to touch.

I build computational models of these nerve cells to explore the significance of these special features on the initiation of swimming in the tadpole. Modelling allows experiments to be performed that are not possible in living tissue. Firstly, by matching the models to experimental records, I narrow down the most likely locations for electrical connections. I find that their presence causes the activity of the nerve cells to synchronise and also has a dramatic impact on the behaviour of the individual neurons. Secondly, I investigate the effect of the long-lasting, self excitation in the model and show that when a brief 'touch' is given to a single side of the simulated network, the feedback within this population leads to a long-lasting rhythm, which can be stopped by activating the 'stop' pathway. Finally, I investigate how the two populations of nerve cells on each side of the brain need to be excited in order to start swimming-like activity. I find that by exciting the populations of nerve cells differently on each side, the tadpole is able to respond quickly and reliably to touch.

ACKNOWLEDGMENTS

I have been very lucky to work with my supervisor, Alan Roberts, for his patience, enthusiasm and commitment to science. His actions have taught me much more than I expected. It has been a pleasure to be part of the tadpole lab in Bristol and many thanks go to Steve Soffe who has been very supportive and patient with my endless questions over the past four years, and Edgar Buhl with whom it was a pleasure investigating the head skin initiation pathways.

In Edinburgh, many thanks to my supervisors David Willshaw and Mark van Rossum as well as David Sterratt for their advice on modelling. I would also like to thank Bob Meech, for his patience in explaining the subtleties of interpreting electrophysiological data; to Roman Borisyuk for his help with mathematics; and Wenchang Li and Crawford Winlove, for access to their experimental data. On the tool building front, many thanks to Andrew Davison and Eilif Muller for their support and encouragement. Many thanks to Carolina Doran, Bob Meech and my supervisors for reading drafts of this thesis. Finally, a special thankyou to Mum, Dad & my sister Lauren, who have been incredibly supportive over the past few years.

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 specified for any other degree or professional qualification.

Edinburgh, March 2013

Michael Hull

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF GRAMMARS

ACRONYMS

4AP 4-Aminopyridine 37

aIN Ascending Interneuron 17

AMPA 2-Amino-3-(3-hydroxy-5-methyl-isoxazol-4-yl) Propanoic Acid 23

AMPAR AMPA-receptor 53

API Application Programmer Interface 69

ASCII American Standard Code for Information Interchange 77

AST Abstract Syntax Tree 79

BNF Backus-Naur Form 199

cIN Commissural Interneuron 17

CNS Central Nervous System 6

CPG Central Pattern Generator 4

CSA Connection Set Algebra 70

CSV Comma Separated Variable 78

DE Differential Equation 67

dIN Descending Interneuron 10

DL Dorsolateral 37

EPSP Excitatory Post-Synaptic Potential 52

GABA Gamma-Aminobutyric Acid 53

GHK Goldman-Hodgkin-Katz 50

HH Hodgkin-Huxley 11

HOC HOC 68

INCF International Neuroinformatics Coordinating Facility 80

IPSP Inhibitory Post-Synaptic Potential 51

LEMS Low Entropy Model Specification 69

MEA multi-electrode array 190

MHR Mid-Hindbrain Reticulospinal neuron 53

MLR Mesencaphlic Locomotor Region 6

MN Motoneuron 17

MODL MOdel Description Language 68

NMDA N-Methyl-D-aspartic acid 10

NMDAR NMDA-receptor 10

ODE Ordinary Differential Equation 79

PIR Post Inhibitory Rebound 10

PSP Post-Synaptic Potential 185

RB Rohon-Beard 16

SBML Systems Biology Markup Language 69

STG Stomatogastric Ganglion 12

TEA Tetraethylammonium 37

tIN Trigeminal Interneuron 25

TTX Tetrodotoxin

XML eXtensible Markup Language 69

XSLT eXtensible Stylesheet Language Transformation 69

Part I

INTRODUCTION

1

INTRODUCTION

"The poetry of motion! The *real* way to travel!
The *only* way to travel! Oh bliss! Oh poop-poop!"

– Mr. Toad
The Wind in the Willows

1.1 HOW DO ANIMALS CONVERT A STIMULUS INTO A BEHAVIOURAL RESPONSE?

1.1.1 *Overview*

To survive, animals need to be able to out-manoeuvre both those that want to eat them and those they want to eat. Effective mobility requires not only fast, accurate control systems, often with tight feedback loops, but also more complex sequences of sustained action. Advanced sensory systems have evolved to match their environments. For example, fish have a system of lateral lines for detecting vibrations in the surrounding water, elasmobrach fishes like sharks have ampullae of Lorenzini for detecting electric fields in the water and certain insects and birds are able to detect magnetic fields and light polarisation for orientation. Throughout nature, the behaviour exhibited by organisms shows an exceptional refinement to the challenges they face in their environments, spanning a diverse range from simple movement through to complex social interactions. These behaviours can be singular events like a bird turning its head in response to noise or sustained responses like a mouse running

away after being startled. It is sustained responses that will be considered in this thesis.

1.1.2 *The initiation of activity in rhythmic systems*

Most sustained behaviour is underpinned by repetition, for example in breathing, the heartbeat, mastication and locomotion. In many systems, once the behaviour has been established, a group of neurons is able to maintain the generation of motor commands intrinsically, without the need for sensory feedback. These networks of neurons are known as Central Pattern Generators (CPGs) [Marder, 2000] and were proposed to be central to rhythm generation in cat locomotion following experiments by Brown at the beginning of the 20^{th} century [Brown, 1911, 1914]. More recently, deafferation and electrophysiological studies have shown CPGs underlying various rhythmic patterns in both invertebrates and vertebrates. Flight (locust), swimming (*Tritonia*, *Clione*, *Lymnaea*, leech, dogfish, *Xenopus*), feeding and chewing (*Aplysia*, cat, rabbit), digestion (crustaceans), walking (cat, cockroach), breathing (cat, goldfish, cockroach) and the heartbeat (lobster, leech) have been shown to be driven by a central pattern generator that does not require sensory feedback to operate [Wilson, 1961; Clarac and Pearlstein, 2007; Delcomyn, 1980; Selverston, 2010; Roberts et al., 2010; Grillner, 1975; Grillner and Wallen, 1985]. It has been shown that CPGs can be frequency modulated and in some cases can be reconfigured to produce different behaviours in response to different inputs [Berkowitz et al., 2010]. Although sensory feedback is not essential to rhythm generation in a CPG, in some cases its absence can affect the output [Ayers et al., 1983]. CPGs provide an explanation of how rhythmic activity is generated and maintained once the network is active, but how activity is first initiated, often remains unclear. Specifically, which biological mechanisms allow a neuronal network to exist in either a state of quiescence, or in a state of generating rhythmic output, and how can these states be switched between? In some systems that are always active, such as the heartbeat and breathing, this is not a concern, but in neuroethology it is a central question.

Sensory information is first received by sensory receptors before being transmitted via neuronal pathways to the central nervous system. There, it is integrated with

other sensory inputs and the animal's behavioural context and finally can result in the initiation, maintenance, modulation or termination of behaviour. The initiation or termination of behavioural outputs are often seen as discrete *decisions*. I might *decide* to stop reading a book or to call out to a friend. We presume that the same thing goes on in other mammals and for example a scared cat at some point will *decide* to start to run away when it is frightened. In these cases, the organism is receiving continuously variable sensory input over time, which it must analyse and, at some point, convert into a binary decision (Fig. 1.1). In response to increasing fear, the cat does not start by slowly wandering away from the situation and instead a definite action is taken.

Decision-making is a subjective term. One broad perspective is that nervous systems will constantly receive stimuli, which are integrated over a continuum of timescales. When particular internal thresholds within the animal are reached, discrete changes (decisions), are made, which are observable as the initiation or termination of behaviours. According to this perspective, the same basic decision-making processes also exist in simpler creatures, such as escape responses in crayfish [Krasne, 1969], amphibians and fish [Korn and Faber, 2005] or the initiation of flight in insects [Boyan et al., 1986].

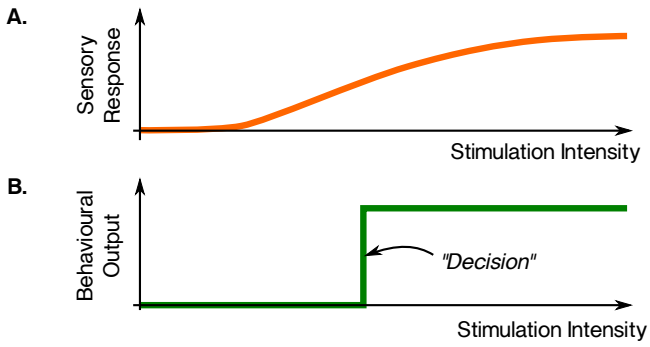


Figure 1.1 – A simplified view of decision making. **A.** The response of an animal's sensory pathways to increasing levels of stimulation. **B.** The behavioural response of an animal to increasing levels of stimulation. The continuously varying sensory response is converted into a discrete, behavioural output. When a change in the behaviour of the animal occurs, it is interpreted as a *decision.*

1.1.3 *The initiation of locomotion in vertebrates and invertebrates*

Effective locomotion in a complex environment requires a vast amount of sensory information to be processed very quickly. A monkey swinging from tree to tree has to time the decision to grab a swaying branch to within milliseconds, all the while continuously updating its internal model of the surrounding environment. In general, it is an open question how planning, sensory and motor data is encoded in the brain. A common view is that in higher animals, the control of locomotion is hierarchical; in higher brain areas abstract commands controlling locomotion are generated which are then processed through a chain of lower subsystems in the brainstem and spinal cord which are responsible for generating the detailed motor commands. For example, a signal to start running may be produced in the motor cortex, which is then passed through to the brainstem, which specialises in integrating this command with additional information such as the dynamics of the animal's balance or the presence of uneven terrain, to produce the final muscle activation commands [Stein et al., 1997; Jordan, 1998; Orlovsky et al., 1999].

In mammals, the regions of the Central Nervous System (CNS) associated with basic locomotor behaviour like walking and its initiation have been investigated in lesion studies and it is surprising how little nervous system is required to generate basic locomotor patterns. Brown showed that the neurons in the lumbar spinal cord of cats are capable of generating rhythmic activity in the hind limbs [Brown, 1911, 1914]. In cats in which the entire forebrain is lesioned, electrical stimulation of different regions of the brainstem can lead to walking, trotting and galloping. In such animals, although movements are well coordinated and balance is maintained, the behaviour is not goal-orientated and the motion is robot-like [Stein et al., 1997]. Surprisingly, decorticated kittens are still able to perform complex patterns of behaviour, such as searching for food or attacking other individuals [Stein et al., 1997]. In general in more complex organisms, it has been difficult to explain the detailed mechanisms responsible for the initiation and maintenance of locomotion definitively. However, although there is variation across species, certain regions such as the motor cortex, cerebellum, Mesencaphlic Locomotor Region (MLR) and the basal ganglia are thought to play important roles [Stein et al., 1997; Jordan, 1998; Orlovsky et al., 1999].

In lower vertebrates, there are substantially fewer neurons and in some cases investigation has been possible at the level of individual neurons [Fetcho et al., 2008; Roberts et al., 2010]. The lamprey is a 13 to 40 cm long fish, with a distinctive jawless, toothed mouth. It can initiate swimming in response to a variety of stimuli including mechanical stimulation, water waves and illumination of the eyes. The spinal cord, which consists of ~100 segments, each containing ~1000 neurons, is able to produce rhythmic swimming-like activity [Buchanan, 2001]. Although specific classes of spinal neurons have been identified based on morphology and electrical properties [Buchanan, 2001] finding individual neurons which initiate locomotion has not yet been possible. However, brain regions that are involved such as the MLR have been identified and connections between them have been characterised and show similarities to those in more complex mammalian brains. [Stein et al., 1997; Mullins et al., 2010]. Larval fish and amphibians have also been studied extensively. The zebrafish is an attractive model system in neuroscience because of its amenability to genetic studies and its transparency, which allows non-invasive photo-ablation and imaging of neurons [Fetcho and Liu, 1998]. As in other vertebrates, neurons in the spinal cords of larval *Xenopus* and zebrafish are able to generate rhythmic motor output [Fetcho et al., 2008; Roberts et al., 2010] (see below for *Xenopus*). Zebrafish can generate swimming at distinct frequencies, which is achieved by recruiting more pools of spinal neurons into the CPG at higher frequencies [McLean et al., 2007]. It has recently been proposed from work on adult zebrafish that the neurons in the CPG in the spinal cord are able to maintain swimming following brief input from descending inputs in the brain [Kyriakatos et al., 2011].

Some of the most detailed information on the initiation of locomotion comes from simpler invertebrates[Sattelle and Buckingham, 2006]. In many invertebrate systems, an individual neuron can be reliably identified across individual animals by its morphology, electrophysiology and connectivity, in contrast to vertebrates, in which the neurons are usually members of larger homogeneous populations. In invertebrates, the stimulation or inactivation of individual neurons can lead to behaviourally detectable consequences, making it possible to identify pathways at the level of individual neurons. For example, in certain systems such as the sea slug *Tritonia* and leech, stimulation of a single neuron activates locomotion in the whole animal [Willows, 1967;

Weeks and Kristan Jr, 1978]. Medicinal leeches show a range of behaviours, including local bending, shortening, swimming, crawling and feeding [Friesen and Kristan, 2007]. Although the nervous system of the leech contains over 10,000, neurons, the animal is segmented and many of the neurons are strongly stereotyped in each of the 21 segmental ganglia. They are an interesting model of simple decision making because they switch between two distinct behaviours, crawling and swimming, in response to the depth of surrounding water. [Friesen and Kristan, 2007; Kristan et al., 2005]. In the leech, pathways for the initiation of swimming have been identified at the level of individual neurons and individual trigger neurons have been identified [Weeks and Kristan Jr, 1978; Kristan et al., 2005; Lamb and Calabrese, 2011]. The marine mollusc *Tritonia diomedea* also shows a variety of interesting behaviour including escape swimming, magnetic field orientation, rheotaxis, crawling, feeding, mating. The entire nervous system is complex; over 180 types of neuron have been identified and the central ganglion contains about 8000 neurons [Boyle et al., 1983]. However, the central pattern generator underlying escape swimming consists of just three types of neuron. An initiation pathway from sensory neurons to the CPG has been described at the level of the specific individual neurons and the connections between them [Frost et al., 2001].

Another well studied invertebrate is the nematode *C. elegans (Caenorhabditis elegans)* which has a nervous system of just 302 neurons, in which the connectome of the rostral part of the nervous system, including morphologies, synaptic connections and gap junctions have been revealed in serial-section electron micrograph reconstructions [White et al., 1986]. The animal is able to move backwards and forwards, which can be triggered by touching the head and tail respectively with fine hair. The individual neurons and the synaptic connections in these response pathways from sensory neurons to motor neurons are known [Chalfie et al., 1985], although the electrical characteristics of the specific neurons are not well characterised because the size of the neurons and internal pressure within the animal makes electrode recordings difficult. Microlasers can be used to ablate cells *in vivo* and investigate effects on behaviour [Sulston and White, 1980] and more recently, impressive use of real-time video analysis, steerable lasers and injection of light-dependent channels into specific neurons has allowed non-invasive manipulation of behaviour by remotely exciting or

inhibiting neurons [Leifer et al., 2011]. However, *C. elegans* shows some fundamental differences to other common models of rhythm generation. It is not clear whether locomotion in *C. elegans* is generated independently by a CPG or whether sensory feedback is necessary for locomotion. Interestingly, the locomotive subsystem contains relatively few inhibitory synapses which would be expected in reciprocally inhibitory CPG networks and it is not clear whether the neurons fire action potentials or instead show graded responses [Bryden and Cohen, 2004; Lockery et al., 2009].

1.1.4 *Networks producing sustained responses*

Once activity has been initiated within a population of neurons, how is it maintained? Two main proposals have been made: a network-based mechanism and a cellular-based mechanism [Grundfest, 1968]. The first proposal was made over 100 years ago to explain rhythmic limb movements in cats [Brown, 1911, 1914]. In this model, two populations of neurons called half-centres, make mutual, reciprocal, inhibitory connections onto each other (Figs. 1.2 & 1.3). Activity in one half-centre causes the opposite half-centre to become inhibited. After some time, the inhibition decays, releasing that half-centre and allowing it to fire and in turn inhibit the opposite half-centre. Again, after some time, this inhibition decays and the first half-centre becomes active, and so forth (Fig. 1.2). The second proposal is a pacemaker mechanism, in which a population of neurons has a tendency to fire at a particular frequency without the need for inhibitory feedback. For example, in the heart, there is a small population of cells in the sinoatrial node which drive the contractions and these cells contain specific currents to produce rhythms of the appropriate frequencies [Noble, 1962; Hille, 2001]. In pacemaker mechanisms, synchronisation between the individual pacemaker elements will clearly be essential, which in the heart is achieved by electrical coupling between cells.

Generally, it has been difficult to disentangle which mechanism is primarily responsible for rhythm generation in different networks and whether they are mutually exclusive. Many rhythm-generating networks contain neurons that fire on rebound in response to fast inhibitory input (*Clione* [Satterlie, 1985; Pirtle and Satterlie, 2007]; motoneurons in leech swimming [Angstadt et al., 2005]; interneurons in leech heart

[Calabrese, 1998]; Descending Interneurons (dINs) in the tadpole [Roberts et al., 2010]; lamprey [Matsushima et al., 1993]; zebrafish [Gabriel et al., 2010]; motoneurons in neonatal rat [Bertrand, 1998]). It is not clear whether Post Inhibitory Rebound (PIR) mechanisms alone are sufficient to generate rhythm. In some animals, lesion studies suggest that a single side of the nervous system is able to generate rhythm, but these hemisection experiments may have side-effects [Hoffman and Parker, 2010]. As to background excitation, in some systems, for example *Clione*, slow swimming patterns can be generated in the absence of tonic drive for variable periods of time [Pirtle and Satterlie, 2007]. In higher animals, is it often presumed that background, tonic, depolarising drive is needed, which could come from either feedback connections within the population itself [Roberts et al., 2008] or from higher brain areas [Orlovsky et al., 1999] (Fig. 1.3). Such a drive could come from long-lasting synaptic input, and in many vertebrates a perfusion of N-Methyl-D-aspartic acid (NMDA), a drug which activates excitatory NMDA-receptors (NMDARs), leads to rhythmic activity (for example in lamprey [Grillner et al., 1981]; tadpole [Dale and Roberts, 1985]; mudpuppy [Wheatley and Stein, 1992]; turtle [Currie, 1999]; chick [Barry and O'Donovan, 1987]).

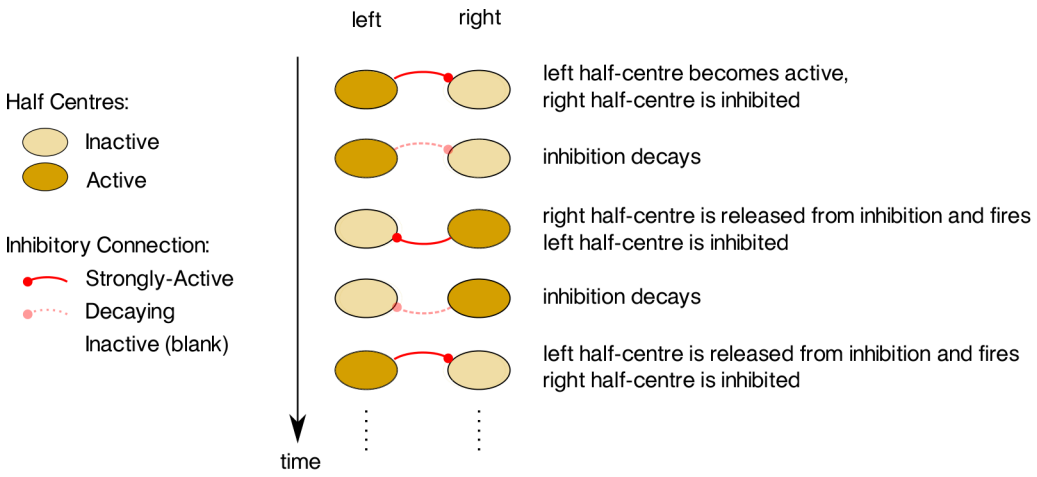


Figure 1.2 – The generation of antiphasic rhythm in the half-centre model. Two opposing populations of neurons, *half-centres*, make reciprocal inhibitory connections onto each other. When one side becomes active, it inhibits the opposite population. This inhibition decays over time, which releases the opposing half-centre and allows it to fire. This inhibits the previously active half-centre and the process repeats.

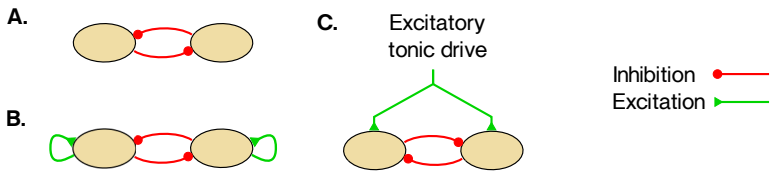


Figure 1.3 – Configurations of the half-centre model. **A**. In some systems (e. g. *Clione*), neurons at rest have been observed to show PIR. **B, C.** In other systems, neurons require a level of tonic background excitation for rebound firing to occur, which from could come from either feedback excitation (B; e. g. tadpole) or from other brain areas (C).

1.1.5 *Modelling perspective*

Mathematics is an essential tool for building and testing hypotheses in science, but unfortunately complex systems such as neuronal networks are analytically intractable. Instead, computational modelling can be used to investigate the effects of different building blocks of the system on the dynamics of its behaviour. At the level of individual neurons, Perkel and Mulloney demonstrated models that could produce self-sustaining alternating activity in simulations of reciprocally inhibitory neurons with PIR [Perkel and Mulloney, 1974]. Since then, many models of rhythmic systems have been built for various rhythmic behaviour at various levels of detail. A model of the *Tritonia* swimming network was built by Getting, using a hybrid of Hodgkin-Huxley (HH) and integrate-and-fire dynamics (see Chapter 2, [Koch, 1999]) , which was able to produce a realistic escape swimming pattern [Getting, 1983]. More recently, modelling the initiation of *Tritonia* swimming suggested that the network may not oscillate initially and that substantial reconfiguration at multiple sites in the network to increase recruitment and coupling between specific groups of neurons would be necessary for a network that could maintain oscillations [Calin-Jageman et al., 2007]. Models of the segmented structure of the lamprey spinal networks have been built that combine physical and electrical properties and can generate different swimming frequencies depending of the intensity of input stimulation [Ekeberg, 1994; Várkonyi et al., 2008]. In *C. elegans*, an unresolved question is whether rhythm is produced intrinsically by a CPG, independently of sensory feedback, and modelling has been used to investigate how the physical and electrical properties of the system could be interact [Bryden and Cohen, 2004; Boyle and Cohen, 2008]. Locomotor be-

haviour has also been studied at a higher level of abstraction. In the leech, the body segments each contain a group of interneurons that operate as a segmental oscillator. Locomotion in the leech is dependent on the coordination of the relative phases of each segment and models of leech locomotion have mainly focused at the level of coupled oscillators [Zheng et al., 2004]. A similar type of model has been built for lamprey [Várkonyi et al., 2008].

In general, there has been little modelling on the pathways which initiate rhythmic activity at the level of individual neurons. In CPG models, initiation is often sidestepped and activity is often started using specifically timed artificial current injections into the neurons on each side, for example [Dale, 1995a; Knudsen et al., 2006; Sautois et al., 2007].

A common problem with building neuronal network models is the lack of good data on the properties of neurons and synapses and also for the individual ion channel's kinetics, distributions and densities. The active dynamics of the voltage-gated channels lead to chaotic, non-linear interactions, meaning that even small changes in parameters can have large effects on model behaviour. The dynamics are often determined by high dimensional parameter spaces, in which values can be difficult to determine biologically. Moreover, it is likely that the 'correct' parameters for a model form a region, rather than a single point, meaning that our estimates for parameters will not improve by simply making more measurements and taking averages. This is particularly well documented for the crustacean Stomatogastric Ganglion (STG), in which parameter sweep approaches have been used to systematically analyse the model space [Marder et al., 2007; Taylor et al., 2009; Prinz, 2010].

1.2 WHY ARE HATCHLING *Xenopus laevis* TADPOLES A USEFUL MODEL SYSTEM?

Amphibian embryos have been one of the most important animals used in studies of animal development and have played a critical role in studies of the nervous system [Coghill, 1929; Sanes et al., 2006]. *Xenopus* is an attractive animal for study because it can be induced to breed at any time of year and its cells are unpigmented, which makes them amenable to anatomical and physiological investigation. In comparison

to many other vertebrate model systems, *Xenopus* tadpoles have a relatively small number of neurons; it is estimated that the circuits for swimming and struggling contain approximately 2000 neurons [Roberts et al., 2010]. Certain invertebrate model systems also have relatively small numbers of neurons, for example, *C. elegans* has 302 [White et al., 1986], but each neuron has distinct electrophysiological and morphological characteristics . In contrast, in *Xenopus* tadpoles, neurons fall into a small number of distinct classes (described below).

1.2.1 *The behavioural response of the animal*

When it hatches, at stage 37/38, approximately 48 hours post fertilisation, the *Xenopus laevis* tadpole is approximately 5 mm long (Fig. 1.4). During this transient, developmental stage, many internal systems of the animal are not yet developed to a functional state. The animal cannot eat and relies on internal nutrition supplied through the yolk and is starting to form eyes but is still blind [Faber and Nieuwkoop, 1956]. It is estimated that tadpoles at this stage of development remain stationary for 99% of the time [Jamieson and Roberts, 2000]. To conserve energy and avoid predation, they can remain still by attaching to solid surfaces using a pressure activated cement gland on their heads, which releases a sticky mucus and allows them to hang from objects [Lambert, 2004].

Remaining still in a safe, unchanging environment would be a good strategy, but environments are more dangerous and it is important for a tadpole to react to stimuli and avoid predation. In response to brief stimulation to the body, a stationary tadpole will start to swim. Swimming is characterised by antiphasic contractions of the muscle on each side of the animal at 10 to 25 Hz [Kahn et al., 1982]. These contractions propagate rostrocaudally from the hindbrain towards the tail of the animal, which propels the tadpole forward through the water (Fig. 1.5).

In response to strong and sustained stimuli, the tadpole will start to struggle. Struggling is also characterised by the antiphasic contractions of each side, but the movements are much more powerful, at lower frequencies of 2 to 5 Hz, and the wave propagates from the tail to the head [Kahn and Roberts, 1982b]. This would help the

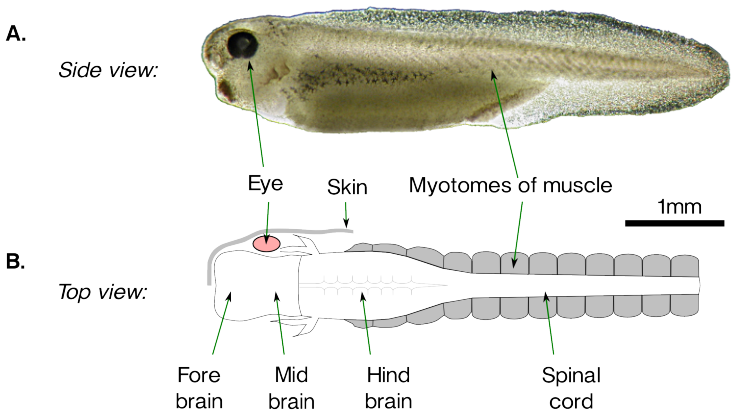


Figure 1.4 – The stage 37/38 hatchling *Xenopus laevis* tadpole. **A**. Side view photograph. **B.** Top-down cartoon representation. The division of the nervous system into the fore/mid-brain, hindbrain and spinal cord and the major anatomical features are shown.

animal to escape, for example, from the jaws of a predator that have latched onto it. Struggling behaviour is not considered in this thesis.

1.2.2 *How Xenopus tadpoles have been studied*

Simple sensory-motor systems are useful models to investigate neuronal networks, because both the sensory inputs and motor outputs are directly observable, leaving a blackbox inbetween to decipher (Fig. 1.6). In the hatchling tadpole, sensory neurons send out processes that innervate the skin [Roberts, 1980]. Touching the skin can depolarise the local area of sensory neuron processes in the skin and lead to the initiation and transmission of an action potential back to the sensory soma: the input to the system. On the output side: motor neurons send out axons to innervate muscles at neuromuscular junctions [Roberts et al., 1999]. Action potentials initiate near the soma and are conducted along the axons, reaching the neuromuscular junctions where they cause the muscle fibres to contract and hence produce movement (Fig. 1.6). Simultaneous intracellular and ventral root recordings were made in preparations of immobilised tadpoles in which the neuromuscular junction had been blocked [Kahn et al., 1982]. These showed that swimming patterns of motor neurons are generated intrinsically by a CPG and do not require continuous sensory feedback.

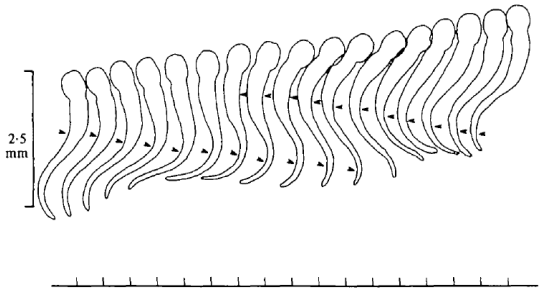


Figure 1.5 – Tracings of swimming movements in stage 37/38 *Xenopus* tadpoles, from film taken at 300 frame/s. Waves of bending pass alternately down either side of the body (arrowheads). The sequence reads from left to right, each frame is displaced to the right by the distance shown on the scale divisions of the baseline. Taken from [Kahn et al., 1982].

We would like to understand what happens inbetween these two groups of neurons (i. e. Fig. 1.6; grey boxes) in the systems which initiate and maintain behavioural patterns. To understand the neuronal network, the different types of neuron within the nervous system have been characterised. The morphologies and the locations of neurons have been investigated with dyes, applied to neuron axons or injected into the somata of individual neurons, which diffuse into the processes and make the entire cell visible under a microscope [Roberts, 2000; Roberts et al., 2010]. The electrophysiological properties of the neurons have been investigated by *in situ* whole-cell recordings in immobilised tadpoles. Based on these properties, the neurons involved in the initiation and maintenance of both swimming and struggling behaviours have been classified into approximately 10 groups. Most of the classes form symmetrical, longitudinal columns of neurons, one column on each side of the nervous system, that descend from the hindbrain into the spinal cord (Fig. 1.7). Connectivity between neurons has been investigated using paired recordings, in which two neurons in the nervous system are simultaneously patch-clamped [Li et al., 2004b]. By triggering an action potential in a presynaptic neuron and simultaneously recording a postsynaptic neuron, the connections between the two neurons and the synaptic kinetics can be inferred. Based on measurements from >1500 pairs of neurons, in which the neurons' classes were also identified, a reasonably full picture of the synaptic connectivity between the neurons of the different classes is now available [Roberts et al., 2010].

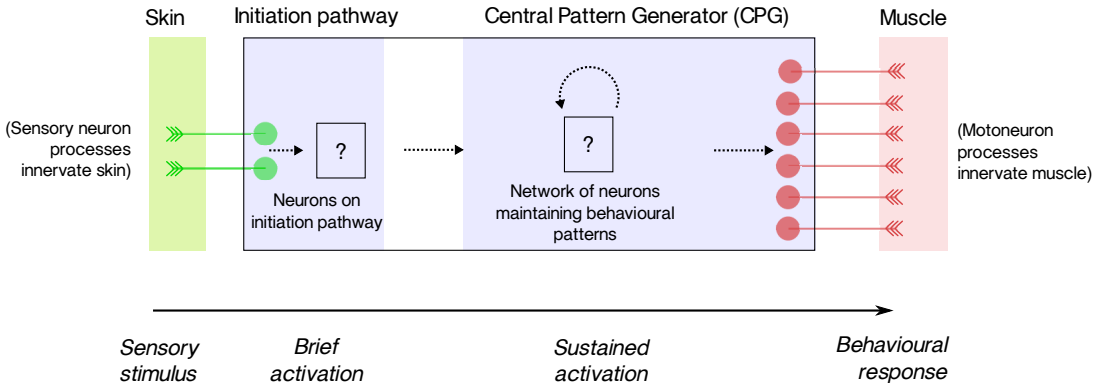


Figure 1.6 – The inputs and outputs of the network controlling behaviour in tadpole. Sensory stimulation leads to activation of sensory neurons. In swimming, this brief excitation is transmitted along an initiation pathway, which activates the CPG. Once active, the CPG is able to maintain activity intrinsically and produces a rhythmic output. The CPG neurons excite motor neurons, which cause muscle to contract and the animal to swim.

1.2.3 *Initiation pathways*

The accessibility of the neurons in the tadpole has made intracellular and ventral root recordings possible in immobilised tadpoles during swimming, which has allowed the definition of pathways at the level of small populations of neurons [Li et al., 2004b; Buhl et al., 2012; Roberts et al., 2010]. For swimming, two independent sensory pathways have been defined. The first responds to stimulation of the head-skin and consists of trigeminal touch sensitive afferent neurons [Roberts, 1980] which excite a group of trigeminal interneurons that relay excitation to the same side [Buhl et al., 2012] (Fig. 1.8), and another group of interneurons that relay excitation to the opposite side (not shown). The second pathway consists of Rohon-Beard (RB) sensory neurons [Clarke et al., 1984], which respond to stimulation of the trunk skin and also excite interneuron populations carrying excitation from a stimulus on one side of the body to both sides of the CNS [Li et al., 2003, 2007b].

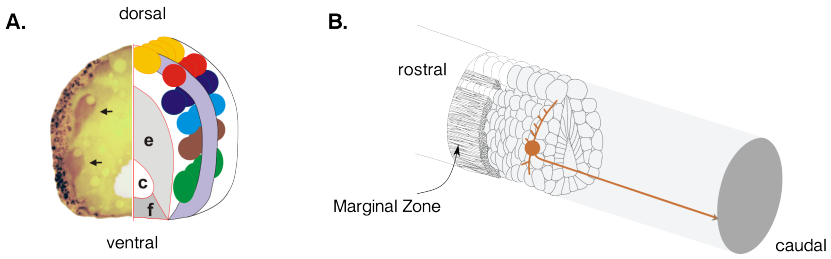


Figure 1.7 – The layout of CPG neurons in the tadpole spinal column **A.** A cross-sectional view of the spinal column showing neuron types (coloured circles) forming longitudinal columns. **B.** A perspective drawing of the spinal cord, in which a single neuron has been highlighted in brown, with a descending axon and dorsoventral dendrites.

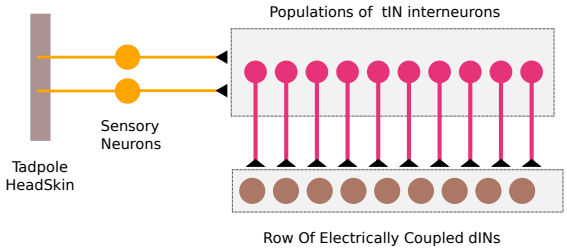


Figure 1.8 – The ipsilateral head-skin pathway. Trigeminal touch sensitive afferent neurons have processes that innervate the head-skin and synapse onto a group of trigeminal interneurons (tINs) which relay excitation to the same side [Buhl et al., 2012].

1.2.4 *The swimming networks & the role of an electrically coupled group of neurons driving swimming (dINs)*

In the *Xenopus* tadpole, a combination of lesion studies and electrophysiological recordings have shown that swimming patterns can be maintained by a small region of the rostral spinal cord and caudal hindbrain containing a population of roughly 300 neurons [Li et al., 2006]. Four types of neurons are involved in the CPG for swimming: Ascending Interneurons (aINs), Commissural Interneurons (cINs), dINs and Motoneurons (MNs), which form symmetrical populations on each side of the animal (Figs. 1.7 & 1.9). The connectivity between the types of neurons is summarised in Fig. 1.9. During swimming, each of these neurons fires a single action potential during each cycle.

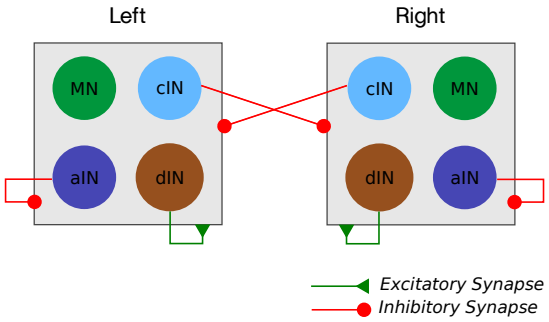


Figure 1.9 – An overview of the synaptic connections between interneurons in the tadpole swimming CPG. On each side of the nervous system are populations of neurons, represented by circles: Motoneurons (MNs), Commissural Interneurons (cINs), Ascending Interneurons (aINs) and Descending Interneurons (dINs). Connections to a box represent synaptic connections to all neurons within that box. The dINs excite ipsilateral neurons, and the cINs and aINs inhibit contralateral and ipsilateral neurons respectively.

A population of excitatory interneurons with descending axons (dINs) [Dale and Roberts, 1985] are thought to play a central role in the swimming CPG [Roberts et al., 2010]. These neurons are the first to fire on each side during swimming and make excitatory synaptic connections to the other neurons on the same side, and are therefore thought of as the *trigger* neurons for swimming [Roberts and Alford, 1986; Li et al., 2006; Roberts et al., 2008; Soffe et al., 2009]. In response to *in situ* step current injections, these neurons only ever fire a single action potential, irrespective of the strength of stimulation, but have the property that they can *fire on rebound* [Li et al., 2006] from fast inhibition when they are depolarised. During swimming, the dINs receive mid-cycle inhibition from the cINs [Roberts et al., 1988], which causes them to fire on rebound after the inhibition decays. This is thought to be important in maintaining the antiphasic firing of the two sides, however the relative importance of the network and cellular mechanisms remains unclear in rhythm generation, since a single side of the nervous system perfused with NMDA can also generate rhythm. The dINs have been shown to make excitatory feedback NMDAR synapses onto other dINs on the same side and it is thought that during swimming, this self-excitation of the dIN population causes them to remain rhythmically active and fire as *pacemakers* [Li et al., 2010]. (In some systems, for example STG networks, the term *pacemaker* is used to refer to cells that fire intrinsically without external stimulation [Marder and Bucher, 2007]. In the context of tadpole rhythm generation and this thesis, *pacemaking*

is used in the context of dINs in the respect that their low firing rate is hypothesised to directly drives the frequency of swimming [Li et al., 2010]).

Whole-cell recordings from pairs of dINs have shown that they are electrically coupled [Li et al., 2009]. Electrical coupling has been observed in a range of studied systems, from simple invertebrates to mammalian cortex, and many roles have been suggested. Electrical synapses are widespread during development [Szabo and Zoran, 2007], but are also seen in developed systems, in a diverse range of contexts [Marder, 1998; Simon and Goodenough, 1998; Simon, 1999; Connors and Long, 2004; Bennett and Zukin, 2004]. Electrical coupling is often observed in systems controlling locomotion, and especially in systems requiring rapid reaction for escape; for example, the tail flip escape response in crayfish was the first place where neuron to neuron electrical transmission was found [Furshpan and Potter, 1959] and the Mauthner neurons which produce escape responses in goldfish are both highly electrically coupled [Lin and Faber, 1988]. In the tadpole, pharmacologically blocking this coupling leads to less reliable swimming: much shorter swimming episodes, a reduction in the synchronisation of spike timing within the population and a reduction in the number of dINs firing on each cycle [Li et al., 2009]. It is likely that the efficiency of a tadpole's swimming may be dependent on the synchronous contractions of the swimming muscles. The motor neurons innervating these fibres are themselves locally electrically coupled [Perrins and Roberts, 1995] which may synchronise their firing. However, motoneurons are directly excited by dINs, so the synchronisation of firing in the dIN population may also play an important role during swimming.

1.2.5 *Models of tadpole swimming*

Network models of the tadpole CPG circuits have been built which are capable of producing swimming and struggling-like motor outputs [Roberts and Tunstall, 1990; Dale, 1995a; Sautois et al., 2007; Li et al., 2007b].

Dale [1995a] used channel data from voltage-clamp recordings to build a single compartment neuron model. The tadpole network was conceptually simplified to two neurons, which made mutual reciprocal inhibitory synapses onto each other and each neuron made excitatory feedback synapses onto itself. This network was able to

sustain locomotor-like output following brief antiphasic synaptic excitation to the two sides. More recently, in a study investigating the roles of the different neuron types, point-neuron models of the CPG interneurons (dINs, cINs, aINs and MNs) with similar firing properties to those seen experimentally were built and when they were connected together with synapses based on experimental data (as shown in Fig. 1.9), they were shown to be able to maintain rhythmic swimming-like activity following a brief sensory excitation, which was modelled as timed current-clamp injections to the two sides [Sautois et al., 2007]. These models are of bilateral, symmetrical animals, and use network-based mechanisms for generating rhythm. The modelling studies have suggested that the specific firing properties of classes of neurons, particularly the single-spike firing in dINs, are central to the network generating behaviour [Sautois et al., 2007; Li et al., 2007b]. To build this model, the firing properties for each type of neuron were matched against physiological voltage responses by adjusting the kinetic properties of the membrane currents for each neuron type. Electrical coupling has not previously been incorporated into models [Roberts and Tunstall, 1990; Sautois et al., 2007]

In network models, the numbers and types of synaptic connections between the neurons of different classes are thought to be important to network operation. In general, it is unclear how synaptic connections between neurons are regulated [Sanes et al., 2006]. In the tadpole, it has been proposed that the probability of different pairs of neurons making synaptic connections onto each other can be explained in terms of the relative locations of their axons and dendrites. Rather than using complex recognition mechanisms, axons simply tend to make synapses onto dendrites that grow in the same dorsal-ventral positions, and so connectivity is a consequence of neuronal geography [Li et al., 2007a]. The tadpole spinal cord (Fig. 1.7) is imagined to be cut along the top and opened *like a book* to produce a flattened, 2 dimensional CNS. Using modelling it has been shown that if the somata of populations of neurons are laid out in this flat CNS and simple growth rules are used to create the axons and dendrites, then it is possible to produce a network where the pattern of synaptic connections between the neurons in the CNS is similar to what is found experimentally [Li et al., 2007a]. Next by using the connectivity from this model and using Morris-Lecar type neuron models, this network was capable of generating swimming-like

rhythmic activity [Morris and Lecar, 1981; Borisyuk et al., 2008, 2011]. Currently a model is being developed to test whether a system of three chemical gradients and appropriate responses from neuronal *growth cones* could generate an axonal layout and pattern of synaptic connections capable of producing a functioning swimming network [Li et al., 2007a; Borisyuk et al., 2008, 2011].

1.2.6 *Conclusion*

One simple animal, where significant progress has been made to understand the general question of how nervous systems respond to stimuli and produce behaviour, is the hatchling *Xenopus laevis* tadpole. The tadpole is an attractive model for several reasons. Firstly, the relative simplicity of its nervous system makes it tractable at the level of the entire organism. In more complex systems, in which investigation focuses on a smaller region, assumptions have to be made about the input and outputs to that subsystem. For the tadpole, such assumptions are unnecessary because the inputs and outputs to the black-box are well defined and so when testing our model with a given input stimulus, we can be confident about the behavioural output we expect to see. Secondly, the animal is in a developing, transient phase: it has only been alive for a short while, and will only stay in this *scaffold* state for a short while longer. The hope is that its regulatory mechanisms might be cruder and less refined than in more developed systems. A combination of behavioural, anatomical, lesioning and electrophysiological studies have identified the classes of neurons and connectome responsible for producing swimming and struggling behaviours. The *Xenopus* tadpole is arguably the best characterised and understood vertebrate spinal locomotor system [Parker, 2009].

1.3 HOW *Xenopus* TADPOLES WILL BE USED TO ADDRESS MAJOR QUESTIONS?

In this thesis I will explore what are some of the important features surrounding the initiation of swimming in the hatchling *Xenopus* tadpole. Biologically, this is a very reliable and significant process, as it may help the tadpole escape being eaten [Lambert, 2004]. I will use new experimental data about the head-skin pathway to

build computational models of this process [Buhl et al., 2012]. A central issue in the initiation process is how the columns of electrically coupled dINs on each side of the tadpole body react to excitation following head-skin stimulation in order to start swimming effectively.

Building models often involves estimating unknown parameters. In this modelling, I have tried to constrain parameter spaces by ensuring that the dIN model reproduces experimental observations. The following chapters outline the journey taken in incrementally adding complexity to a model in order to reach a bilateral model of the swimming CPG and its response to head-skin initiation pathways. Broadly, the following steps were taken:

A. Building kinetic models of the voltage-gated channels.

B. Inferring gap junction layouts in a small population of axo-axonically coupled passive dINs.

C. Building an active model of an individual dIN.

D. Investigating the effect of NMDAR feedback excitation for sustaining rhythm in a small population of dINs.

E. Investigating the effect of electrical coupling on the recruitment of a single population of dINs to head-skin pathway input.

F. Investigating how the structure of the head-skin initiation pathways affects the initiation of swimming-like activity in a bilateral swimming CPGs network.

This process involves building and experimenting on many models and variations. It is important to have suitable tools to effectively manage the models and experiments, and therefore a series of tools and libraries have been developed, which are described in overview in Chapter 3. The implementation details and example code for these libraries are given in Appendices B-G.

1.3.1 *Chapter 2: Modelling components of a brainstem network (A)*

[A] The ability of the tadpole locomotor networks to produce different behaviours is crucially dependent on the firing properties of the different classes neurons in

response to synaptic input [Sautois et al., 2007], and the voltage-gated channels in the membrane play an important role in defining these [Hille, 2001; Winlove and Roberts, 2012]. New voltage-clamp data on potassium currents in tadpole spinal neurons is available, which is used to build new models of potassium channels [Winlove and Roberts, 2012]. Existing models of sodium and calcium channels based on [Dale, 1995a,b] were also implemented as well as models of 2-Amino-3-(3-hydroxy-5-methyl-isoxazol-4-yl) Propanoic Acid (AMPA), NMDA and inhibitory synapses which are used in Chapters 4, 5 & 6.

1.3.2 *Chapter 3: Managing complexity in simulation tools*

Computational models of the electrical behaviour of a neurons have existed for many years, and software tools and libraries have been written to solve the equations efficiently, which allows large populations of neurons to be simulated on desktop computers. Modelling, however, still remains a time-consuming endeavour, and I discuss some of the sources of bottlenecks in computational sciences generally. I demonstrate prototypes of tools designed to shift unnecessary complexity from the modeller back into software. I demonstrate a high-level, simulator-independent Python library which allows simulations of small populations of multicompartmental neurons, including analysis and visualisation, to be defined entirely within a single Python file. I also propose a simple grammar for defining quantities and sets of equations involving units, and a tool for managing large numbers of simulations. These tools are used to build the simulations in Chapters 4, 5 & 6.

1.3.3 *Chapter 4: A population of electrically coupled dINs (B, C)*

[B] Electrophysiological recordings provide data about the strength of electrical coupling between dINs in the tadpole [Li et al., 2009] but it has not been possible to define the distribution and resistances of gap junctions, which are most likely to be axo-axonic. Using anatomically realistic, passive, multicompartmental models of somata and axons I investigate possible spatial distributions of gap junctions in a population of 30 dINs by matching models to experimentally measured coupling coef-

ficients. The relatively small number of parameters allows constraints to be placed on possible axon diameters and gap junction locations on the axons. Next, using the channel models implemented in Chapter 2, I match the dIN model behaviours to those observed physiologically, and find more constraints on the distribution of channels over the soma, initial segment and axon.

[C] Using this electrically coupled dIN population model, I next investigate how electrical coupling could affect the firing properties of neurons. In particular, it is important to understand how dINs, which only ever fire a single action potential in response to step-current injections during whole-cell patch recordings, are able to generate rhythmic, pacemaker-like activity during swimming. By reproducing *in situ* experiments *in silico,* I show that electrical coupling can have a dramatic effect on firing behaviour of neurons. I investigate the effect of the strength of electrical coupling, to investigate how electrical coupling can facilitate the modulation of behaviour by the activity in surrounding neurons.

1.3.4 *Chapter 5: Sustained pacemaker activity in a population of dINs (D)*

[D] Long-lasting NMDA-mediated feedback synaptic connections are introduced into the single-sided population of dINs, and I investigate the response of the network to brief synaptic input. Based on experimental estimates for population sizes, synapse strengths and connectivities, I show that the NMDA synapses between the members of the population of 30 dINs would allow the network to sustain rhythmic pacemaker firing following a brief synaptic excitation. I investigate the effects of perfusing NMDA over the population of dINs, and investigate the effects of the voltage-dependency of the NMDARs due to magnesium block. I investigate how such a network can be switched on and off at short latency by synaptic input from experimentally defined excitatory and inhibitory sensory pathways. I generalise the result by demonstrating that a population of HH-type neurons with NMDA-mediated feedback synapses can also be switched on and off synaptically and show that networks of other types of neurons with feedback NMDA-mediated synaptic connections could also be switched on and off by brief synaptic input.

1.3.5 *Chapter 6: The decision to initiate swimming (E, F)*

[E] By building a network of electrically coupled network of dINs, using results from the two previous chapters, I investigate the effect of electrical coupling on the recruitment of the members of a single dIN population, by driving the neurons with synaptic input with strengths and timings corresponding to those measured electrophysiologically from the head-skin pathway (Trigeminal Interneurons (tINs)) [Buhl et al., 2012]. In particular, I investigate the effects of electrical coupling on the recruitment of the dINs to fire as a group.

[F] Finally, swimming in the tadpole involves the antiphasic firing of populations of neurons on each side of the animal. I build a network consisting of two populations of electrically coupled dINs and two populations of inhibitory cINs, in order to investigate bilateral initiation of swimming in the tadpole. Experiments have shown that the excitation to the populations of dINs on each side of the body is asymmetrical. In a series of experiments I investigate the effects of input from simple hypothetical initiation pathways, and find that by introducing asymmetry, it is possible to produce a model that has short latencies until the start of swimming, is able to start swimming activity on a random side, and also robustly initiates antiphasic firing across different levels of stimulation.

Part II

METHODS

2

MODELLING COMPONENTS OF A BRAINSTEM NETWORK

2.1 OVERVIEW

Nervous systems control and regulate all aspects of an animal's life through complex interactions between electrical and chemical signals. In neuroethology, we are interested in decomposing complicated nervous systems into simpler components, in order to understand how animals generate behaviour. This goal requires finding important features and appropriate levels of detail to allow explanation of interesting aspects of the system, whilst simplifying less significant details to a suitable level of abstraction.

Almost 250 years ago, Galvani observed that electric current applied to frog legs caused them to twitch [Bresadola, 1998]. Later, at the end of the 19^{th} century, Cajal applied stains to the brain and found individual neurons with diverse morphologies which he proposed acted as discrete units in the nervous system [Ramón y Cajal, 1909]. It was shown that individual neurons are electrically excitable and that activity in one can affect its neighbours (reviewed in [López-Muñoz and Alamo, 2009]). Fifty years later, Hodgkin and Huxley proposed that voltage-gated channels could regulate ionic currents flowing across an excitable membrane and demonstrated a simple model that reproduced the major features of neuronal action potentials [Hodgkin and Huxley, 1952]. Over the last fifty years, the improvement of electrophysiological techniques allowed the isolation of individual currents in recordings, and demonstrated the opening and closing of individual voltage-gated channels [Hille, 2001; Sakmann and Neher, 2010].

In parallel, two different classes of connections between neurons have been identified, electrical [Bennett, 1966] and chemical synapses [López-Muñoz and Alamo, 2009]. Electrical synapses are simpler and make a direct connection between the cytoplasm of the two neurons. Between two electrically coupled neurons, A & B, subthreshold voltage changes in A can have an almost immediate influence on voltage of B and the coupling can be unidirectional or bidirectional [Bennett, 1966]. By contrast, chemical synapses are more complex: they are unidirectional and the arrival of an action potential in the presynaptic neuron leads to a discrete change in the currents flowing in the postsynaptic neuron [López-Muñoz and Alamo, 2009].

In simple nervous systems, such as the hatchling tadpole, sensory input and motor outputs can be seen directly as electrical potentials in the corresponding sensory and motoneurons [Delcomyn, 1998]. A natural question to ask is - can observed behaviours in simple animals be described primarily in terms of electrical excitability of individual neurons and the interactions between them? If so, since flows of electrical currents are well understood mathematically, is it possible to build models of nervous systems at the level of currents flowing in neurons and synapses, which produce animal-like behaviours? In this chapter, I outline the mathematics of the models of the neurons and synapses which form the building blocks used in later chapters.

2.1.1 *The Hodgkin-Huxley model*

What causes neurons to exhibit non-linear responses to electrical stimulation? In the Hodgkin-Huxley (HH) model, the voltage across a neuron's membrane is dictated by the flow of sodium (Na), potassium (K), and leak (Lk) currents across its membrane (Fig. 2.1) [Hodgkin and Huxley, 1952; Squire et al., 2012]. In the model, these currents result from the movement of ions across the membrane, due to a *driving force,* which results from an imbalance in the intracellular and extracellular concentrations of these ions (Fig. 2.1). This driving force was modelled as a *constant voltage source* for each current, E_{Na}, E_K & E_{Lk} (Fig. 2.2). The membrane of the neuron itself is a thin bi-lipid layer, which does not allow any particles to across and acts as a capacitance (Fig. 2.1). The ions cross the membrane through channels, some of which can open and close.

These channels can be ion-specific and voltage-gated. These channels regulate the flux of ions, and are modelled as time-varying conductances, g_{Na}, g_K & g_{Lk} (Fig. 2.2).

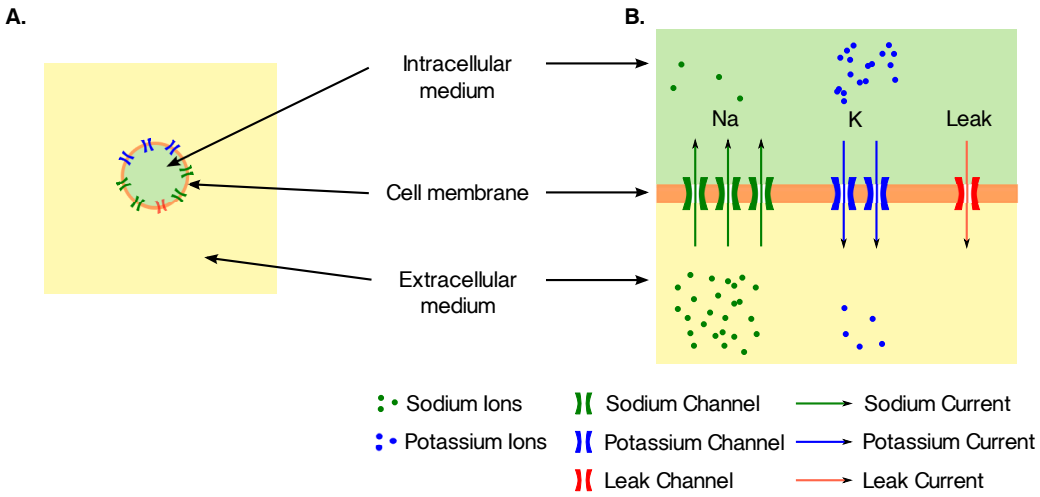


Figure 2.1 – Ion channels control current flow across an excitable membrane. **A.** We consider an isolated neuron in the extracellular medium. The neuron is enclosed in a bi-lipid membrane (orange), which is impermeable to particles and separates the intracellular medium from the extracellular medium. Embedded within the membrane are channels (green, blue & red). **B.** The ionic concentrations are assumed to be uniform inside and outside the membrane. The gradient in intracellular and extracellular concentrations causes diffusive forces on the sodium, potassium ions, which act as driving force for ions across the membrane. If the channels are open (as shown), this will result in a current flowing across the membrane.

In the HH model, the sodium and potassium channels are thought of as being composed of independent *gating particles*, each of which can be either activated or inactivated at a particular time[1]. Sodium channels have four gating particles (three m's and an h) and potassium channels have four (all n's). For a channel to be open, all of its gating particles must be activated. Since it is assumed that there are many individual channels distributed over the membrane (Fig. 2.1) and it is assumed that individual gating particles can either be activated or inactivated, the HH model represents the proportion of activated gating particles in the whole membrane as a continuous variable, and therefore the total conductance of a type of channel is calculated as the

1 The leak channel has no voltage-gated channels and is considered always open

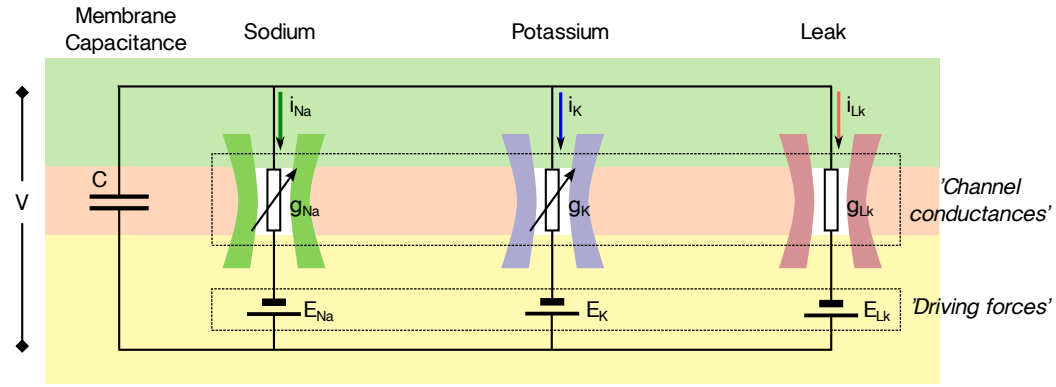


Figure 2.2 – Modelling ion flow across a neuron's membrane as an electrical circuit. The driving forces due to concentration gradients are modelled as voltage sources (E_{Na}, E_K & E_{Lk}) and the channels are modelled as resistors, which can vary in resistance as the channels open and close (g_{Na}, g_K & g_{Lk}). The membrane is very thin and acts as a capacitance.

product of the gating variables. The leak, potassium, and sodium currents in the HH model are given in Eqn. 2.2. Each of the gating particles, m, h & n can activate and inactivate over time and are modelled as state variables, whose evolution is dictated by voltage-dependent forward, $\alpha_x(V)$, and backward, $\beta_x(V)$, rate constants (Eqn. 2.3). The gating variables evolve according to a first order linear differential equation [Koch, 1999] (Eqn. 2.3) and since channels can be composed of heterogeneous gating particles (e. g. sodium: m and h) which activate and inactivate over different voltage ranges and over different durations, the interplay between the opening and closing of different gating particles can give rise to both inactivating (i. e. transient) and non-inactivating (i. e. intransient) currents [Hille, 2001]. A schematic of the HH model is given in Fig. 2.2[2], whose general solution is given by Eqn. 2.1. The evolution of the voltage across the membrane is calculated by combining Eqn. 2.1 with the equations governing specific currents, for example Eqns. 2.2 & 2.3.

2 Since the publication of the original model, variants of the HH model have been used to described behaviours in other types of neurons. I use *"Hodgkin-Huxley model"* to refer to the original model and parameters given in [Hodgkin and Huxley, 1952], and *"Hodgkin-Huxley-type model"* to refer to any model in which the membrane voltage is calculated from currents flowing through channels across the membrane, in which the conductances of the channels vary over time determined by voltage-dependent opening and closing rate constants.

$$C\frac{dV}{dt} = \sum_{x \epsilon \substack{membrane \\ currents}} i_x = i_{Lk} + i_{Na} + i_K \tag{2.1}$$

$$\begin{aligned} i_{Lk} &= \bar{g}_{Lk} \times (V - E_{Lk}) \\ i_{Na} &= \bar{g}_{Na} \times (V - E_{Na}) \times m^3 h \\ i_K &= \bar{g}_K \times (V - E_K) \times n^4 \end{aligned} \tag{2.2}$$

$$\frac{dx}{dt} = \frac{x_\infty(V) - x}{\tau_x(V)} \qquad x_\infty(V) = \frac{\alpha_x(V)}{\alpha_x(V) + \beta_x(V)} \qquad \tau_x(V) = \frac{1}{\alpha_x(V) + \beta_x(V)} \tag{2.3}$$

In general, a voltage describes the difference in electrical potential energy between two points [Halliday et al., 2004]. In neurons, the voltage across the membrane is the difference between intracellular and extracellular potentials rather than an absolute value. In the original HH papers the values of neuronal voltage are given relative to the resting potential (i. e. $0\,mV$ at rest). In this thesis, I use the convention of measuring all voltages relative to the extracellular medium, i. e. the extracellular medium is considered to be ground ($0\,mV$). The term 'neuron's voltage' is used to mean 'the intracellular voltage relative to the extracellular voltage', i. e. at $0\,mV$ the intracellular and extracellular mediums are isopotential.

2.1.2 *Roles of different channel types*

In the HH model, a neuron's membrane voltage is negative at rest and the sodium channels are closed. When the neuron is given a depolarising input, its voltage increases, causing these channels to open and resulting in an inward current which further depolarises the neuron. If a neuron's voltage reaches the *firing threshold*, a positive feedback loop begins which results in the membrane voltage shooting briefly towards the reversal potential of sodium. However, the sodium current quickly inactivates, causing the membrane voltage to return to rest.

Across animal species, many inward (e. g. sodium, calcium) and outward currents (e. g. potassium, chloride) both inactivating and non-inactivating, operating over different voltage ranges and time scales have been characterised [Hille, 2001]. Some currents can be modulated by particular intracellular and extracellular substances, including intracellular and extracellular concentrations of ions of other transmembrane currents. If sodium currents are sufficient to cause electrical excitability, and action potentials, why such a diverse range of currents are observed in general an open question. The interactions between these currents, their ionic concentrations and membrane voltages make neuronal membrane responses highly non-linear, and although many currents have been characterised electrophysiologically, in most cases their specific roles have been hard to delineate. It is thought that the particular cocktail of channels that regulate the current flows in a neuron allow it to specialise for a task, for example pacemaker cells in the heart, require the interplay of at least four currents for producing a slow, heartbeat frequency [Hille, 2001; Noble, 2004] and the dependence of current flows on intracellular and extracellular modulators allows cellular, and consequently network, dynamics to be shaped in response to different inputs.

A more diverse range of potassium currents have been observed than other currents. Briefly, delayed rectifier currents have been proposed to keep action potentials short and help rehyperpolarise the neuron after firing [Hille, 2001], inactivating potassium currents can reduce the rate of firing by spacing action potentials during repetitive firing [Connor and Stevens, 1971], calcium-dependent potassium currents have been implicated in long hyperpolarising pauses, frequency adaptation, and burst termination [Hille, 2001], and hyperpolarisation-activated currents have been implicated in pacemaker and rebound firing [Luthi, 1998; Hille, 2001]. Fewer types of inward currents have been observed. Within a single neuron, different types of sodium channels have been observed in the soma and the axon hillock, and those in the hillock, where action potentials may initiate, are often more excitable [Schmidt-Hieber and Bischofberger, 2010]. High-voltage–activated calcium currents are often non-inactivating, and can lead to longer spike durations, for example in the heart [Hille, 2001]. In general, the flow of calcium ions is thought to have significance beyond directly affecting voltage because other currents can be dependent on the level

of intracellular calcium, and calcium ions also affect other cellular processes as an intracellular secondary messenger [Hille, 2001].

2.1.3 *Multicompartmental modelling*

The individual currents flowing in neurons can be isolated and characterised using pharmacological and voltage-clamp experiments. Models of new voltage-gated membrane currents can be incorporated into the original HH formulation by adding more terms to the right-hand side of Eqn. 2.1. The original HH model assumed that the membrane was isopotential. However, neurons can have complex morphologies and non-uniform channel distributions. In multicompartmental modelling, the model neuron is divided into smaller compartments to allow different voltages at different locations. This allows us to capture the propagation of an action potential along an axon for example. A common approach in multicompartmental modelling is to divide a neuron into a set of cylindrical compartments, where each compartment has its own voltage and set of state variables (Fig. 2.3). The current flows, I^i, across the membrane are calculated separately for each compartment, (for example using Eqns. 2.1, 2.3 & 2.2; Fig. 2.2). These smaller compartments are electrically connected together via resistances, which are calculated based on the intracellular resistivity, R_i, of the neuron, and the surface area that connects the two compartments (Fig. 2.3). The mathematics is described further in Koch [1999]; Carnevale and Hines [2006].

Unfortunately the HH model contains a lot of parameters, is complex to fit and in many cases good experimental data is lacking about the current kinetics and distributions of channels over the neuron. An alternative approach, often used in modelling the behaviours of large networks of neurons, is to treat the neurons as *blackboxes* and approximate the voltage response of a neuron in response to stimulation as a simple abstract model (e. g. [Morris and Lecar, 1981; Izhikevich, 2003, 2007]). These reduced models are able to reproduce similar firing patterns to those observed physiologically, but have fewer state variables, which makes them analytically tractable, and involve fewer parameters, which makes them easier to fit. In this work, I am interested to understand the effects of axo-axonic electrical coupling within the population of neur-

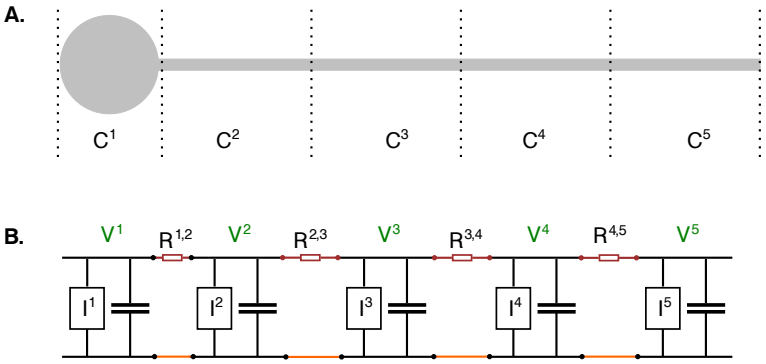


Figure 2.3 – Extension of a single compartment Hodgkin-Huxley type model to use multiple compartments. **A.** The neuron is divided into smaller compartments, (C^1,C^2, ...), and within each compartment the neuron membrane is considered isopotential. **B.** Each compartments C^i has a voltage, V^i, and a set of state variables for each channel (e. g. m^i, h^i, & n^i). Within each compartment, membrane currents flow across the membrane, I^i, (for example sodium, potassium and leak flow as Fig. 2.2) and currents also flow between adjacent compartments. The current flow between two compartments C^i and C^{i+1} are determined by the length of the compartments, surface area of the face connecting the compartments and the cytoplasmic resistivity, R_i which modelled as a resistance, $R^{i,i+1}$(maroon). The extracellular potential is considered isopotential (orange).

ons and these types of reduced models are not suitable because they do not capture multicompartmental current flows.

2.1.4 *Membrane currents in Xenopus tadpoles*

Xenopus laevis tadpoles have been used for investigating neuronal membrane ion flows and their functional effects [Dale, 1993, 1995b; Winlove and Roberts, 2011]. Some advantages of performing electrophysiology in this animal model are that the firing patterns of different neuron types are well characterised during swimming and struggling *in situ*, only a limited number of types of channels have ever been observed, the dendrites of the neurons are often electrotonically compact making the neurons easier to space-clamp [Wolf et al., 1998] and since the soma and dendrites are assumed to be isopotential, there is no need to consider any effects resulting from different distributions of the channels over these regions when constructing models.

Moreover, since the animal has only been alive for a short time, it is hoped that its regulatory mechanisms are simpler than those of older animals.

Voltage-clamp recordings have been made from unidentified dissociated neurons from the tadpole spinal cord and models made of the predominant currents, which included an inactivating sodium current, a high-voltage-activated calcium current and a fast and a slow non-inactivating potassium current [Dale, 1995a,b]. More recently the effects of potassium currents on the firing properties of neurons were investigated using voltage-clamp recordings from two classes of spinal neurons, RB and Dorsolateral (DL) neurons [Winlove and Roberts, 2011, 2012]. RB neurons, like dINs, only ever fire a single action potential in response to step current injection during *in situ* recordings. This is in contrast to most types of neurons recorded in *Xenopus* tadpoles (including DL neurons) which fire repetitively. Two families of potassium currents were identified that were sensitive to different pharmacological blockers: slower currents that were sensitive to Tetraethylammonium (TEA) and faster currents that were sensitive to 4-Aminopyridine (4AP). Using voltage-clamp recordings, it was found that repetitively firing DL neurons have more fast potassium currents, and less slow potassium currents than RB neurons, and by applying the blockers to the neurons *in situ* it was suggested that the ratios of these potassium currents could affect whether a neuron fired once or multiply in response to step current injection.

Direct voltage-clamp data about the currents present in dINs is currently lacking. The dINs are electrically coupled, which prevents good space clamping in whole-cell patch recordings [Li et al., 2009]. I implemented existing models of calcium and sodium channels based on Dale [1995a] and used new voltage-clamp data from Winlove and Roberts [2012] to build new models of potassium currents from *in situ* recordings, described in the next sections. All data used for modelling potassium channels is courtesy of Dr C Winlove.

2.2 MODELS OF MEMBRANE CURRENTS

2.2.1 *Slow potassium currents*

Voltage-clamp recordings from RB neurons, in which sodium, calcium currents and fast potassium currents were blocked, show a slow, non-inactivating potassium current which fully activates after ~30 ms (Fig. 2.4) [Winlove and Roberts, 2012]. The current shows no obvious inactivating component, so was fitted to a simple HH type conductance model of the form $i = \bar{g} \times (E_k - V) \times n^\gamma$, where $\bar{g}$ is the open conductance of the channel, n is a gating variable governed by forward and backward rate constants and γ is an integer constant to be found. E_k is the reversal potential of the potassium current which was determined experimentally to be $-81.5\,mV$.

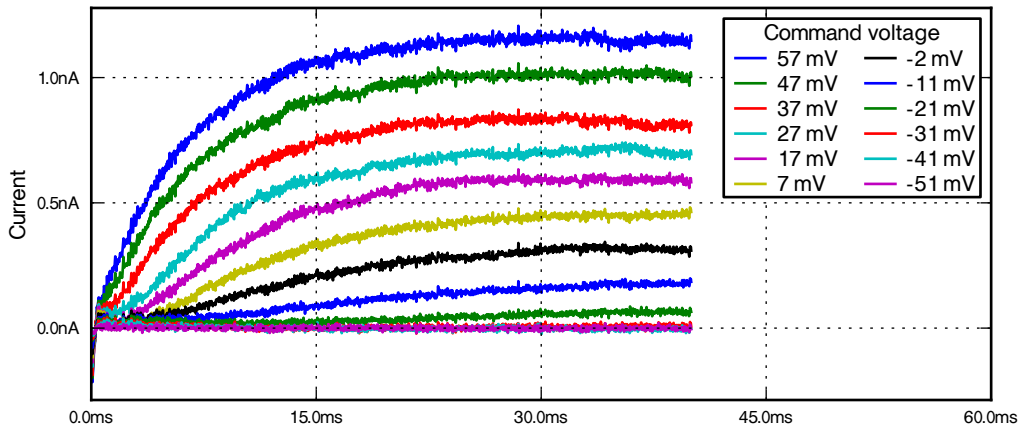


Figure 2.4 – An example set of voltage-clamp recordings showing the *in situ* slow potassium current in RB neurons as the command voltage is varied in steps from -51 to $57\,mV$. (The current monotonically increases with voltage and currents corresponding to command voltages less than $-30\,mV$ are all close to $0\,nA$) The other currents were blocked pharmacologically: sodium (using TTX), calcium (using amiloride and nimodipine) and fast potassium (4AP). Data from [Winlove and Roberts, 2012].

I assumed that both the forward and backward rate constants would take the form $(A + BV)/(C + \exp((D + V)/E))$ and initial attempts were made to directly optimise coefficients A, B, C, D & E. The sum of the squared distance between the cur-

rent traces calculated from the model and the experimental data was used as a cost function for a gradient-descent optimising routine. This was unsuccessful, since the routine would get stuck in local minima. A more successful approach was to fit the model in stages, finding smooth curves over the range of command voltages at each stage. Although this process is described linearly, in practice it was iterative.

The currents reached a steady state after ~30 ms, so steady state conductance densities for each command voltage level were calculated by taking the mean of the current traces between 30 and 45 ms (Fig. 2.5A) and dividing this by the driving force. The HH model assumes that the gating variables, (i. e. n), vary between zero (closed) and one (open), so the conductance ($g = \bar{g} \times n^{\gamma}$) saturates at high values of V. Unfortunately, the conductance did not saturate over the range of command voltages tested (up to 60 mV), which made it impossible to directly derive a value for the open conductance, $\bar{g}$ (Fig. 2.5B). Instead I initially estimated that for each data-set, the conductance at the highest value of command voltage was the open-conductance of the channel and fitted a model of the channels opening and closing kinetics based on this. Later, after the forward and backward rate constants had been estimated, I renormalised this open conductance based on the shapes of the steady-state curves for the gating variable. Five datasets were used in this fitting process [Winlove and Roberts, 2012].

The proportion of channels open relative to the maximum observed conductance in the steady state were calculated, and time constants were calculated by fitting decaying exponentials to each of the current traces (not shown). In the HH model, γ is an integer which represents the number of gating variables and controls the shape of the activation of the currents. Simple gradient descent was used to optimise fits of decaying exponentials raised to the power of γ = 1, 2, 3 & 4 for each current trace, using a cost function that minimised the sum of the squared distance between the measured and calculated values of a trace over the transient phase of the current (the first 30 ms) . I found that a value of $\gamma = 2$ gave good fits across the range of command voltages and datasets (Fig. 2.6). This process gave us time constants for the command voltages, $n_{\tau}(V)$ and also allowed us to calculate the steady states, $n_{\infty}(V)$ for each of the command voltages. When the values of steady state conductance were low, (i. e.

$V < -20\,\mathrm{mV}$), the traces were shallow and the time constants were less well defined (Fig. 2.5A).

$n_\infty(V)$ and $n_\tau(V)$ relate directly to the forward and backward rate constants $\alpha_n(V)$ and $\beta_n(V)$ (Eqn. 2.3). The forward rate constant could be fit to the form $(A + BV)/(C + \exp((D + V)/E))$ and the backward rate constant as a straight line. I built a tool which allowed us to interactively adjust the values of the rate constants at particular command voltages and see the resulting effects on a simulation of a voltage-clamp experiment on a model neuron (see Section 3.6.4). I paid more attention to matching the currents at the normal operating voltages of CPG neurons (from -60 to $30\,\mathrm{mV}$) than to more extreme voltages ($> 40\,\mathrm{mV}$). Since the backward rate constant is linear, rather than asymptotic to zero, at very high voltages ($V > 70\,\mathrm{mV}$), the proportion of open channels does not saturate. However, since these values are above the reversal potential for sodium, this situation is unlikely to occur in this model. The final model is given in Eqns. 2.4 and Fig. 2.6. Note that the denominator of $\beta(V)$ effectively evaluates to the constant $\sim 1.62 + \exp(2/3)$ because $200e3 \gg V$, however I use this form for consistency with the other channel models.

$$\alpha_n(V) = \frac{0.46 + 8.2e-3 \times V}{4.59 + \exp\left(\frac{-4.21 + V}{-12.0}\right)} \qquad \beta_n(V) = \frac{0.0924 - 1.354 \times V}{1.62 + \exp\left(\frac{200e3 + V}{300e3}\right)}$$

$$i = \bar{g}_{kslow} \times (E_{rev} - V) \times n^2 \tag{2.4}$$

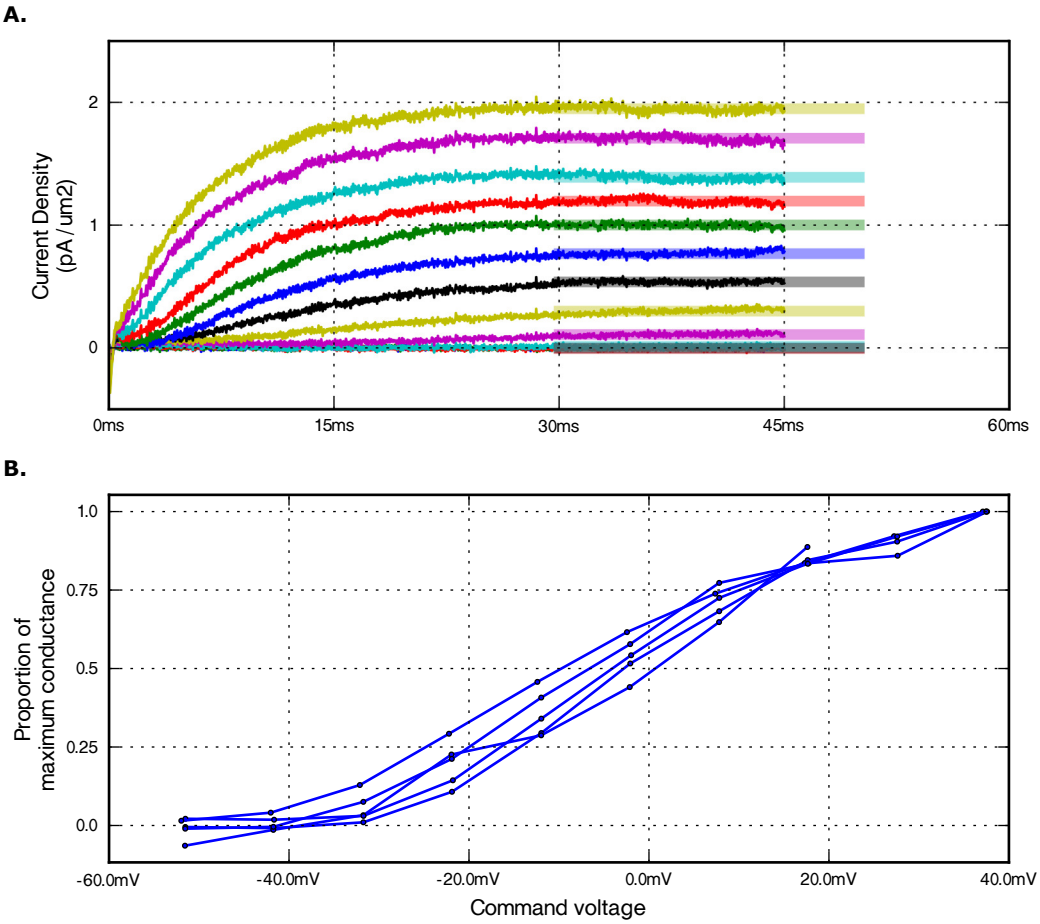


Figure 2.5 – **A.** The steady-state currents were calculated from the mean current flowing in a window between 30 and 45 ms as a function of command voltage for each dataset (shown as thicker, transparent lines). **B.** The maximum conductance did not saturate over the range of command voltages tested. Therefore to estimate the proportion of channels open at a particular command voltage, the calculated conductances were normalised so the maximum for a dataset was one.

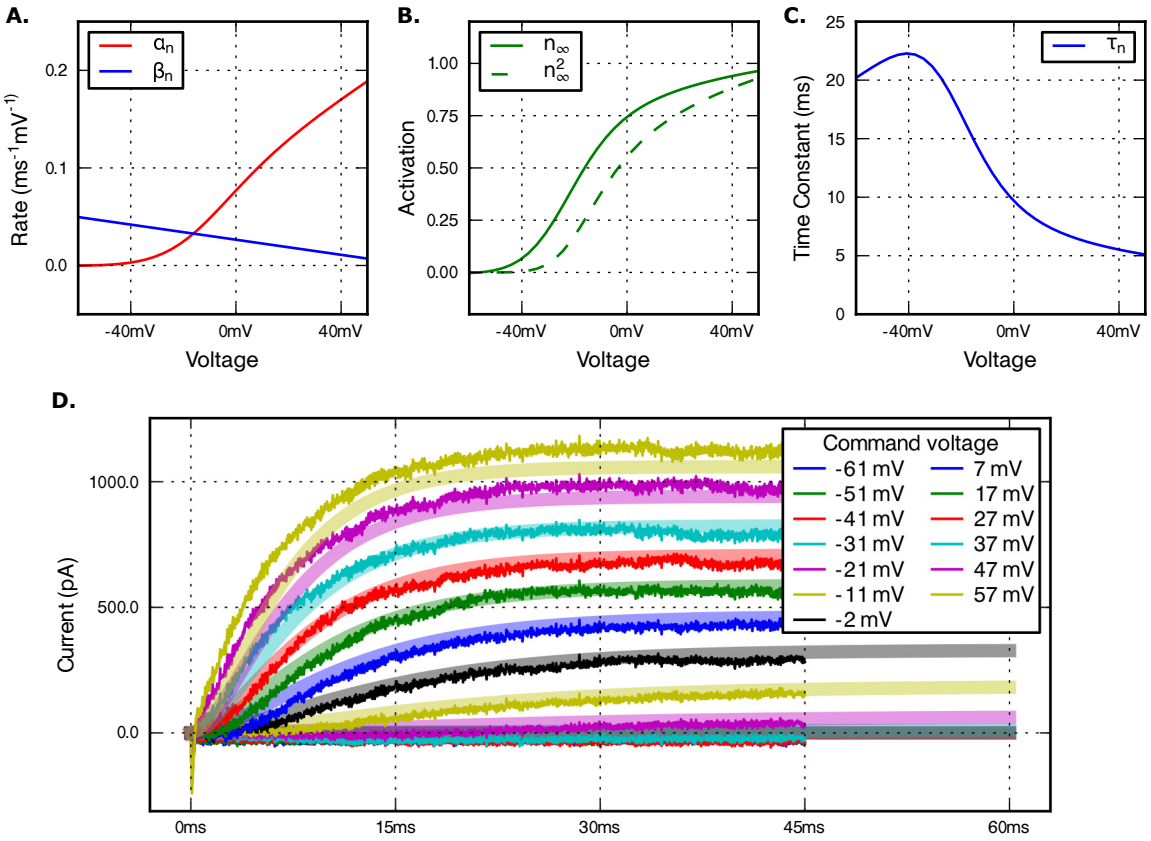


Figure 2.6 – The kinetics of the model slow potassium channel and a comparison against voltage-clamp recordings. **A.** The forward and backward rate constants. **B.** The steady state activation of an individual gating variable (n_∞: solid-line) and of the channel (n_∞^2: dashed line). **C.** The time constants of the gating variable (n). **D.** A comparison of the currents from voltage-clamp simulations of the model channel (thicker, transparent lines) with experimental recordings (thinner, solid lines) at different command voltages.

2.2.2 *Fast potassium currents*

The analysis of faster potassium currents was more complex. During voltage-clamp experiments, currents are blocked by applying a combination of pharmacological blockers in order to isolate the current of interest. However the blockers are imperfect, have side-effects, and can have similar selectivity to different channels which makes it hard to isolate individual currents [Hille, 2001]. The current traces from voltage-clamp experiments of RB neurons in which blockers for sodium, calcium and slow potassium current were used suggest the presence of at least two fast potassium currents with different time-courses: one current activates with a time constant of ~4 ms (Fig. 2.7A) and another current activates with a time constant of less than 2.5 ms (Fig. 2.7B) [Winlove and Roberts, 2012]. The records suggest that the slower component is non-inactivating, but is it more difficult to draw conclusions about the faster component, since, for example, it could be the combination of a fast inactivating component and a slower non-inactivating component. Voltage-clamp recordings from similar experiments in DL neurons show a current with a characteristic *bump* at onset (Fig. 2.7C) and one hypothesis is that this current is the combination of a fast, inactivating potassium current (e. g. an A-type current [Connor and Stevens, 1971]) and a slightly slower non-inactivating component (see below). Together, these recordings suggest the presence of possibly two or three fast, non-inactivating potassium currents and at least one very fast, inactivating potassium current.

In both the RB and DL recordings, the fast potassium currents activate very quickly; the fastest component activates within ~1.5 ms and the slowest within 5 ms. The normalised steady state currents and conductances of fast potassium currents from both RB and DL neurons suggested that the non-inactivating components of the recorded currents could contain either slightly different current components or different ratios of these current components (Fig. 2.8). The time courses for activation of the two hypothesised, non-inactivating fast potassium channels are both within the range of 2 to 4 ms and the levels of steady state activation across the command voltages were similar. Since we do not have voltage-clamp recordings from dINs, I treated these as a single current and I assumed that the dINs would have a single, fast, non-inactivating

potassium current. I based the kinetic model on data from DL neurons, since the data was less noisy (Fig. 2.8 green).

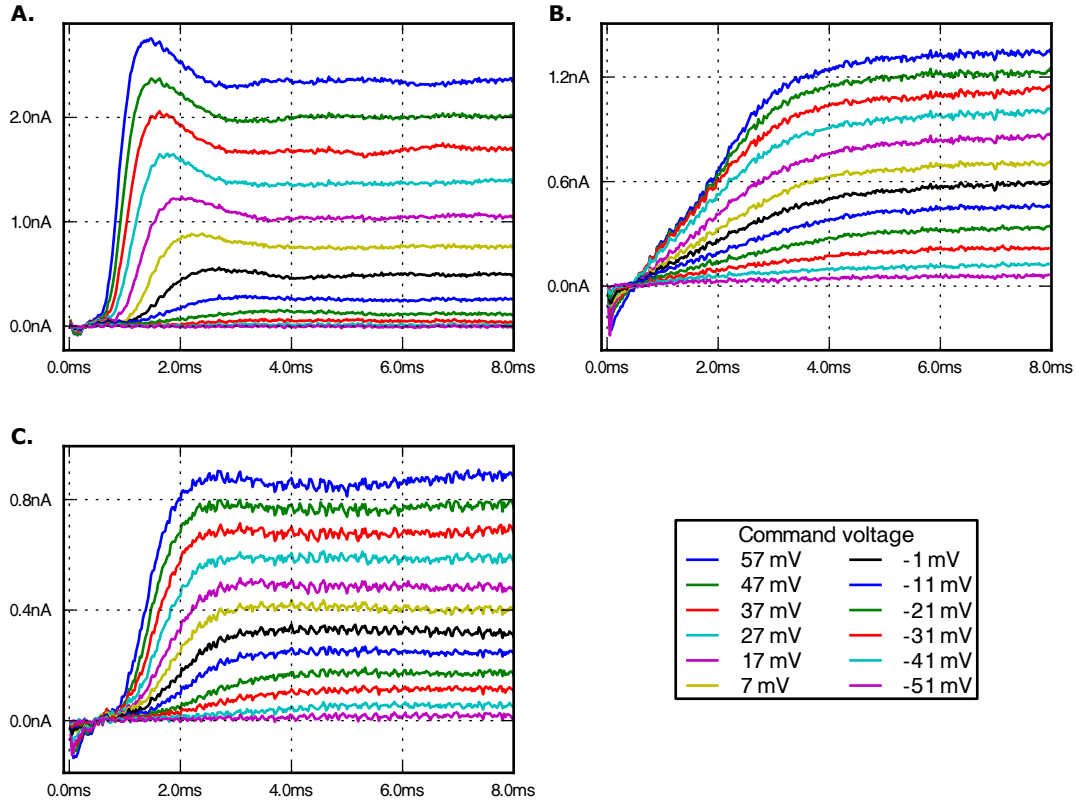


Figure 2.7 – Examples of the fast potassium currents seen in RB and DL neurons. **A, B.** Fast currents recorded from RB neurons. **C.** Fast current recorded in DL neuron.

Modelling the fast non-inactivating potassium current

Although there was good data for the steady-state conductances of the channels, the existence of inactivating currents makes it much more difficult to fit the time constants. The time constants and steady-state variables are linked because they are both functions of the forward and backward rate constants, which I assumed would be smooth functions. I assumed that a fast inactivating potassium current was responsible for the initial bump (Fig. 2.7A) and matched the current model to have a time constant so that it it would rise to meet the steady-state value after the bump. I built a model of this fast non-inactivating potassium current using the same strategy as the slow non-inactivating potassium current, based on four sets of voltage-clamp data. A

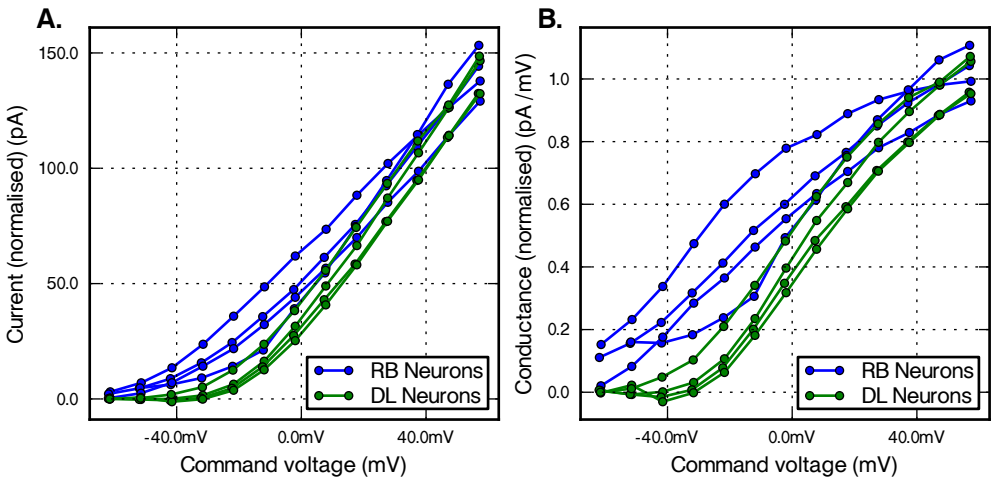


Figure 2.8 – The steady-state currents and conductances of fast potassium currents seen in RB and DL neurons. **A.** The steady-state currents of the neurons. The magnitudes of the currents varied across neurons so were manually normalised for comparison. **B.** The steady-state conductances calculated from the normalised steady-state currents and driving forces.

value of $\gamma = 4$ was found to give a good fit. The fitting of these channels was again an interactive process. The equations for the forward and backward rate constants are given in Eqns. 2.5 (Fig. 2.9).

$$\alpha_n(V) = \frac{5.06 + 0.0666 \times V}{5.12 + \exp\left(\frac{-18.4 + V}{-25.4}\right)} \qquad \beta_n(V) = \frac{0.5}{\exp\left(\frac{28.7 + V}{34.6}\right)}$$

$$i = \bar{g}_{kslow} \times (V - E_k) \times n^4 \tag{2.5}$$

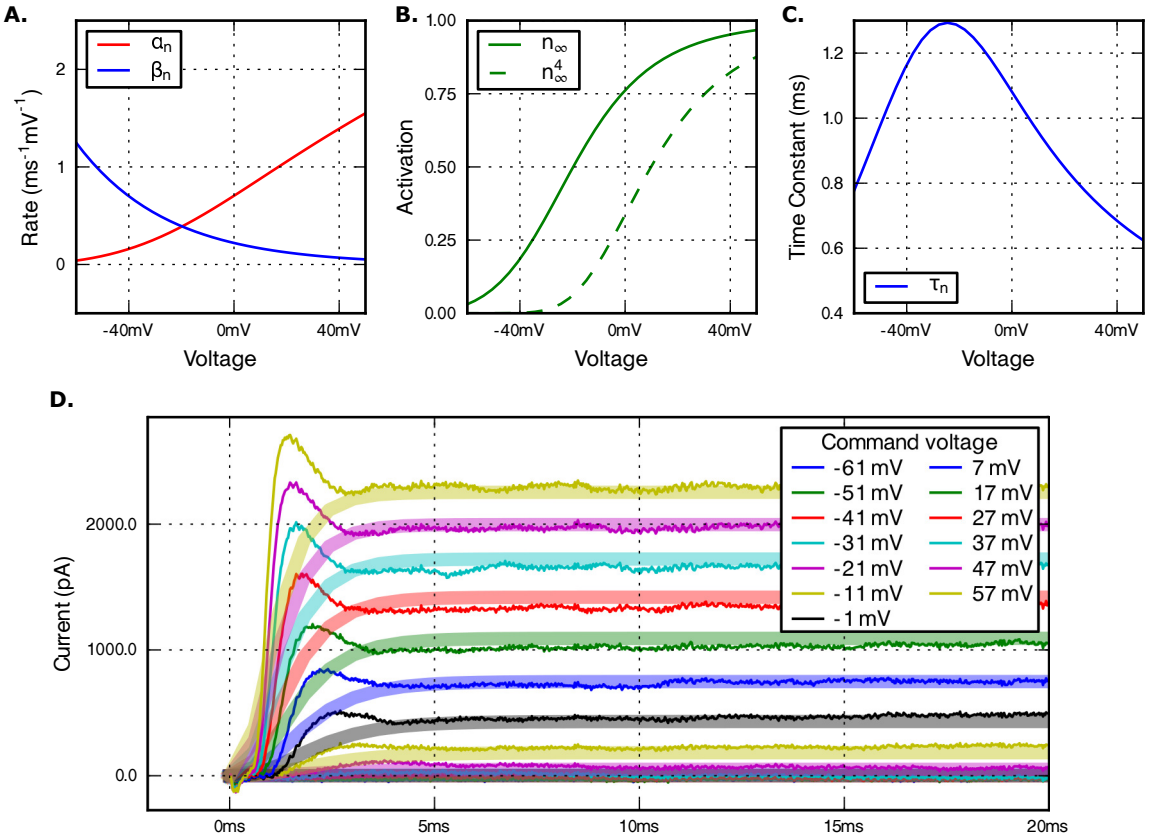


Figure 2.9 – The kinetics of the model fast potassium channel and a comparison against voltage-clamp recordings. **A.** The forward and backward rate constants. **B.** The steady-state activation of an individual gating variable (n_∞: solid-line) and of the channel (n_∞^4: dashed line). **C.** The time constants of the gating variable (n). **D.** A comparison of the currents from voltage-clamp simulations of the model channel (thicker, transparent lines) with experimental recording (thinner, solid lines) at different command voltages.

Inferring an inactivating fast potassium current

Inactivating outward currents have been proposed to play a role in limiting the firing frequencies of neurons in other systems [Connor and Stevens, 1971; Hille, 2001]. A one-sided population of dINs in the presence of NMDA fires repetitively at low frequencies (20 to 30 Hz). I investigated whether the bump seen in DL voltage-clamp recordings could be the result of an inactivating and a non-inactivating current. I subtracted simulation traces of the fast potassium current from the original experimental recordings from DL neurons (Fig. 2.7A) to infer this current (Fig. 2.10).

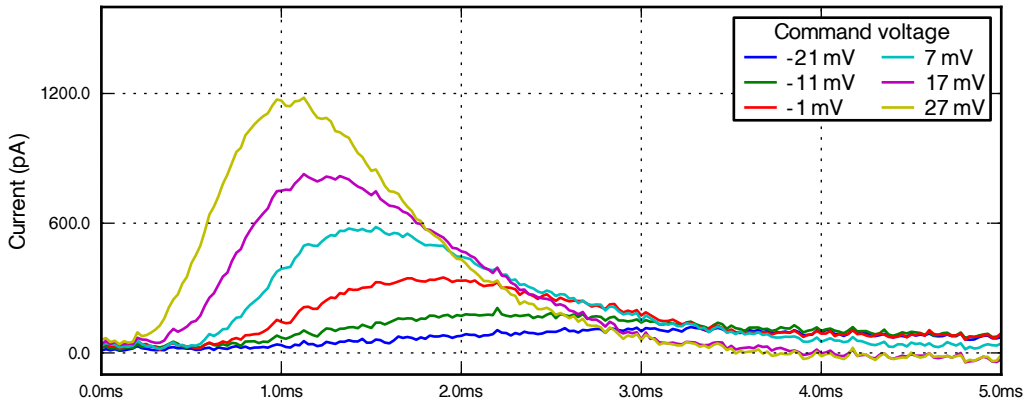


Figure 2.10 – The hypothesised very fast, inactivating potassium current in a DL neuron, inferred by subtracting the model of fast potassium current from experimental voltage-clamp recordings containing *bumps* (Fig. 2.7A).

The hypothesised inactivating potassium current is very fast and both activates and inactivates in under ~4 ms (cycle period of swimming ~40 ms). In order to fit this current I would need to find rate constants for both the activating and inactivating components of the model. However, since the inferred time course of inactivation of this current will depend strongly on the activation dynamics for the fast non-inactivating current, no further attempt was made to model it.

2.2.3 *Comparison of potassium currents with existing model*

The new models of fast and slow, non-inactivating currents are based on small datasets (fast: $n = 4$; slow: $n = 5$) and the fitting process involved manual intervention. Both fast and slow non-activating potassium currents have previously been modelled from voltage-clamp recordings in unidentified, dissociated neurons [Dale, 1995a]. Figures 2.11 & 2.12 show comparisons of the time-courses and proportion of channels open in the steady states in the existing *Dale-95*[3] model (as defined in [Dale, 1995b]) and the new *Hull-12* model (as defined in the previous sections). In the case of the fast channels models, the activation appears similar over the range of voltages from -60 to $0\,mV$ and begins to diverge at higher voltages (2.11A). The time constants were

3 There appears to be a mistake in β_2 for the slow-potassium current in Dale [1995a], so I assumed β_1 is used for all voltages in these graphs

also similar (0.5 to 2 ms) over the operating range of neurons. In the case of the slow potassium channel models, again the steady state activation curves are similar over the range of voltages from −60 to 0 mV (2.12A) and the time constants also vary from ~25 ms at low voltages (~−40 mV) to ~5 ms at higher voltages (> 20 mV). For both the fast and slow currents, the time constants in the models diverged at low voltages (< −30 mV) (2.11B & 2.12B) . In both cases, when modelling the currents, the time-courses were difficult to fit to the corresponding current traces because the steady state conductances of the channels are very low at these voltages (2.11A, green & blue dotted lines, 2.12A, green dotted & blue lines). To resolve this fitting problem, more experimental data is needed. The models that are built in the following chapters are highly non-linear, dynamical systems. In general, it is difficult to predict the effects of small changes , such as, the differences between the kinetics of the potassium channels of *Dale-95* and *Hull-12*, to these systems without resorting to simulation.

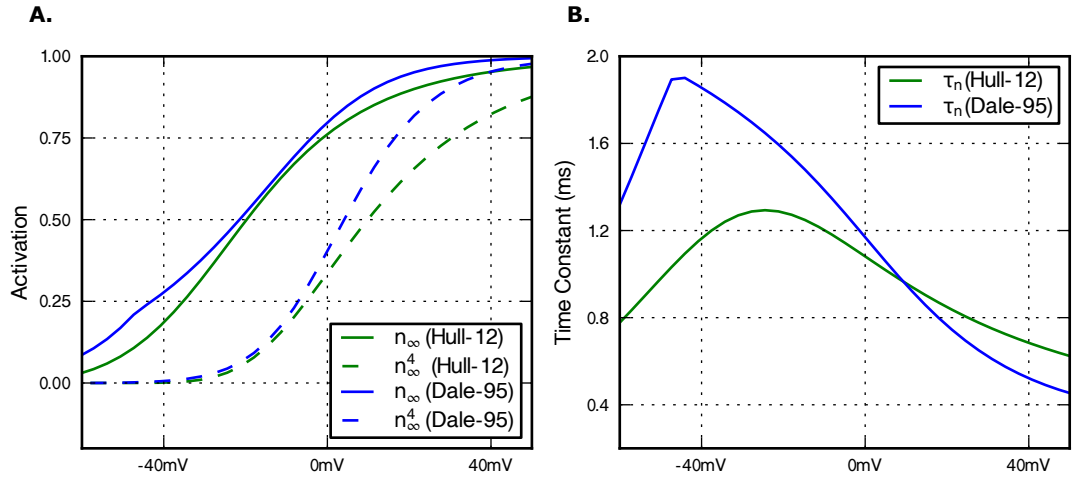


Figure 2.11 – Comparison of the *Hull-12* (green) and *Dale-95* (blue) fast, non-activating potassium channel models. **A.** The steady state values of a single gating particle (solid-line) and the channel (dashed line: product of all gating particles) **B.** The time constants of the two channel models.

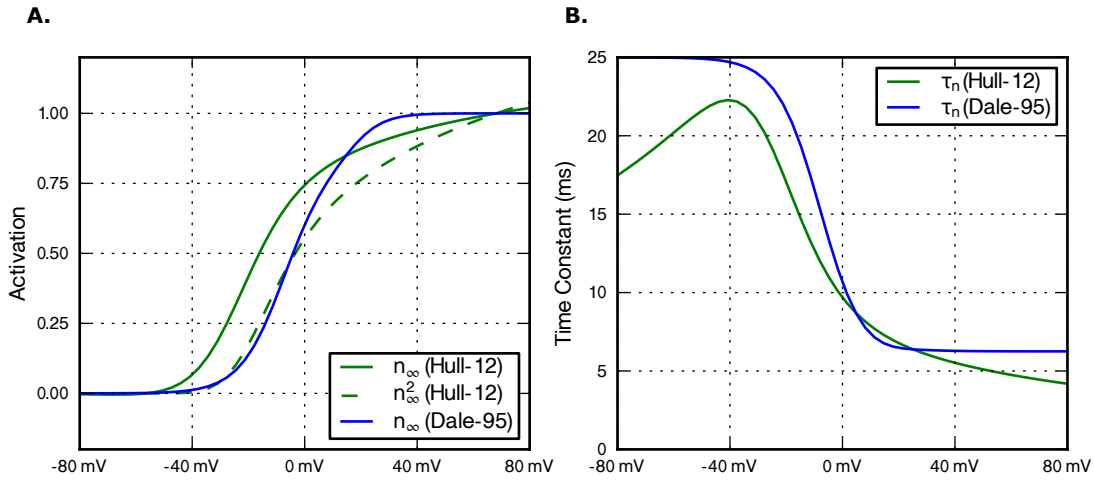


Figure 2.12 – Comparison of the *Hull-12* (green) and *Dale-95* (blue) slow, non-activating potassium channel models. **A.** The steady state values of a single gating particle (solid-line) and the channel (dashed line: product of all gating particles) (Note, the *Dale-95* channel model only uses a single gating particle, i. e. $\gamma = 1$) **B.** The time constants of the two channel models.

2.2.4 *Sodium currents*

Xenopus spinal neurons possess an inactivating sodium current, which when blocked prevents action potentials from firing [Dale, 1995b]. I used an existing model of this sodium current from Dale [1995a] (Fig. 2.13). This channel model has three activating gating particles, m, and one inactivating particle, h. The forward and backward rate constants and current equations are given in Eqn. 2.6.

$$\alpha_m(V) = \frac{8.67}{1 + \left(\frac{1.01 - V}{12.56}\right)} \qquad \beta_m(V) = \frac{3.82}{1 + \exp\left(\frac{9.01 + V}{9.69}\right)}$$

$$\alpha_h(V) = \frac{0.08}{\exp\left(\frac{38.88 + V}{26.0}\right)} \qquad \beta_h(V) = \frac{4.08}{1 + \exp\left(\frac{5.09 - V}{10.21}\right)}$$

$$i_{na} = \bar{g}_{na} \times (E_{na} - V) \times m^3 h \tag{2.6}$$

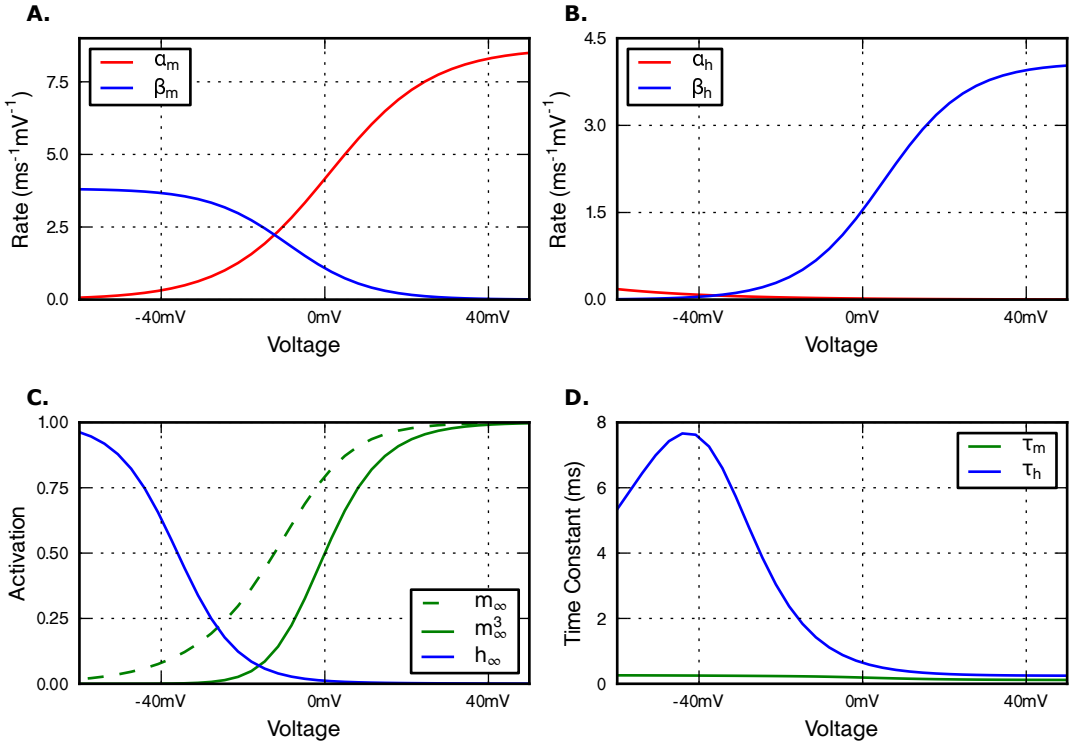


Figure 2.13 – Kinetics of sodium channel described in [Dale, 1995a]. **A, B.** The forward and backward rate constants for m and h. **C.** The steady state values and **D.** the time constants of the gating particles m and h at different voltages.

2.2.5 *Calcium currents*

In voltage-clamp recordings of *Xenopus* neurons, Dale observed fast activating, calcium currents which did not inactivate. These were modelled together as a single current [Dale, 1995a]. The calcium current differs from the other currents used in the model, because there is a much higher extracellular concentration of calcium ions than intracellular. In the model the flow of calcium current is described by the Goldman-Hodgkin-Katz (GHK) equation [Hille, 2001; Koch, 1999; Dale, 1995b] (Eqn. 2.7).

$$\begin{aligned} I_{Ca} &= P_{Ca} 2\nu F \frac{[Ca^{2+}]_i - [Ca^{2+}]_o e^{-\nu}}{1 - e^{-\nu}} \times m^2 \\ \text{where} \quad \nu &= \frac{2V_m F}{RT} \end{aligned} \tag{2.7}$$

In Eqn. 2.7, F is Faraday's constant (96485 C/mol), R is the ideal gas constant (8.3144 J mol^{-1} K), T is the absolute temperature (300 K) and $[Ca^{2+}]_i$ & $[Ca^{2+}]_o$ are the intracellular and extracellular calcium concentrations (100 nM and 10 mM) [Dale, 1995b].

In the original *Dale-95* model, the backward rate constant for calcium channels uses two functions for voltages each side of −25 mV, but there is a discontinuity at $V = -25$ mV (Fig. 2.14). In life, dINs can be sensitive to small step inputs (for example a single, fast-acting Inhibitory Post-Synaptic Potential (IPSP) to a depolarised dIN can cause a rebound spike). I adjusted the coefficients for the backward rate constant for $V < -25$ mV to remove this discontinuity. (Note that both the forward and backward rate constants for this channel are continuous, but the derivatives are not). The equations used for the modified forward and backward rate constants are given in Eqns. 2.8.

$$\alpha_m(V) = \frac{4.05}{1 + \left(\frac{15.32 - V}{13.57}\right)} \qquad \beta_m(V) = \begin{cases} \dfrac{1.24 + 0.093V}{\exp(10.63 + V) - 1} & \text{if } V < -25\text{mV} \\ \dfrac{1.28}{1 + \exp\left(\frac{5.39 + V}{12.11}\right)} & \text{if } V \geqslant -25\text{mV} \end{cases} \tag{2.8}$$

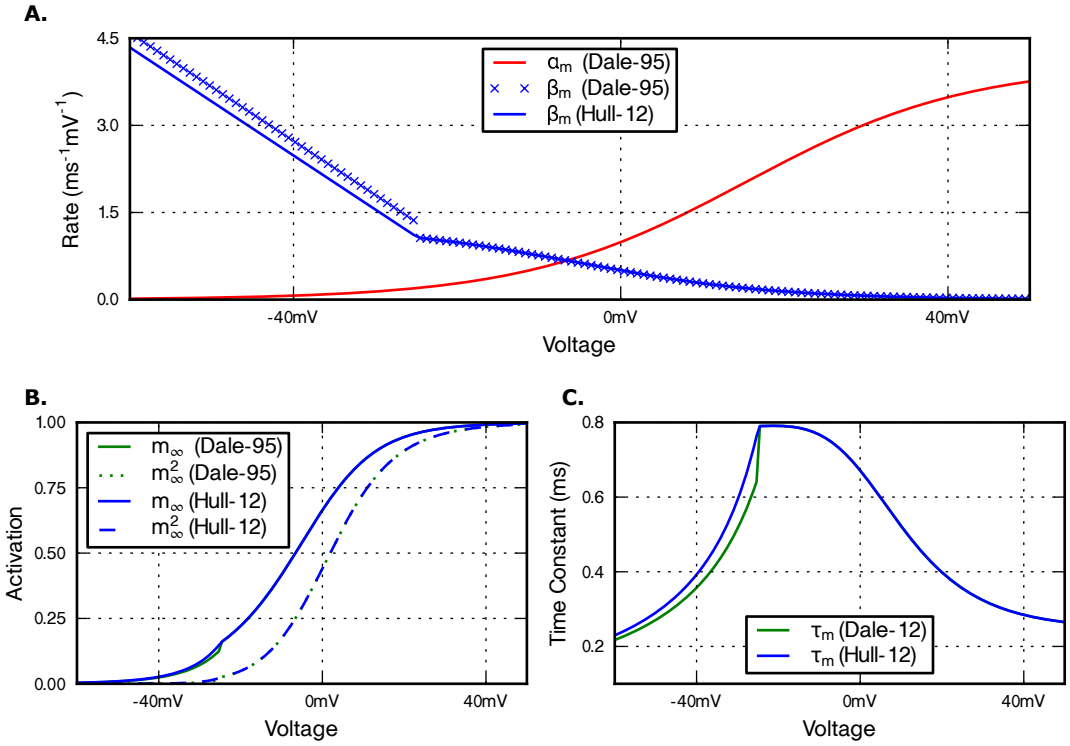


Figure 2.14 – Kinetics of the calcium channel. **A.** The *Dale-95* model contains a discontinuity in the value of beta at −25 mV (dark blue crosses) which was removed in the *Hull-12* model by shifting the function defining β for $V < -25$ mV down (dark blue line) **B.** A comparison of the steady state values and **C.** time constants for the *Dale-95* (dark blue) and the *Hull-12* (green) models.

2.3 MODELS OF SYNAPSES

2.3.1 *Chemical synapses*

One form of interaction between neurons in the nervous system is through chemical synapses [Squire et al., 2012] which are specialist junctions which form between two neurons. Briefly, when the voltage in the presynaptic neuron crosses a threshold, the presynaptic terminal releases neurotransmitter into the synaptic cleft. The neurotransmitter diffuses across the cleft and can bind to receptors on the postsynaptic neuron, which causes channels to open. This allows postsynaptic currents to flow and can cause an Excitatory Post-Synaptic Potential (EPSP) or an Inhibitory Post-Synaptic Potential (IPSP). After some time, the neurotransmitter is removed from the cleft and

receptors, and channels close [Squire et al., 2012]. Across nervous systems, a range of chemical synapses have been observed, which act over different time constants and use different neurotransmitters [Delcomyn, 1998]. Within a single synapse, multiple neurotransmitters can be coreleased from the presynaptic terminal [Hnasko and Edwards, 2012], and different types of receptors can exist at a single postsynaptic site, for example AMPA-receptors (AMPARs) and NMDARs in synapses onto *Xenopus* MNs [Dale and Roberts, 1985].

Synaptic connections between classes of neurons have been characterised in *Xenopus* tadpoles using paired recording experiments [Dale and Roberts, 1985; Dale et al., 1986; Roberts et al., 1987, 1988; Perrins and Roberts, 1995; Perrins et al., 2002; Li et al., 2004b, 2007b; Roberts et al., 2008; Soffe et al., 2009; Buhl et al., 2012]. In the tadpole, the predominant excitatory neurotransmitter is glutamate, which can bind to both fast-acting 2-Amino-3-(3-hydroxy-5-methyl-isoxazol-4-yl) Propanoic Acid (AMPA) receptors and slow-acting N-Methyl-D-aspartic acid (NMDA) receptors. At least two types of inhibitory synapse have also been observed. In the swimming CPG, cINs make contralateral glycinergic synapses onto other interneurons [Dale, 1985; Roberts et al., 1988] and in a pathway which terminates swimming, Mid-Hindbrain Reticulospinal neuron (MHR) neurons make Gamma-Aminobutyric Acid (GABA) ergic synapses onto the dINs (see Chapters 6 & 7) [Perrins et al., 2002]. In this thesis, for the sake of brevity, I will refer to a glutamatergic synapse with postsynaptic NMDARs as an *'NMDA synapse'* and a glutamatergic synapse with postsynaptic AMPARs as an *'AMPA synapse'*.

Models of synapses were built as presynaptic and postsynaptic components (see Section D.1.1). The presynaptic component produces an event when the presynaptic voltage crosses a threshold ($0\,mV$). After a delay, this event is transmitted to the postsynaptic component, which generates a transmembrane current in the postsynaptic neuron. A synaptic delay of $1\,ms$ between ipsilateral neurons and $2\,ms$ between contralateral neurons was used to represent conduction delays [Sautois et al., 2007].

In models of postsynaptic receptors, I used a conductance-based model in which the postsynaptic currents were calculated based on a synaptic reversal potential, E_{Syn}, and a time-varying conductance, g_{syn} which represents the activation of the receptor (Fig. 2.15, Eqn. 2.9). The receptor has opening and closing times which are described by the difference of two state variables, A & B. Each state variables evolves

according to a first-order differential equation which cause it to decay exponentially towards zero (Eqn. 2.9) and when a receptor receives a synaptic event, the values of A and B are increased by one. In order to simplify comparisons with physiology, I expressed the strengths of the synapses in terms of peak conductance g_{peak} and since the height of the curve described by a difference of two exponentials depends on the time constants, a normalising term, tc_{max}, was factored into the conductance equation, so that the maximum value of $(B - A)$ was one (Eqn. 2.9).

$$\begin{aligned}
g_{syn} &= g_{peak} \times \frac{B - A}{tc_{max}} \\
i_{syn} &= g_{syn} \times (V - E_{syn}) \\
tc_{max} &= \frac{(\tau_o \times \tau_c)}{(\tau_c - \tau_o)} \times \log\left(\frac{\tau_c}{\tau_o}\right) \\
t_{max} &= \exp\left(\frac{-tc_{max}}{\tau_c}\right) - \exp\left(\frac{-tc_{max}}{\tau_o}\right) \\
\frac{d}{dt}A &= \frac{-A}{\tau_o} \\
\frac{d}{dt}B &= \frac{-B}{\tau_c}
\end{aligned} \tag{2.9}$$

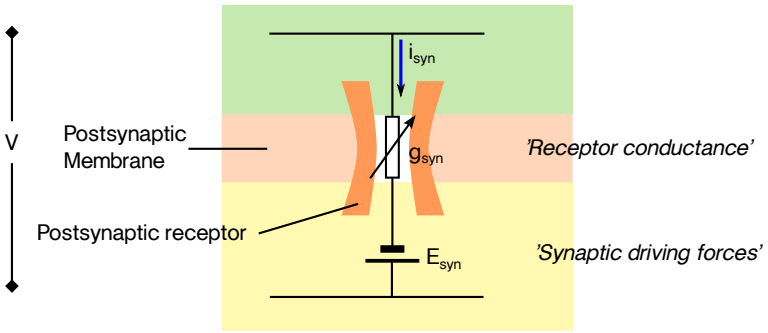


Figure 2.15 – The circuit diagram for the model of a postsynaptic receptor. Postsynaptic currents, i_{syn}, flow as a result of a driving force, $(V - E_{Syn})$ across the membrane at the receptor. The receptors open and close in response to synaptic events (not shown) and change the conductance, g_{syn}.

This formulation was used to model the postsynaptic receptors for both the excitatory (AMPAR & NMDAR) and inhibitory (Glycinergic and GABAergic synapses). The parameters values for the different synapses and their sources are summarised in

Table 2.1. Example traces of dIN to dIN, AMPAR and NMDAR-mediated EPSPs are given in Fig. 2.16.

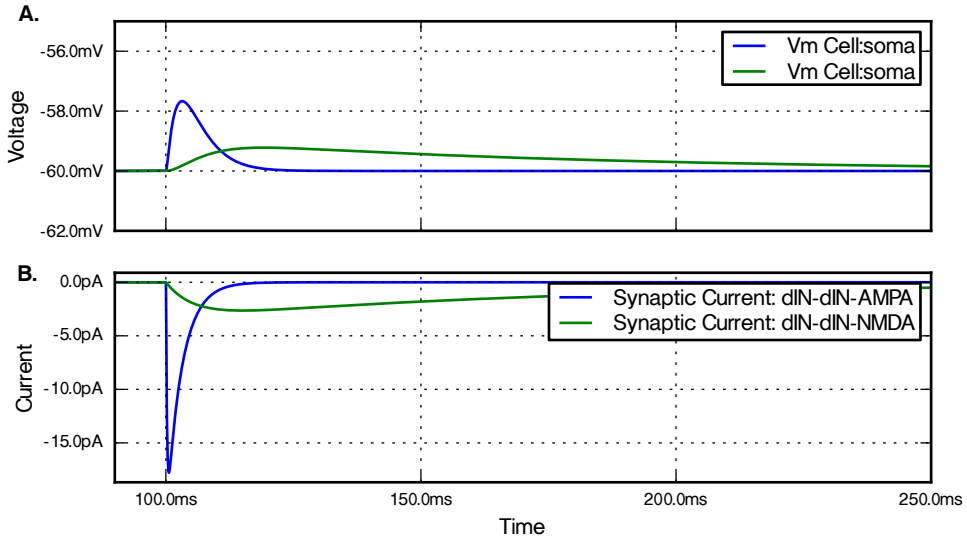


Figure 2.16 – Example traces from the synaptic models. The responses of a passive neuron with an input resistance of 300 MΩ (the same as that measured in a dIN) to single event at 100 ms to an AMPAR (blue) and an NMDAR (green) synapse. In both cases, g_{peak} = 300 pS. **A.** The responses of the membrane voltages. **B.** The current flow though each of the synapses.

In the case of the NMDA synapse, an additional term was used to describe the voltage dependency of the receptor due to magnesium block [Nowak et al., 1984] (Eqn. 2.10; more details about the NMDAR voltage dependence are given in Chapter 5). The parameters used were $\eta = 0.1\,\mathrm{mM}^{-1}$, $\gamma = 0.08\,\mathrm{mV}$ and $\left[Mg^{2+}\right]_o = 0.5\,\mathrm{mM}$, taken from [Sautois et al., 2007]. In simulations of zero extracellular magnesium, $vdep^{Mg2+}(V)$ was set to 1.0.

$$\begin{aligned} i_{syn} &= g_{peak} \times \frac{B - A}{tc_{max}} \times (V - E_{syn}) \times vdep^{Mg2+}(V) \\ vdep^{Mg2+}(V) &= \frac{1}{1 + \eta[Mg^{2+}]_O \times \exp(-\gamma V)} \end{aligned} \tag{2.10}$$

Table 2.1 – Parameters used in synapse models between pairs of neurons

SYNAPSE TYPE	PRESYNAPTIC	POSTSYNAPTIC	E_{syn}	g_{peak}	τ_{open}	τ_{close}
AMPA	dIN	dIN	$0\,mV$[a]	$300\,pS$[b]	$0.2\,ms$[a]	$3.0\,ms$[a]
	dIN	cIN	$0\,mV$[a]	$400\,pS$[c]	$0.2\,ms$[a]	$3.0\,ms$[a]
	tIN	dIN	$0\,mV$[a]	$250\,pS$[d]	$0.2\,ms$[a]	$3.0\,ms$[a]
NMDA	dIN	dIN	$0\,mV$[a]	$300\,pS$[b]	$5.0\,ms$[a]	$80.0\,ms$[a]
	tIN	dIN	$0\,mV$[a]	$300\,pS$[d]	$5.0\,ms$[a]	$80.0\,ms$[a]
Glycinergic	cIN	dIN	$-64\,mV$[a]	$300\,pS$[b]	$1.5\,ms$[a]	$4.0\,ms$[a]
GABA ergic	MHR	dIN	$-70\,mV$[e]	$2000\,pS$[f]	$1.5\,ms$[f]	$20\,ms$[f]

[a] Taken from Sautois et al. [2007].
[b] Taken from unpublished measurements from Soffe, based on dIN-dIN paired recording measurements. The recordings were done in zero magnesium.
[c] Derived from values in Sautois et al. [2007].
[d] Found by fitting to current-clamp traces from tIN - dIN paired recordings [Buhl et al., 2012].
[e] Taken from Koch [1999].
[f] Found by fitting to current-clamp recordings from MHR - dIN recordings [Perrins et al., 2002].

2.3.2 *Electrical synapses*

Another form of interaction between neurons is through electrical synapses. Gap junctions are regions between two adjacent neurons which contain specialist channels spanning the two membranes (Fig. 2.17A) [Simon and Goodenough, 1998]. In vertebrates, the individual channels are roughly cylindrical in shape and composed of two opposing hemichannels called connexons, one in each membrane (Fig. 2.17B). Each connexon is composed of a hexagon of 6 subunits called connexins. The channels have a diameter of approximately $1.5\,nm$, creating a pathway between the two neurons which allow small molecules to flow between the two neurons. Gap junctions can show selectivity to particular molecules and in some cases they can open and close, for example in response to the voltage across it [Qu and Dahl, 2002]. About 20 connexins have been identified and it is thought that particular configurations of connexins produce particular selectivity and also affect the dynamics of the channel. Gap junctions are not just observed in neurons but also play an important role beyond electrical coupling since they allow communication between cells by the passage of small molecules and they are particularly widespread during development [Simon and Goodenough, 1998]. The significance of gap junctions in other systems is discussed further in Chapter 4.

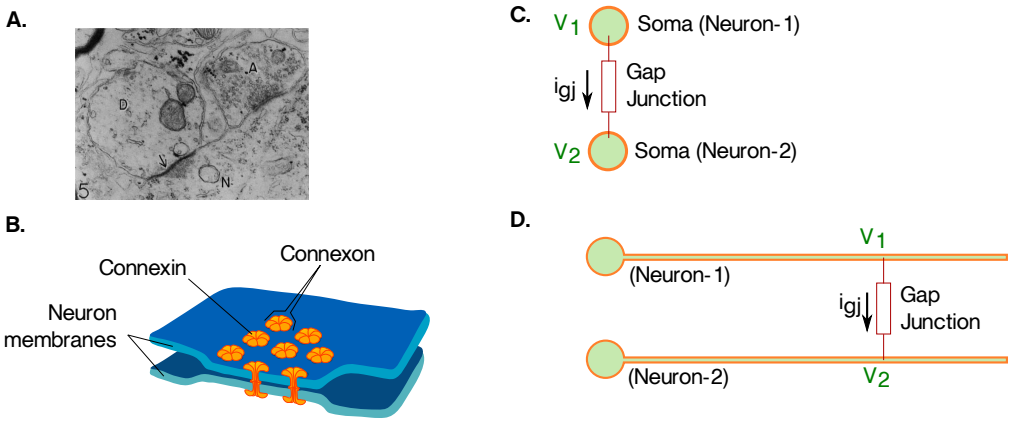


Figure 2.17 – Gap junctions between neurons. **A.** An electron micrograph of a small portion of the surface of a spinal motoneuron from the swim bladder nucleus of a toadfish. A dense junction, at the arrow, is formed between apposing plasma membranes of a dendrite (D) and of the neuronal soma (N). (Taken from [Pappas and Bennett, 1966]; Fig. 5). **B.** A cartoon of a gap junction between two neurons. At the gap junction, the membranes of the two neurons come into close proximity and are joined by apposing connexons in each membrane. **C, D.** Gap junctions are often modelled as simple resistances, allowing a direct current flow between the somata of each neuron (C), or on a neuronal process (D). In both cases, the current flow depends on the voltage difference at the points of connection membrane. In the case of ohmic gap junctions, the current flow is calculated using Eqn. 2.11.

Gap junction channels provide an electrical pathway between two neurons. In many cases, the gap junctions behave like simple resistances (Fig. 2.17C, D; Eqn. 2.11), although more complex current-voltage relationships have been recorded (see Chapter 4). In this thesis, all gap junction are modelled as ohmic, where the current flow through the junction, i_{GJ}, depends on the voltage of the presynaptic and postsynaptic neurons, V_{pre} and V_{post} and the resistance of the gap junction, R_{GJ}, as given in Eqn. 2.11.

$$i_{GJ} = (V_{pre} - V_{post})/R_{GJ} \tag{2.11}$$

When current is injected into a presynaptic neuron, it will cause a voltage change in that neuron, ΔV_{pre}, and it will also cause a voltage change in an electrically coupled neuron, ΔV_{post}. The strength of electrical coupling between two neurons is often quantified as a coupling coefficient which is given by $\Delta V_{post}/\Delta V_{pre}$. In the tadpole, the coupling coefficients between electrical coupled dINs were shown to be slightly stronger from caudal to rostral (9%) than rostral to caudal (8%) [Li et al., 2009]. Al-

though the gap junctions could be rectifying, a more likely explanation could be due to the input resistances of the neurons. In the simple case of two isopotential neurons with input resistances R_1 and R_2, coupled by a gap junction of resistance R_{GJ}, the coupling coefficients measured between the neurons, due to a current injection into the first neuron, K_{12}, and due to a current injection into the second, K_{21}, can be calculated by applying the Wye-Δ transform to the equivalent circuit (Eqn. 2.12) [Bennett, 1966].

It can be seen that differences in the input resistances of the electrically coupled neurons (R_1 and R_2) would be observed as differences in coupling coefficients. This could be due different sizes of neurons or the formation of different numbers of gap junctions to other neurons[4] for example.

$$\begin{aligned} K_{12} &= \frac{1}{1 + R_{GJ}/R_2} \\ K_{21} &= \frac{1}{1 + R_{GJ}/R_1} \end{aligned} \tag{2.12}$$

2.4 SPECIFIC METHODS FOR CHAPTERS

2.4.1 *Evaluating the effects of channel densities on firing behaviours*

In Chapter 4, parameter sweeps are used to investigate the effects of different channel densities on firing properties of an electrically coupled dIN. The kinetics of the channels (as described in Chapter 2) are summarised in Table A.1. Initial values for channel conductances, $\hat{g_X}$, where X is the channel type, were estimated based on input resistance and current densities measured in voltage-clamp recordings from CPG neurons (Table 2.2). The small number of channel types meant that parameter sweeps could be used to investigate the effects of channel densities on the firing properties of the neurons and the robustness of the neurons to noise. For each channel type, a set of scaling factors (K_X) was chosen (Table 2.2), resulting in parameter space of 288 possible combinations. Since the experimental recordings of dINs are from *in situ*

4 When neurons are electrically coupled to other neurons, their effective input resistance, for example in response to current injections, decreases (discussed further in Chapter 4)

recordings, in which the dINs are embedded in the electrically coupled network, the simulations were performed in the same scenario. For each parameter set in the parameter space to be investigated, a network of 30 electrically coupled dINs was created, in which each dIN had a conductance density of $\bar{g}_X = \hat{g_X} \times K_X \times n$. Normally distributed variability was introduced with a noise term, $n \sim \mathcal{N}(\mu = 1.0, \sigma = 0.1)$, which was calculated separately for each model dIN, for each channel, and for each simulation.

Table 2.2 – Conductance densities of membrane channels used in the dIN parameter sweep and final model.

CHANNEL	REV. POTENTIAL	BASE CONDUCTANCE	VALUES OF K_x TESTED	K_x USED
Calcium (Ca)	-	$\hat{p_{Ca}} = 0.016\,cm/s$ [(a)]	0, 0.5, 1.0, 1.5	1.0
Fast Potassium (Kf)	$-81.5\,mV$ [(b)]	$\hat{g_{Kf}} = 2.5\,mS/cm^2$ [(b)]	0.5, 1.0, 1.5	1.0
Slow-Potassium (Ks)	$-81.5\,mV$ [(b)]	$\hat{g_{Ks}} = 2.0\,mS/cm^2$ [(b)]	0.5, 1.0, 1.5	1.0
Sodium (Na)	$50\,mV$ [(b)]	$\hat{g_{Na}} = 10\,mS/cm^2$ [(b)]	0, 0.5, 1.0, 1.5, 2.0, 3.0	3.0
Leak (Lk)	$-52\,mV$ [(c)]	$\hat{g_{Lk}} = 0.25\,mS/cm^2$ [(c)]	0.5, 1.0, 1.5	1.0

[(a)] from Dale [1995b].
[(b)] from Winlove and Roberts [2012].
[(c)] from Sautois et al. [2007], based on an input resistance of $300\,M\Omega$. (Note that this leak-conductance value was calculated based on measurements of input resistance of an *in situ* dIN, which is electrically coupled).

2.4.2 *Sustained activity in a population of Hodgkin-Huxley type neurons*

In Chapter 5, in the experiments where Hodgkin-Huxley (HH)-type neurons were used, point neurons models were used with leak, sodium and potassium currents [Hodgkin and Huxley, 1952; Koch, 1999]. The populations of HH-type neurons had NMDA and AMPA feedback synapses, onto other neurons in the population, with a probability of making a connection of 0.15 between any pair of neurons, and neurons did not make synapses onto themselves. The gap junctions were modelled as a resistors of $100\,M\Omega$ connecting the somata, and a pair of somata had a 0.2 probability of forming a connection.

3

MANAGING COMPLEXITY IN SIMULATION TOOLS

3.1 GENERAL ISSUES IN COMPUTATIONAL SCIENCE

> "Increasingly, the real limit on what computational scientists can accomplish is how quickly and reliably they can translate their ideas into working code. "
>
> – Greg Wilson [Wilson, 2006]
> *Where's the Real Bottleneck in Scientific Computing?*

Dramatic developments in computational hardware over the last 30 years mean that vast amounts of data can now be acquired, processed and stored inexpensively. In parallel, a solid base of high-quality open-source platforms and libraries have been developed by research communities around the world. In science this has facilitated the collection and analysis of large quantities of data and today, the expectations of what can be understood quantitatively have ballooned: the climate, medicine, the origin of the universe, and the brain. These quantitative models are often analytically intractable dynamical systems and it is difficult to use traditional mathematical techniques to reason about their behaviours. Instead simulations are used to gain insight into their properties; computational modelling has become a widespread tool used across many fields of science: from subatomic collisions to the formation of galaxies, from rockets to climates, from tadpoles to neocortex.

The software ecosystem is constantly evolving on top of rapidly changing hardware platforms. In contrast to the traditional sciences, scientific computing is a young discipline: the problems of the field are slowly being identified and the best practises are starting to emerge [Merali, 2010]. Surprisingly, the difficult issues around modelling

often do not lie in the numerics of computation because the core code required for simulation kernels is often well studied and highly optimised [Press et al., 2007]. If modelling studies can be summarised in a few hundred words and a few equations and in general the simulations themselves do not take much time to run, why is modelling time-consuming - what limits productivity of individual modellers and what are the bottlenecks in collaborating and extending existing work?

I argue that most modelling is not spent at the cutting edge of a model; implementing new sets of equations, adding new features or trying to understand the numerics of a problem. Instead time is spent on more mundane problems such as managing files and data; converting between formats; interfacing between tools; finding mistakes in code and learning new library interfaces. The problems do not come from our inability to understand complex algorithms or conceptual issues of a model but instead the mundane day-to-day implementation issues: how do we effectively make a computer do what we want? Modelling has many similarities with software development, particularly the open source model, in which small geographically distributed groups of people, often working on loosely related problems, work together to produce technically complex software. What can we learn from the experience of the wider software development community?

The software community has recognised that the surrounding *softer* issues, for example data-management, reproducibility and sharing are central to effective sustained progress in complex software projects. Traditionally the focus of a software project was primarily the resulting executable code and the surrounding *scaffolding* such as documentation, testing, versioning, distribution and coding style was considered an optional nicety. The model has dramatically shifted: Python builds documentation directly into the language, test-driven-development proposes writing tests before any functionality [Beck, 2002], and toolchains are developed so that most software can be built, tested and documented with a single command. Although these issues are not as scientifically stimulating as exciting new algorithms, they must be solved to facilitate collaborative and sustained workflows. Both software projects and computational models require many smaller pieces of software to operate smoothly together. In order to build more advanced models, it is essential to get a handle on complexity: as the number of components, n, in a project increases, the possible

number of interactions between them scales as $O(n^2)$ [Brooks, 1995]. How are large software projects successfully developed if our brains can't reliably process such complexity [Parnin and Rugaber, 2012]?

Across all approaches to software development, a central tenet is effective partitioning [Sommerville, 2004]. Effective languages and libraries encourage interfaces to structure large codebases by encapsulating functionality in modules [Woodfield et al., 1981]. This allows developers to understand software both at a conceptual level, as the interactions of abstract components, but also allows them to focus their attention on a small, isolated part of the program to solve implementation issues [Sommerville, 2004].

In large projects, developers go to great lengths to reduce the user intervention required for often repeated, mundane tasks. Many commonly used tools were developed following the recognition of the difficulties surrounding the management of large amounts of data in software projects, for example the development of filesystems, compilers, development environments and version control tools. By automating a simple task, we make it easy to repeat and can be more confident that it has been performed correctly, which frees developers to work at a higher level. Experienced software developers know that mistakes are made. It is estimated that every 1000 lines of code delivered by industry contains 10-50 bugs [McConnell, 2004] and science is surely not immune to similar mistakes (e. g. [Krug and Kresnow, 1999; Miller, 2006; Gronenschild et al., 2012]). Successful development approaches take this into account. One approach is to avoid writing new code and instead reuse existing, tested libraries. Another approach is to pinpoint and isolate *mistake hotspots*, specifically tasks that are repetitive but conceptually well defined, and delegate these to libraries to remove human intervention as far as possible.

Experienced software developers are acutely aware of their own shortcomings in reliably reading and writing complex code. Tools for automatically quantifying complexity in code have existed for decades [McCabe, 1976]. Rules of thumb about where errors commonly occur have emerged, for example, functions that take more than five parameters or contain more lines than can fit on a screen which can be automatically detected by tools [McConnell, 2004]. Refactoring, simplifying the interfaces of code without affecting functionality, is an area of active research. Coding styles have

emerged because unpredictable program layouts are another form of complexity to be assimilated. The issue here is not that a function with more than five parameters, or without proper indentation is impossible for a developer to understand, but that more mental resources will be spent doing so. Each of these examples taken alone may only have a small effect, but their net result is a significant increase in the unnecessary complexity that a developer contends with.

Modelling is rarely a *hole-in-one* and often may require many cycles of iteration in conjunction with experimental work, suggesting where data is lacking and experiments needed, and incorporating new information as it becomes available. Modelling can be used to perform experiments that are impossible in reality, such as investigating the effects of a channel by removing it. In many cases, the exact issues that need to be investigated may not be known from the outset; instead, for a model to be useful it needs to be quickly and easily manipulated in order to answer specific questions. This scenario presents different problems beyond just solving the sets of equations. Interestingly, many similar issued have been raised in agile programming, a development process used in industry to develop software with shifting requirements [Olague and Etzkorn, 2008]. Specifically in computational modelling, the following particular classes of problems have been identified within the communities and potential solutions already proposed, both at the level of the individual modeller and to facilitate effective group collaboration:

MANAGEMENT OF SIMULATIONS An advantage of building a model, is that it can be used to investigate the effects of changing parameters on output, but how do we store these different simulation setups and results effectively? [Gil et al., 2007; Hudson et al., 2011]. If a modelling framework requires a new file for each experimental setup for every set of parameters, then even a small parameter sweep in which two parameters are tested at three different values and three experiments are run per set of values will require 27 input files. If we naïvely duplicate files, this experiment will become very difficult to maintain: if new experimental data becomes available and a parameter is revised, for instance, then it will need to be changed in 27 places.

DEPENDENCY MANAGEMENT Often, several tools are used in conjunction with each other and several processing steps produce the final output. For example, a set

of parameters, which are used by a simulator to produce a file containing results, which is then plotted using a graphing library. If the original parameter set changes, how can it be ensured that the output graphs are up to date? Mature, general purpose tools solve this problem by automatically tracking the dependencies between inputs and outputs (e. g. make [Mecklenburg, 2004]). To integrate with these dependency tracking tools, our own tools have to be designed with such automation in mind and it is particularly important that the command line is well supported [Groth and Gil, 2008].

REUSE OF COMPONENTS: A central tenet in software development is to avoid duplication of the same information [Kernighan and Ritchie, 1988; Fowler et al., 1999; Hunt and Thomas, 1999]. Similarly, parameters for a model need to be referenced from a single, authoritative place, if not it is inevitable that future changes to the model, for example due to new experimental data, will be made in some places and not others. This may lead to false conclusions being drawn because of running two different versions of a model. It is essential that our modelling tools facilitate component reuse, and more subtly, it must be easy to override particular parameters in a component without *copying and pastin*g, so we can investigate explicit changes.

REPEATABLE WORKFLOWS AND REPRODUCING RESULTS: How do I know which parameters produced a set of output graphs? How do I reproduce a graph I saw six months ago? One approach to this problem is to use version control systems (e. g. Sumatra [Davison, 2012]). Modelling often involves systems with many interacting components. If I change part of my model, how do I know that the effects do not knock-on to other parts of the model and whether previous results from that model are still valid? In software development, a comprehensive *test suite* is often created, which can be run to ensure that changes made in one place does not have unexpected side-effects in other parts of the code [Beck, 2002; Sommerville, 2004]. For this to be effective, it is essential that running this process is simple and has lead to the development of automated *continuous integration* tools. Many general tools have been written which can be adapted to neuroinformatics specific toolchains [Zaytsev and Morrison, 2012].

In modelling it is harder to write tests to check all output; yet we still need to be able to see effects of changing one part of a model on all previous simulations to ensure that our model still behaves within constraints as expected. Our tools need to make it easy to remain in control of a *library* of simulations during model creation, and after publication it should be easy for someone reading a modelling paper to quickly reproduce the graphs.

REINVENTION OF THE WHEEL Many tasks initially seem trivial, for example, extracting the spike times from an analogue voltage signal, but after the edge cases have been handled and bugs have been ironed out, many people may have wasted time solving the same problem. To avoid this we need common platforms and object models that are open, modular and easy to extend, so that parts can be reused across different projects, for example Bruederle et al. [2010].

COMMUNICATION OF MODELS How do we reliably communicate the details of a complex model? Following consistent notations allows our brains to assimilate a problem quicker. A recent literature survey finds that very similar concepts are communicated using a range of different conventions in computational modelling papers [Nordlie et al., 2009]. On a more practical level, how do we reuse models other people have written? Mistakes can be made in manually transcribing parameters from models into papers. One possibility is to generate documentation directly from the models [Nordlie and Plesser, 2010]. This has parallels to software documentation tools (e. g. tangle & weave [Knuth, 1992], doxygen [van Heesch, 1997], sweave [Leisch, 2002]).

In order to develop more complex models, unnecessary complexity needs to be trimmed so that scientific questions remain the focus of attention. Development of interpreted languages, such as Matlab and Python, around low-level numerical algorithms written in C and FORTRAN has already dramatically increased productivity of computational scientists. To overcome today's modelling bottlenecks, we can learn from the experience of the wider software community. Specifically, we need to remove user intervention from mundane activities, standardise interfaces and libraries, develop tools and toolchains to manage simulations and data which support repro-

ducible workflows and develop modelling platforms which encourage the reuse of model components and algorithms.

3.2 EXISTING TOOLS IN COMPUTATIONAL NEUROSCIENCE

As in many fields of computational science, simulating the activity in neuronal networks is achieved by modelling the individual components of the network as Differential Equations (DEs) which are solved over a period of time given an initial starting state. This is a widely used approach for many problems in computational science, and highly optimised, general purpose libraries have been written for efficiently solving large sets of these equations (e. g. [Hindmarsh et al., 2005]). Although models can be written using these libraries directly, there has been a trend towards using specialist neuronal simulators. These act as intermediate layers between the DE solvers and the context of the biological problem, for example NEURON [Carnevale and Hines, 2006] and GENESIS/MOOSE [Bower and Beeman, 1998] for simulations of multicompartmental neurons, and NEST [Gewaltig and Diesmann, 2007] and BRIAN [Goodman and Brette, 2008] for large networks of single compartment neurons [Brette et al., 2007]. These simulators offer a more natural interface for building biologically-based models by allowing modellers to express concepts such as *neuron population, cell segmentation* or *ion diffusion* in a single, high-level statement rather than by manipulating the underlying equations directly. By specialising the generic DE solvers, these tools can be highly optimised for solving the forms of equations occurring in neuronal simulations [Hines, 1984], can support specialist features such as events & delays and can automatically parallelise a simulation over a compute cluster. By hiding the numerical details from the modeller, it is simpler to write and manage complex models. The opaque interface provided by the simulator can be invaluable in debugging models: a simulator such as NEURON provides a thoroughly tested platform to build upon, in contrast to code written by an individual scientist, which is unlikely to have had any testing. This encapsulation dramatically reduces the complexity that a modeller has to consider and allows a great jump in productivity over coding from scratch. The kernels of modern simulators are efficient and support a diverse range of model types.

Over the last few years, Python has emerged as the *de facto* programming language in computational neuroscience [Davison et al., 2009]. Python combines powerful, dynamic typing, elegant architecture and clean syntax. Python is a general purpose programming language, with a large, open-source community providing powerful libraries from databasing to distributed processing to graphical user interfaces. Python provides an excellent basis for computational science by providing a high-level interface to mature, efficient numerical and plotting libraries [Oliphant, 2007; van der Walt et al., 2011]. It is simple to interface to existing code in languages such as C and FORTRAN. Python is a flexible language and which allows high-level concepts to expressed concisely, making it ideal for *plumbing components together*. All major neuroscience simulators now provide Python interfaces. [Eppler et al., 2008; Ray and Bhalla, 2008; Hines et al., 2009; Cornelis et al., 2012].

Both NEURON and GENESIS use custom languages for model specification. NEURON supports high-level simulation configuration through HOC (HOC), and the specification of individual components through MOdel Description Language (MODL) [Hines and Carnevale, 2000; Carnevale and Hines, 2006]. This allows the simulator to provide a high-level interactive workspace for the modeller to define the structure of a simulation, whilst leveraging existing compilers to generate efficient code for the core loops of the computation. NEURON was designed as a modelling environment for humans and unfortunately it is harder to use NEURON as a library. A Python interface to NEURON exists [Hines et al., 2009], which gives access to the HOC interpreter, but has shortcomings; (a) it is difficult to clear a simulation and restart from scratch, making it difficult to run multiple simulations in a single script; (b) it is difficult to define new simulation object types (e. g. channel & synapse models) directly from Python.

To promote model sharing, central repositories have been created such as ModelDB [Peterson and Healy, 1996]. The difficulties in sharing models in computational neuroscience are well documented and best formats for sharing complex models remain unclear [Cannon et al., 2007]. In general, approaches for defining models range between fully declarative to imperative. Declarative descriptions provide a cleaner, focused interface, reducing a complex model to set of parametrisable components, which are defined with values such as *conductance density* for a channel *or opening time constant* for a synapse, while imperative descriptions offer more flexibility be-

cause features from the host language, such as `if` statements, `for` loops and libraries, can also be used.

One example of a declarative format is NeuroML [Goddard et al., 2001; Gleeson et al., 2010], which provides a set of primitives, for example `DoubleExponentialSynapse`. Models are implemented by defining an eXtensible Markup Language (XML) file which specifies the parameters to these primitives, for example, `max_conductance`, `rise_time`, `decay_time` and `reversal_potential`. To use these components in a simulation, simulator-specific code is generated from this file, for example via an eXtensible Stylesheet Language Transformation (XSLT). An advantage of this approach is that it allows the user to specify a model once in a high-level fashion, and use it across multiple simulators. One drawback is that models need to be written in XML, which can make parametrisation difficult, and the indirection involved also necessitates a form of dependency tracking. In another XML-based model description language, Systems Biology Markup Language (SBML), a toolchain has been proposed for managing a set of patches that should be applied to an original XML file in order to change parameters and produce the *working model* [Saffrey and Orton, 2009]. Another problem with declarative frameworks in general is the requirement that a model must be expressed as one of the components predefined by the framework. Solutions are being proposed to these problems in the NineML language and NeuroML V2 via Low Entropy Model Specification (LEMS), which allow the specification of the parametrisable modelling components in terms of more general mathematical primitives, for example, *state variables*, *operating regimes* and *parameters*.

PyNN, a high-level simulator independent Application Programmer Interface (API) for building networks of point neuron models, takes an alternative approach to model specification. Network models are defined directly in Python and interoperability between simulators supported by allowing a model setup to be specified once and the simulator used to run the simulation chosen with a single `import` statement. PyNN avoids the dependencies on external files by predefining the types of neuron and synapse models that can be used. By defining simulations within a scripting language, more flexibility is afforded to the modeller in setting up complex network connectivity than would be available by defining the simulation in a declarative format, al-

though the clean separation between concept and implementation is no longer enforced.

A novel approach to defining network connections, proposed in the Connection Set Algebra (CSA) [Djurfeldt, 2012], is to supply a set of mathematical primitives which are powerful enough to represent algorithmic components for defining connectivity, allowing a complex description in a small number of symbols. More generally, how do we represent components of models that are best expressed as algorithms? The problem is rooted in the historical design of programming languages; traditionally functions are considered separately from data. Passing data around a program is commonplace but passing algorithms around has been more involved (requiring for example, function pointers [Kernighan and Ritchie, 1988]; polymorphism [Stroustrup, 1997] or architectural patterns [Gamma et al., 1994]). Modern languages are blurring this distinction and function objects, *functors,* can be treated in the same way as data structures, capable of being passed as arguments to other functions, greatly lowering the barriers to reusing algorithms.

3.3 ISSUES IN MODELLING SMALL, BIOLOGICALLY REALISTIC POPULATIONS OF NEURONS

In modelling small biologically realistic populations of neurons, beyond the more general issues surrounding computational modelling discussed above, we are also interested in the following topics:

DIMENSIONS & UNITS Quantities in neuroscience are expressed in a variety of units; for example conductance densities can be specified in $\mathrm{pS}/\mu\mathrm{m}^2$, $\mathrm{mS}/\mathrm{cm}^2$ or indirectly, for example 'a neuron with surface area of $1200\,\mu\mathrm{m}^2$ with an input resistance of $300\,\mathrm{M}\Omega$'. Although the conversions between these quantities are not complex, we want to avoid making these tedious conversions by hand. We want to be able to express the parameters used in our model in an explicit, flexible form so it is simple to verify that the values used in a simulation are the same as those measured experimentally.

DEFINING MORPHOLOGIES Neuronal morphologies are used in a variety of contexts; for example, electrophysiological simulations, and connectome reconstruction. Declarative formats for storing morphologies exist, for example SWC and MorphML [Crook et al., 2007], but currently no standard Python object-model has emerged. Our format should be interoperable with existing formats and make common operations simple.

VARIATION & STOCHASTICITY Even within a homogeneous population of neurons, we would expect variation amongst the members and the parameters used in a model usually reflect an average of those measured experimentally. It is important that our tools make it easy to reintroduce this variability seen in biology between components and also allow investigation into the effects of changing particular parameters explicitly

CHANNEL DISTRIBUTIONS There is strong evidence that channels are not uniformly distributed over neuronal membrane; for example the axon hillock and the nodes of Ranvier have been shown to have higher densities of sodium channels, which has already been modelled in studies (e. g. [Schmidt-Hieber et al., 2008]). We would like to be able to express these distributions concisely.

PARAMETER SWEEPS Often, exact parameters are unknown, and we need to try out parameters over a range to understand their effects. It must be very easy to perform parameter sweeps, ideally presenting a simple interface from within a single script to avoid potential complex data-management issues arising due to intermediate files.

INTERACTIVE TOOLS It is useful to explore a model, in order to get intuition into how it works, to understand how crucial particular parameters are to behaviour of a system. The less the number of steps required between a change being made, and the result appearing, the more valuable this process will be and ideally we would like to be able to build interactive tools on top of our toolbox.

REUSABLE COMPONENTS We would like to be able to define composite objects at a high level which can be reused. For example, given the definition of a neuron, including its morphology and channel distributions, or a small subnetwork con-

taining neurons, and their associated connectivity, it should be possible to embed this in several simulations directly without the need to replicate the code.

To tackle some of these issues; I present three pieces of software designed to make it easy to build and manage models for computational neuroscience: (a) `mreorg`, a tool for managing large numbers of Python simulations graphically; (b) `neurounit`, a library for specifying and working with units in computational neuroscience in Python; (c) `morphforge`, a high-level API for simulating multicompartmental neuron models in Python. These are briefly described in the next sections.

3.4 MREORG

Modelling is often an iterative process, and as the complexity of a model increases, managing a large number of associated scripts on the filesystem can quickly become unwieldy. `mreorg` is a tool for managing a set of Python simulations and their results in order to make it easy to monitor the behaviour of an evolving model (Fig. 3.1). `mreorg` automatically captures and stores output from Python simulations including generated graphs and text, and allows the user to later browse simulation results. It differs from Sumatra in that focus is not on tracking the history of the model [Davison, 2012], rather `mreorg` assumes that we are only interested in the results of the current state of the model . The central goal of `mreorg` is to make it as easy as possible to re-run and visualise the new output from a set of simulation experiments when the model changes.

`mreorg` is a tool consisting of two components: (a) a low-level Python library and (b) high-level web-based interface to managing large numbers of simulations. The low-level library leverages Python's dynamic typing and powerful introspection to allow behaviours of scripts to be controlled via environmental variables. For example, certain environmental variables control whether figures are displayed on the screen, or are instead saved to image files by dynamically changing the behaviour of matplotlib at runtime (i. e. *monkey-patching*). This allows the same script to produce interactive figures on the screen (normal behaviour) or be used as part of a batch file without any modification. It is unintrusive and only requires one line (`import mreorg`) to be added near the top of the script to use it.

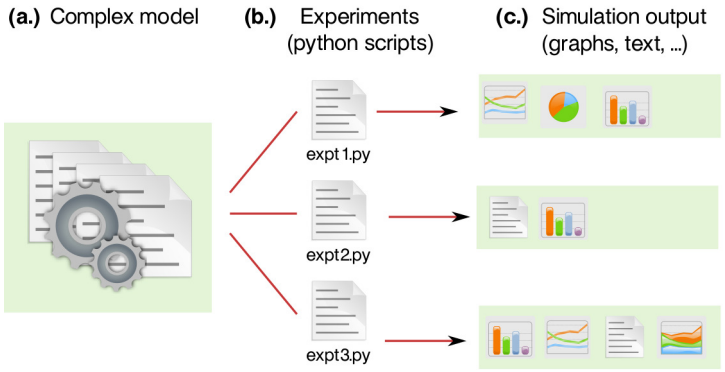


Figure 3.1 – An overview of `mreorg`. The use-case is that a model (a) is tested in many simulation experiments (b), which each produce a set of text and graphical outputs (c). A the goal of `mreorg` is to minimise the effort needed by the user to re-run and visualise new output from the simulation experiments as the model evolves over time.

The high-level tool makes it easier to maintain large numbers of scripts, when changes are being made and provides a simple way to run them in parallel. The tool consists of a django-based *web interface*, and a *backend* that runs the simulations. Using the web interface, a set of Python simulation files on a filesystem can be marked as *Monitored*. Several pages are then available, including an *Overview* page (Fig. 3.2), which summarises the current state of each simulation in a table, an *Output* page (Fig. 3.3), which displays all the output graphs captured from all simulations, and a *Details* page (not shown), which gives all the output (figures, standard-out/error) associated with a particular simulation run. The *Overview* page shows each simulation file as a row in a table including the date and time taken to run the simulation. The colour of the row makes it easy to see the current state of the file (Fig. 3.2), and simulations can be placed in a queue to be run by clicking on *'Run'*. Because the tool is interactive, it is possible to manually control runs of simulations that may take a long time to run.

Unlike Sumatra [Davison, 2012], mreorg does not track Python package dependencies. The goal of this software has been to make it very simple to ensure everything is up to date, quickly to track down errors in batch runs, and easily inspect large numbers of output graphs. More detailed information about `mreorg` can be found in Appendix G.

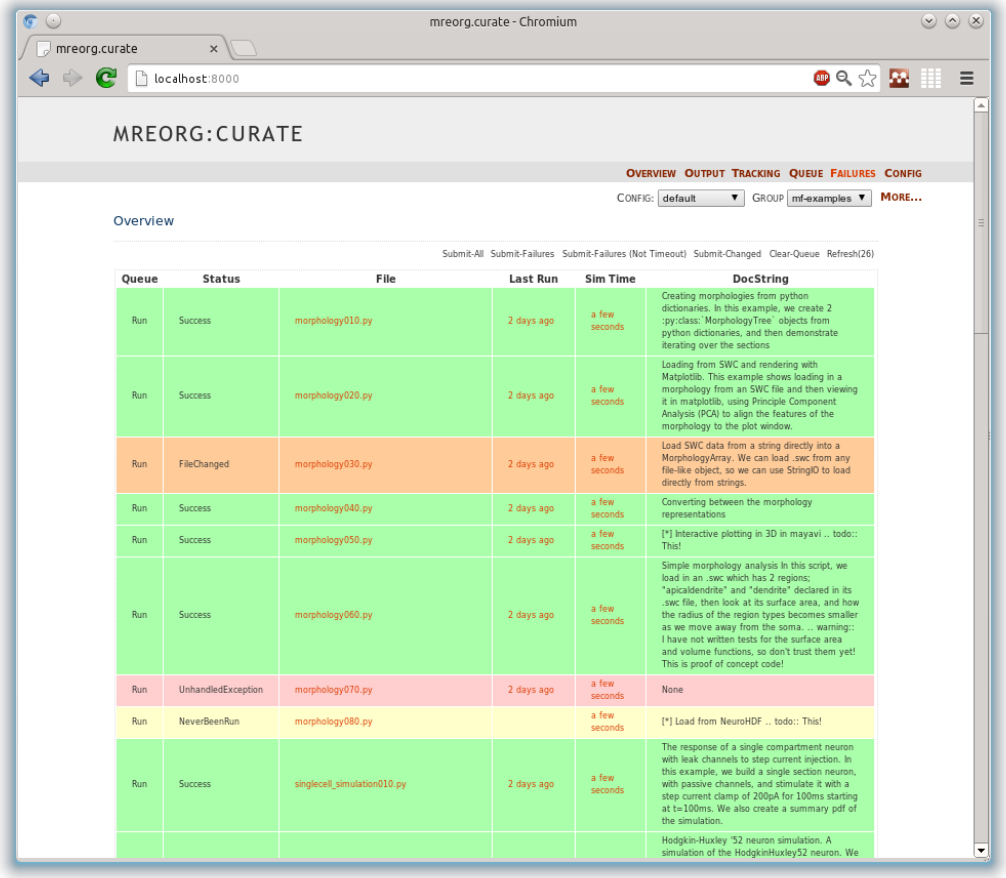


Figure 3.2 – The *Overview* page in `mreorg`, which shows each simulation file as a row in a table including the date and time taken to run the simulation and the docstring. Simulations can be placed in a queue to run by clicking on them. The colour of the row makes it easy to see the status of the simulation file, for example: *green*: simulation successfully ran; *red*: error running the simulation; *blue*: simulation timed out (not shown); *orange*: the file contents have changed since the last run. Simulation files can be grouped and in this screenshot the group *mf-examples* is shown.

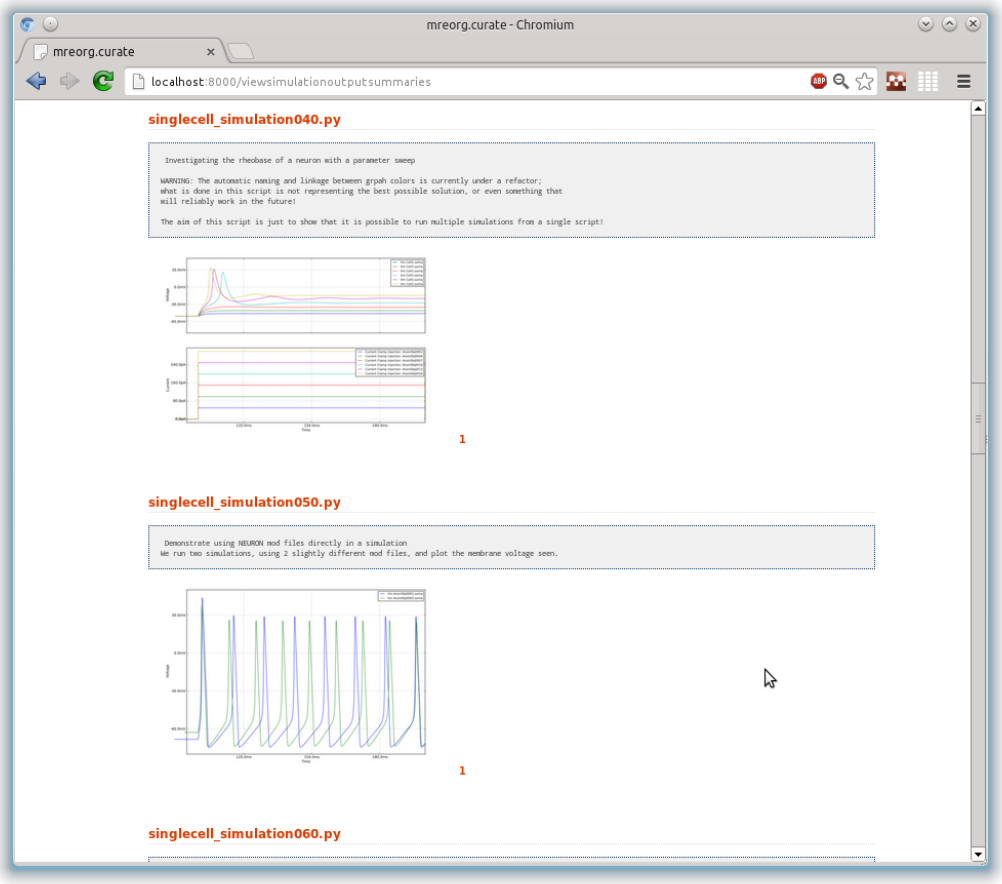


Figure 3.3 – The *Output* page in `mreorg`, which allows all the output graphs from a simulation run to be browsed offline. In this example, the output of two simulations, *singlecell_simulation040.py* & *singlecell_simulation050.py* are shown. This page shows a summary for the simulations (taken from the file's docstring) as well as the output graphs. Clicking on the name of the simulation gives more detailed information, such as the standard output & error and how long the simulation took to run (not shown).

3.5 NEUROUNIT

> "Root Cause: Failure to use metric units in the coding of a ground software file"
>
> *– Mars Climate Orbiter Mishap Investigation Board Phase I Report* [Stephenson et al., 1999]

The Mars Climate Orbiter crashed in 1999 because of a confusion over SI *Newton*s and imperial *pound-force* values. Over $500 million and five years of development was lost because a multiplication factor of 4.45 was missed [Stephenson et al., 1999].

Although conversions between units are mathematically trivial, they can be a source of difficult-to-find bugs. In neuroscience, the scales are small and often particular units of measurements chosen which are suitable for experimentalists. Simulations of biologically realistic neurons often involve many parameters, and it is essential that models are simple for humans to read to find mistakes. The values are not hard to calculate on a single occurrence, but in complex models, errors involving units are more likely to be found if the units of parameter values are specified explicitly and flexibly in a form that is easy to read. To illustrate, if a neuron has a surface area neuron of $1000\,\mu m^2$ and is voltage-clamped at $-30\,mV$, how much calcium current flows in the steady state, assuming $10\,mM$ intracellular and $100\,nM$ extracellular concentration of calcium, a permeability of $0.01\,cm\,s^{-1}$ and assuming that all the channels are fully open (i. e. Eqn. 3.1)?

$$\begin{aligned} I_{ca} &= P_{Ca} 2\nu F \frac{[Ca^{2+}]_i - [Ca^{2+}]_o e^{-\nu}}{1 - e^{-\nu}} \times m^2 \\ \text{where} \quad \nu &= \frac{2V_m F}{RT} \end{aligned} \tag{3.1}$$

Although the value is simple to calculate, unambiguously writing this in code succinctly is more difficult. In the implementation given in Listing 3.1, it is difficult to detect a mistake without resorting to pen and paper. Even if it is correct, when our simulations do not behave as expected, we are likely to scour this code to determine whether it is correct.

```
V_in_mV = -30
F_in_C_per_mol=96485; R_in_J_per_Kmol=8.314; T_in_K=300
pca_in_cm_per_sec=0.01; so_in_mM=10; si_in_mM=100e-3; area_in_um2=1000
nu = (2 * V_in_mV * 1e-3 * F_in_C_per_mol) / (R_in_J_per_Kmol * T_in_K)
ica = pca_in_cm_per_sec * (0.01) * 2.0 * nu * F_in_C_per_mol * (si_in_mM - so_in_mM * exp(-nu)) * ↙
    ↳ (1e-3) / (1-exp(-nu)) * area_in_um2
```

Listing 3.1 – An implemention of the GHK equation in a C-type language (possibly incorrect). It is difficult to quickly ascertain whether this is a correct implementation of Eqn. 3.1.

Computers are excellent at menial, well defined tasks and libraries already exist which perform dimension checking and inference in scientific calculations (e. g. [Petty, 2001; Alexandrescu et al., 2001; Karlsson, 2005; Dale, 2011]). Modern interpreted programming languages such as Python support mechanisms that allow dimensions to be explicitly embedded within the data itself rather than being specified in variable names, in comments, by convention or in documentation. Libraries such as `python-quantities` overload mathematical operators so that a voltage measurement divided by a current measurement will automatically have dimensions of resistance for example. In computational neuroscience, some simulator packages provide tools for checking unit consistency (e. g. `modlunit` in NEURON), and more modern simulators, for example BRIAN, provide built-in support for units [Goodman and Brette, 2008].

Two issues surrounding dimensions remain unresolved: firstly, there is no standard notation for defining simple quantities with associated dimensions as text strings and secondly, we lack suitable libraries that allow sets of equations with units to be specified and transparently handle dimensional inference *behind the scenes*. The first is needed to allow integration of data from different software packages and to allow tools and simulators to converge and interoperate by moving towards more standardised interfaces. The second is essential for building complex models of equations that both humans and machines can easily read.

I propose a grammar for defining units, quantities and systems for equations involving dimensions. The proposed syntax is designed to be unambiguous, but human readability is the central priority. Simple American Standard Code for Information Interchange (ASCII) strings are used because they have no dependencies, are human readable, implementation independent, and can be used within a variety of contexts,

for example in the header of a Comma Separated Variable (CSV) file. A prototype library is available in Python (see Appendix B & G), but a similar library could be implemented using standard libraries available in most mainstream programming languages. Three layers have been defined to support increasingly complex use cases:

LEVEL-1 supports the definitions of units and quantities involving units, for example: `'mV', '10pA/cm2', '0.1e-2mm', '14 centimeter second'`. LEVEL-1 is designed to be simple to parse using simple tokenising routines from standard libraries and predefined lookup tables. Such strings could be used in a simulation configuration files or in the header of a CSV file for example. Using `neurounit` and `python-quantities`, the code in Listing 3.1 can be rewritten as Listing 3.2.

```
from neurounit import Q1
# Define quantities with units using Neurounit Level-1:
V=Q1('-30 mV')
F=Q1('96485 C/mol'); R=Q1('8.314 J/K mol'); T=Q1('300K')
pca=Q1('0.01 cm/s'); so=Q1('10 mM'); si=Q1('100 nM')
area=Q1('1000 um2')

# use python-quantities to calculate the result:
# (note: 'nu' and 'ica' are internally recording the units)
nu = (2*V*F)/(R*T)
ica = pca * 2.0 * nu * F * (si - so * exp(-nu))/(1-exp(-nu)) * area


print ica.rescale('pA')
# Displays: "array(-0.4966447845995212) * pA"
```

Listing 3.2 – Example of neurounit LEVEL-1. We define a series of variables using neurounit LEVEL-1, which we then use to perform a calculation. Since the units are part of the objects (V, F, R, etc), the correct units are automatically propagated to `ica`.

LEVEL-2 supports the calculations of quantities involving units, for example using the mathematical operators `'+-*/'`, and also the use of constants and functions. LEVEL-2 only supports a single expression[1]. In LEVEL-2 (and LEVEL-3), all numerical constants in expression must appear within curly braces,`{...}`, for example `'100mV + 3V'` would be written as `'{100mV}+{3V}'`. This is to prevent ambiguous statements such as `'1 pA / F'` which could refer to either the *one pico-amp-per-farad* or *one pico-amp divided by Faraday's constant*. (These two cases

1 Although the result of one expression could be used in another in a script, for example, 3.2

would instead by written as `{1 pA/F}` or `{1pA}/F`). A built-in library is provided, which contains basic mathematical functions (e. g. `sin()`, `exp()`) and physical constants (e. g. `e`, `pi`, `F`). This allows us to make derivations of parameter values more clear (Listing 3.1).

```
from neurounit import Q2
area_circle = Q2('4 * 3.141 * {10um}**2')
area_sphere = Q2('4 * std.math.pi * {15 micrometer}**2')
leak_conductance = Q2('(1/{300 MOhm}) / {590 um2}')
current_flow = Q2('({1mV}+{200 uV}) * {400 nS}')
```

Listing 3.3 – Examples of neurounit LEVEL-2

LEVEL-3 supports the grouping of differential and normal equations that should be solved together, for example, in a model of a transmembrane current specified by HH equations (see Section. 2.1.1). In `neurounit`, a group of equations, possibly including Ordinary Differential Equations (ODEs), can be grouped together as an equation-set (`eqnset`), in which the symbols are shared among the equations. LEVEL-3 also supports the definition of libraries (`library`), which contain only function definitions and constants, and can be accessed through using a Python-like syntax *(*i. e. using *namespaces* and `import` statements*)*. Both `eqnset` and `library` are specified as blocks, as shown in Listing 3.4. Equation-sets allow the definition of time-evolving state variables, intermediate variables, parameters and functions. LEVEL-3 allows synaptic and channel kinetics to be defined in an unambiguous, easy to read fashion. The syntax is general purpose, and neuroscience-specific information is specified through metadata, such as, which variables represents transmembrane voltages or currents. LEVEL-3 syntax is more complex to read than LEVEL-1 & LEVEL-2 and involves more comprehensive parsing and interpreting phases. `neurounit` internally constructs an Abstract Syntax Tree (AST), which can then be used to produce various outputs, for example, MODL, summary pdf-documents via LaTeX and Python functors.

A discussion surrounding parsing strings with units, design decisions about the proposed syntax and some of the implementation issues are given in Appendix B.

LEVELS 1 & 2 of `neurounit` are used extensively in morphforge (Section 3.6) to specify parameters and LEVEL-3 can also be used to define channel and synapse models (Section E)

Neurounit was developed following discussions at an International Neuroinformatics Coordinating Facility (INCF) NineML taskforce meeting about how to specify the parameters involving units for models. The implementation of LEVELS-1 & 2 provided a more standardised syntax for specifying parameters morphforge, the extension to LEVEL-3 was an experiment to investigate how the syntax would scale. NineML supports a wider range of features that `neurounit` does, for example *regimes* and *transitions*. In the future, it would be interesting to extend NineML to include some of the syntax from `neurounit`.

```
from neurounit import L3
k_chl_def = """
library myneurolib {

   # We define a function for calculating rate constants:
  RateConstant5(V:(V), a1:(s-1) ,a2:(V-1 s-1), a3:(), a4:(V), a5:(V)) = \
     (a1 + a2*V)/(a3+std.math.exp((V+a4)/a5))
}
eqnset chlstd_hh_k {
   from myneurolib import RateConstant5
   i = g * (v-erev) * n**4
   ninf = n_alpha_rate / (n_alpha_rate + n_beta_rate)
   ntau = 1.0 / (n_alpha_rate + n_beta_rate)
   n' = (ninf-n) / ntau
   n_alpha_rate = RateConstant5( V=v,a1=n_a1,a2=n_a2,a3=n_a3,a4=n_a4,a5=n_a5)
   n_beta_rate = RateConstant5( V=v,a1=n_b1,a2=n_b2,a3=n_b3,a4=n_b4,a5=n_b5)

   <=> PARAMETER g
   <=> PARAMETER erev
   <=> PARAMETER n_a1, n_a2, n_a3, n_a4, n_a5
   <=> PARAMETER n_b1, n_b2, n_b3, n_b4, n_b5
   <=> OUTPUT i:(mA/cm2) METADATA {"mf":{"role":"TRANSMEMBRANECURRENT"} }
   <=> INPUT v: mV METADATA {"mf":{"role":"MEMBRANEVOLTAGE"} } }
}
"""

# Parse the string:
k_chl_lm = NeuroUnitParser.File(k_chl_def)
k_chl = k_chl_lm.eqnset['chlstd_hh_k']

# Now, we can do various things with the eqnset:
k_chl.to_nmodl(...)
k_chl.to_latex(...)
k_chl.to_functor(...)
```

Listing 3.4 – Example of neurounit LEVEL-3. We demonstrate using a library and an equationset. The library is defined, which defines a function RateConstant5 which calculate the forward and backward rate constant at a particular voltage, V, given a set of parameters, A, B, C, D & E (see Chapter 2). We next define a potassium channel in an equationset called chlstd_hh_k. After this string has been parsed by neurounit into an object, it can be used in a simulation, written to a LaTeX document or used as a Python functor.

3.6 MORPHFORGE

3.6.1 *Overview*

morphforge is a high-level, simulator independent, Python library for building simulations of small populations of multicompartmental neurons, in which membrane voltage is calculated from the sum of individual ionic currents flowing across a membrane. The use-case of the API is to allow the modeller to quickly construct simulations of small populations of neurons and synaptic connections, with particular focus on:

a) allowing simulation specification, with visualisation and analysis in a minimal, clean, human readable language;

b) reduction of complex simulation toolchains to a single Python script;

c) promoting reproducible research through automatic documentation generation;

d) encourage the reuse of components such as morphologies, neurons and channels such so that specific and stochastic variation in parameters is simple;

e) transparent handling of different units;

f) allowing the use of established formats, (e. g. MODL files), but also simplify the definition and sharing of new channel types easily, including the possibility to support other libraries and standards easily (for example PyNN, NineML, NeuroML, CSA)

Morphforge is not a simulator itself; it is a high-level interface to simulators (currently NEURON) and provides a set of high-level primitives for specifying morphologies, channel distributions and network connections in Python. Morphforge is not designed for large-scale simulations and a design choice was taken to prioritise the interface to the modeller over simulation speed. Morphforge provides a single interface to building models of multicompartmental neurons: an entire *in silico* experiment, including the definition of neuronal morphologies, channel descriptions, stimuli and plotting of results at publication quality can be written in a single short Python script,

without the need for external files. Reusability is central to the library, and consideration has been given to how to reuse both algorithmic and parametrisable components. The design of the API allows models to be written in a declarative style, but because the full Python object model is exposed, complex objects can also be built. It is possible to define new, simulator-independent objects, although the architecture aims not to restrict what can be done to the lowest common denominator. For example, it is possible to use HOC and MODL files in morphforge simulations using the NEURON backend directly.

Morphforge consists of four layers which each define a set of classes which work together as an object model (Fig. 3.4). The higher layers depend on the lower levels, but lower levels do not need the higher ones, for example, `Morphology` objects are used by the `simulation` layer, but can also be used without it, for example for anatomical reconstructions.

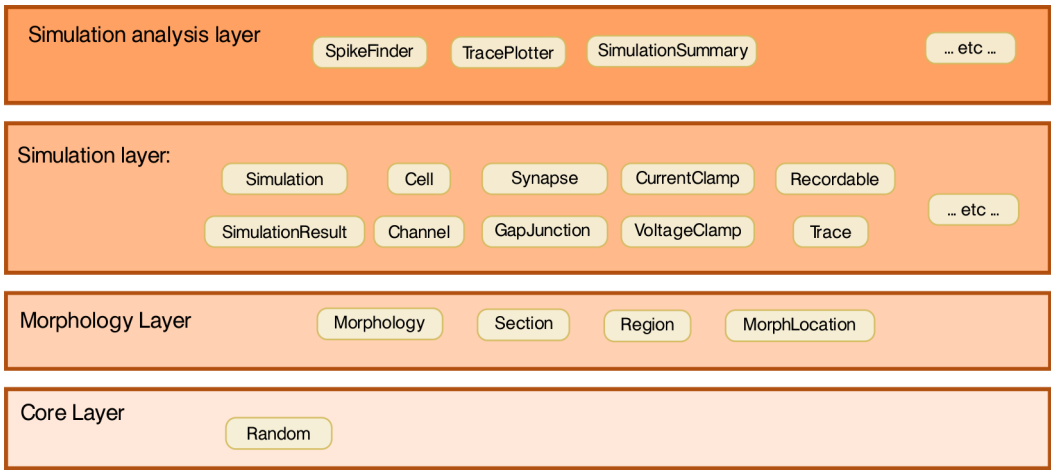


Figure 3.4 – The layers in morphforge and example classes from each layer. The details of each layer will be described in the following sections.

The `core` layer provides a single point of access to control random number seeding, simulation settings and locations on the filesystem access as well as the plug-in infrastructure and utility functions. The `morphology` layer provides classes that represent neuronal morphologies as a tree of cylinders and functions for their creation, import, export, rendering, traversal & manipulation. The `simulation` layer defines a simulator-independent object-model on top of the morphology objects for defining multicompartmental neurons with complex channel distributions. Primitives for de-

fining network connectivity and an interface for recording values during a simulation are also provided. It provides a library system for allowing components (e. g. channel & synapses) to be defined once and reused with different parameters, as well as an extensible high-level object model for representing analogue signals with units. Finally, the `simulationanalysis` layer provides functions for analysing the output of simulations such as spike detection, a visualisation system for easily viewing the outputs of simulations and infrastructure for automatically generating summaries of simulations including the details of components such as channels and synapses.

In the following sections, I will briefly introduce the `morphology` and `simulation` layers. Further details about the high-level architecture, design decisions and implementation of the `simulation` and `simulationanalysis` layers can be found in Appendix D. Example simulations are given in Section E. Further example scripts and API documentation can be found in the morphforge documentation (see Section G).

3.6.2 *`morphology` layer*

Morphforge defines a simple object model for representing the morphologies of neurons, independently of electrical properties. As with many existing formats, morphologies are represented as a tree of joined conical frustra (for example see [Carnevale and Hines, 2006; Crook et al., 2007]). The object model is minimal and it is simple to traverse the tree structure using Python iterators. Morphologies can be imported from SWC & NeuroML files or constructed in code. Morphologies can be rendered as 2D projections using matplotlib and in 3D using MayaVI. The object-model consists of just four classes: `Morphology`, `Section`, `Region` and `MorphLocation`.

The `Section` objects, which correspond to the individual conical frustra, contain information about their length and diameters as well as their connections to other `Sections`. The `Morphology` object, is a container for `Sections`, and represents the morphology of a single neuron. Within a `Morphology`, a group of `Sections` can be defined as a `Region`, and particular locations on the morphology can be specified as `MorphLocation` objects. `Region`, and `MorphLocation` objects are used by the `simulation` layer, for specifying channel distributions and synapse locations.

3.6.3 *simulation layer*

General structure of a simulation

The simulation layer provides an object model for building simulations of multicompartmental neurons and networks entirely within Python. The object-model in the simulator layer is designed to be both flexible and allow complex simulations to be written concisely. A script to run a simulation normally performs the following steps:

a) Instantiate a `Simulation`

b) Define types of `Channel` and `Synapses`

c) Populate the `Simulation` with `Cells`

d) Specify the biophysics of the `Cells`, (e. g. using `Channels`)

e) Connect the `Cells` with `Synapses`

f) Specify experimental stimuli (e. g. `CurrentClamp`))

g) Define which values should be `recorded` during the simulation

h) Call `Simulation.run()`, which runs the simulation and returns a `SimulationResult`

i) Retrieve the recorded values from the `SimulationResult` for plotting or analysis

A simple example script is given in Listing 3.5 that simulates a single neuron with passive channels in response to a step current injection. No external file dependencies are needed, and running the code will cause Figure 3.5 to be displayed on the screen.

LINE [4] A `Simulation` is created, which will use the NEURON backend and last for 200 ms.

LINES [7,8] We create a `Cell` object. We use a dictionary to define the morphology of the cell, which is a single `Section` (cylinder) of length 20 µm and diameter 20 µm. The `Section` has the id of *soma*, which will be used later.

LINES[11-14] A leak channel is defined with a conductance and reversal potential.

```
from morphforge.stdimports import *
from morphforgecontrib.stdimports import *
# Create the environment & simulation
sim = NEURONEnvironment().Simulation(tstop=200*units.ms)

# Create a cell:
cell = sim.create_cell(name="Cell1",
                 morphology=MorphologyBuilder.get_single_section_soma(area=1000*units.um2) )

# Define a leak channel
lk_chl = sim.environment.Channel(StdChlLeak,
         name="LkChl",
         conductance=0.25*units.mS/units.cm2,
         reversalpotential=-51*units.mV)

# Apply the leak channel to the cell, and set the capacitance:
cell.apply_channel(lk_chl)
cell.set_passive(PassiveProperty.SpecificCapacitance, 1.0*units.uF/units.cm2)

# Create the stimulus
cc = sim.create_currentclamp(
   name="Stim1", cell_location=cell.soma,
      amp=200*units.pA,
   dur=100*units.ms,
   delay=100*units.ms)


# Define what to record:
sim.record(cell, what=StandardTags.Voltage, name="SomaVoltage", cell_location=cell.soma)
sim.record(lk_chl, what=StandardTags.CurrentDensity, cell_location=cell.soma)
sim.record(cc, what=StandardTags.Current)

# run the simulation
results = sim.run()

# Display the results:
TagViewer(results, figtitle="The response of a neuron to step current injection", timerange=(95, ↙
     ↳ 200)*units.ms)
```

Listing 3.5 – An example script using morphforge to simulate a single compartment neuron containing only leak channels in response to a current injection and plot the results

LINES[17] The leak channel is *applied* to the cell. (By default it is applied everywhere on the neuron with a uniform density, but this can be specified. This is discussed further in Section D.1.1).

LINES[18] The capacitance of the cell is set.

LINES[20-21] A step current-clamp is created which injects 200 pA for 100 ms starting 100 ms after the simulation starts into the soma of *cell1*.

LINES[29-31] We specify that we want to record the voltage from the neuron at the soma, the current density flowing through the leak channels at the soma and the total current flowing through the current-clamp.

LINES[34] Until this line, we have built an object model in memory, When a `Simulation` is `run()`, morphforge runs the simulation, and returns the result. (Internally, this step can be complex, for example, in this example, morphforge will generate and compile suitable MODL files, spawn an instance of NEURON in a new process and link it with the compiled MODL code. Next, NEURON will run and solve the equations. Morphforge will capture the output, and finally make it available in the original process through the `SimulationResult` object stored in the variable `results`).

LINES[37] Plot the output results. The `TagViewer` inspects the results object, and works out what should be placed on which axes, and deals with the units of the output automatically. The figure displayed by this call is shown in Fig. 3.5.

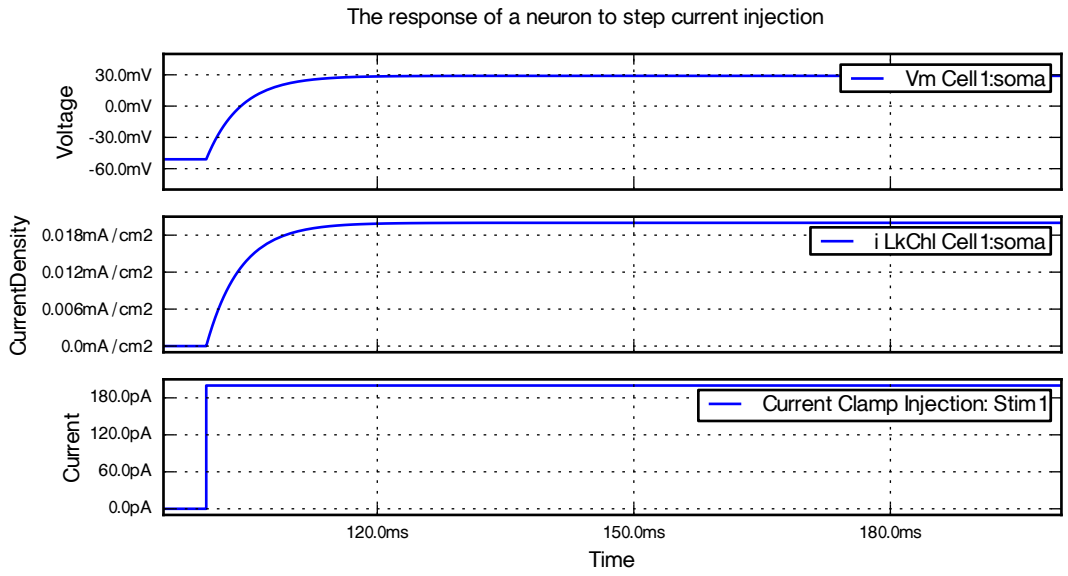


Figure 3.5 – The figure produced from running Listing 3.5. Three axes have been automatically created, for *voltage*, *current density* and *current*, and the values recorded on lines 29-21 automatically displayed with the correct units. The `TagViewer` object is discussed further in Section D.1.2.

Further examples are given in Appendix E and details about the architecture and implementation are given in Appendix D.

3.6.4 *Building on the object model*

One advantage of building an object model over a standalone program is that it can be used as a basis to build more complex tools. I built a tool that allowed us to interactively explore the effects of different parameters on a single tadpole neuron. The tool allowed us to visualise the effects of the forward and backward rate curves functions on the simulated voltage-clamp curves interactively (not shown) and to explore the effects of the different parameters of the model on the responses to current-clamp experiments (Figure 3.6). The tool uses the interactive plotting library Chaco.

Each type of channel (sodium, fast and slow potassium, calcium and leak) in the neuron has a tab, which allows the reversal potential and conductance to be adjusted, as well as a set of interactive graphs for each gating particle for that channel. These graphs show the forward & backward rate constants and the resulting steady state and time constant graphs as a function of voltage The curves are approximated as piecewise line segments, and the connecting points can be adjusted by dragging with the mouse. The graphs are interconnected and moving a point on either the time constant or steady state activation graphs for example will automatically adjust the forward and backward rate equation graphs in real-time, and vice versa. Changing any parameters in the simulation will cause the simulation to be run in NEURON and the results to be displayed in the centre panel. The entire tool is less than 700 lines of code.

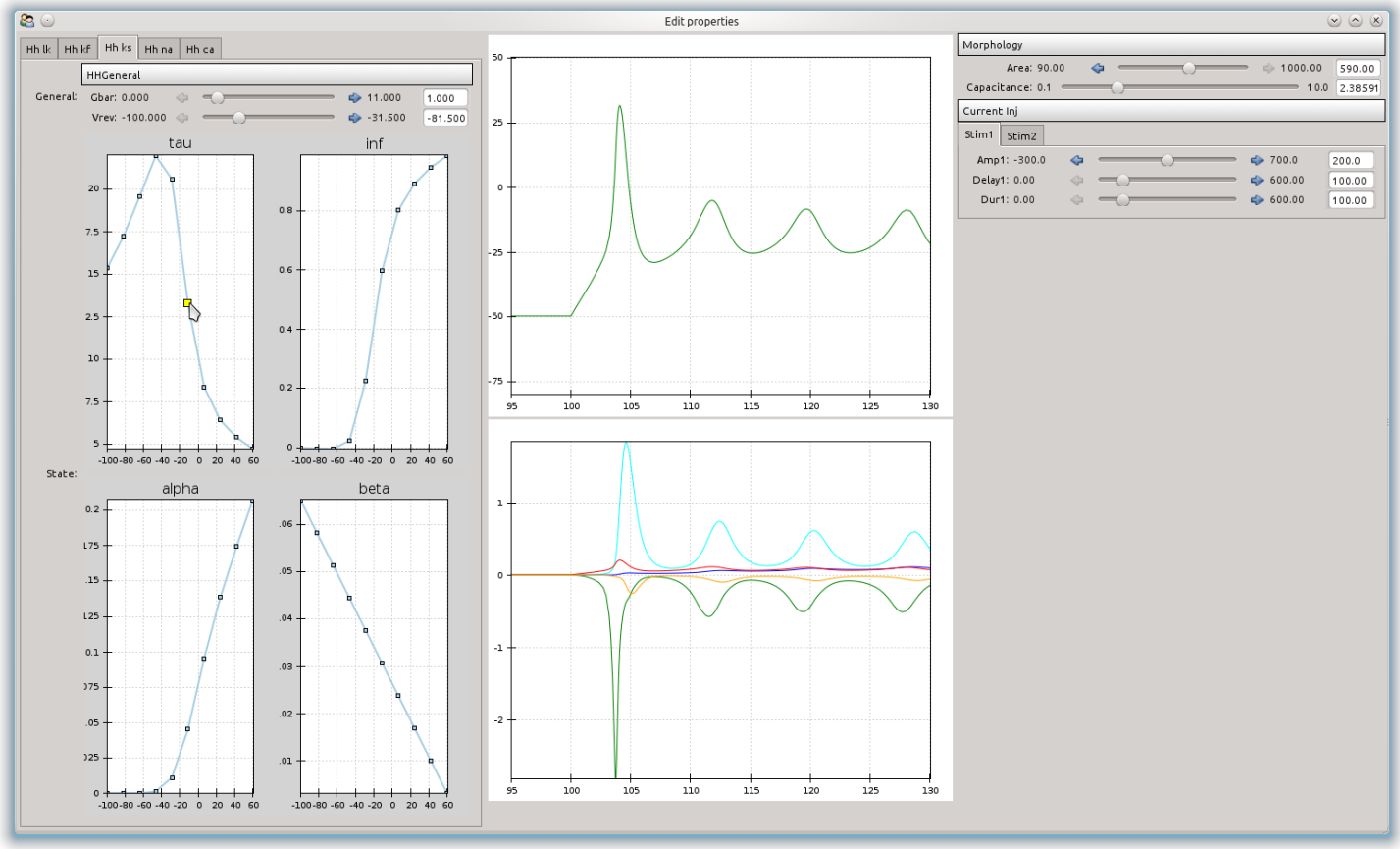


Figure 3.6 – A screenshot of a simple tool that was built on top of morphforge. The tool allows parameters of a simulation to be interactively changed, including passive properties & stimulation protocols (top-right), the conductance densities and reversal potentials of each channel (top-left). For each state variable, for each channel, the forward and backward rates, steady state and time constants as functions of voltage are shown in plots. The curves are approximated by line segments, which can moved interactively using the mouse in order to adjust the channel kinetics. Changing any of the parameters of the model runs the simulation with the new parameters and automatically updates the graphs.

3.6.5 *Testing morphforge*

Testing is an integral part of software development and many proposals have been made for how to test different aspects of software systems in general. One particular difficulty with testing scientific software is that the exact purpose of the code can be difficult to define ahead of time. The approach used to develop morphforge was a combination of defensive programming [McConnell, 2004] in conjunction with high-level functional testing. Morphforge has been tested using a new Simulator-TestData repository, which defines a set of scenarios in a simple, consistent, human & machine readable text file.

Each scenario file describes the setup for a simulation; the start of 'scenario-001' is shown in Listing 3.6. The description is given as a text string. The specification allows parameters to be used in the description (for example `<A>`, `<VS>`, `<C>`, `<GLK>`, `<EREV>` & `<I>`), and defines what should be recorded (for example the voltage of `cell1` as `$V`). The file defines which units to use for all the parameters and recorded values, and also defines the values that should be used for each parameter for parameter sweeps (not shown).

```
scenario_short= scenario001
title = Responses of passive cell to step current injection
description = """
In a simulation lasting 350.0ms
Create a single compartment neuron 'cell1' with area <A> and initialvoltage <VS> and capacitance <C>
Add Leak channels to cell1 with conductance <GLK> and reversalpotential <EREV>
Inject step-current of <I> into cell1 from t=100ms until t=250ms
Record cell1.V as $V
Run the simulation
"""
```

Listing 3.6 – An example of a scenario description from the Simulator TestData repository. (Only the first part of the file is shown)

The repository is designed for testing simulation results in two ways. Firstly, results can be compared against a set of hand-calculated results, which are given as a table in the scenario file. Secondly, result traces from different simulators can be compared against each other. More details about the format and a complete example scenario file are given in F.1.

The Simulator-TestData repository was written to check that no mistakes had been made in unit conversion internally within morphforge. The predominant primitive

objects in morphforge, including current & voltage clamps, channels, synapses and gap junctions, were tested using 10 scenarios, which include tables of hand-calculated expected values. A set of HOC and MODL files were written to ensure that the results from code written directly in NEURON matched those from morphforge.

3.6.6 *Further work*

Morphforge has developed incrementally out of the requirements for the modelling for this thesis. However, I have tried to develop it as a reusable platform for more general use cases. Morphforge is not *production-ready* code, it is a prototype object model for modelling and simulation small networks of multicompartmental neurons. Some decisions were taken because they were simple to implement, rather than being the optimum design decision and improvements can be made.

EFFICIENCY OF THE `trace` CLASS: Currently, there is no sharing of the time values amongst `Trace` objects. This means that if 300 traces are recorded from a simulation, 300 distinct `numpy.arrays` will be created with exactly the same data, which is not efficient for large simulations. There is no architectural reason why this needs to be the case and was done for simplicity. Moreover, further work on morphforge would ideally interface to the classes in NEO[], to provide maximum interaction with electrophysiological data-formats.

SYNAPSES: As was discussed in Section D.1.1, in some cases it is possible to share postsynaptic receptors among several synapses, provided they are linearly superposable. Morphforge currently allows postsynaptic receptors to be shared, however, there are some limitations, for example it is not possible to record internal values, such as current, from shared receptors using the interface described in Section D.1.1. One possibility that would need further development is that the modeller would not specify that receptors are shared, and instead, morphforge could make optimisations after `Simulation.run()` was called. At this point, the object model could detect instances of the same type of receptor onto the same postsynaptic neuron, and, provided that they were linearly su-

perposable and not being recorded from, merge the postsynaptic objects automatically, in order to allow much faster simulations.

ION FLUXES By defining the ionic species of currents, NEURON is able to calculate ion fluxes across the membrane and therefore keep track of changes to intracellular and extracellular ionic concentrations, which can be used by other channels, for example in models of calcium-dependent potassium channels. This is not directly supported by the morphforge object-model, but nor does it it limit it and incorporating MODL using this feature should work as expected.

PYNN INTERFACE morphforge provides an interface for building small populations of multicompartmental neurons. By contrast, `PyNN` provides an interface for specifying simulations of larger populations of neurons. Since they both use Python object models, it should be simple to provide a morphforge-backend to `PyNN`. One benefit of this would be a standardisation of connectivity primitives between software packages, as well as providing a single platform for building multiscale models.

ERROR HANDLING morphforge has been written incrementally as a tool. The code is written defensively [McConnell, 2004] and will cause exceptions to be raised in the case of ambiguous circumstances. morphforge is a prototype, and the software is not polished, which means that sometimes the resulting error stack traces can be hard to diagnose.

SUMMARY GENERATION Morphforge contains infrastructure for automatically generating summaries of the simulation setup. The summaries contain details about the morphologies of the neurons, synapses between them and stimuli (e. g. Fig. E.4). Diagrams are automatically drawn showing the connections between individual neurons or between populations of neurons. The `Channel` objects and `PostSynapticTemplate` objects are able to document their kinetics in a summary. More work is needed to improve the quality of the output diagrams and to display summaries with more appropriate levels of detail depending on the complexity of the simulation.

Part III

RESULTS

4

A POPULATION OF ELECTRICALLY COUPLED DINS

4.1 ABSTRACT

Axo-axonic gap junctions allow neurons with distant somata to be electrically coupled and can decrease response latencies, synchronise firing or even generate oscillations in higher brain centres, but unfortunately, experimental techniques to investigate the distribution and function of electrical coupling in network dynamics are limited. I modelled a column of 30 dIN brainstem neurons with axo-axonic electrical coupling using anatomically realistic multicompartmental models of somata and axons. I first built models of a passive network to infer the layouts of gap junctions between neurons. It was difficult to distribute gap junctions in the network so that the measured coupling coefficients between neurons were similar to those observed experimentally, and the best matches were achieved when gap junctions formed in a region close to the soma of the more caudal neuron. The diameters of axons had a strong influence on the strength of the electrical coupling, and it was not possible to match the coupling coefficients in the model to experimental data when the diameter was reduced to less than 0.3 µm. The numbers of gap junctions in the system suggested that most pairs of neurons would not be directly coupled by a gap junctions and instead the coupling was often via the axon of a third neuron. When active channels were added to the model, I found that in a network at rest, the gap junctions to other (hyperpolarised) neurons acted as current sinks and prevented action potentials propagating along the thin axons. I found I had to increase the density of sodium channels and decrease the density of potassium channels in just the initial region of the axon to

allow action potentials to propagate without the population becoming over excitable. In life dINs only fire a single action potential in response to step current injection, but they fire like pacemakers when they are perfused with NMDA. I ran parameter sweeps over the conductance densities of the different channels and matched firing properties using a model dIN *in situ* (i. e. in an electrically coupled network). The firing behaviour of a dIN in the electrically coupled network of neurons was dependent on the activity of other neurons: it would fire once to a step current injection but as a pacemaker when all members of the population were excited simultaneously, as is the case with a perfusion of NMDA. This raises the possibility that in life, an isolated dIN will fire repetitively and that the observed single-spike firing behaviour is actually a consequence of the way experiments are performed rather than a situation that will occur in reality. Finally, when all the dINs in the small electrically coupled population were simultaneously given step current injections, they fired repetitively like pacemakers in synchrony, but when the electrical coupling was removed the firing became desynchronised within a few cycles, suggesting that within a one-sided network, electrical coupling plays an important role in generating synchronous pacemaker activity during swimming.

4.2 INTRODUCTION

Animal behaviour is controlled by interactions amongst neurons via chemical and electrical synapses. Electrical synapses are widespread in nervous systems, from snails to monkeys, from motoneurons to neocortex [Bennett and Zukin, 2004]. What effects do they have on coupled neurons and on network behaviour? Electrical coupling has a long history of study for example in giant motor synapses of crayfish [Furshpan and Potter, 1959], Mauthner cell synapses in goldfish [Lin and Faber, 1988], the electric organ of fish [Bennett et al., 1967], and the stomatogastric system of lobsters [Eisen and Marder, 1982; Marder and Bucher, 2007]. In these relatively simple systems, electrical coupling between cells has been proposed to be important in reducing response latency, synchronisation of firing, and more recently, in more complex computation such as coincidence detection [Edwards et al., 1998].

Electrical coupling has also been found within larger populations of neurons in the brain and in some cases this coupling is axo-axonic: in the inferior olive [Llinas et al., 1974; Hoge et al., 2011], in subpopulations of inhibitory neurons in the neocortex [Gibson et al., 1999; Galarreta and Hestrin, 2002; Pangratz-Fuehrer and Hestrin, 2011], in hippocampal pyramidal neurons where it is axo-axonic [Draguhn et al., 1998; Traub and Bibbig, 2000; Mercer et al., 2006; Vivar et al., 2012] and in the cerebellar cortex [Mann-Metzer and Yarom, 1999; Traub et al., 2008]. Although the effects of electrical coupling are likely to be diverse in these larger populations [Hormuzdi et al., 2004; Connors and Long, 2004], one hypothesis that has received considerable attention is in the synchronisation of network activity over particular frequency ranges [Traub and Bibbig, 2000; Maex and De Schutter, 2007; Traub et al., 2008]. Links have also been suggested with epileptic seizures [Volman et al., 2011].

Experimentally, electrical coupling is easy to demonstrate between pairs of recorded neurons, but difficult to investigate within populations of neurons. The problem starts with defining the distribution of electrical junctions within a network. Low molecular weight tracer dyes can be injected and may pass through gap junctions to reveal other electrically coupled neurons and show potential gap junction sites. However, dye spread often depends on the types of gap junctions so electrically coupled neurons frequently show no dye coupling (e. g. [Li et al., 2009]). The junctions can also influence the uptake and transport of the dye [Mills and Massey, 1998] and within a group of tightly coupled neurons, resolution of contact sites can be difficult [Fukuda and Kosaka, 2000]. Secondly, within an electrically coupled population of neurons, the properties and responses of individual neurons cannot be isolated in electrical recordings. In order to get around these problems and understand the effect of the gap junctions on network properties, pharmacological blockers have commonly been used. However, it is very likely that most blockers have side effects on other types of membrane channels [Pan et al., 2007; Li et al., 2009]. Genetic knockouts are another possibility, but since gap junctions are widespread in development [Belousov, 2011] and allow chemical signalling between cells, there are likely to be side-effects [Simon and Goodenough, 1998]. Since direct experimental investigation has these problems, one approach to understanding is through modelling networks of the electrically coupled neurons [Calì et al., 2008].

48 hours after fertilization, tadpoles of the frog *Xenopus laevis* are approximately 5 mm in length (Fig. 4.1A) and respond to a brief touch stimulus by swimming for several seconds [Roberts et al., 2010]. The locomotive circuits generating swimming-type rhythms consist of ~2000 neurons subdivided into ~10 classes. These neurons form longitudinal columns, starting in the hindbrain and descending into the spinal cord. Lesion studies have shown that a small region of the nervous system, approximately 0.3 to 0.4 mm long in the hindbrain and rostral spinal cord, is sufficient to produce sustained swimming-like rhythms in response to stimulation [Li et al., 2006] (Fig. 4.1B). dINs play a central role in the generation of swimming rhythms and directly drive motoneurons (Fig. 4.1C). Whole-cell recordings from pairs of these dINs up to 200 µm apart have shown that they are electrically coupled [Li et al., 2009]. This coupling is likely to be axo-axonic via their very thin (< 0.5 µm), unmyelinated, longitudinal axons since they have dendrites with very limited longitudinal extents (< 30 µm). When gap junction blockers were applied swimming became unreliable, suggesting that the electrical coupling is crucial for reliable rhythm generation. Later analysis showed that the dINs are the first neurons to fire on each side on each cycle of swimming, and concluded that they are the drivers of swimming [Soffe et al., 2009]. Curiously, whole-cell patch-clamp recordings from dINs show they only ever fire a single action potential in response to step-current injection, a property which has been suggested is important for PIR-based rhythm generation at the network level [Sautois et al., 2007]. This single firing response seems at odds with the pacemaker properties during NMDAR activation which have recently been shown to be fundamental to the role of dINs in driving swimming within a single side of the CNS [Li et al., 2010]. All experimental analysis of this population of electrically coupled neurons is limited to whole-cell recordings from single neurons or pairs of neurons. In order to understand the effects of electrical coupling in this population, I have built a model of part of the dIN neuron population.

The aim was to build a computational model of part of a population of electrically coupled, rhythm driving neurons in the frog tadpole in order to conduct *ideal* experiments, facilitate investigation into the effects of different parameters and experimental protocols and monitor all the internal states of the system simultaneously. Specifically, I build a small population of model brainstem dINs and use it to ad-

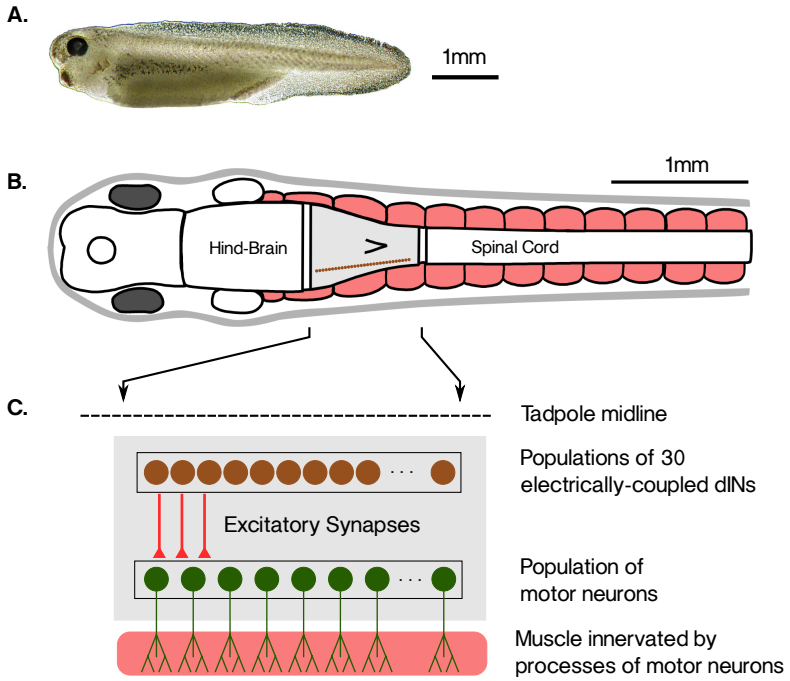


Figure 4.1 – A population of electrically coupled dINs in the hatchling tadpole CNS. **A.** Side view of a hatchling *Xenopus laevis* tadpole at Stage 37/38, 48 hours after fertilisation. **B.** Top view diagram of tadpole showing skin (grey), swimming muscles (pink), and CNS with hindbrain and spinal cord. The CNS region able to generate swimming rhythm when isolated (grey) contains a population of ~30 dINs (brown) on each side **C.** (B, expanded) On each side of the nervous system, a population of dINs makes excitatory AMPA synaptic connections onto motoneurons (green), which directly innervate muscle fibres.

dress the following questions about the significance of their electrical coupling: a) is it possible to infer the locations of axonal gap junctions using passive models of their somata and axons subject to anatomical constraints; b) what is the effect of axonal electrical coupling on experimentally determined firing properties of individual neurons; specifically, can it explain the observed single-spike firing in dINs, a property that is thought to be important in rhythm generation; c) what is the effect of axonal electrical coupling on overall network firing responses, and what significance might this have for the pacemaker properties of the dIN population?

4.3 RESULTS

4.3.1 *A passive model of electrically coupled descending interneurons (dINs)*

The electrical coupling between dINs in the tadpole is likely to be axo-axonic [Li et al., 2009] and the strength of coupling between two dINs will therefore be determined by the resistance of the gap junctions and their positions along the axon. The effects of neuronal morphology were taken into account by building multicompartmental models of dINs which included an axon. dINs have an electrotonically compact soma and dendrites that can be considered as a single isopotential compartment [Wolf et al., 1998], and a single thin axon ($< 0.5\,\mu m$) which descends towards the tail for 280 to $2050\,\mu m$ [Li et al., 2001]. In some cases dINs also have an ascending axon (Fig. 4.2A) but I have not considered these here. The surface area of dIN soma and dendrites were estimated from tracings of motoneurons with very similar size and morphology as $\sim 1200\,\mu m^2$, corresponding to a sphere of diameter $\sim 17.5\,\mu m$ and a small taper into the axon. A multicompartmental model was built with four sections to represent dIN morphology (Fig. 4.2C), which was further compartmentalised during simulations (see below). Initially, the model dINs were given a uniform leak conductance (g_{Lk}) over their surface, to match the dIN input resistance measured experimentally, an intracellular resistivity $R_i = 80\,\Omega cm$ and a specific capacitance $C = 1.0\,\mu F/cm^2$ [Koch, 1999]. The somata of the dINs driving swimming form a longitudinal column. The axons of dINs grow into the marginal zone (Fig. 4.2B), where they could contact dendrites or axons of other dINs. Based on experimental measurements of cell distributions and reconstructions from fills with neurobiotin [Li et al., 2001, 2006], I created a column of 30 model dINs in the hindbrain and rostral spinal cord, with somata spaced at $10\,\mu m$ intervals.

In the tadpole, current injections during paired whole-cell patch-clamp recordings have shown electrical coupling between dINs with somata up to nearly $200\,\mu m$ apart [Li et al., 2009]. The coupling coefficient is defined as the change in potential expressed as a percentage of the change in potential in another neuron where hyperpolarising current is injected. The coupling coefficient was found to decrease with distance between the two neurons, from ~15% to 5% (Li et al., 2009; Fig. 3A). The location of the gap junctions responsible for coupling has not yet been established ex-

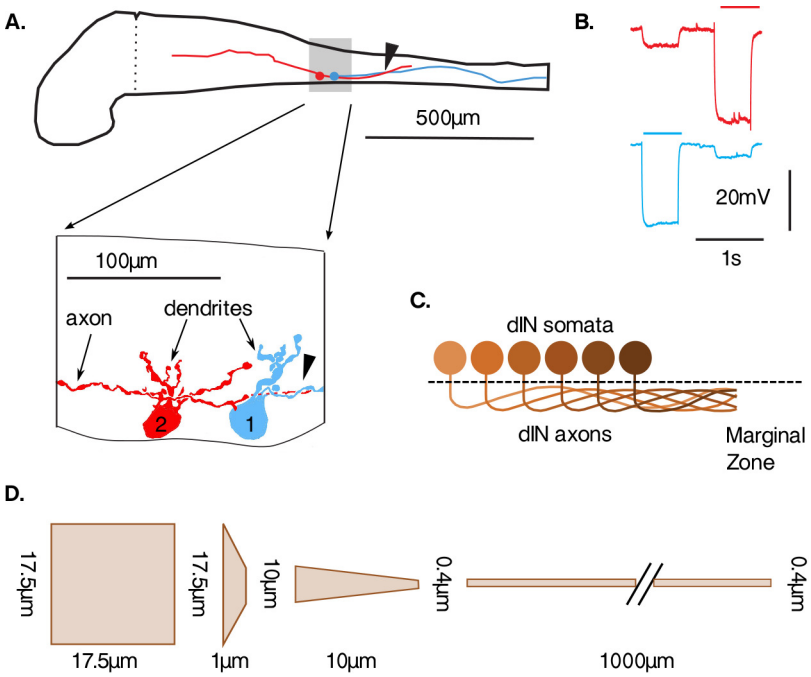


Figure 4.2 – Descending interneurons in the hindbrain and rostral spinal cord **A.** Side view of the hindbrain and spinal cord showing tracings of two filled dINs ([Li et al., 2009]: Fig. 1) with somata and short dendrites. dIN-1 (blue) only has a descending axon. dIN-2 (red) also has an ascending axon (arrow). The intertwining of the axons can be seen (arrowheads). **B.** Electrical coupling was shown using simultaneous voltage recordings from two dINs (red and blue). Coupling was present when hyperpolarising current injection into either dIN (bar) caused a small voltage deflection in the other (from [Li et al., 2009]) **C.** Diagram of dIN axons leaving the soma and intermingling in the marginal zone, where axo-axonic gap junctions could lie. **D.** To model a single dIN, morphology was approximated as four sections each modelled as a cylinder or tapered cylinders, one for the soma and axon, and two for the hillock (not to scale).

perimentally. They could be dendro-dendritic if the dINs are very close to each other, or axo-dendritic or axo-axonic if they are further apart. Since the dendrites are electrotonically compact, both dendro-dendritic and axo-dendritic coupling can be seen as a special case in which the distance along the axon from the soma-dendrite compartment was 0 µm and only axo-axonic coupling was investigated. An advantage of using the dINs in *Xenopus laevis* to study axo-axonic electrical coupling is that the layout is constrained to a single dimension; the location of a gap junction coupling two dINs can be expressed simply as the distance of the gap junction from the more caudal dIN.

4.3.2 *Inferring the locations of dIN axonal gap junctions using passive models of their somata and axons?*

I investigated how the numbers, strengths and location of gap junctions in the network would affect the electrical coupling between the dINs within the column (Fig. 4.3). The gap junctions were modelled as simple resistors of resistance of R_{GJ}. I tested several layout schemes for electrically coupling the neurons which were consistent with the anatomy and layout of the dINs. Each scheme created gap junctions between pairs of overlapping axons either: (i) at a fixed position from the caudal soma with a certain probability (Fig. 4.3C); (ii) with a fixed probability density over a defined region of the axon (Fig. 4.3D); or (iii) with a probability density defined as a function of the distance from the caudal somata (Fig. 4.3E). The layout of the network leads to more gap junctions forming caudally, because the axons descend (Fig. 4.3, 4.4). For a bundle of axons travelling in parallel, if the axons are thought of as cylinders of equal diameter, one axon can be in contact with a maximum of six others at any point. Therefore at each point of each axon, I limited the possible connectivity to a maximum of six neighbouring axons.

Gap junction distribution schemes were evaluated by examining the distribution of coupling coefficients in a population of 30 passive neurons. The dIN axons were compartmentalised according to gap junction density, in order to maintain simulation speed (compartment lengths: 5 µm in the hillock, 5 µm for the first 400 µm of the axons and 100 µm thereafter, electrotonic length in the axon is ~250 µm). In 50 simulations, hyperpolarising current was injected into a randomly chosen source neuron (N_{src}) and the steady-state voltage deflections measured in all neurons. Coupling coefficients between N_{src} and all other neurons were calculated and plotted against the distance between somata (Fig. 4.4).

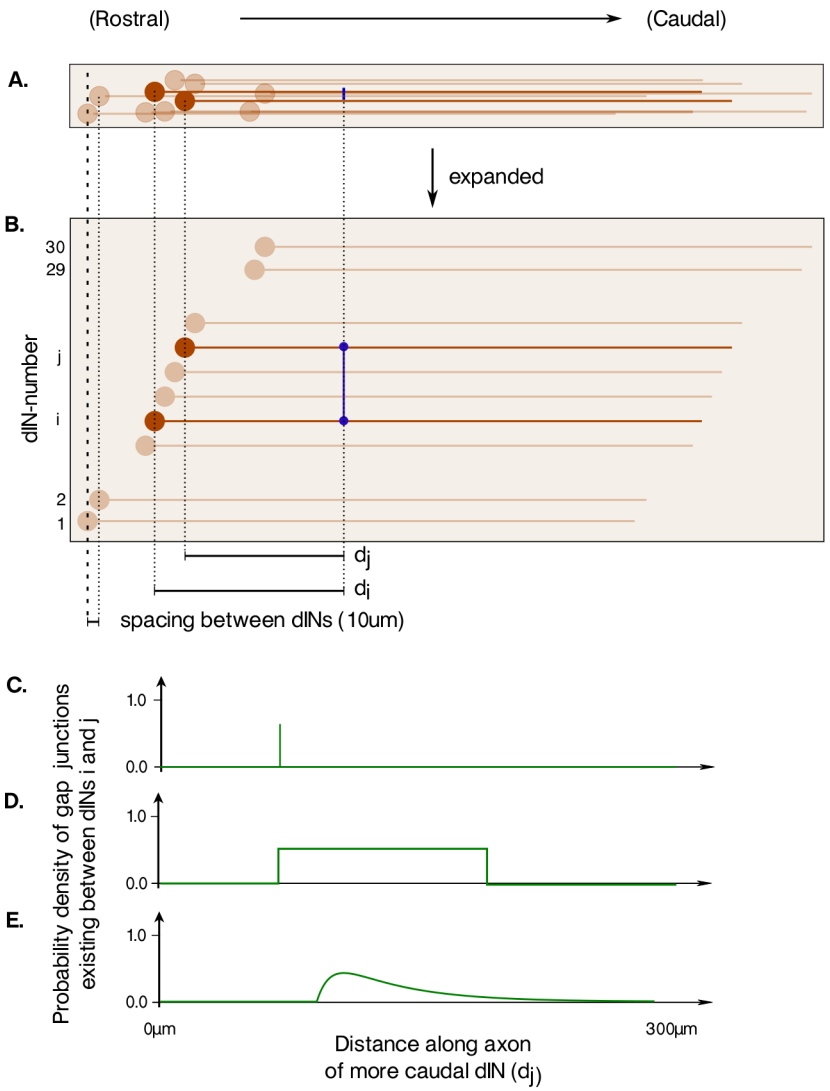


Figure 4.3 – Distributions of gap junctions in the linear network of dINs **A.** The somata of the dINs form a rostral-caudal column (brown: circles-somata, lines-axons). In this example, there is a single gap junction (blue) between a pair of dINs {i, j} (dark brown) (Not all dINs shown) **B.** (A-expanded vertically). **C-E**. Three layout schemes of gap junctions were tried to evaluate effects on coupling coefficients, for example: gap junctions form at a fixed point along the caudal axon with a given probability (C); gap junctions form only within a given region of the axon of the more caudal dIN with a given probability (D); and gap junctions form as a function (difference of two exponentials) of the distance along the axon of the more caudal dIN (E).

The small number of parameters in the system meant that each layout strategy could be tested using a parameter sweep through different values of g_{Lk}, R_i and R_{GJ}. Model networks were evaluated by the distribution of their coupling coefficients. I found most of the layout schemes and most parameters produced very low coupling coefficients between distant neurons compared to experimental observations. For all schemes, in order to get good coupling over distance, gap junctions needed to be close ($< 50\,\mu m$) to the somata of the more caudal neuron, but interestingly when the coupling was too close, although neurons closer together showed stronger coupling, this also reduced the strengths of coupling over long distances.

A gap junction layout scheme of the second type (uniform probability density) gave coupling coefficients similar to those observed experimentally, and was biologically plausible. Gap junctions were generated between overlapping axons with a fixed probability density of $0.015\,\mu m^{-1}$ over the first $30\,\mu m$ of the axon of the more caudal neuron (Fig. 4.3D). For each dIN, the initial $30\,\mu m$ of the axon was divided into $1\,\mu m$ bins. For each bin, six other dINs with axons at the same position along the rostrocaudal axis were picked at random. Each of these potential contact sites was given a fixed 0.015 probability of forming a gap junction. A range of gap junction resistances (R_{GJ}) were tested and a value of $600\,M\Omega$ gave coupling coefficients similar to those measured experimentally. This method generated 80-120 gap junctions in the network of 30 dINs which is about 5 to 8 hemichannels per axon (Fig. 4.4B-D). The small number of hemichannels on each axon suggests that much of the coupling between neurons could be indirect. About 25% percent of pairs of the neurons were directly coupled (Fig. 4.4B; blue dots) but most pairs were indirectly coupled via the axon of one other neuron (Fig. 4.4B; yellow dots). I found that when a dIN was electrically coupled in the network, its observed input-resistance, calculated by injecting hyperpolarising step current injections, dropped by ~50% compared to when isolated. The intracellular recordings of dINs are always of them embedded in an electrically coupled network, and therefore the experimental measurements of dIN input resistance comprise of the intrinsic leak conductance of that neuron itself as well as the current flow through gap junctions into surrounding dINs. To take this into account, the intrinsic leak conductance of the model neuron was reduced by 50%, so that the observed input resistances calculated by hyperpolarising current injections into an electrically coupled neuron were the same in the experiments and model. In this

thesis I use the term leak conductance to refer only to the current that flows through a neurons leak channels and does not include the flow through gap junctions.

Studies have suggested that in other systems, the minimum diameter of axons may be determined by the necessity of action potential propagation and prevention of spontaneous action potentials [Faisal and Laughlin, 2007]. The axons in tadpole are very thin (~0.3 to 0.4 µm), their diameters are difficult to measure accurately, in part because they can vary along their lengths. Therefore I investigated the effect of axon diameter on electrical coupling using modelling. I found that diameter had a drastic effect on the strength of the coupling in the network; reducing diameter to values of less than 0.3 µm led to very low coupling coefficients and therefore poor fits to the experimental data regardless of the other parameters or layout scheme used.

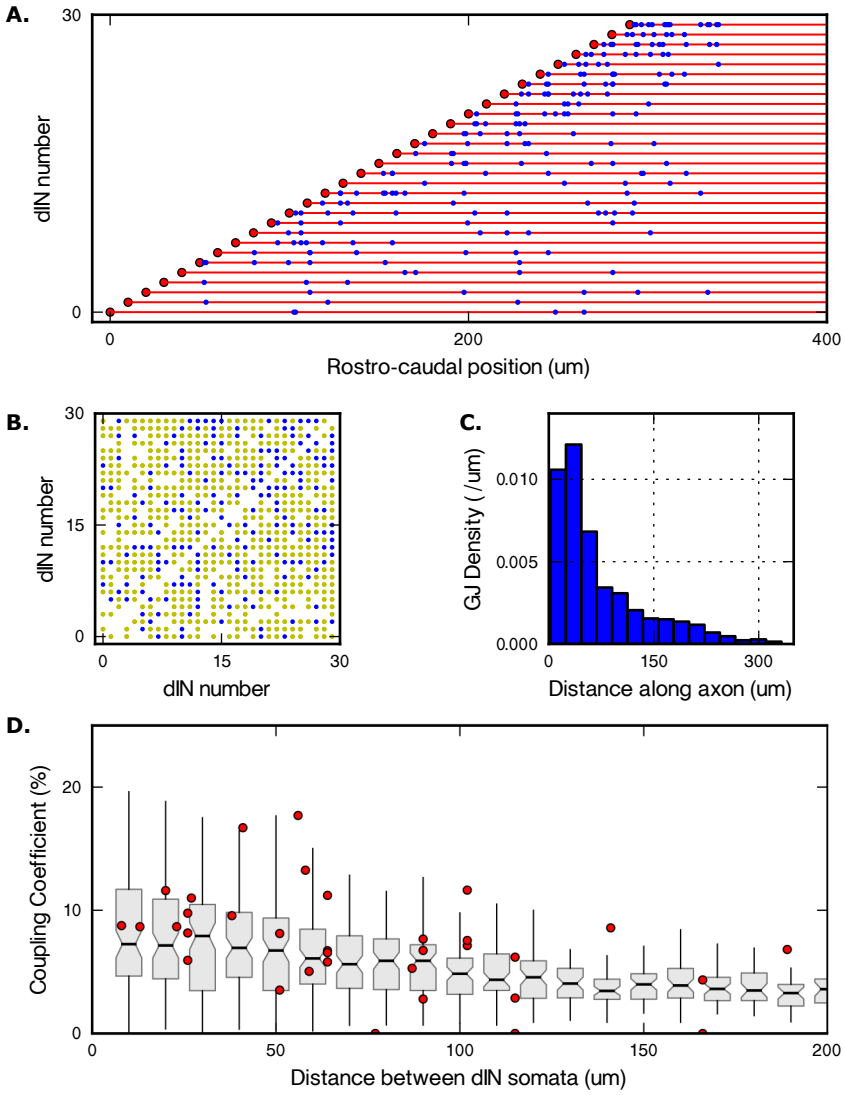


Figure 4.4 – Gap junction distribution and coupling coefficients between dINs **A.** The chosen layout of gap junctions in the column of 30 dINs (somata red circles spaced 10 µm apart) with descending axons (red lines). Blue dots represent one side of a gap junction (hemichannel), which connects to another gap junction at the same position along the column. **B.** Direct and indirect dIN to dIN coupling: directly coupled neurons (blue dots); indirectly coupled via the axon of another neuron (yellow dots); coupled via more than 3 axons (empty square). **C.** Histogram of gap junction distributions along the axons of dINs. In order to get good coupling between dINs, I found that gap junctions needed to form close to the somata. **D.** Comparison of the coupling coefficients between pairs of dINs measured experimentally (red circles) and in the model. 100 networks were generated, the notched grey bars show the median and 25% to 75% quartiles of the coefficient at each distance. The lines represent the 5% to 95% percentiles.

4.3.3 *Adding active channels to the model dINs and their axons*

The dINs play a critical role in driving the firing of other neuron types on the same side during fictive swimming as they are the first neurons to fire on each side on each cycle [Li et al., 2010]. It is known from current-clamp experiments that dINs have an unusual response to a step current injection and only fire a single action potential, even at levels of injected current up to twice rheobase [Li et al., 2006]. Furthermore, it has been proposed that dIN action potentials during swimming are partly the result of post inhibitory rebound following inhibition from the opposite side of the spinal cord [Soffe et al., 2001; Li et al., 2006]. Rebound firing is not seen when dINs are hyperpolarised from rest but during swimming they are depolarised [Roberts et al., 2010]. In line with this, while dINs are depolarised by injected current, short hyperpolarisations or IPSPs [Soffe et al., 2009] can lead to rebound firing (Fig. 4.6B).

The densities and kinetics of the voltage-gated channels play an important role in determining the firing properties of neurons [Hille, 2001]. In *Xenopus* tadpole spinal neurons, voltage-clamp experiments on dissociated neurons [Dale, 1995b] and neurons *in situ* [Winlove and Roberts, 2011] have suggested the presence of about 8 types of voltage-gated ion channels. In dINs, the majority of the voltage-gated currents are thought to be carried though a sodium channel (Na), a fast and a slow potassium channel (Kf, Ks) and a high-voltage-activated calcium channel (Ca). The kinetics of these channels were described in Chapter 2.

To evaluate the responses of model dINs and their axons to current injection, I used a parameter sweep over the channel densities and evaluated three behaviours at every point in the parameter space: a) whether the model neuron reliably only fired a single action potential in response to step current injections of 50, 100, 200 and 300 pA; b) whether an action potential initiated in the soma would propagate along the axon; and c) whether the model neuron would 'fire-on-rebound' to short, hyperpolarising current injections given during step current injections. Since variability is introduced into model parameters (see below), 10 neurons were evaluated at each point. In the first parameter sweep, channels were applied uniformly to the neurons. I found that, in general, increasing sodium and calcium channel density and reducing potassium

channels increased the excitability of the neurons (Fig. 4.5). Details about the parameter sweep are given in the Methods section.

I found it difficult to produce a dIN which could produce all three expected behaviours. With fewer calcium and sodium channels but more fast and slow potassium channels, dINs robustly fire only a single action potential (Fig. 4.5B) but its propagation along the axon only occurs in <10% of cases (Fig. 4.5D). With more calcium and sodium channels but fewer fast and slow potassium channels, dINs fire once at threshold (rheobase) but repetitively as current is increased (Fig. 4.5D). Action potential propagation occurred in 20% to 40% of tests (Fig. 4.5E). Interestingly, at high levels of sodium and calcium conductance, rhythmic firing could continue after the current injection or in some cases occur without any stimulation.

Why is action potential propagation unreliable in the model dIN axon? When gap junction resistance was increased from 600 MΩ to 2000 MΩ, action potentials were always able to propagate, suggesting that the failure to propagate was due to the shunting of current through the gap junctions. In other fine, unmyelinated axons densities of sodium channels up to 50 times those found elsewhere have been observed in the initial segment of the axon proximal to the soma [Wollner and Catterall, 1986; Kress et al., 2008; Kole et al., 2008; Grubb and Burrone, 2010]. In order to increase the proportion of action potentials propagating over the regions with 600 MΩ gap junctions, without increasing the overall excitability of the model dINs, I carried out a parameter sweep in which channels were not distributed evenly over the neuron. Initially, I varied the density of sodium and potassium channels in the entire axon independently of the rest of the neuron. Although this improved action potential propagation to ~40%, it also produced networks that were unstable, and fired repetitively without any input. The next step was to adjust the model so that a small region of the axon, (length of 70 µm, starting 20 µm from the soma) had higher densities of sodium channels than the remainder of the axon and the soma. I was able to increase the density of sodium channels dramatically in this small region by up to 10 times that in the soma, without the network becoming unstable. However, increasing above 4-5 times the density of sodium channels in the soma did not result in further improvement in action potential propagation. I therefore chose an increase factor of 5, which improved action potential propagation rates to over 50%. This rate is low but *in vivo*

dINs are likely to be active together so the current leak to other dINs will be less than when only a single dIN is active.

The final dIN model had properties close to those of real dINs (cf. [Li et al., 2006; Sautois et al., 2007; Soffe et al., 2009]): input resistance of 300 $M\Omega$; firing threshold of 80 pA, action potentials with similar rise times and durations, single firing in response to large step current injections (> 300 pA; Fig. 4.6A) and firing on rebound in response to short hyperpolarising current injections during a long depolarising step current injection. (Fig. 4.6B). The rise time of the model potential to current injection was slightly quicker than the physiology. This could be compensated for by the addition of an A-type potassium channel [Hille, 2001] but since its introduction had little effect on network dynamics, it has been omitted. Action potentials propagated more slowly over regions of gap junctions. The conduction velocity over a region with gap junctions was calculated as ~0.16 $m\,s^{-1}$, which increased to ~0.32 $m\,s^{-1}$ when they were removed (not shown). Conduction velocities of ~0.25 to 0.5 $m\,s^{-1}$ have been recorded in the central axons of sensory Rohon-Beard neurons [Clarke et al., 1984].

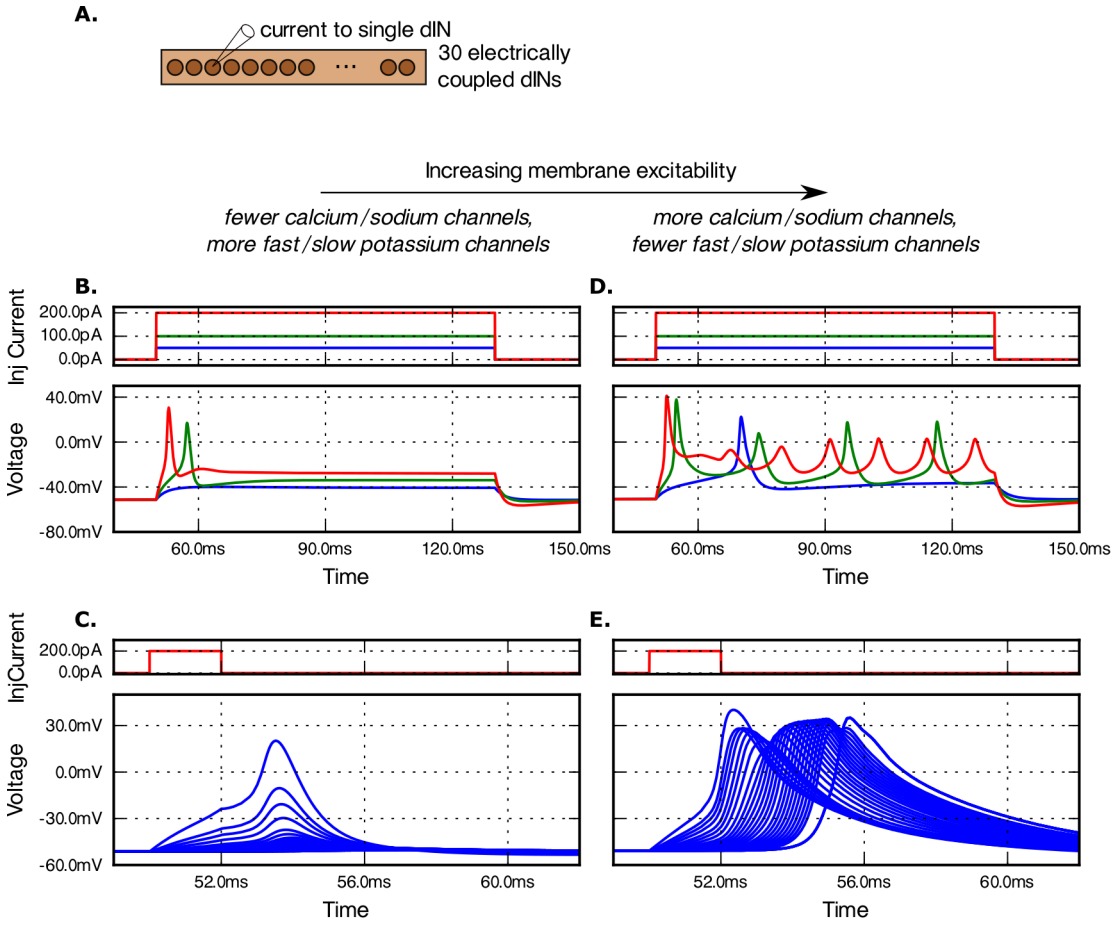


Figure 4.5 – Effects of uniform changes in densities of channels on model dIN membrane excitability, firing and action potential propagation **A.** Tests were conducted on a population model of 30 dINs electrically coupled via axo-axonic gap junctions. **B, C.** When the membrane excitability is low, model dINs reliably fire a single action potential at the onset of current injection (B) but would not reliably conduct action potentials along their axons (C). **D, E.** With increased excitability, the model neurons fire repetitively to higher levels of current injection and **E.** the reliability of action potential propagation increased. [C, E] Action potential propagation in the model was tested by recording the voltage in the soma and at 20 points along the axon (20 µm spacing) of a random neuron in the network. The voltage traces are shown by the different colours. A short current pulse injected into the soma is used to initiate an action potential (first blue trace). (The values for k_X in these traces were [B, C]:k_{Ca} = 0.5, k_{Na} = 1.0, k_{Kf} = 1.0, k_{Ks} = 1.5, and k_{Lk} = 0.5 and [D, E]:k_{Ca} = 1.0, k_{Na} = 1.5, k_{Kf} = 0.5, k_{Ks} = 0.5, and k_{Lk} = 0.5)

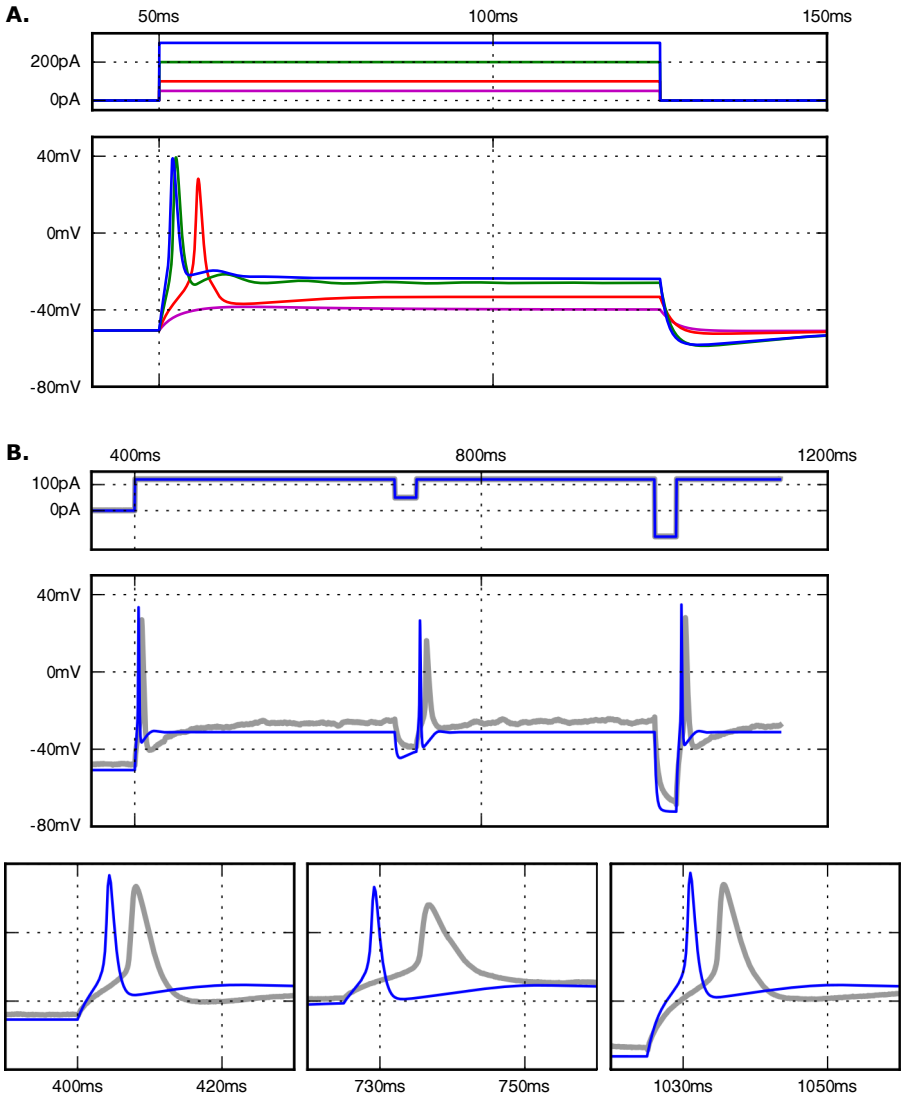


Figure 4.6 – Responses of final active dIN model to current injections. **A.** In response to increasing levels of step current injection (top), the model neuron soma has a firing threshold of ~80 pA (red), and only ever fires a single action potential, even in response to currents of 3 times the rheobase (blue). **B.** Rebound spikes to negative current steps (middle; expanded in bottom traces) during depolarisation. Model responses (thin blue line) and whole-cell somatic recordings (thick blue line; from Dr Wenchang Li, unpublished).

4.3.4 *What is the effect of electrical coupling on experimentally determined firing properties of individual neurons?*

All the known classes of tadpole spinal neurons in the circuits underlying swimming fire multiple action potentials in response to a positive current injection with the exception of dINs and sensory RB neurons [Sautois et al., 2007], which reliably only fire a single spike in response to step-current injections up to rheobase. Previous models of dINs have relied on specific sodium-channel kinetics to produce a single action potential to step current injection [Sautois et al., 2007]. In contrast, the model dIN was built by adjusting conductance densities of physiologically characterised voltage-gated channels within an electrically coupled network. I therefore investigated whether the single spike firing was intrinsic to the properties of individual dINs or a result of the electrical coupling.

I compared the response of a model dIN to step current injections in an electrically coupled network, and in isolation (Fig. 4.7). When electrically coupled within the network, dINs only fired once to all levels of injected current (Fig. 4.7B). Removing all coupling in the network caused the dIN to start firing repetitively to the same levels of step current injections (Fig. 4.7C). This suggests that the single spiking response is the result of coupling the dIN via gap junctions to other dINs.

Why does the electrical coupling lead to single spike firing? Consider a dIN which is axo-axonically coupled via gap junctions to other dINs. Initially, when there is no input to any dIN, all regions of the neurons are isopotential, and so no current flows from the soma into the axon. On slight depolarisation of the soma, current will flow from the soma, into the axon, through gap junctions and into the other neurons, because the coupled neurons are not depolarised. However, if all the dINs are driven synchronously, the neurons will be isopotential and so no current will flow across the gap junctions. In effect, the electrical coupling provides a hyperpolarising current that depends on the relative membrane voltages of the other neurons (Fig. 4.8).

Interestingly, since the driving force across a gap junctions is proportional to the voltage difference of the two neurons, the surrounding neurons would not necessarily need to fire in exact synchrony for this modulation to occur and even subthreshold depolarising stimulation to the surrounding neurons may be sufficient. For example, if

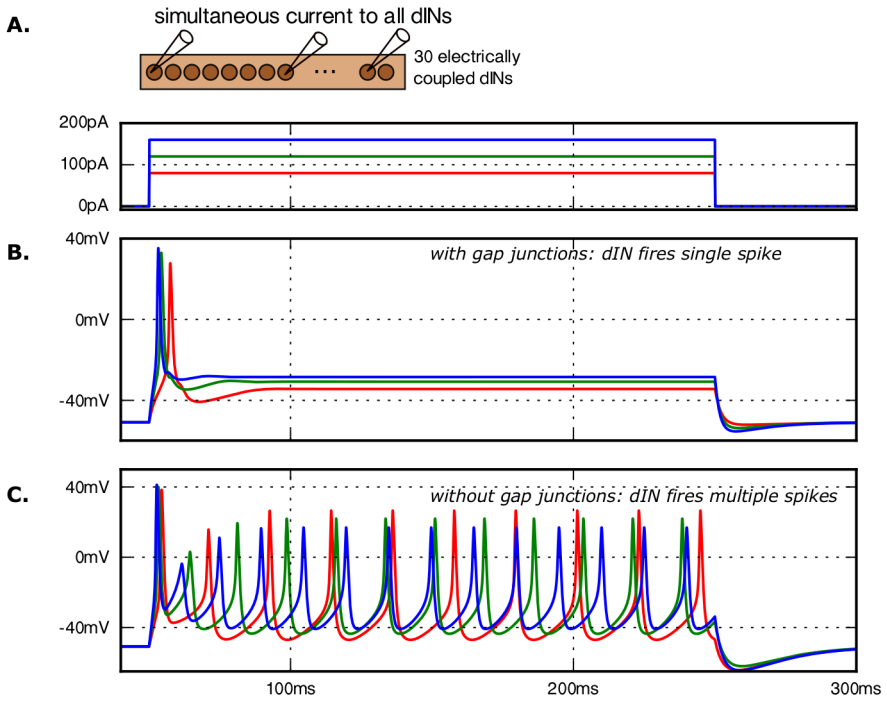


Figure 4.7 – Effects of electrical coupling on firing properties of network of 30 electrically coupled dINs **A, B.** Injection of 200 ms step current into a single electrically coupled dIN reliably produces only a single spike, in response to increasing levels of current injection (red, green, blue) **C.** When gap junctions were removed, the injected dIN fires repetitively.

the surrounding neurons were stimulated so their membrane potentials were brought from resting (−52 mV) to close to the firing threshold (−35 mV), then magnitude of this hyperpolarising current to a neuron which was also close to the firing threshold (−30 mV) would be reduced to less than a third of the magnitude of that when the network was at rest.

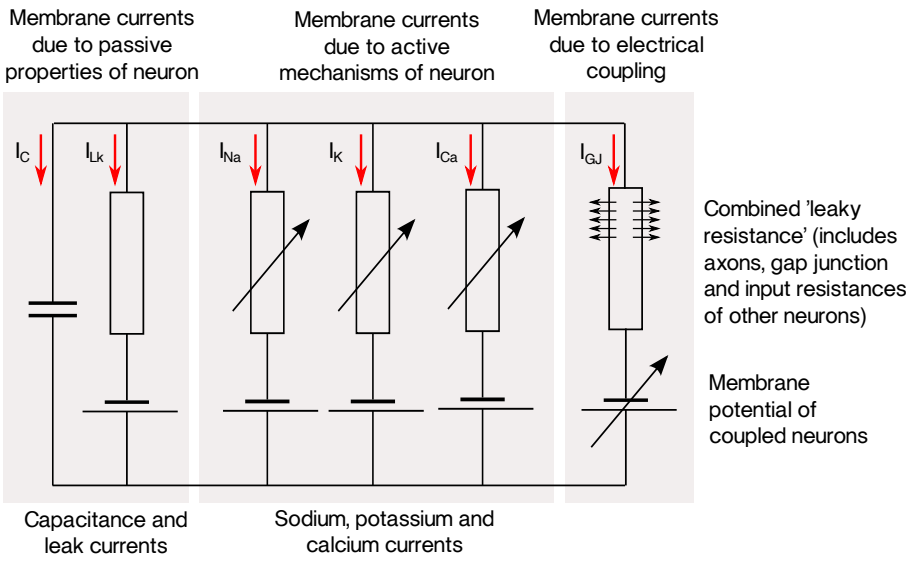


Figure 4.8 – Simplified circuit diagram of membrane and gap junction currents in the dINs. In addition to active and passive currents due to the neuron itself, a current flows from the soma, into the axon, through gap junctions and eventually into the somata of other neurons. This current is dependent on the relative membrane potential of the coupled neurons. (Although illustrated in terms of a simple resistance and reversal potential, the exact calculation is more complex and requires the use of cable theory)

4.3.5 *What is the effect of axonal electrical coupling on overall network firing responses?*

In physiological experiments recording can only be made from one or two neurons at the same time. The modelling led to the hypothesis that the single spiking response seen physiologically to step current injection into a single dIN is caused by shunting of current into nearby dINs whose membrane voltages remain close to rest. To investigate this hypothesis, I injected a 200 ms step current pulse into all 30 dINs in the electrically coupled network simultaneously (Fig. 4.9A). With sufficient current, all the dINs in the population fire repetitively during the current injection (Fig. 4.9B) and as the current level was increased the frequency of dIN firing increased to reach a plateau at ~80 Hz (Fig. 4.9E).

I investigated the effect that the gap junctions played in synchronising network firing by reducing the number of gap junctions in the network. Electrically coupled networks were built as before, but in different simulations the probability of gap junctions forming in a given location was reduced by 0, 25, 50 or 75% (Fig. 4.9C).

As the level of coupling was reduced, firing became less synchronised, revealing the variability in the dIN population (Fig. 4.9C, D).

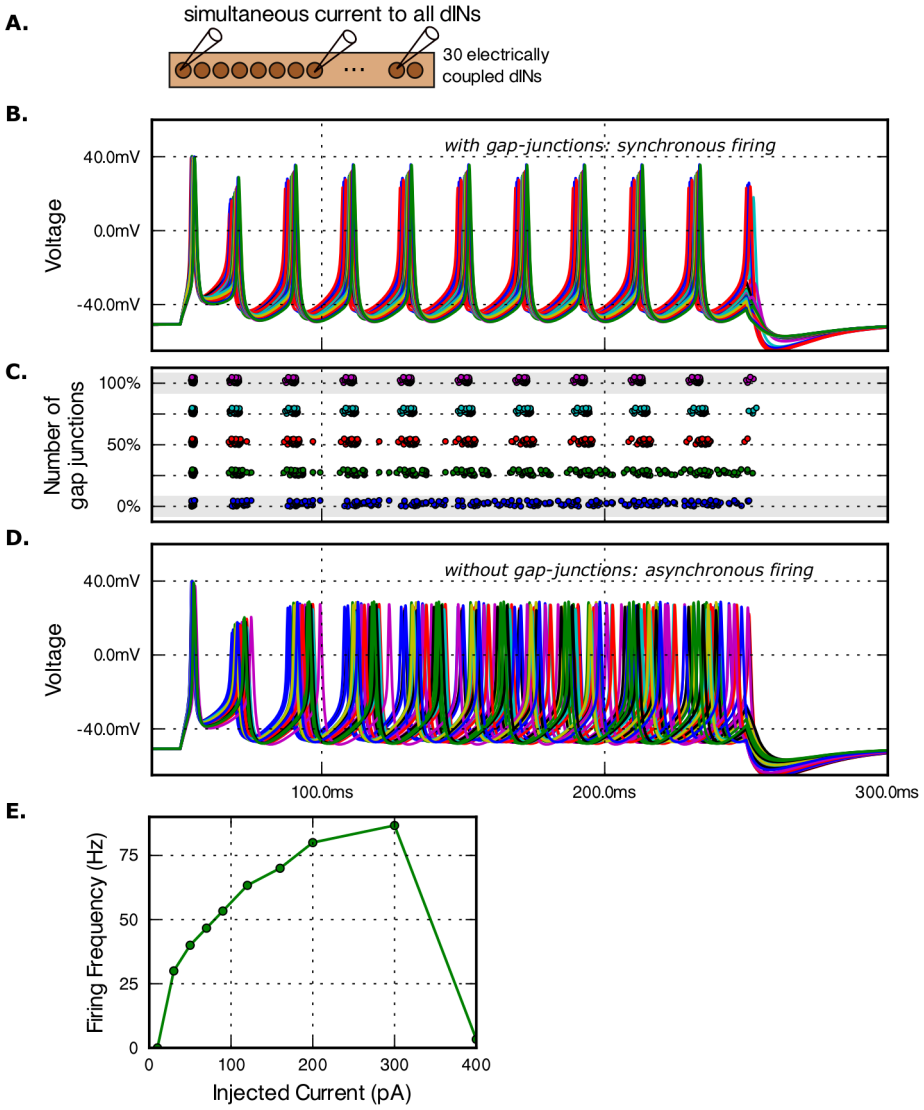


Figure 4.9 – The effects of electrical coupling on firing properties of network of 30 electrically coupled dINs. **A, B.** When current is injected into all the electrically coupled dINs, the whole population fires repetitively, and shows strong synchronisation. **C.** Raster plots of spike times show that as the number of gap junctions in the network was reduced, neurons still fired repetitively but lost synchronisation. The percentage of gap junctions in the network is plotted on the y axis) **D.** Firing is desynchronised with 0% gap junction coupling. **E.** Current-frequency curve for a dIN in the electrically coupled network. (100% gap junction coupling, equal current injected into all dINs).

4.4 DISCUSSION

I built models matching experimental results in order to investigate a small population of dIN neurons in the central nervous system that drive swimming in young frog tadpoles. I used parameter sweeps to investigate parameters that were not well specified experimentally and found gap junction layouts and conductance densities that produced a model that could match experimental observations and had constrained parameters. Using a reduced preparation with well-studied functional properties, modelling suggests that for long range axo-axonic electrical coupling it is necessary that gap junctions form as close to the soma as possible (>50 µm). Although most pairs of dINs are electrically coupled, this is mostly indirect and only about 20% of these pairs are directly coupled by a gap junction. I found that the diameter of the thin axons has a strong influence on coupling strength and action potential propagation; experiments in which the diameter of the axons were reduced to <0.3 µm produced very weak levels of coupling. Action potentials in these thin axons failed to propagate over regions containing many gap junctions unless they had a higher density of sodium channels and a lower density of potassium channels in the initial segment of the axon than in the soma.

It was important that the modelled neuron was embedded in the electrically coupled network when it was being matched to experimental data. The electrical coupling was shown to have a dramatic impact on the firing of the dINs. The single spike firing observed in the model was a direct consequence of their electrical coupling, and when this was removed they would fire repetitively. I have shown how the firing of an individual neuron can be modulated by current flow to coupled neurons, and how this leads to pacemaker-like firing in a simultaneously stimulated population of neurons. I have demonstrated a model that is consistent with experimental observation, which is able to explain how dINs are able to act as pacemakers during swimming although they only fire single action potentials in response to experimental step-current injections. Electrical coupling is proposed to play many roles in nervous systems, from ionic transport between cells [Langer et al., 2012], to producing rapid responses [Furshpan and Potter, 1959], synchronising action potentials and contributing to computation [Marder, 1998; Kiehn and Tresch, 2002]. The modelling of axo-axonic coupling

within a linear population of brainstem neurons raises questions about the axonal distribution of gap junctions, spike propagation past gap junctions in fine unmyelinated axons, the effects of coupling on the apparent firing properties of neurons recorded during experiments, and the role of coupling in the operation of a rhythmic network controlling locomotion. Another additional possibility: causing changes to neuronal firing patterns dictated by the behaviour of surrounding neurons. In the model, electrical coupling effectively provides a current that acts to hyperpolarise the neuron, which is modulated by the sub-threshold membrane voltages of surrounding coupled neurons. I have shown how the interaction of this current with other membrane currents could allow the behaviour of a neuron to be regulated by the subthreshold activities of surrounding neurons. I will now consider these questions in turn.

4.4.1 *Effects of gap junction distribution on long-range, axo-axonic electrical coupling*

Electrical coupling is widespread and serves a range of functions. The coupling between tadpole dINs which form a longitudinal column in the hindbrain acts over distances up to 200 µm and therefore is most probably via axonal gap junctions [Li et al., 2009]. Such long-range axonal coupling has been found in the mammal brain but is unusual [Bennett and Zukin, 2004]. Unfortunately, there is no anatomical evidence yet about the distribution of neuronal gap junctions in the tadpole. I used parameter sweeps over several different distribution algorithms to reproduce both the strength and variability of coupling seen between the tadpole dINs particularly over long (>100 µm) distances during experiments. The modelling suggests firstly, that gap junctions need to be within 30 µm of the soma to get realistic coupling, and secondly, that much coupling is indirect. However, adding gap junctions does not necessarily increase coupling, particularly if a more proximal gap junction already exists between a pair of axons. Realistic coupling was achieved using around 100 gap junctions in the network of 30 neurons. This suggests that most coupling is indirect, so two dINs are coupled via the axon of a third.

In the model, removing the electrical coupling between the dINs approximately doubled the observed input resistance of a dIN, which is a much larger than the <10% increase observed experimentally when a pharmacological gap junction blocker was

applied (see Fig. 4 in Li et al., 2009). This suggests that the pharmacological block of gap junctions is not equivalent to removing gap junctions from the models. Since 90% of the dINs recorded were coupled with a coefficient of between 5 and 20% [Li et al., 2009], one possibility is that the gap junction blockers may have effects on other membrane channels besides blocking electrical coupling [Srinivas et al., 1999; Juszczak and Swiergiel, 2009].

4.4.2 *Problems modelling small unmyelinated axons*

Neurons in the developing tadpole nervous system have very fine, unmyelinated axons (<0.5 µm in diameter). Despite their widespread occurrence, such fine axons are not well understood because it is so difficult to obtain recordings from them (e. g. hippocampal mossy fibres, rod-cell axons, parallel fibres in granule cells, olfactory receptor axons, [West et al., 1982; Kress et al., 2008; Perge et al., 2012]. In general, thicker axons will have faster action potential propagation velocities, but will consume more energy [Perge et al., 2012]. Studies have investigated the role of axonal diameter, and suggested that although the minimum diameter to house the 'molecular machinery' is quite small, axons narrower than 0.1 µm may be prone to spontaneous initiation or failure of action potentials due to the stochastic nature of ion channel opening and closing [Faisal et al., 2005; Faisal and Laughlin, 2007]. In the modelling of the electrically coupled network of hindbrain dINs, axon diameter was found to be critical to achieving realistic coupling. Passive coupling between dINs could not be achieved with an axon diameter of < 0.3 µm regardless of the gap junction layout algorithm and parameters. Since electrical coupling is critical to the reliability of swimming in the tadpole [Li et al., 2009], the modelling suggests there is a limit on minimum axon diameter for a dIN.

The effect of the densities and spatial distributions of membrane channels has been investigated in axons from many systems [Catterall, 1981; Safronov et al., 1997; Boiko et al., 2003; Schmidt-Hieber et al., 2008; Kress et al., 2008]. Modelling the population of tadpole dINs as a passive network suggests that most of them were coupled indirectly via the axons of other dINs. This hypothesis is supported by the failure of action potential propagation over regions of axons with high densities of gap junctions. Such

failures are in contrast to physiological recordings that show action potentials initiated in one dIN reliably cause EPSPs in more caudal neurons [Li et al., 2006]. Simply increasing the density of sodium channels uniformly in the whole neuron or over the entire of axon produced networks that were unstable and could fire repetitively even without stimulation. Action potential propagation became more reliable when the density of sodium channels was increased and the density of potassium channels was decreased in the initial segment of the axon close to the soma.

Higher densities of sodium channels have been observed experimentally in the axon hillock and initial axon segment in other neuron types [Catterall, 1981; Safronov et al., 1997; Boiko et al., 2003; Kress et al., 2008]. It could also be that there is localised clustering of sodium channels around the gap junctions, as seen at the nodes of Ranvier [Ritchie and Rogart, 1977; Zeng and Tang, 2009]. Another possibility is the existence of two types of sodium channel; those near the gap junctions having a higher voltage-activation threshold allowing them to reliably conduct action potentials, without initiating them. Another possibility comes from the duration of action potential, which is long in dINs compared to other neuron types in the tadpole [Sautois et al., 2007]. This was difficult to match in the model and the mechanism underlying this long duration remains unclear; however in other systems, long duration action potentials have been suggested to improve conduction reliability in unmyelinated axons [Bostock et al., 1978].

In the model, I could not get both robust individual neuron firing properties and reliable action potential propagation from a single dIN within a network at rest. I chose to prioritise robust single spiking dIN behaviour at the expense of reliable action potential propagation. Although this is not consistent with what is observed experimentally, in future modelling this is unlikely to have significant functional effect, since under normal circumstances, the population of dINs tends to receive similar input, reducing the voltage drop across the gap junctions and allowing action potentials to propagate. From an experimental perspective, the modelling suggests that in electrically coupled axons, action potential propagation failure may occur in the axon near gap junctions particularly when the network is at rest. If this is the case, then when paired recordings are used to investigate whether an action potential in a presynaptic neuron leads to an EPSP in another, the absence of a EPSP may be the result

of propagation failure and does not necessarily imply there is no synapse between the neurons.

4.4.3 *Effects of gap junction coupling on apparent firing properties of neurons*

An unexpected outcome of modelling the dIN population was the demonstration of the significant effect that electrical coupling could have on their firing properties. In vivo, in an unexcited network, step-current injection into a single dIN will lead to a single action potential [Li et al., 2003, 2006; Sautois et al., 2007]. The modelling suggests that a neuron that fires multiple action potentials in isolation can be transformed into one that only ever fires a single action potential at the onset of a step current injection, by electrically coupling it to other neurons.

In both life and the model, when a single dIN in an electrically coupled population is driven with a step current injection, it fires a single spike, and subsequent spikes are prevented (Fig. 4.3A). I conclude that in the model this is caused by a combination of; 1) sodium channel inactivation and activation of a slow potassium channel; 2) a current flowing through the gap junctions that acts like a leak to hyperpolarise the neuron. Both of these mechanisms are needed to prevent successive action potentials; individually they are not sufficient. I infer this because just prior to the first spike, although currents will flow across gap junctions, there is no sodium inactivation or slow potassium channels activation and the neuron is able to spike. Similarly, when all the surrounding neurons are also depolarised, (Fig. 4.5C) there is no current flowing through the gap junctions, and the sodium inactivation and potassium currents are unable to prevent repetitive spiking. This would explain why dINs only ever fire a single spike in response to step current injections during whole-cell recordings. Currently it is impossible in experiments to inject current simultaneously into 30 individual dINs. However, it is possible to depolarise many dINs at the same time by perfusion of NMDA, and this leads to repetitive firing [Li et al., 2010] just as I have found with current injection into the model dIN population.

It is interesting that Dale proposed a similar explanation for the single action potential response to current injection found in some neurons during sharp-electrode recordings [Dale, 1995b]. He suggested that the electrode damaged the neuron to

produce a *leak current* which would have played a similar role to the current sink I find due to gap junctions. (More recent recordings suggest that although attaching the sharp electrodes effect the leak of the neurons, this did not have an effect on the cell activity, and moreover that attaching whole-cell patch-clamps did not have any significant effect on the neurons [Li et al., 2004a]).

4.4.4 *Significance of electrical coupling and firing properties for rhythm generation*

Two proposals have been made about the mechanisms producing swimming-type activity in the tadpole. The first is a pacemaker mechanism [Li et al., 2010; Dale, 1995a, 2003]. It has been shown that when NMDA is applied to an isolated single side of the hindbrain, the dINs fire rhythmically at frequencies similar to those seen during swimming, without the need for inhibitory feedback [Li et al., 2010]. The second is a network mechanism [Sautois et al., 2007; Roberts and Tunstall, 1990; Tunstall et al., 2002] where reciprocal inhibitory synaptic connections between the two sides of the CNS play an important role in rhythm generation by producing PIR firing in dINs already depolarised by their own mutual NMDAR-mediated excitation. It is likely that both mechanisms play a role in rhythm generation in the tadpole. However, a problem arises in marrying these two proposals. According to the cellular pacemaking proposal, the dINs will fire repetitively in response to tonic excitation. However, according to the network hypothesis, during swimming, both sides of the network are maintained at a level of excitation, and only generate spikes following brief inhibition from the opposite side because of PIR. The first proposal is supported by the lesion experiments showing that a single side of the nervous system is able to generate rhythm in the presence of NMDA [Li et al., 2010]. The second proposal is supported experimentally by whole-cell patch-clamp recordings of long-duration step-current injections into the dINs, which only ever produce a single spike even to amplitudes up to twice rheobase (Fig. 4.2C).

I have demonstrated a model in which these two proposals can coexist. The modelling suggests that the experimental observation of robust single-spike firing of the

dINs to step current injections could result from a current flowing into other electrically coupled dINs. This effectively makes the dINs much less excitable. When the entire population is stimulated when swimming is initiated, the dINs will fire like pacemakers. This does not mean that post inhibitory rebound does not play an important role in pattern generation, particularly in maintaining the antiphasic relationship between the side, but it might not be essential to the repetitive firing of the dINs during swimming.

Many hypotheses have been made in this chapter, and with suitable histological techniques, it might be possible to investigate the distributions of sodium and potassium channels on the cell membrane. Unfortunately, many hypothesis are difficult to test because of current experimental limitations: the gap junction blockers do not seem to act as would be expected; it is difficult to isolate dINs electrically to investigate their firing properties and the gap junctions have been difficult to locate because of their small size. A summary of the hypotheses made from the modelling are given in Chapter 7.

5

SUSTAINED PACEMAKER ACTIVITY IN A POPULATION OF DINS

5.1 ABSTRACT

Rhythmic activity is widespread in nervous systems and is central to life processes. Rhythms producing motor-output are often generated in the CNS by a population of neurons. What cellular and network properties facilitate reliable rhythm generation that can be initiated and terminated by brief synaptic input? I investigated how a population of brainstem neurons that drive swimming locomotion in young frog tadpoles can be switched on and off via synaptic pathways. I built a biologically realistic computational model of the minimal 30 electrically coupled neurons in the hindbrain and spinal cord, which produce rhythmic firing that drives tadpole swimming. Based on experimental estimates for population sizes, synapse strengths and connectivities, I showed that the NMDA synapses between the members of the population would allow the network to sustain rhythmic pacemaker firing following a brief synaptic excitation. I investigated the effects of the voltage dependency of the NMDA channel due to magnesium block, and found that it doubles the range of synaptic feedback strength over which the network is able to sustain rhythm. By driving the network with synaptic input from defined excitatory and inhibitory sensory pathways, I showed that the network can be switched on and off at short latency. I generalised the result by demonstrating that a population of Hodgkin-Huxley type neurons with NMDA feedback can also be switched on and off synaptically, and propose that the key feedback mechanisms revealed in this network could operate in other networks.

5.2 INTRODUCTION

Rhythmic activities are fundamental to an animal's survival. From maintaining a regular heartbeat and chewing food, to swimming through the ocean and galloping over the desert, a core requirement of all CNS is effective control of rhythmic processes. In many cases rhythmic motor patterns are generated intrinsically within the nervous system by the firing of populations of interconnected central pattern generator neurons [Brown, 1914; Nakamura and Nobuo, 1995; Feldman et al., 2003; Kristan et al., 2005; Goulding, 2009; Selverston, 2010]. The mechanisms allowing these networks to generate activity are well studied across vertebrates and invertebrates, although many important questions, germane across all fields of neuroscience, remain open, such as: the relative importance of the properties of individual neurons as distinct from network dynamics; the roles of electrical and chemical synapses; and the mechanisms of neuronal synchronisation and oscillations. The key mechanisms involved in rhythm generation in particular systems are well characterised and it has been possible to construct computer models of the networks which produce output patterns similar to those observed in real animals (locust flight: [Wilson and Waldron, 1968]; sea slug swimming:[Getting, 1983; Calin-Jageman et al., 2007]; leech swimming:[Taylor et al., 2000; Kristan et al., 2005]; salamander walking: [Ijspeert, 2001]; stomatogastric ganglion: [Marder and Bucher, 2007]; tadpole swimming: [Sautois et al., 2007]).

What is less clear is how rhythmic activity is controlled. Under many circumstances brief sensory stimuli to an animal lead to a change of state, for example, beginning to walk or adopting a new posture. In both cases, the brief stimulus leads to a long-lasting change in the motor output from the CNS. This raises the question of how networks which generate rhythmic activity in the CNS can be switched on and off. The processes governing the initiation and termination of activity in such rhythmic networks are less understood. Rhythmic systems are often chosen for study because of their simplicity and well defined outputs. Unfortunately, even the simplest of such circuits are highly non-linear dynamical systems often with complex input pathways. In this chapter, I investigate the roles of cellular pacemaker properties, feedback excitation, electrical coupling and start/stop signals in the operation of the network generating swimming rhythms in the hatchling tadpole.

The hatchling tadpole responds to brief touch stimulation with swimming lasting several seconds. The specific populations of CNS neurons involved in generating the swimming rhythms have been well studied. When the CNS is viewed from the top (Fig. 5.1A), the spinal cord expands into the brainstem or hindbrain and surgical isolation has shown that sufficient neurons to generate a swimming rhythm lie in this tapered region (grey in Fig. 5.1A,B). During swimming, neurons on each side of the rostral hindbrain and spinal cord fire a single action potential in antiphase with those on the opposite side and drive the firing of motor neurons. This produces alternating contractions on each side of the body at frequencies from 10 to 25 Hz which progress from the head to the tail [Kahn et al., 1982]. On each side, on each cycle of swimming, a population of hindbrain and spinal cord neurons with descending axons (dINs) play a critical role [Li et al., 2006]. They are the first neurons to fire action potentials on each cycle of swimming, providing synchronous excitatory synaptic drive to the other ipsilateral neurons [Soffe et al., 2009]. The antiphasic timing of firing on each side is organised by a population of reciprocal inhibitory commissural neurons.

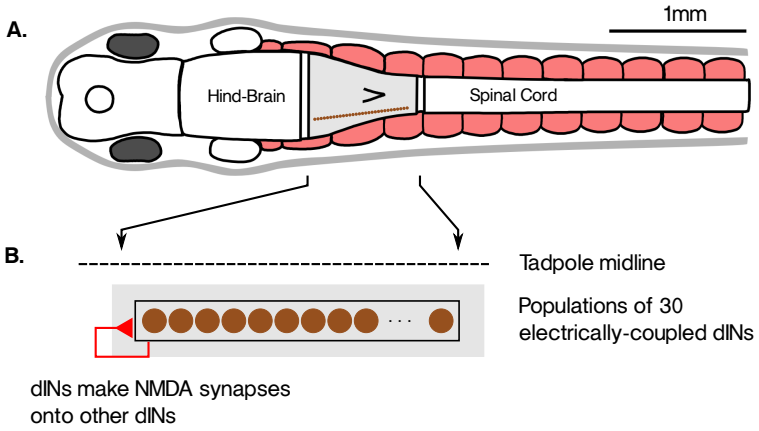


Figure 5.1 – A population of electrically coupled dINs with feedback NMDA synapses. in the hatchling tadpole CNS. **A.** Top view diagram of tadpole showing skin (grey), swimming muscles (pink), and CNS with of hindbrain and spinal cord. An isolated half of the nervous system is able to maintain rhythmic activity. The CNS region able to generate swimming rhythm when isolated (grey) contains a population of ~30 dINs (brown) on each side **B.** (A, expanded) On each side of the nervous system, a population of dINs which makes excitatory feedback NMDAR synapses onto themselves.

Network mechanisms involving PIR and cellular pacemaker properties have been proposed to complement each other in producing rhythms in many networks, from

molluscs to mammals, [Perkel and Mulloney, 1974; Satterlie, 1985; Marder and Calabrese, 1996; Calabrese and Feldman, 1997; Grillner, 1999; Arbas and Calabrese, 1987; Angstadt et al., 2005; Arshavsky, 2003; Bertrand, 1998]. This also seems to be the case in the tadpole. The network mechanism [Roberts and Tunstall, 1990; Tunstall et al., 2002; Sautois et al., 2007] involves reciprocal inhibitory synaptic connections between the two sides of the CNS, which play an important role in rhythm generation by producing PIR firing in dINs already depolarised by their own mutual NMDAR-mediated excitation [Sautois et al., 2007]. The cellular pacemaker mechanism proposes that swimming rhythms are generated primarily by the slow, repetitive firing properties of the dINs [Dale, 1995a, 2003; Li et al., 2010]. It has been shown that when NMDA is applied to an isolated single side of the hindbrain the dINs fire rhythmically at frequencies similar to those seen during swimming [Li et al., 2010]. Since the dINs form excitatory NMDA synapses onto each other [Li et al., 2006], it has been proposed that once the dIN population becomes active, the long-lasting reciprocal NMDAR activation provides a background tonic drive to the population and produces self-sustained rhythmic output.

How is the swimming rhythm produced and controlled in the hatchling tadpole? Lesion studies in immobilised tadpoles have shown that a region of hindbrain and rostral spinal cord 0.3 to 0.4 mm long containing only about 30 dINs on each side is able to produce the fictive swimming pattern for many seconds in response to a brief current stimulation (Fig. 1; [Li et al., 2006]). In the unlesioned tadpole, brief stimulation to the head or trunk initiates swimming via short sensory pathways which activate the dINs [Roberts et al., 2010; Buhl et al., 2012]. Swimming can continue for several seconds but when the tadpole swims into a solid object causing pressure to the front of the head, the tadpole immediately stops swimming and attaches to the object with mucus. A GABA-ergic inhibitory pathway terminating swimming following such head stimulation has been defined [Perrins et al., 2002].

Previous modelling work has investigated the network mechanisms for generating rhythm and shown they are able to generate swimming-type rhythms [Sautois et al., 2007; Roberts et al., 2008]. The modelling of an electrically coupled population of dINs in the previous chapter suggested that although reliable single-spike firing is recorded in dINs in response to step current injection, swimming rhythms could still be

generated within a single side of the nervous system without the need for inhibition. It was suggested that isolated dINs could fire repetitively to step current injection and that the observed single-spike firing seen experimentally was the result of electrical coupling to other, hyperpolarised neurons. It was demonstrated that simultaneous stimulation to all the neurons would result in synchronous, repetitive firing. The pacemaker hypothesis supported by previous experimental studies proposing that CPG neurons have intrinsic pacemaker properties [Dale, 1995a; Aiken et al., 2003], however until now it has been unclear why the neurons appeared to change between single spiking and pacemaker behaviours.

My aim is to try to understand more quantitatively how a small population of electrically coupled interneurons on one side of the CNS could act as pacemakers and intrinsically generate self-sustaining rhythm which can be turned on and off by brief external synaptic input. Computational modelling will allow us to test proposals about the roles of cellular and pacemaker properties in generating rhythmic activity in tadpoles. I use an existing model of the minimal population of the electrically coupled dIN neurons located in a small tapered region of the hindbrain and spinal cord based closely on the evidence from biology [Li et al., 2006]. In particular, I address the following questions: *a)* based on experimental estimates for population sizes, synapse strengths and connectivities, would a small population of dINs with mutual NMDAR-mediated synaptic connections be able to sustain rhythm generation, switched on by brief synaptic excitation; *b)* how does the activation of NMDARs affect pacemaker-firing of the population of electrically coupled dINs and what is the effect of the voltage dependency of the channel; *c)* can brief synaptic inhibition switch off dIN rhythmic activity?; and *d)* can mutual NMDAR-mediated synaptic connections allow rhythm generation in populations of other types of neuron?

5.3 RESULTS

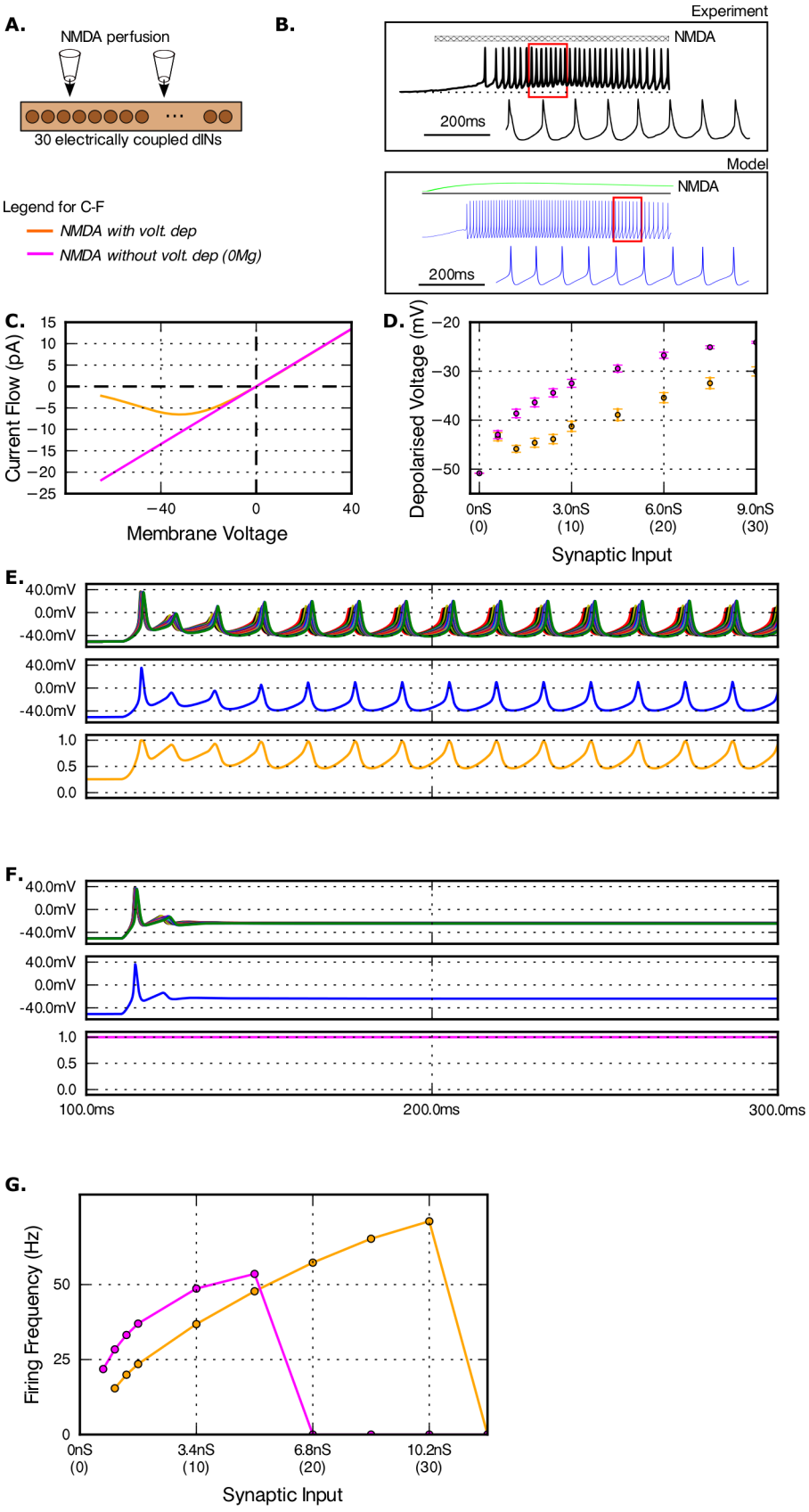
5.3.1 *Pacemaker responses of dIN population to NMDA perfusion*

In the tadpole during swimming dINs fire once on each cycle and release glutamate to excite each other [Li et al., 2006]. The glutamate activates NMDAR and is proposed

to produce a sustained background depolarisation. [Li et al., 2010] investigated the effects of NMDAR activation by perfusing NMDA over one half of the CNS while recording from a dIN. They found that NMDA perfusion could led to depolarisation and rhythmic, pacemaker-like firing at swimming frequencies in the recorded dINs so long as $Mg^{2+}$was present (Fig. 5.2B). The NMDAR-mediated current differs from a simple depolarising current injection and other synaptic currents because it is voltage dependent in the presence of $Mg^{2+}$and when extracellular $Mg^{2+}$is removed, this voltage dependency is not seen (Fig. 5.2C) [Nowak et al., 1984]. The first aim was to model this process.

To investigate rhythm generation induced by NMDA perfusion over the dIN population, I implemented a model of the NMDAR [Sautois et al., 2007], using a very slow closing time-course of 10 s (Fig. 5.2B; [Li et al., 2010]). The NMDAR conductance could be simple (without extracellular Mg^{2+}) or have voltage dependency (with extracellular Mg^{2+}) (Fig. 5.2C). In all tests I used the same model of 30 dINs from Chapter 4, where multicompartmental dINs were arranged in a longitudinal column and coupled via gap junctions on their axons. To investigate the effects of NMDAR activation on

Figure 5.2 ***(facing page)*** – Perfusing NMDA onto the dIN population. **A.** The population of electrically coupled dINs, onto which NMDA was perfused. **B.** Perfusion of NMDA onto a dIN causes depolarisation and repetitive firing (black traces from Fig. 2 in [Li et al., 2010] where hatched bar denotes NMDA perfusion). Similar firing is seen in the model (blue trace) where green line shows the NMDA activation. Red box shows region expanded below. **C.** Current-voltage curve of a single NMDA synapse, with (yellow) and without (pink) voltage dependence. **D.** The steady-state potentials of dINs as a function of NMDAR conductance (with sodium channel conductance set to zero to prevent firing) with and without voltage dependency of the NMDAR. (x-axis: number in brackets denotes the number of synapses onto each dIN) **E, F.** The response of the network of 30 dINs to NMDA perfusion with (E) and without (F) voltage dependency. top: somatic membrane voltage records of all dINs, middle: somatic membrane voltage of dIN-15, bottom: voltage-dependent component of NMDAR. **E to G**. The effect of increasing amounts of NMDAR activation on the firing frequency of the dIN network with (yellow) and without (pink) voltage dependency. **G.** Plots of dIN firing frequency vs NMDAR conductance. At high levels of NMDAR conductance, without voltage-dependency, the dINs are unable to rehyperpolarise and repetitive firing breaks down (F).


A.
NMDA perfusion
30 electrically coupled dINs
B.
Experiment
NMDA
200ms
Model
NMDA
200ms
Legend for C-F
NMDA with volt. dep
NMDA without volt. dep (0Mg)
C.
Current Flow (pA)
Membrane Voltage
D.
Depolarised Voltage (mV)
0nS (0)
3.0nS (10)
6.0nS (20)
9.0nS (30)
Synaptic Input
E.
40.0mV
0.0mV
-40.0mV
1.0
0.5
0.0
F.
100.0ms
200.0ms
300.0ms
G.
Firing Frequency (Hz)
0nS (0)
3.4nS (10)
6.8nS (20)
10.2nS (30)
Synaptic Input

dINs their firing was turned off by removing sodium channels, and a series of simulations were run, in which the NMDAR conductance was activated in all dINs. Increasing NMDAR activation caused increasing steady state depolarisation, which was larger without NMDAR voltage-dependency (Fig. 5.2D).

When the dINs had voltage-gated sodium channels, sufficient NMDAR activation caused dINs to fire repetitively, as in experimental recordings (Fig. 5.2B). This firing occurred both with (Fig. 5.2E) and without (not shown) NMDAR voltage dependence. Simulations without voltage dependency required fewer synapses to reach firing threshold than those with voltage dependency, since the resulting conductance at each synapse was higher. When firing frequency was measured higher conductance provided a stronger drive, causing a higher firing frequency (Fig. 5.2G). At higher conductances, dINs driven by NMDARs without voltage-dependency were unable to rehyperpolarise sufficiently and did not fire full action potentials (Fig. 5.2F). However, those with voltage dependence still showed reliable spiking (Fig. 5.2E). The voltage dependence of the NMDARs effectively doubled the range of conductances over which the network could produce robust rhythmic firing (Fig. 5.2G).

5.3.2 *Effects of mutual dIN excitation on their response to brief synaptic excitation*

A brief stimulus to the tadpole skin is normally sufficient to initiate swimming which can last for many seconds [Roberts et al., 2010]. Recently, a simple pathway has been identified in the tadpole, which can initiate swimming in response to head-skin stimulation [Buhl et al., 2012]. Sensory neurons innervate the tadpole's head-skin, and form excitatory synapses onto a population of tINs which fire briefly to excite dINs in the hindbrain. At low levels of head-skin stimulation EPSPs are seen in the dIN population, but as stimulus intensity increases the whole dIN population is recruited to fire and swimming starts. The question then is: what keeps swimming going after input from the tINs declines? Recordings have shown that dINs in the hindbrain and rostral spinal cord make reciprocal, glutamatergic, excitatory synaptic connections with each other [Li et al., 2006]. It was therefore proposed that when the dIN population fires, these synapses activate the NMDARs on members of the dIN population, and act like a perfusion of NMDA to turn on their pacemaker firing (see [Li et al., 2010]; Fig. 5.2B).

The NMDAR synaptic conductance between the dINs is expected to sum from cycle to cycle during swimming [Dale and Roberts, 1985; Li et al., 2006]. In the model, synaptic conductance is calculated as the difference of two state variables, which receive a step increase when the presynaptic voltage crosses a threshold and decay to zero during the remainder of the cycle [Sautois et al., 2007]. I investigated the effect of driving a single NMDAR synapse with spike trains of different frequencies (Fig. 5.3) and found that for typically swimming frequencies the conductance would reach a maximum between 2 and 3 times the peak conductance of a single synaptic event.

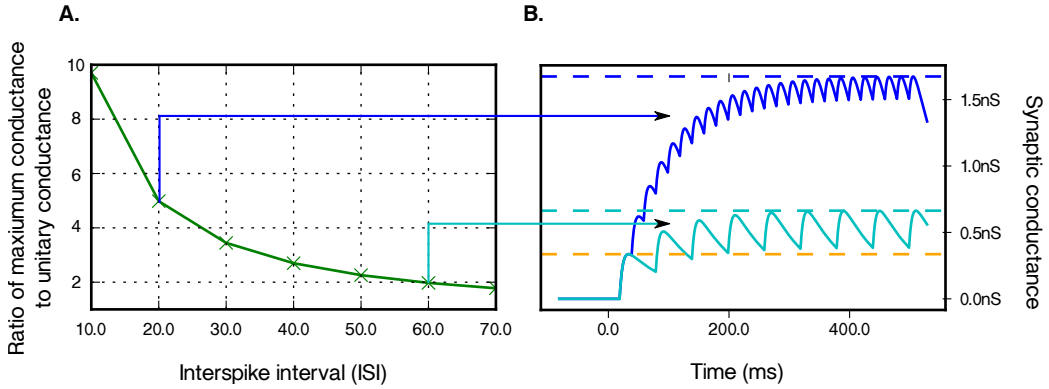


Figure 5.3 – Maximum conductance of feedback NMDARs as a function of frequency **A.** Spike trains of different frequencies were delivered to model NMDARs with a closing time of 80 ms, and the maximum conductance recorded. **B.** Faster spike trains had a larger maximum conductance; at the frequencies of tadpole swimming, the conductances will not rise above ~3 times the conductance of a single synaptic event.

I therefore investigated the role of glutamate-mediated feedback excitation between members of the dIN population. Would it allow the network to be switched into sustained swimming mode by a brief synaptic excitation from a 'sensory' activation pathway? To answer this question I modelled glutamate synapses with a fast AMPA component and a slow voltage-dependent NMDA component [Sautois et al., 2007; Li et al., 2010]. The electrically coupled network of 30 dINs was used as before, but mutual glutamatergic synaptic connections between the dINs were added (Fig. 5.4A, B). These synapses were formed between pairs of neurons with a probability of 0.2 (Fig. 5.4C). To model the sensory activation of the network observed experimentally, I activated excitatory glutamatergic synapses onto dINs at times and with synaptic strengths

based on experimental measures of tIN firing times in response to head-skin stimulation and EPSP amplitudes measured in dINs when tINs fired [Buhl et al., 2012].

The model of a dIN has a higher firing frequency than recorded dINs [Li et al., 2010] where current-clamp injections suggest that this rate is limited at the level of individual neurons. This means that dINs in the model will receive higher levels of feedback excitation (Fig. 5.3). I therefore tested two models of dynamics on the feedback synapses; in both the synaptic conductance increased by a fixed amount following each synaptic event, and in the second the conductance saturated at $2 \times g_{peak}$ (Fig. 5.4G, red).

Synthetic sensory pathway synaptic input to the dINs produced long-lasting conductance increases (Fig. 5.4D) resulting in long-duration EPSPs. If these were large enough then the dINs started to fire and could maintain their own rhythmic firing if their mutual synapses were sufficiently strong (Fig. 5.4D - F). I found that firing was not sustained below certain levels of NMDA feedback between the dINs (Fig. 5.4G), and that as feedback strength was increased, the network frequency increased. Saturation of postsynaptic conductance led to more robust firing at higher frequencies.

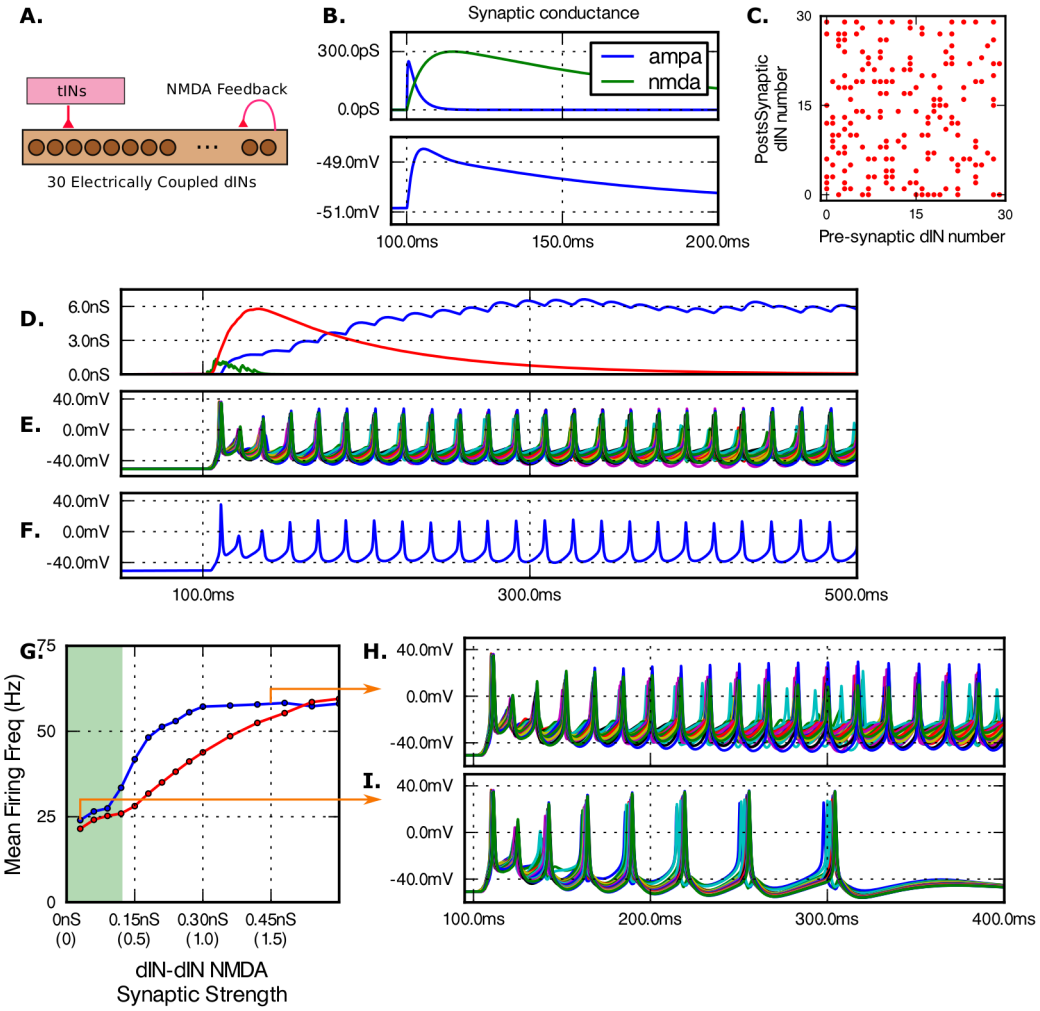


Figure 5.4 – The response of the dIN network with feedback excitation to brief sensory excitation. **A.** The network of 30 electrically coupled dINs excited by sensory pathway tINs and with feedback glutamatergic synapses. **B.** The conductance time-courses of the faster AMPAR (top: blue) and slower NMDAR(top: green) result in a combined EPSP seen in a dIN (bottom:blue) **C.** Synapses are created between pairs of dINs with a fixed probability of 0.2. **D.** The conductance of synapses onto dIN-15 from the tIN sensory pathway (red: NMDAR, green: AMPAR) and from feedback (blue: summed NMDAR conductance from other dINs, the voltage-dependent term of the conductance has been excluded for clarity) **E.** The membrane voltage traces in all the dIN somata. After the input from the sensory pathway has decayed (after 300 ms), the network is able to maintain rhythmic activity because of the feedback NMDAR conductance. **F.** A single dIN voltage trace from E. **G - I.** The effect of the strength of the NMDA feedback excitation on firing frequency of the network. At low levels of excitation, in some cases rhythmic activity is sustained, but often the network is unable to sustain rhythmic firing (green area in G, I). Higher levels of feedback lead first to regular firing then to higher firing frequencies. (red: feedback synapse conductance saturates at twice peak-conductance; blue: no saturation). At high feedback (>0.45 nS), some dINs do not fire full action potentials (as in Fig. 5.2F), although rhythmic activity is still generated by the population (not shown).

5.3.3 *Synaptic termination of dIN rhythmic activity*

In life, a swimming tadpole can be stopped by pressure to the head-skin via an identified GABA ergic pathway [Perrins et al., 2002]. Primary trigeminal afferent neurons innervating the cement gland and head-skin form excitatory synapses onto MHRs. These in turn form inhibitory GABA-A synapses onto neurons in the CPG. MHRs fire multiply when the cement gland is pressed and in response to step current injection at frequencies between 40-140 Hz. It has been shown that activation of a single MHR, producing ~5 spikes, is sufficient to stop swimming in the tadpole.

To investigate whether the dIN population could be switched off using a biologically realistic pathway I implemented a simple model of an MHR inhibitory GABA-A synapse connected to all the dINs. The synapse was based on current-clamp recordings of IPSPs in spinal neurons produced by MHR stimulation and from the literature (Fig. 5.5B) (τ_o = 1.5 ms, τ_c = 20.0 ms, E_{rev} = –70 mV, g_{peak} = 2 nS) [Perrins et al., 2002; Koch, 1999]. The 30 dIN network was built, and the strength of the feedback NMDAR excitation was reduced by 50% to reduce the firing frequency of the population. The network was activated to start producing rhythm using the tIN pathway as before, and after 700 ms, the inhibitory synapses were activated 5 times at 15 ms intervals, equivalent to 66 Hz, which is at the low end of MHR firing frequencies. This form of inhibitory input produced a large compound IPSP like the MHR pathway and reliably switched off rhythm generation like the MHR pathway (Fig. 5.5C).

I investigated the effectiveness of the stopping pathway and found that a single IPSP (maximal conductance of g_{peak} = 2 nS) delivered to all dINs simultaneously would stop spiking in 25% of simulations (Fig. 5.5D, E). The effectiveness of a single IPSP in stopping activity depends on the time in the cycle when the inhibition arrives. A single inhibitory synaptic event (g_{peak} = 2 nS) delivered to all the dINs simultaneously could have little effect, cause a delay in firing, or terminate firing (Fig. 5.5D).

I ran a series of simulations in which the number of spikes ($nspikes$) and the interspike intervals (ISI) EPSPs from a single MHR were varied ($nspikes$:1, 2, 3, 4 & 5 spikes, ISI: 10, 15 & 20 ms, g_{peak} = 2 nS) . The probability of stopping increased with the total duration of inhibition, either by adding additional IPSPs or increasing the interval between them. (Fig. 5.5E blue histogram). The duration of inhibition plays

an important role in preventing spiking until the background NMDA excitation had decayed sufficiently to prevent further dIN activity (not shown). I also reran the final experiments but I increased the level of NMDA feedback between the dINs to 1.0 and made the feedback NMDAR conductance saturate at two times the unitary conductance to limit firing frequency (Figs. 5.3 & 5.4 G). When the network was driven with synaptic input from three MHRs, swimming was reliably stopped when the MHRs provided inhibition for a sufficiently long duration (Fig. 5.5E red line). In life when the head is pressed and swimming stops, it is likely that dINs will receive input from many MHRs, both contralateral and ipsilateral [Perrins et al., 2002]. This modelling suggests that the number of MHR to dIN synapses could be low and still provide a reliable stopping pathway.

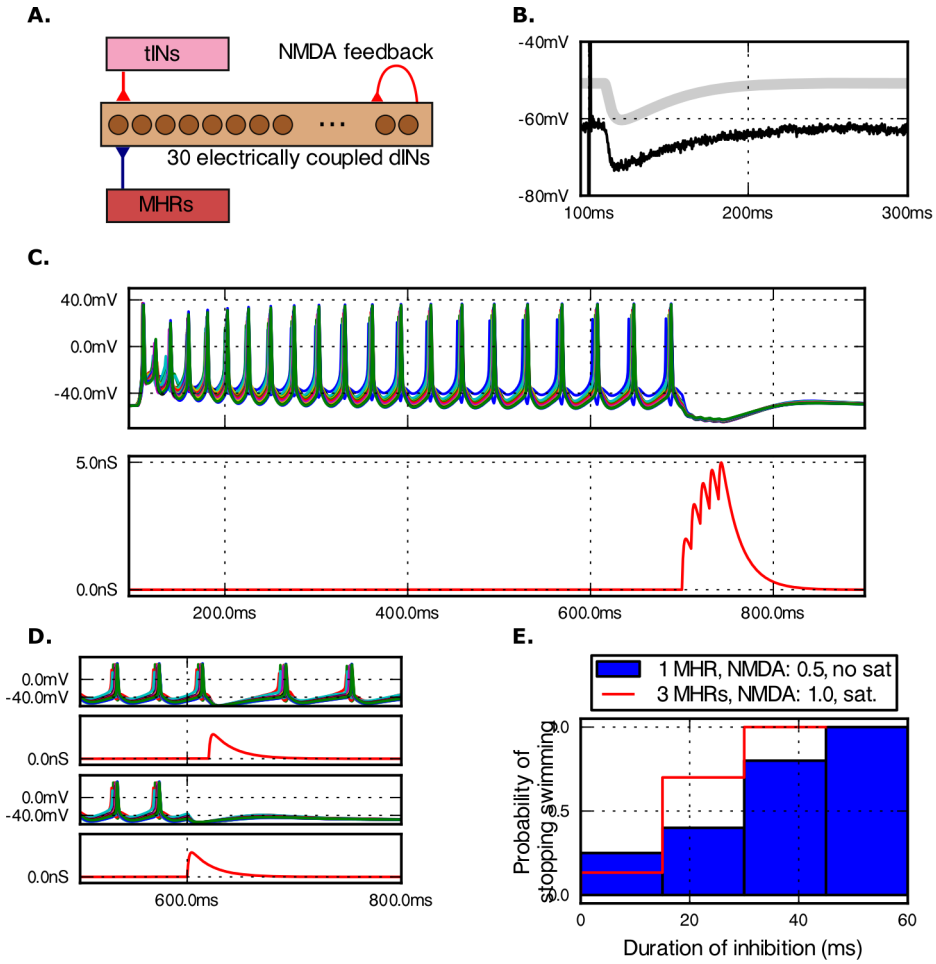


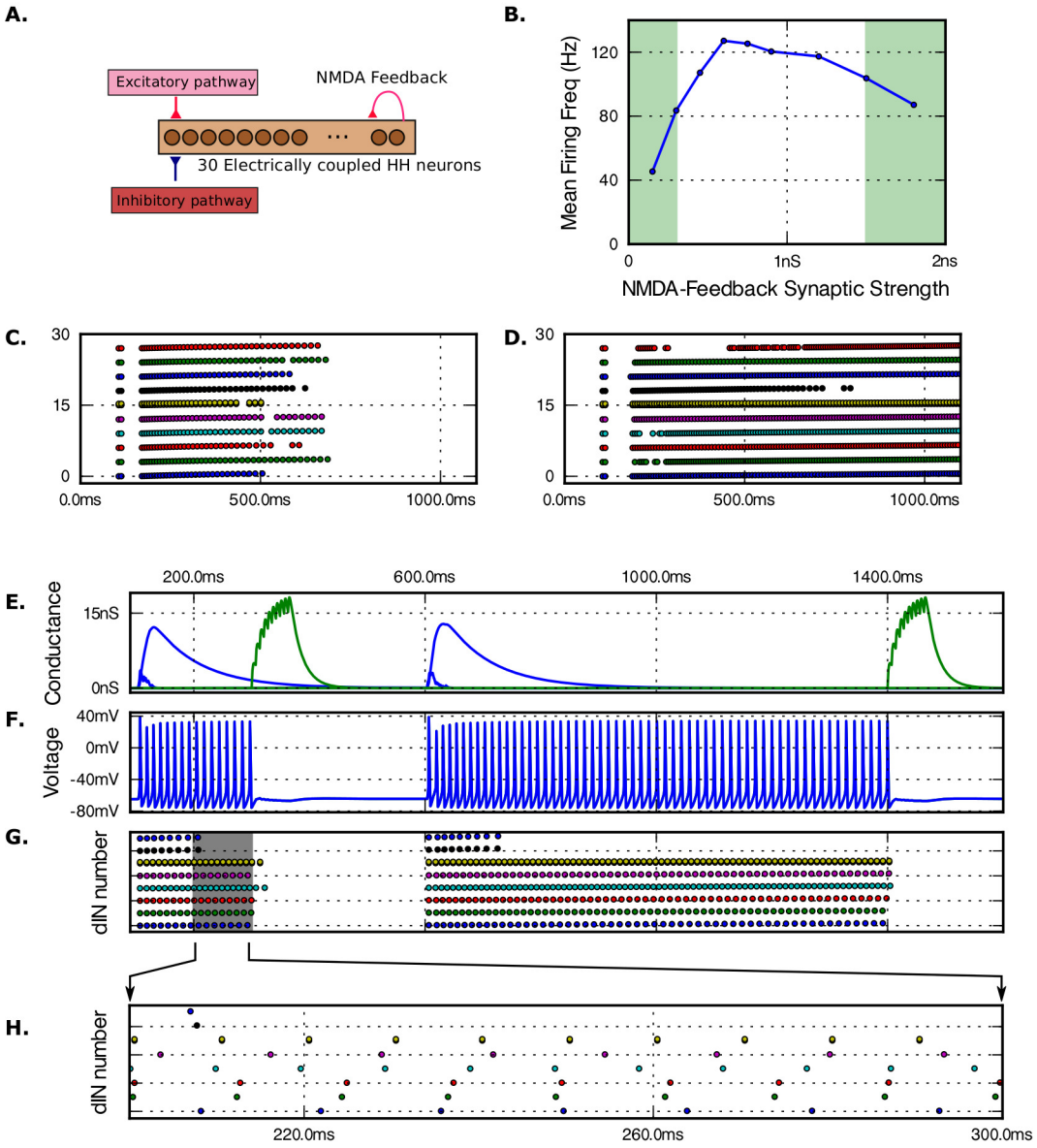
Figure 5.5 – Switching off the swimming network using a simple inhibitory pathway. **A.** The electrically coupled population of 30 dINs with excitatory NMDA & AMPA synaptic feedback. tINs excite dINs to start rhythm and MHRs inhibit all dINs synchronously. **B.** Matching the temporal dynamics of the inhibitory synapse. Recording of an MHR GABA-A IPSP (black; [Perrins et al., 2002]) in a spinal neuron and model IPSP in a dIN (light grey). **C.** Rhythm generation in the model network was switched on at 100 ms via synaptic input from the sensory pathway (top voltages traces), and then at 700 ms five IPSPs from the MHR pathway in each dIN (lower trace) turn activity off. **D.** The phase within the firing cycle of a single inhibitory synaptic event determines whether it stops activity. A synaptic event delivered to all the dINs simultaneously could have little effect (not shown), cause a delay in firing (top) or terminate firing (bottom). (The top traces show the voltages, the lower trace (red) show the inhibitory conductances onto a single dIN) **E.** The stopping pathway became more reliable as inhibition was made longer, either with more spikes or with a longer interspike interval (ISI) in both the case when the NMDAR feedback was reduced (blue) and the postsynaptic receptors saturated (red). The duration of inhibition is calculated as $(spikes - 1) \times \text{ISI}$.

5.3.4 *Rhythm generation by other networks of neurons*

Can other populations of neurons with mutual excitatory connections also generate self-sustained activity? Such connections are often thought to lead to instability so I explored whether this was the case. In the tadpole the dINs play a specific role in swimming and it is likely that they have a low and limited firing frequency range, because of their role in a locomotor system [Li et al., 2010]. Most neurons in the tadpole and in the nervous systems of other animals typically fire at much higher rates (up to 200 Hz). I therefore investigated whether populations of more typical neurons with higher intrinsic firing frequencies could generate self-sustained firing if they made mutual excitatory synaptic connections. I also investigated the effect of electrical coupling on such rhythm generation.

Each of the 30 dINs in the network was replaced by a single compartment model of a HH neuron [Hodgkin and Huxley, 1952]. The neurons in the network were electrically coupled like the dINs, had excitatory NMDA and AMPA synaptic feedback connections and were driven by brief glutamate excitation at 100 ms to switch them on (Fig. 5.6) (details in the Methods section). The network was able to generate sustained rhythm, and inhibitory synaptic input at 500 ms (using the MHR pathway as in Fig. 5.5) was able to turn off firing. Although the model neurons were identical, random differences in synaptic input lead them to fire at different frequencies. Even with low coupling resistances (100MΩ) between every pair of neurons, the whole network would not synchronise and instead, small groups of neurons would fire together. Removing electrical coupling from the network had little effect (not shown).

Figure 5.6 ***(facing page)*** – Generalising the feedback excitation rhythm generation mechanism. **A.** A population of 30 electrically coupled HH neurons with feedback AMPA and NMDA synapses was driven with excitatory synaptic input to switch it on and inhibition to switch it off (on/off; 100 ms/300 ms, 600 ms/1400 ms). **B.** Below a certain level of feedback, the network is unable to sustain rhythm (left-green area, C). Increasing the strength of the NMDA feedback synapses leads to a faster firing frequencies, but when feedback strength is too high, firing also begins to break down (right-green area, D). **C.** When feedback NMDAR conductance was too low (g_{peak} = 150 pS); the network is unable to sustain rhythm. (Left, green region of B). In this simulation, following brief stimulation at t=100 and with no external inhibition, some members of the population generate action potentials for several hundred milliseconds, but eventually the network returns to rest. **D.** When NMDA feedback strength was too high (g_{peak} = 2 nS), some neurons were over stimulated and locked up, as with the dINs. In some cases, neurons that were overstimulated at initiation, would fire later when some of the excitation decayed, for example (e. g. upper blue raster) **E.** The network was able to be switched on and off at short latency in response to synaptic input (excitatory in green; inhibitory in red) **F.** A sample voltage trace is shown in blue. **G.** A raster plots shows the times of action potentials. **H.** Expansion of the grey area in F. Although the neurons have identical cellular properties, variation in gap junction and glutamatergic synaptic connections causes them to fire at slightly different frequencies. Even though the electrical coupling is strong (100 MΩ), the neurons only partly synchronise.


A.
Excitatory pathway
NMDA Feedback
30 Electrically coupled HH neurons
Inhibitory pathway
B.
Mean Firing Freq (Hz)
120
80
40
0
0
1nS
2ns
NMDA-Feedback Synaptic Strength
C.
30
15
0
0.0ms
500.0ms
1000.0ms
D.
30
15
0
0.0ms
500.0ms
1000.0ms
E.
200.0ms
600.0ms
1000.0ms
1400.0ms
Conductance
15nS
0nS
F.
Voltage
40mV
0mV
-40mV
-80mV
G.
dIN number
H.
dIN number
220.0ms
260.0ms
300.0ms

5.4 DISCUSSION

Modelling allows us to ask questions about neuronal networks: do we understand the key mechanisms? Do we have sufficient data to make a robust model capable of reproducing *in silico*, what is seen *in vivo*? Can we make quantitative predictions? In this work I built a biologically realistic computational model of a population of neurons in the hatchling tadpole brainstem. This model has provided a quantitative platform to ask questions that cannot be addressed directly *in vivo* about the mechanisms controlling action potential firing in neurons driving swimming.

I propose that the voltage dependence of NMDAR channels allows the dINs to be activated to fire over a wider range of NMDAR conductance strengths and this improves robustness of the network rhythm generation. For the network controlling hatchling tadpole fictive swimming, I have demonstrated that mechanisms identified experimentally: voltage-gated membrane channels; feedback NMDAR excitation; and electrical coupling; [Li et al., 2006; Soffe et al., 2009; Winlove and Roberts, 2011] are sufficient to generate self-sustaining, rhythmic, synchronous pacemaker-like firing activity in a population of neurons in one side of the hindbrain without the need for any inhibitory feedback. I have shown how this population can be switched on and off robustly by brief synaptic input and that non-linear saturations in the cellular and synaptic dynamics facilitate stable, positive feedback. I suggest that although the dIN network forms part of a bilateral network, the firing frequency is partly determined by the cellular properties of the dINs themselves rather than resulting from network properties (as proposed by [Dale, 1995a; Li et al., 2010]). Finally I suggest that this *synaptically switchable* pacemaker network maybe a useful building block in the CNS toolbox; and could be applicable in other contexts in which populations of neurons need to be switched on and off at low latency.

5.4.1 *Role of NMDAR voltage dependence in population pacemaker firing*

One of the main conclusions of this modelling study is that a population of neurons with mutual glutamatergic synapses can generate self-sustaining pacemaker-like activity by activating their own NMDARs. During long-lasting excitation, after an isol-

ated dIN has fired, it is likely that several mechanisms act to prevent another action potential firing again, including sodium channel inactivation and a slow-activating, persistent potassium current [Dale, 1995a; Winlove and Roberts, 2011]. Both of these mechanisms act against the neuron firing again until they have been *reset* by the rehyperpolarisation of the membrane. In the case of a simple current injection to the neuron, if the injected current is too high, the membrane voltage will not rehyperpolarise, and the neuron will fire only a single spike. However, as discussed in Tabak and Moore [1998], the NMDARs allows excitatory drive to be delivered to the dINs, whilst still allowing them to rehyperpolarise to negative levels following action potentials. The voltage dependence of the NMDARs *cooperates* with rehyperpolarising mechanisms and allows repeat firing. How could this work? Consider the NMDAR at these rehyperpolarisation potentials (~−40 to −60 mV) in the case of no voltage dependency (0 Mg^{2+}). There is a large driving force across the receptor, causing a current that acts to depolarise the membrane and drive the membrane voltage strongly towards 0 mV (Fig. 4.8C). However, at similar voltages in the case of NMDAR with voltage dependence, (with Mg^{2+}) this current is blocked, and so there is no current acting to depolarise membrane. (see Fig. 4.8D). In effect, the voltage dependence of the channel prevents a current from flowing at membrane voltages of (< −30 mV) which would act to depolarise the neuron. When mechanisms that act to rehyperpolarise the membrane activate (for example Kf channel after an action potential or input from inhibitory synapse), the membrane voltage with voltage dependency is more negative for a longer time, allowing fuller recovery from inactivation mechanisms. Simulations reflect this, suggesting that although the voltage-dependency of the NMDAR is not required for rhythmic activity, a simpler, non-voltage-dependent drive to the dINs would need to be more carefully regulated in order to provide a current in a range that is strong enough to cause an action potential, but not too strong to prevent rehyperpolarisation. A mechanism using voltage-dependent NMDAR would be more robust and require less fine-tuning.

5.4.2 *Comparison of model with biology*

I have produced a model of the tadpole dIN population on one side of the tadpole CNS which is capable of reproducing many experimental results. However, it is an oversimplification. For example, in the model, the network could fire at low frequencies given low levels of excitation, but would fire faster at higher levels of excitation (~60 Hz). Experimentally, the dINs have a low maximum firing frequency (<30 Hz), within the normal swimming frequency range, during NMDA perfusion even when additional positive current is injected (Fig. 3; [Li et al., 2010]). It seems likely that this low and narrow firing frequency band is the result of cellular properties, particularly the activation and inactivation kinetics of the sodium, slow potassium, or other membrane channels [Hille, 2001]. Additionally, action potentials recorded in dINs have a characteristically long rise time [Sautois et al., 2007]. Voltage-clamp records from other spinal neurons have suggested the presence of a fast, inactivating A-type potassium channel (opening/closing time constants < 3 ms; Chapter 2). When an A-type channel was introduced into the dIN model, I found better fits to the rise time and shape of the action potential (not shown). A-type potassium channels are proposed to slow firing frequency in other systems, [Hille, 2001; Connor and Stevens, 1971]. However, tests showed that it had little effect on the frequency of repeated firing, of dINs so it was omitted from the network model. To resolve why the dIN population fires too fast. direct evidence on the membrane channel properties of dINs based on new voltage-clamp data is needed.

This modelling suggests that a single side of the nervous system will sustain rhythmic activity following brief stimulation by the mutual self-excitation of NMDAR synapses among dINs, even when the conductance of NMDAR feedback is reduced to less than 50% of its measured value (Fig. 5.4). Experimentally, this is difficult to demonstrate definitively. Ventral root recordings showed that one side of the nervous system of the tadpole can sustain activity for many cycles in response to brief electrical stimulation. Applying a pharmacological NMDAR blocker (AP5) abolished rhythm generation in a one-sided nervous system, and application of NMDA to a single side of the nervous system made it more excitable and led to an increase in the duration of episodes [Soffe, 1989]. Blocking the inhibitory synapses does not prevent rhythmic activity in

one side of the nervous system, but the duration of activity is generally shorter. It has recently been suggested firstly that these lesion experiments may have side effects [Hoffman and Parker, 2010] and secondly that inhibition is necessary for rhythm generation. Light-driven outward proton pumps were expressed in cINs in the tadpole spinal cord which allowed them to be inactivated rapidly. It was found that inactivating these neurons during swimming could terminate the episode [Moult et al., 2013], which would appear to be at odds with the results in this Chapter. This paper also found that the swimming process is affected by homeostatic mechanisms and that within 23 minutes of inactivating the cINs, one side of the nervous system was able to sustain rhythm. Unfortunately, it is not clear whether inactivation of cINs would prevent the initiation of swimming and one possibility is that the network and cellular mechanisms of rhythm generation are active at different times: at the start of swimming, the pacemaking properties of the dINs could play an important role, and later, once swimming is established, network mechanisms become more important. The excitation due to feedback NMDARs sum from cycle to cycle [Roberts et al., 2008] and one explanation is that once swimming is established, the feedback excitation to the dINs becomes very strong. While the dINs are receiving mid-cycle inhibition from the opposite side, this rehyperpolarises the dINs sufficiently to allow repetitive firing. However, when this mid-cycle inhibition is removed, the dINs are no longer able to rehyperpolarise and instead *lock-up (*similar to Fig. 5.2*)* and so swimming stops. What other hypotheses could exist for why a single side of the nervous system would not fire repetitively to feedback NMDAR excitation? It remains a possibility that the single spike firing observed in dINs is not due to the electrical coupling, as was hypothesised in Chapter 4, and is instead due to an intrinsic cellular property, for example, a slowly activating potassium current [Winlove and Roberts, 2011]. Following this hypothesis, the results due Soffe [1989] and Li et al. [2010] could be the result of homeostatic mechanisms arising from the lesion experiments [Hoffman and Parker, 2010], and the observed recordings of repetitively firing dINs due to Aiken et al. [2003] could be explained as recordings of the later identified dIN*r*s [Li et al., 2007b; Roberts et al., 2010]. Only more direct experimental data, focusing on the start of swimming episodes, will be able to resolve these issues. It is also possible that rather than a single mechanism being responsible for these observed behaviours (i. e. the rebound

or conditional-oscillator properties of the dINs), multiple mechanisms may work together in parallel to provide redundancy, and the relative roles of the different mechanisms may change during the swimming episode. If this is the case, it may be hard to experimentally tease apart *clear-cut* explanations and this possibility also emphasizes the importance of accurately recording the setups under which experiments are undertaken, and the attention to detail that is needed when interpreting experimental observations.

In life it is likely that synaptic inhibition plays a role in limited swimming frequency. As well as reciprocal inhibition from the other side, each side of the CNS contains not only excitatory feedback connections from the dINs onto themselves, but also connections from a population of aINs providing recurrent inhibition. The work suggests that rehyperpolarising mechanisms play an important role in dIN pacemaking; at high levels of stimulation, the dINs will be unable to rehyperpolarise sufficiently to fire repetitively. Inhibitory aINs are most active shortly after sensory stimulation initiates swimming [Li et al., 2004b] which is the time when dINs are most strongly excited [Li et al., 2006]. To investigate whether inhibition from aINs onto the dINs could provide an additional hyperpolarising mechanism I introduced a population of aINs to the network. These were excited to fire by the dINs and made inhibitory synapses onto dINs [Sautois et al., 2007]). When aINs were modelled, the depolarised plateau after their first spike (see Fig. 5.2E, 5.4E, H, I) disappeared and the after-hyperpolarisations deepened. As a result the second spikes fired by dINs became larger and more reliable (not shown).

5.4.3 *Conclusions*

The model of a population of tadpole hindbrain neurons sustains rhythmic firing through positive feedback onto itself. In many domains, positive feedback leads to exponential growth and instability. However, the neuronal circuits in the tadpole nervous system, like those of most neurons, contains many saturating non-linearities. All neurons have limited upper firing frequency which in the dINs is particularly low. Additionally, the saturation of the NMDARs at feedback synapses and the activation of non-inactivating potassium conductances will act to prevent run-away depolarisation

and firing. The model dIN network with a higher firing frequency limit and without synapse saturation still gave stable firing across a range of parameters.

Although this work has focused on a single sided population of dINs, the tadpole has bilateral populations of electrically coupled dINs which form connections with other neuron populations to produce rhythmic, antiphasic firing on each side during swimming [Roberts et al., 2010; Sautois et al., 2007]. This modelling, as well as intracellular recordings from current injections to dINs perfused with NMDA [Li et al., 2010] suggest that the firing frequency of the swimming network depends importantly on the cellular properties of the dINs. Mid-cycle inhibition from the contralateral reciprocal inhibitory cINs may lengthen the cycle period slightly. It has been proposed that cINs organise the phase relation between the two sides through post inhibitory rebound [Roberts and Tunstall, 1990; Tunstall et al., 2002]. Their inhibition will also act as another rehyperpolarising mechanism for the dINs, facilitating repetitive firing at high levels of stimulation.

In a biologically realistic model network the rhythmic activity of a small population of neurons can be switched and off by brief excitatory and inhibitory synaptic input. The NMDA-mediated feedback mechanism underlying rhythm generation in the tadpole CNS will work in other populations of neurons with much higher maximum firing frequencies, for example HH neurons. However, because of the much higher rate of spikes causing feedback, the dynamics of the synapses will likely be more important. The simple models of these networks suggested that at the higher frequencies, electrical coupling would have little effect on spike synchronisation. Such a feedback mechanism would be particularly suited to systems that need to be switched on and off with low latency, such as the control for positioning an outstretched arm.

I have explained major features of neuronal population pacemaker mechanisms driving tadpole swimming and shown that similar mechanisms could lead to controlled and stable rhythmic firing in other neuron populations. I have built a model that reproduces many experimental findings; although the detailed channel currents which limit firing frequencies in tadpoles remain to be clarified. I have demonstrated that electrical coupling could play a subtle but crucial role in the pacemaker firing of the dINs as a population; that the voltage dependence of NMDARs can facilitate strong drive to neurons without causing them to lock up. Finally, I have demonstrated that

small, biologically realistic population of neurons which drive vertebrate locomotion is able to sustain stable rhythmic firing intrinsically and can be switched on and off via experimentally defined neuronal input pathways.

6

THE DECISION TO INITIATE SWIMMING

6.1 ABSTRACT

What happens when we decide to perform an action, for example when we start to move? I investigate how locomotion is initiated in the tadpole by building computational models of small populations of neurons and initiation pathways. In the tadpole, brief stimulation to the head can lead to swimming, and the rhythmic signals which drive swimming are generated intrinsically by a small, well characterised network of neurons in the brainstem. Two small, symmetrical populations of electrically coupled neurons (dINs), one on each side of the body, are particularly central to the generation of these swimming rhythms and have recently been shown to receive asymmetrical synaptic input in response to head-skin stimulation. I use modelling to first investigate the effects of electrical coupling between these neurons on how a single population activates in response to synaptic input, and find that it causes the population to either fire as a group or not at all. Next, I build models of the bilateral initiation pathways based on experimental recordings and show through a series of experiments that although long-lasting symmetrical input to the two populations of dINs on each side can start swimming, often the two populations begin and remain firing in synchrony. However, when the pathways are asymmetrical, consistent with what is observed physiologically, the two sides initiate more reliably in antiphase and swimming can start at a shorter latency over a wider range of stimulation strengths. The modelling suggests that two small populations of neurons, which have not yet been identified experimentally, play an important role in the initiation of swimming

in the tadpole by providing long-lasting NMDAR-mediated excitation to the dINs at the start of swimming.

6.2 INTRODUCTION

In many animals, locomotion is produced by opposing, antiphasic activity in symmetrical parts of the body, such as in swimming and running. Some of the cellular and network properties that allow rhythm to be sustained in an already active CPG have been described in previous chapters; what remains unclear is how these networks of neurons are reliably activated. A working musculoskeletal system and pattern generator circuit are essential to locomotion, but redundant without effective initiation pathways. In humans, 1% to 2% of the population over 60 years old develop Parkinson's disease, in which one of the symptoms is difficulty in initiating movement [de Rijk et al., 1997]. The initiation of locomotion in humans is not yet well understood; it is thought that movement planning occurs in higher brain areas, which is then integrated with sensory information and behavioural context and finally translated into motor commands. Specific regions such as the thalamus and the globus pallidus, subthalamic nucleus and substantia nigra within the basal ganglia are thought to be involved [Shik et al., 1966; Whelan, 1996; Jordan, 1998; Orlovsky et al., 1999; Jordan et al., 2008; Dubuc et al., 2008]. Unfortunately, these higher brain areas are complex, highly interconnected and also involved in a range of other roles, and it has been difficult to tease apart the neuronal pathways initiating locomotion.

In animals, we often think of a change in behaviour as a *decision*, for example at some instant, a cat will run away from an approaching group of people. We can imagine the danger perceived by the cat as a continuously varying input, which increases as the group approaches. We presume that when this variable crosses some threshold in the cat's brain, this causes a discrete change in the behaviour and the cat *decides* to flee, by first moving one foot forward, then the opposite. This scenario raises two questions. Firstly, how does this thresholding happen - how does a nervous system convert a continuous variable into a binary decision, for example to stay or to flee (Fig. 6.1A). Secondly, in order to run, the cat needs to start a rhythm, by first

moving one leg, then the other, not both simultaneously (Fig. 6.1B). How is the initial antiphasic drive to each limb arranged in a coordinated manner?

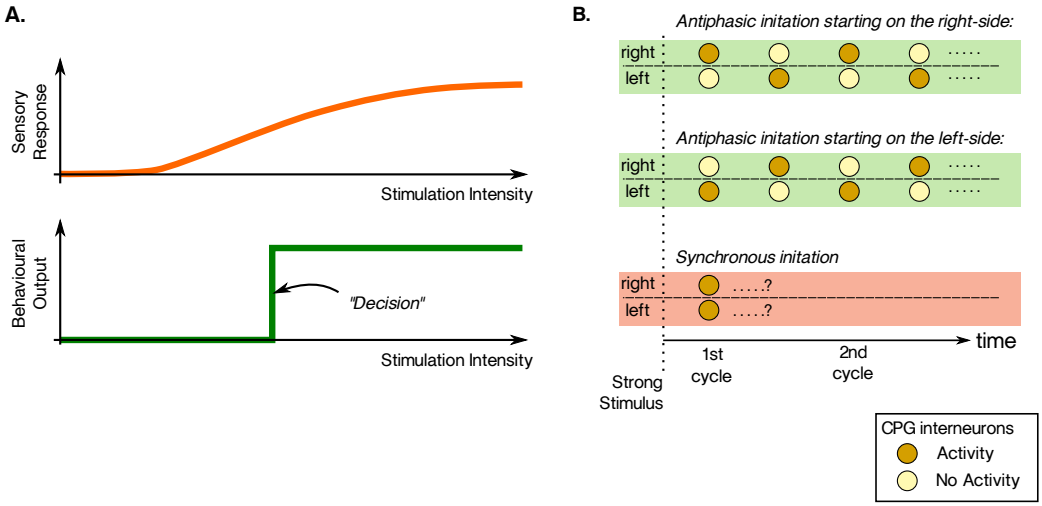


Figure 6.1 – Simplified behavioural responses of an animal to an environment. **A.** Animals need neuronal mechanisms to convert continuously varying environmental stimuli (top) into discrete behaviours (bottom) when the stimulus is sufficiently strong. A switch of behaviours can be thought of as a decision. **B.** When a bilateral animal *decides* to respond to a stimulus, how are the two sides of the locomotor CPG activated to produce an antiphasic pattern of activity (green) rather than a synchronous one (red)?

Hatchling tadpoles have to decide when to swim, so we can use them to investigate the underlying mechanisms of initiation. Tadpole swimming is characterised by antiphasic firing of neurons on opposite sides of the spinal cord [Roberts et al., 2010]. The neuron types and connections in the circuits that generate swimming patterns are well characterised (see Chapter 1). Each side of the nervous system includes a population of electrically coupled dINs which provide excitation to CPG interneurons on the same side and a population of cINs which provide inhibition to CPG interneurons on the opposite side (Fig. 6.2). The relative importance of cellular and network properties within the swimming circuit have already been considered and two proposals have been made (see Chapters 4 & 5). The first is that the dINs themselves have pacemaker properties and fire rhythmically in response to sustained background excitation [Li et al., 2010]. In this proposal, contralateral inhibition is important for arranging the phase delay between the two sides, rather than necessary for rhythmic firing. The second proposal is that the PIR properties seen in dINs require that they re-

ceive mid-cycle inhibition in order to generate rhythm, which comes from IPSPs from contralateral cINs [Li et al., 2006]. Although both of these hypotheses can explain how two populations of dINs that are already firing in antiphase can maintain rhythmic output, neither explains how antiphasic rhythmic activity in the network is initiated. How are the stimuli to the initiation pathways that drive these swimming circuits converted to produce either no activity or robust antiphasic firing in the populations of dINs from the outset?

How is swimming normally initiated in the tadpole? In response to sufficient, local head-skin stimulation, either mechanical with a fine hair or electrical, a tadpole will start to swim [Roberts et al., 2010]. Experiments in which head-skin is touched with a fine hair showed that stimulation to one side of the head can cause swimming to start on either side of the body (~60% on the contralateral side) [Boothby and Roberts, 1995]. The possible advantages of stochasticity in the initial responses of animals are widely documented [Humphries and Driver, 1970; Domenici et al., 2011]. For tadpoles, random variation in the initial direction of swimming could be important for survival, by making them less predictable and hence harder to catch. A short input pathway which relays stimulation to the head-skin and results in synaptic excitation to ipsilateral dINs has been identified [Buhl et al., 2012] (Fig. 6.2). The skin is innervated by trigeminal afferent neurons, which can be excited either by mechanical or electrical stimulation. These neurons make excitatory synapses onto a population of approximately 20 tINs, which in turn excite ipsilateral dINs via glutamatergic synapses. This short pathway can produce swimming within 20 to 30 ms of stimulation. The tINs only make ipsilateral projections and their role in the pathway to the dINs remains unclear. One hypothesis is that routing the excitation from very few (~1-5) sensory neurons through the small population of 20 tINs, amplifies the input to the dINs but also limits it, since no more than the entire tIN population can be recruited. This would prevent over-excitation in response to stimulation of a larger region of skin.

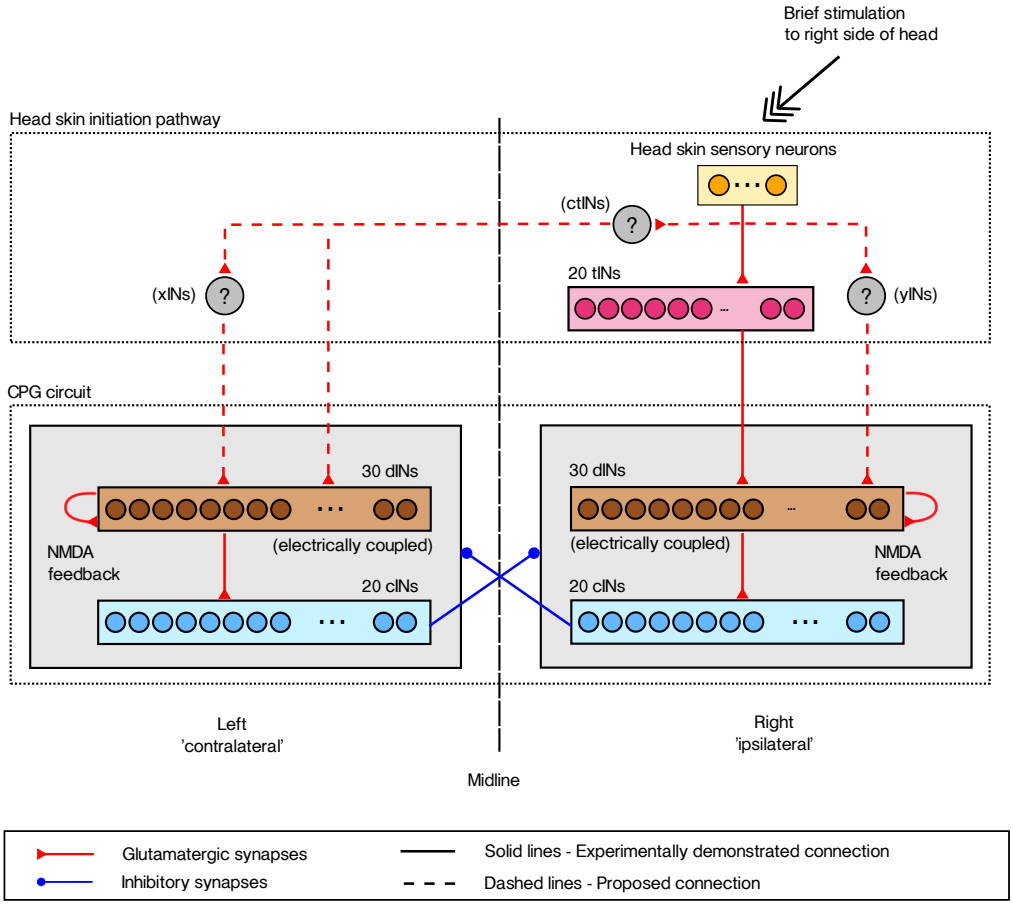


Figure 6.2 – Simplified circuit diagram of the head-skin initiation pathway and bilateral swimming CPG in the tadpole. On each side of the tadpole a population of dINs (brown) makes excitatory glutamatergic connections onto a population of cINs (blue) [Li et al., 2006]. The axons of cINs cross the spinal cord and carry inhibition to the opposite side. A pathway which excites ipsilateral dINs in response to head-skin stimulation has been defined [Buhl et al., 2012], in which excitation is relayed through a small population of tINs (purple). Stimulation of the head-skin also causes excitation to contralateral dINs, and it is also possible that another ipsilateral pathway exists, but the neurons in these pathways have not yet been identified. These unidentified neurons are referred to as xINs and yINs (question-marks, red dashed lines).

Could electrical coupling play a role in the initiation of tadpole swimming? It is widely observed in motor systems, and in systems involving short-latency, synchronous activity within a population of neurons, such as those generating escape reflexes [Edwards et al., 1999; Kiehn and Tresch, 2002]. In the tadpole, it has been proposed that electrical coupling plays an important role in making swimming robust and in synchronising the activity of the dINs [Li et al., 2009] (Chapters 4 & 5). Electrical coupling could also play an important role during the recruitment of dINs during the first cycle of swimming. There is variation between the neurons of the dIN population, both in intrinsic membrane properties and in synaptic connectivity [Li et al., 2001, 2004a; Sautois et al., 2007]. If there were no electrical coupling between the dINs and the tadpole were given threshold levels of stimulation, we would expect that the resulting excitation from the tIN pathway would cause some proportion of dINs to fire. One hypothesis is that sufficiently strong electrical coupling could overcome these variations and synchronise the firing of the dINs. Such an *all-or-nothing* recruitment of the population would provide a mechanism for converting the continuously varying input to the dINs from a sensory pathway, into a discrete decision.

What determines which side fires first following head-skin stimulation? Ventral root and recordings of dINs responses to head-skin stimulation show that the ipsilateral and contralateral sides of the nervous system are excited differently (Fig. 6.3) [Buhl et al., in prep]. When swimming starts ipsilaterally, the latency before ventral root activity is shorter than when swimming starts contralaterally (Fig. 6.3B). Paired intracellular recordings show that the populations of dINs on each side of the body are excited differently. When sufficient excitation is given to initiate swimming, the ipsilateral dINs fire their first spikes at a shorter latency than the contralateral dINs (Fig. 6.3C). The recordings show that the ipsilateral dINs receive short latency, long-lasting excitation that either causes them to fire at short latency (Fig. 6.3D; blue trace), or to become depolarised close to firing threshold (Fig. 6.3E; blue trace). In contrast, the contralateral dINs seem to receive a slower, gradual build-up of excitation, which reliably causes them to fire at longer latency than the ipsilateral dINs (Fig. 6.3E; red trace).

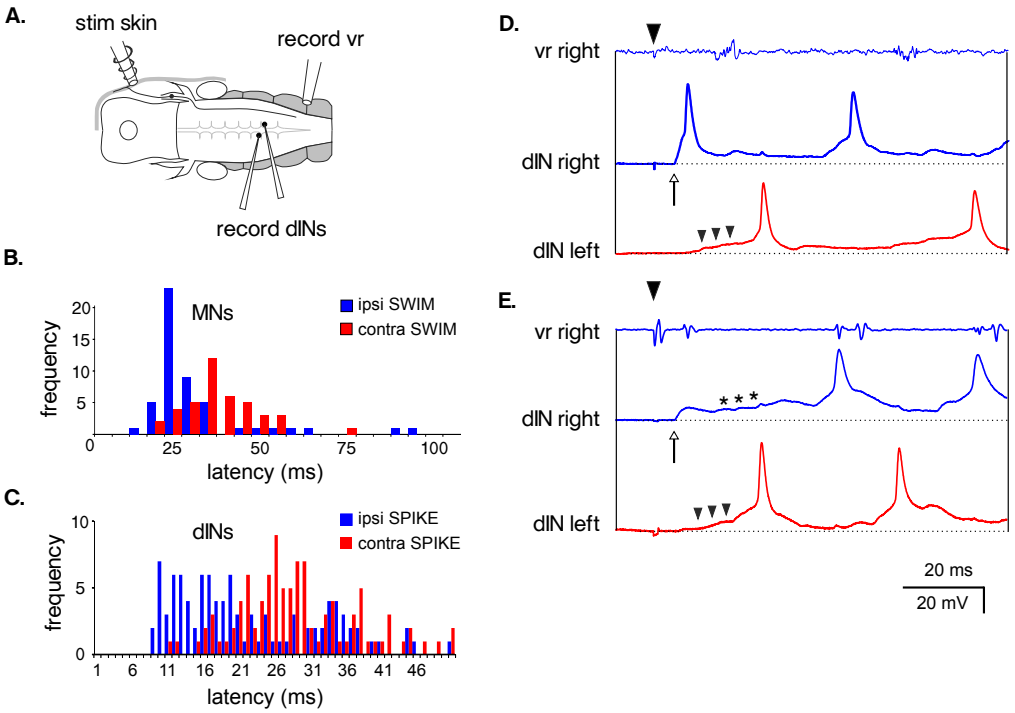


Figure 6.3 – Physiological recording from dINs during head-skin stimulation. **A.** Intracellular voltages from a pair of dINs and extracellular voltages from ventral root (vr) were recorded in response to head-skin stimulation **B.** The latency before activity is seen in the ventral roots in response to head-skin stimulation, when activity starts ipsilaterally (blue) and contralaterally (red). **C.** A histogram of spike latencies for 10 dINs to ipsilateral (blue) and contralateral (red) stimulation, showing that ipsilateral dINs spike earlier than contralateral dINs when swimming starts. **(D, E)** Intracellular recordings from dINs in response to head-skin stimulation to the right side of the body. **D.** When the ipsilateral (blue, 'dIN right') dIN fires first there is a large rapid excitation (at arrow) leading to it firing followed by a slow build-up in excitation to the contralateral dIN (red, 'dIN left', small arrow heads). **E.** In contrast when the contralateral (red, 'dIN left') side fires first, the ipsilateral dIN (blue, 'dIN right', small arrow heads) receives short latency input initially (at arrow), which has a long-lasting component (asterisks). The stimulus time is denoted by the arrow heads above the vr recordings. Data courtesy of Dr E Buhl.

What are the sources of excitation to the dINs following head-skin stimuli? The recordings of ipsilateral dINs show short latency, long-lasting excitation to the dINs (Fig. 6.3E; blue, at arrow, asterisks). Physiological experiments suggest that the excitation from the tINs to the dINs is primarily AMPAR-mediated, although it is possible that an NMDAR-mediated component is also present, which would be hard to detect because the recordings were performed in the presence of extracellular magnesium [Buhl et al., 2012] (In the future, this issue might be avoided by briefly depolarising the cells to remove the magnesium block). One possibility is that the initial, short latency excitation (Fig. 6.3E; blue, at arrow) is AMPAR-mediated from the population of tINs and the longer-latency, longer-lasting excitation (e. g. 6.3E; blue, asterisks) is NMDAR-mediated from another population of ipsilateral neurons (e. g. yINs, Fig. 6.2). In the spinal cord, a population of commissural sensory pathway neurons carries excitation to the opposite side in response to trunk-skin stimulation [Li et al., 2003]. In the head-skin pathway, the recordings suggest that the contralateral dINs receive a gradual build-up of long lasting excitation which is likely to be NMDAR-mediated, but the neurons that relay this excitation to the opposite side have not yet been defined (e. g. ctINs or xINs, Fig. 6.2).

What would be the advantages to the tadpole of arranging the synaptic input to the dINs in this way? One suggestion is that providing synaptic input from a pathway to make a single side fire is simple, but arranging suitable excitation to ensure the second side fires half a cycle later is more complex. A working hypothesis is that the ipsilateral dINs receive short-latency, long-lasting input, which will cause them to fire at short latency in ~50% of cases. They also receive excitation over a longer duration, so that if they don't fire at short latency then they are maintained at a depolarised state (e. g. *firing:* Fig. 6.3D middle blue, at arrow; *no initial firing-held depolarised*: Fig. 6.3E middle blue, asterisks). At the same time, the contralateral dINs receive gradually building excitation which reliably causes them to fire slightly later than the ipsilateral dINs would (e. g. Fig. 6.3D, E lower red, arrowheads). This would mean that if the ipsilateral dINs fire at short latency then the contralateral side would still receive sufficient input to fire slightly later (mid-cycle), setting up an antiphasic rhythm, On the other hand if the ipsilateral side does not fire initially at very short latency, then the contralateral side will still fire, causing inhibition to the ipsilateral

dINs and because the ipsilateral dINs are already depolarised, they will then fire their first spike due to PIR.

I built network models to address two main questions about the synaptic input initiating swimming: *a)* what effect does electrical coupling have on the recruitment of the dIN population in a single sided network of 30 electrically coupled dINs; and *b)* what forms of synaptic drive from initiation pathways would lead to robust initiation of swimming but also allow stochasticity on the side of the body that where swimming starts in a bilateral network, consisting of two populations of dINs and two populations of cINs?

6.3 METHODS

6.3.1 *tIN spike time model*

To model the synaptic input which dINs receive following sensory stimulation, I used data from whole-cell patch-clamp recordings made in the tINs showing responses to head-skin stimulation [Buhl et al., 2012]. In life a tIN will fire between 0 and 5 spikes depending on the stimulus level, so I built a simple model that generated a set of spike times for a single tIN in response to graded stimuli. The stimulus strength, s, is normalised so that $s = 100\%$ corresponds to a head-skin stimulus at the threshold required to initiate swimming (Experimentally, s is the normalised amplitude of injected current used for electrical head-skin stimulation). This model was used to drive EPSPs in the dINs to model excitation from a population of 20 tINs in the following experiments. In the tIN spike time model, the number of spikes fired, n, at given stimulus level, s, is generated from the probability distribution $p(N = n \mid S = s)$ (Fig. 6.4A). This simple model was based on the observations that: *a)* the mean threshold stimulus that leads tINs to fire a single spike is 95% ($94 \pm 6\%$), (i. e. $p(N = 1 \mid S = 95\%) = 0.5$); *b)* at 100% stimulus all tINs fire a single spike (i. e. $p(N = 1 \mid S = 100\%) = 1.0$); *c)* as stimulus strength increases above 100%, some tINs begin to fire multiply, but some always fire a single spike (10/34) ($p(N = 1 \mid S > 120\%) = 0.3$); and *d)* the distributions for the number of spikes firing at high stimuli ($s > 120\%$) were estimated based on counts spikes at higher stimulation levels [Buhl et al., 2012]. Next, the number of spikes, n,

fired by a model tIN is converted into timings, $\{t_1, t_2 \ldots t_n\}$. The time of the k^{th}spike, t_k is generated from a normal distribution $t_k \sim N(\mu = \mu_k, \sigma = \sigma_k)$, where μ_k and σ_k are the means and standard deviations of the k^{th} spike (Fig. 6.4B). The values of μ_k and σ_k were calculated from experimental data taken at all levels of stimulation.

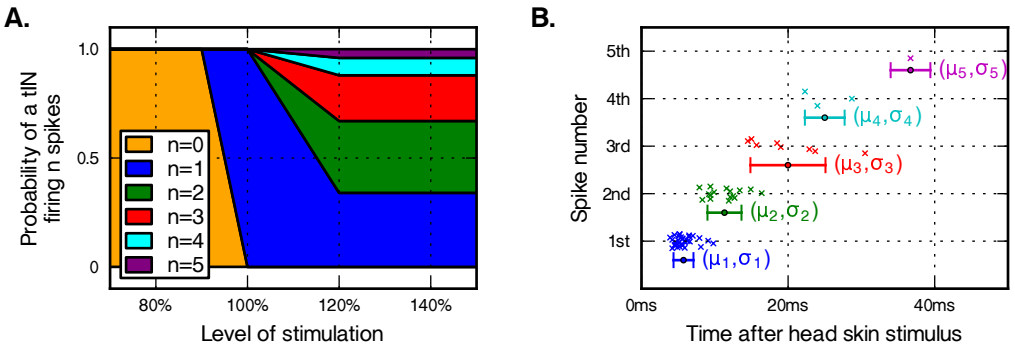


Figure 6.4 – A simple generative model of spike times for a single tIN. **A.** The probability distribution of a single tIN firing different numbers of spikes at levels of head-skin stimulation. **B.** The times of the spikes measured experimentally are shown as coloured crosses and the means and standard deviations (μ_k and σ_k) used to generate the spike times of a model tIN (means are shown as coloured circles, standard deviations are shown as horizontal error bars).

The population of tINs have descending axons, which are thought to make synapses onto the dendrites of the dINs [Buhl et al., 2012] (Fig. 6.2). The close anatomical proximity and short, constant EPSP latencies suggest these connections are monosynaptic (although see [Berry and Pentreath, 1976; Parker, 2010]). Paired tIN-dIN recordings suggest that the contact probability is high (~92%), but a spike in a tIN only causes an EPSP in a dIN in approximately 50% of cases. Experimentally, it is not clear why the probability of causing an EPSP is low, two possibilities are that action potentials are failing to propagate along the tIN axons or there is a low probability of transmitter release at the presynaptic terminal. In the modelling, I assumed the connections were monosynaptic with 100% tIN-dIN connectivity and that a spike in a tIN had a 50% chance of causing an EPSP in each dIN [Buhl et al., 2012]. A set of spike timings was constructed for the population of 20 tINs and a simple model of unreliable synaptic transmission was implemented in NEURON (see Section D.1.1).

6.3.2 *xIN & yIN spike time model*

The neurons in the pathways which carry excitation to contralateral population of dINs in response to head-skin stimulation have not yet been identified, but the EPSPs observed in contralateral dIN recordings show that the latencies are longer than those from the tIN pathway. The latencies of EPSPs in contralateral dINs following head-skin stimulation were measured and a shifted gamma distribution was fit to the times using maximum-likelihood estimation (Fig. 6.5) [Crawley, 2007]. The probability density of a spike occurring at a particular time, t, is given in Eqn. 6.1, where $\alpha = 1.07$, $\beta = 20.15$ and $t_{start} = 7.48$. This distribution is used to generate EPSP times for NMDAR synapses onto dINs in bilateral experiments c & d, where the strength of the stimulation is used to determine how many EPSPs would be generated (described in Section 6.4.5).

$$\begin{aligned} p\,(\text{spike at time } t \mid \alpha, \beta, t_{start}) &= \beta^{\alpha} \frac{1}{\Gamma(\alpha)} (t - t_{start})^{\alpha-1} e^{-\beta(t-t_{start})} \\ \Gamma(\alpha) &= \int_0^{\infty} x^{\alpha-1} e^{-x} dx \end{aligned} \tag{6.1}$$

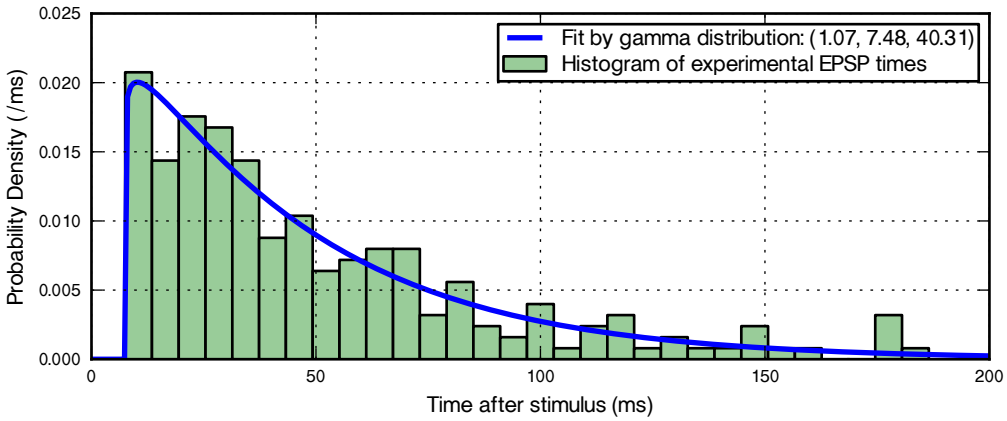


Figure 6.5 – A histogram of the latencies of contralateral EPSPs in dINs measured experimentally (green) and the fit with a gamma distribution (blue).

6.4 RESULTS

6.4.1 *The effect of electrical coupling on the response of a population of dINs to synaptic input*

I began by investigating the effect of electrical coupling on recruitment of dINs on a single side of the nervous system in response to input from the glutamatergic tIN pathway. I ran a series of model experiments in which spike times from 20 tINs were used to drive mixed AMPAR (g_{peak} = 250 pS) and NMDAR (g_{peak} = 300 pS) synapses on a population of 30 dINs. In each experiment, I counted the number of dINs in the population that fired an action potential and the latencies. The level of head-skin stimulation, *s*, used to generate the spike times for the tINs was varied from 90% to 105% in 1% steps. At each stimulus level, I repeated the simulations 10 times. Initially, the dINs were not electrically coupled, and then identical simulations were repeated in which the dINs were electrically coupled by gap junctions on their axons (as in Chapter 4).

The presence of electrical coupling in the population had two effects on the firing of the dINs. Firstly, it synchronised the spike times of individual dINs. In simulations without electrical coupling, the delay between the first and last action potentials was much greater than those with electrical coupling (Fig. 6.6). This effect was clearest at lower levels of head-skin stimulation (Fig. 6.6C). Increasing the level of stimulation lead to shorter latency and more synchronous firing in the population in both the coupled and uncoupled populations. At higher levels of stimulation, electrical coupling still synchronised activity, although the effect was not as pronounced (Fig. 6.6B). Secondly, electrical coupling had a dramatic effect on recruitment of the dINs population as the stimulus level was increased up to the stimulus threshold (s = 100%). When no electrical coupling was present, the number of dINs that fired in an experiment increased steadily, from no dINs firing (s = 90%) to the entire population firing (s = 100%) (Fig. 6.7A). However, when the population was electrically coupled, the number of dINs that fired in a given experiment became polarised and was either none or all of the population (Fig. 6.7B).

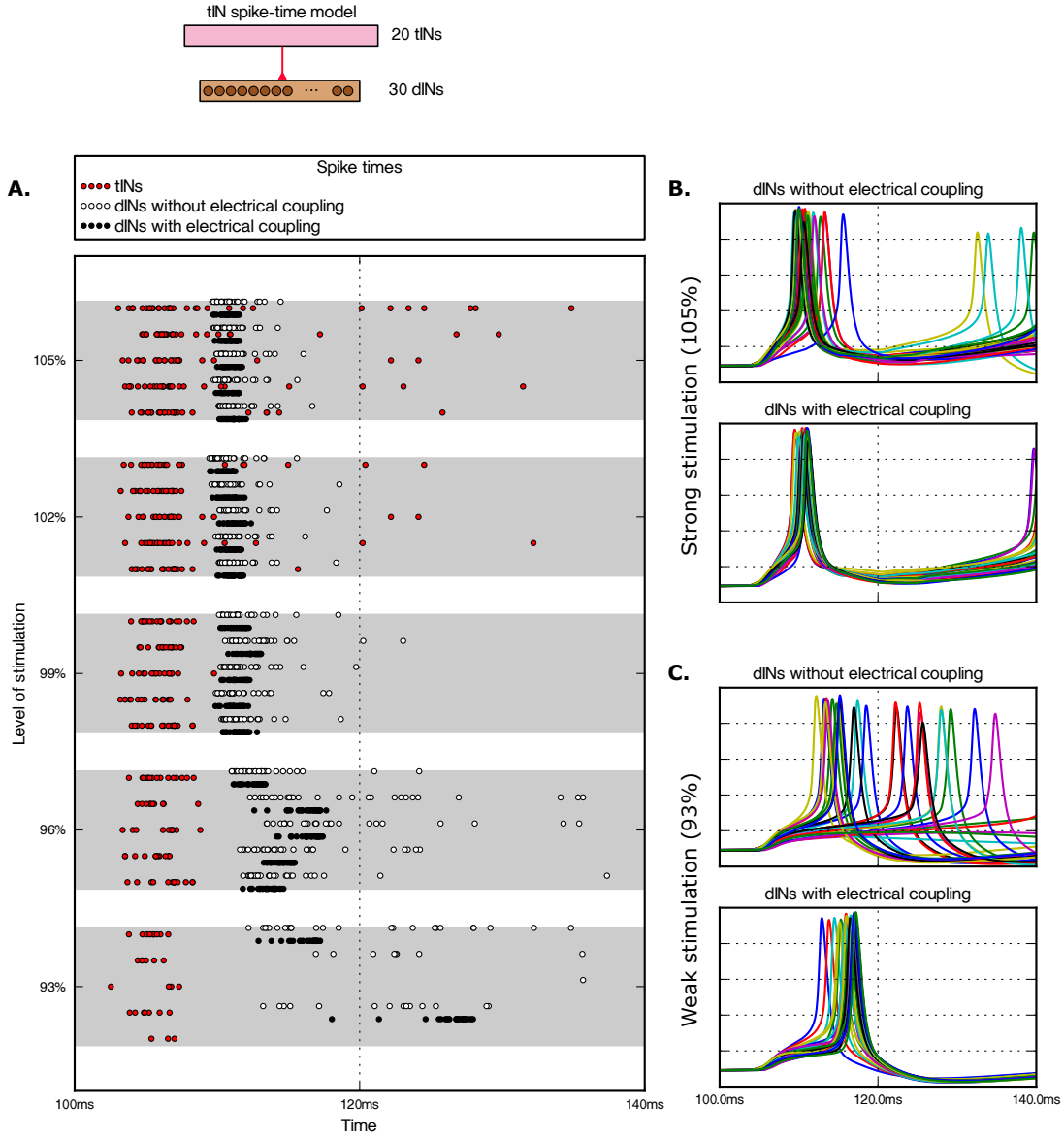


Figure 6.6 – The effects of electrical coupling on a population of dINs in response to synaptic input from a population of tINs at different levels of stimulation. **A.** Spikes times of the tINs (red) and the dINs with (black) and without (white) electrical coupling between the dINs (only the first spike per dIN is shown. five repetitions are shown from the five levels of stimulation. **B, C.** Example voltage traces from the populations of 30 dINs in response to strong (B, s = 105%) and weak (C, s = 93%) stimulation. The examples in B & C correspond to the top raster plots for the stimulation level.

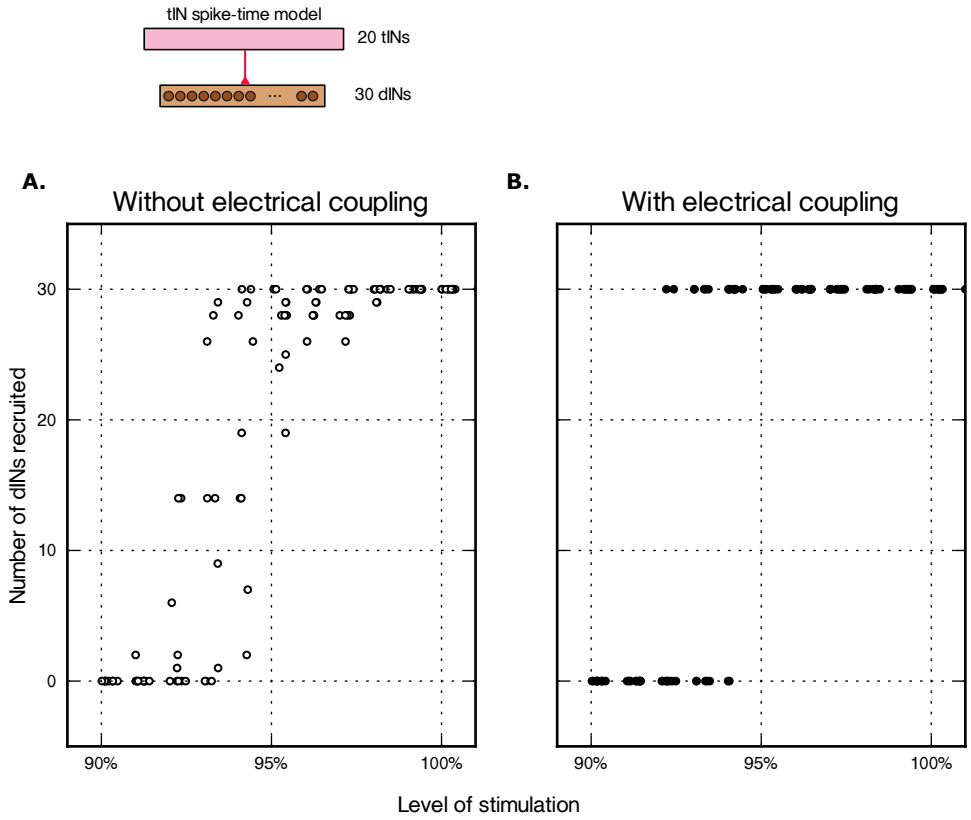


Figure 6.7 – The effect of electrical coupling on recruitment of the dIN population. Each point shows how many dINs from the population of 30 fired in response to input from the tIN pathway in a single simulation. 10 simulations are shown for each level of stimulation (in 1% increments). A small amount of noise has been added to the x-values.

6.4.2 *How does sensory stimulation initiate swimming?*

The previous experiments show that in response to excitatory drive, a single population of electrically coupled dINs could be recruited to fire synchronously and as a group. To investigate how the interactions between the two sides of the network affect initiation, I built a bilateral model in which each side contained a population of 30 electrically coupled dINs, and a population of 20 cINs (Fig. 6.8A, B). The simplified model did not include aINs or MNs [Roberts et al., 2010] and I assumed that activity in the dIN population would represent activity in the muscles on each side (Fig. 6.2). dINs made AMPA synapses onto ipsilateral cINs and cINs made inhibitory synapses onto both contralateral dINs and cINs. The connection probability was 0.3 in all cases [Li et al., 2007a] and the synaptic strengths are given in Section 2.3.1. The dINs made excitatory feedback connections onto themselves as described in Chapter 5. The dIN population model presented in Chapter 4 fires over a wider range of frequencies than is observed experimentally. I therefore reduced the firing frequency to more realistic levels by reducing the strength of dIN-dIN NMDAR-mediated feedback conductance to 100 pS (see Chapters 5 and 7 for discussion about the dIN firing frequency). In a one-sided model, a population of dINs coupled with such synapses sustains rhythmic activity at ~30 Hz (see Chapter 5). In the simulations, the network was symmetrical so I only considered stimulation to the right-side of the head, although the resulting synaptic input to the two populations of dINs could be different on the ipsilateral and contralateral sides (Fig. 6.8B, C; experiment d).

Experimentally, it has been difficult to find the neurons in the contralateral pathway, so I used modelling as a way to investigate hypotheses about the effects of different forms synaptic input to the CPG. Specifically, four types of EPSP distribution patterns were used to drive the dINs: *a)* short-duration, symmetrical input to both sides, like that from the tIN spike time model, mediated by AMPAR; *b)* short-duration, symmetrical input to both sides, like that from the tIN spike time model, mediated by NMDAR; *c)* long duration symmetrical input mediated by to both sides, like that from the xIN EPSP time model; and *d)* an asymmetrical input in which both sides are given long–duration NMDAR-mediated input and the ipsilateral side is also given short-duration AMPAR-mediated input (Fig. 6.8C). These are described in the next sections.

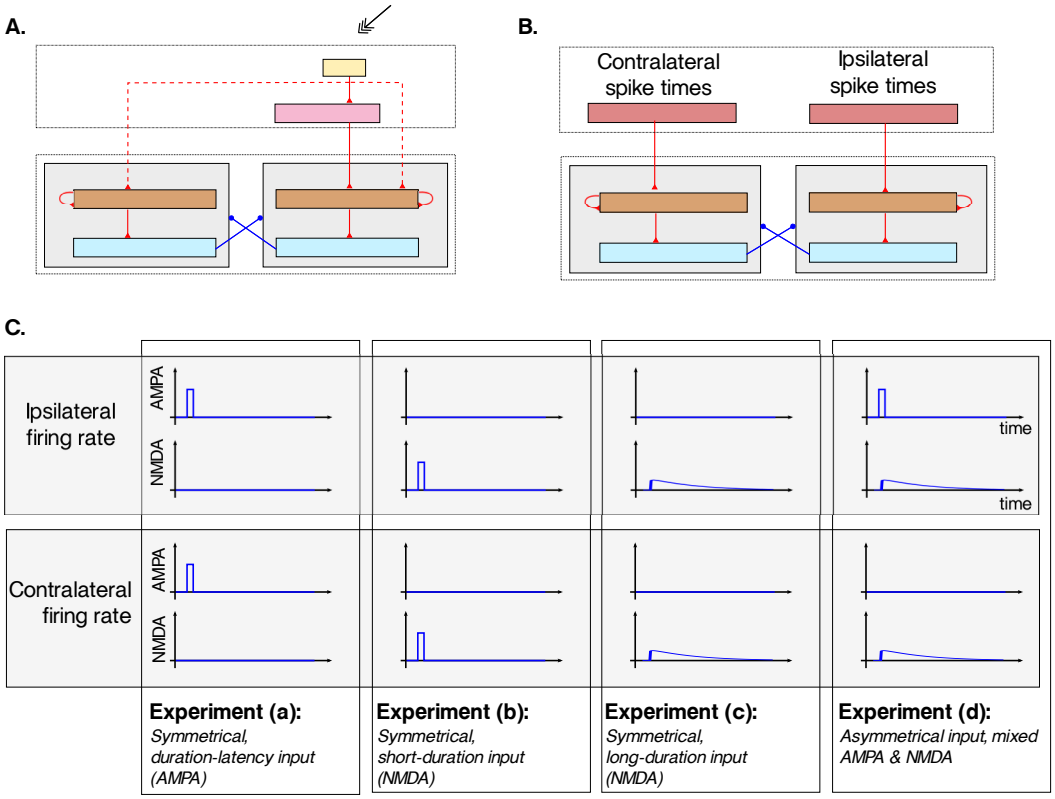


Figure 6.8 – Modelling the effects of synaptic input distributions on initiation of a bilateral network. **A.** The tadpole initiation pathway and simplified CPG model. **B.** The network model used to investigate the effects of different types of synaptic input to the dINs. **C.** A simplified cartoon of the three distributions of synaptic input given to each side. More details about the input are given in the following text.

6.4.3 *(a) Response of the bilateral network to symmetrical, short duration AMPAR-mediated input*

In the first bilateral experiment, I investigated whether symmetrical short-duration, synchronous input, like that from the tINs, could lead to swimming-like activity. Both populations of dINs were driven with symmetrically distributed, short-duration, AMPAR-mediated (g_{peak} = 250 pS) EPSPs (Fig. 6.9). I choose three levels of stimulation which defined the stimulus strength for generating EPSP times from the tIN model (weak s = 92%; medium s = 96%; strong s = 100%). At each level of stimulation, 8 experiments were run. At weak levels of stimulation, there was no activity in the network. As the strength of the stimulation increased, both populations of dINs would fire once in synchrony, but activity was not sustained.

6.4.4 *(b) Response of the bilateral network to symmetrical, short duration NMDAR-mediated input*

In the second experiment, I investigated the effect of using the same short duration distribution of EPSP times (Fig. 6.10), to drive long-lasting NMDAR-mediated excitation on the population of dINs (Fig. 6.10, g_{peak} = 250 pS).

At weak levels of stimulation, no sustained activity was seen in the network. At medium levels of stimulation, often the excitation was sufficient to cause both sides to fire in synchrony at short latency. In some cases, this synchronous firing continued. In other cases, only one side of the network sustained activity, which typically ceased within a few cycles. At strong levels of stimulation, both sides began and remained firing in synchrony, although in some cases it was observed that the network switched into antiphasic firing after several cycles.

6.4.5 *(c) Response of the bilateral network to symmetrical, temporally distributed input*

In the third experiment, the distribution of the input to each side was again symmetrical and mediated by NMDARs, but the times of the EPSPs to the dINs were generated using the xIN spike time model (Fig. 6.11). At medium stimulation, a number of spikes was chosen randomly using a uniform distribution ($n_{med} \sim U(20, 25)$), and a time generated for each spike using the gamma distribution found for the xIN spike time model. EPSP times for weak and strong stimulation levels were generated by using half (weak) and one and a half (strong) the number of spikes generated at the medium level of stimulation (n_{med}).

When the stimulation was weak, the two sides did not start firing in synchrony. Antiphasic activity was not sustained, and activity stopped either on one or both sides within a few cycles. When the level of stimulation increased, the network began generating antiphasic rhythm within a few cycles in most experiments. However, as the excitation was further increased, more simulations began to fire in synchrony from the first cycle, which often did not resolve into antiphasic firing.

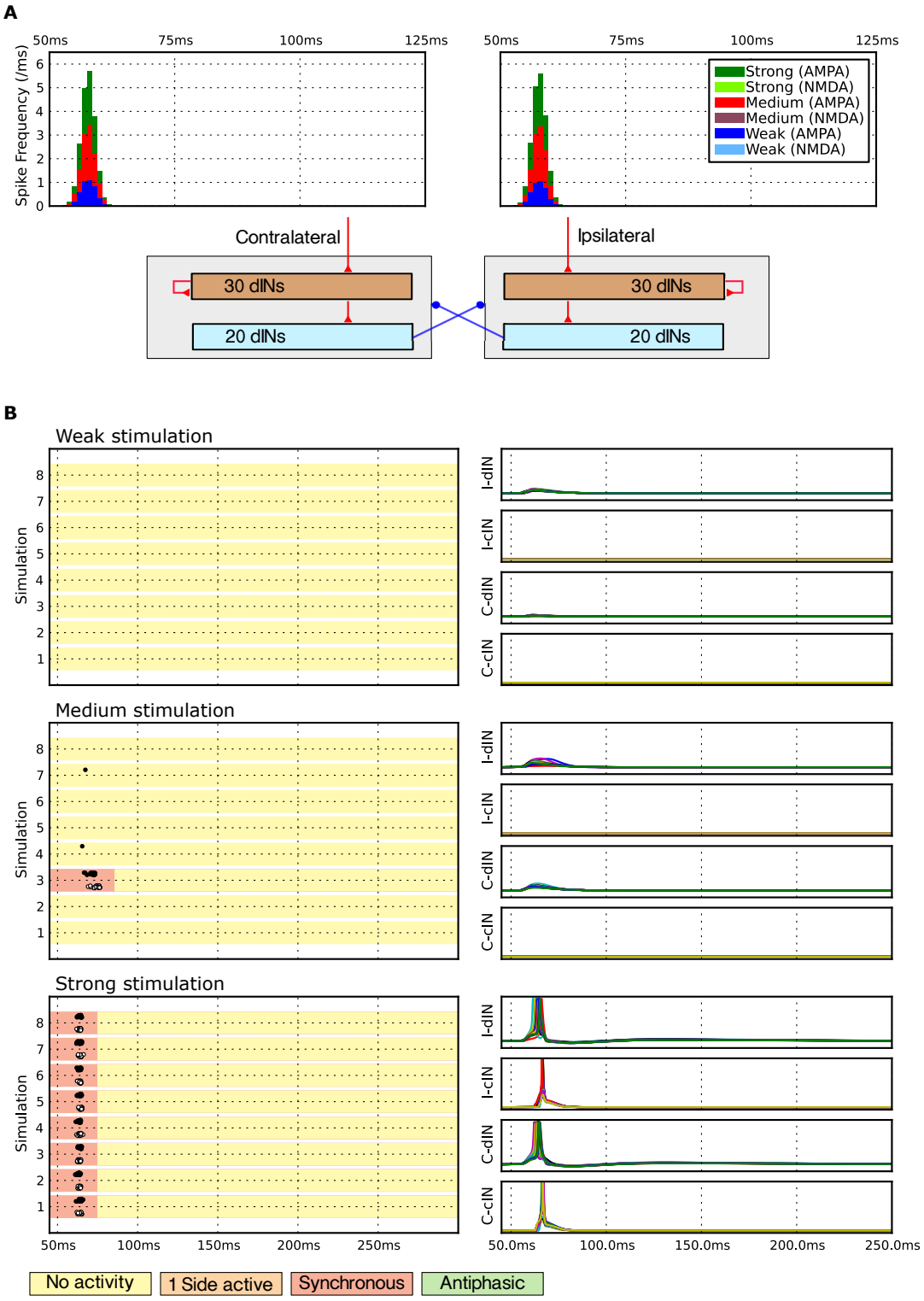
6.4.6 *(d) Response of the bilateral network to an asymmetrical input model*

Finally, I investigated whether a model in which the two sides were driven by asymmetric distributions of synaptic input would be able to initiate antiphasic rhythm. I used three sources of input to drive the dINs: an ipsilateral population of tINs, which provide AMPA-mediated short duration excitation to the ipsilateral dINs and a population of xINs and yINs on each side, that provide spike times distributed over a longer duration to NMDA synapses onto the dINs (Fig. 6.2, 6.12). The spike times for both the xINs and yINs were generated using the xIN model described above. At medium stimulation, the contralateral xIN population produced $n_{med} \sim U(20, 25)$ spikes, and the ipsilateral yIN population produced slightly fewer, $n_{med} \sim U(15, 25)$ (see the discussion).

As in the previous experiment, EPSP times for weak and strong stimulation levels for the xIN and yIN population were generated by using half (weak) and one and a half (strong) the number of spikes generated at medium level of stimulation (n_{med}).

At weak levels of stimulation, there was generally no network activity. As the level of stimulation increased, activity started synchronously or in antiphase. In all cases, within a cycle the activity resolved into antiphase, swimming-like activity. When the level of stimulation was strong, the ipsilateral side always fired first at short latency, which in most cases was followed by direct antiphasic firing, although in some cases, the two sides became synchronous and remained in synchrony.

Figure 6.9 ***(facing page)*** – The response of the bilateral network to short duration, symmetrically distributed AMPAR-mediated excitation. **A**. A histogram showing the mean rate of EPSPs, used to drive the on each side. The histogram was created by generating spikes for 1000 simulations, at different levels of stimulation. **B.** (Left) At each level of stimulation, the results from 8 experiments (colour-coded bars) are shown in a raster plot (left). The firing times of the individual are shown as circles. (black: ipsilateral; white: contralateral). The colour coded bars are given as a guide for the behaviour of the network. (Right) The voltage traces from the first experiments for ipsilateral (I) and contralateral (C) dINs and cINs are shown.


A
50ms
75ms
100ms
125ms
Spike Frequency (/ms)
Strong (AMPA)
Strong (NMDA)
Medium (AMPA)
Medium (NMDA)
Weak (AMPA)
Weak (NMDA)
Contralateral
Ipsilateral
30 dINs
20 dINs
30 dINs
20 dINs
B
Weak stimulation
Medium stimulation
Strong stimulation
Simulation
I-dIN
I-cIN
C-dIN
C-cIN
50ms
100ms
150ms
200ms
250ms
50.0ms
100.0ms
150.0ms
200.0ms
250.0ms
No activity
1 Side active
Synchronous
Antiphasic

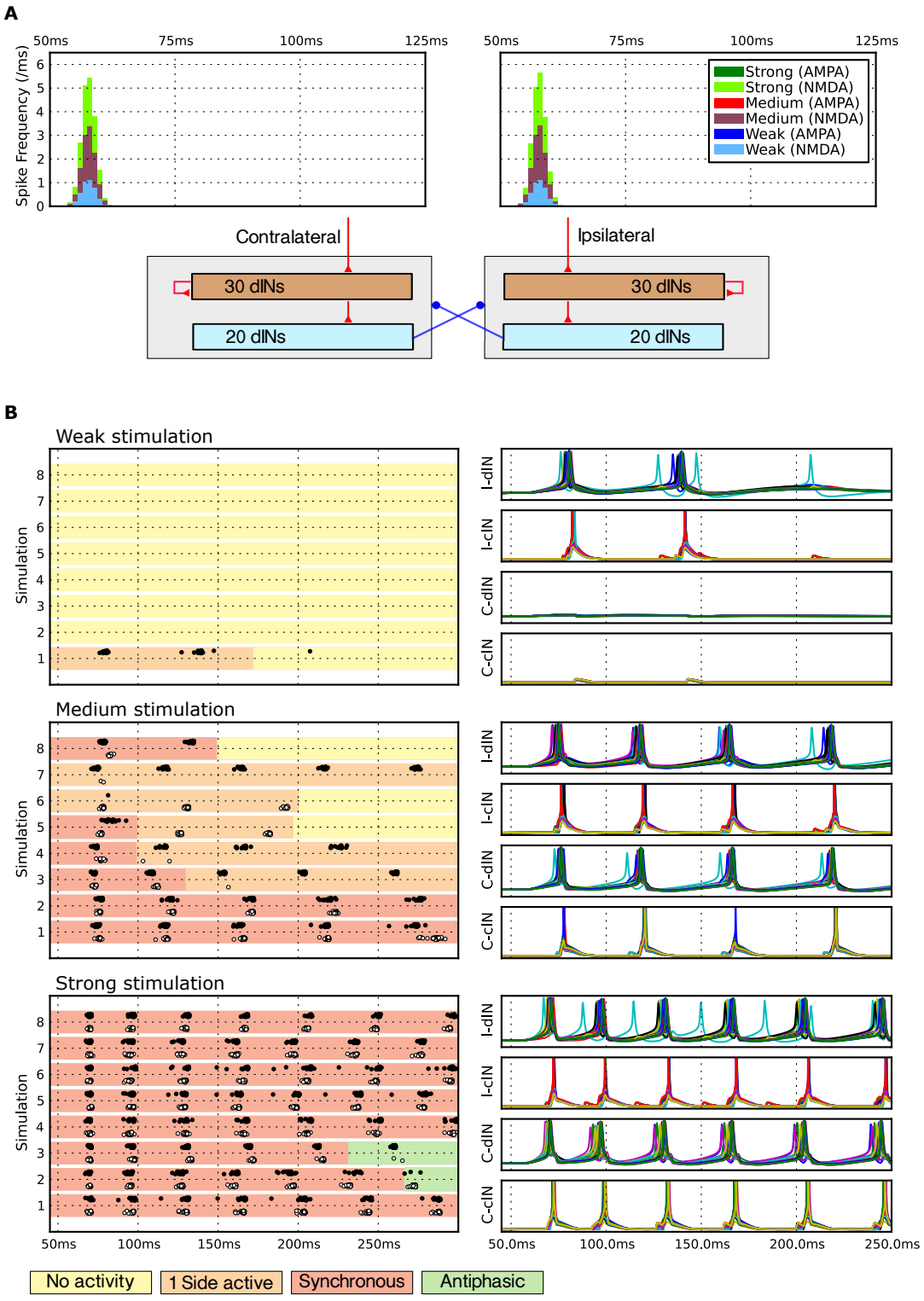


Figure 6.10 – The response of the bilateral network to short-duration, symmetrically distributed NMDAR-mediated excitation. (The layout of the results is the same as Fig. 6.9)

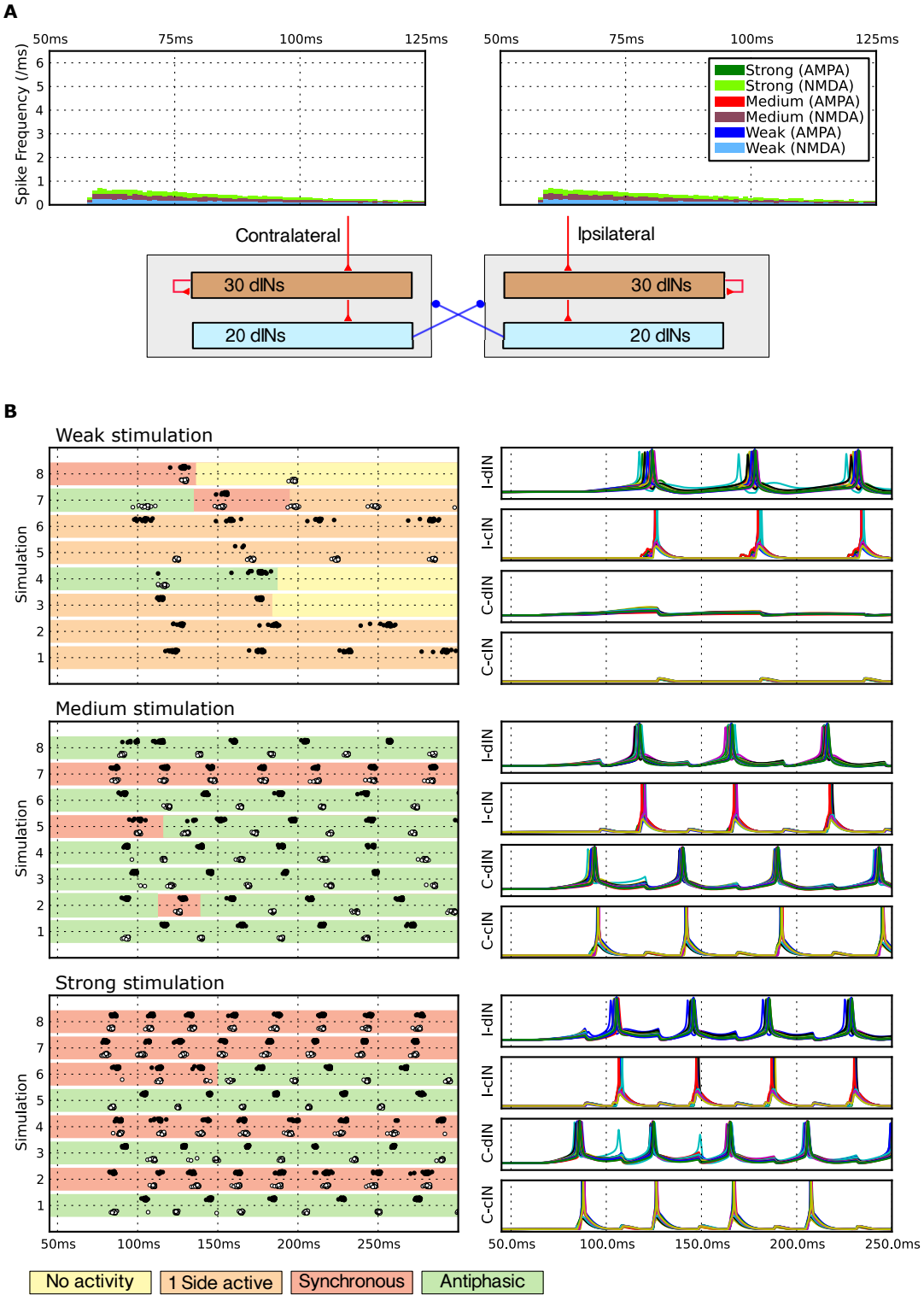


Figure 6.11 – The response of the bilateral network to temporally spread, symmetrically distributed NMDAR-mediated excitation. (The layout of the results is the same as Fig. 6.9)

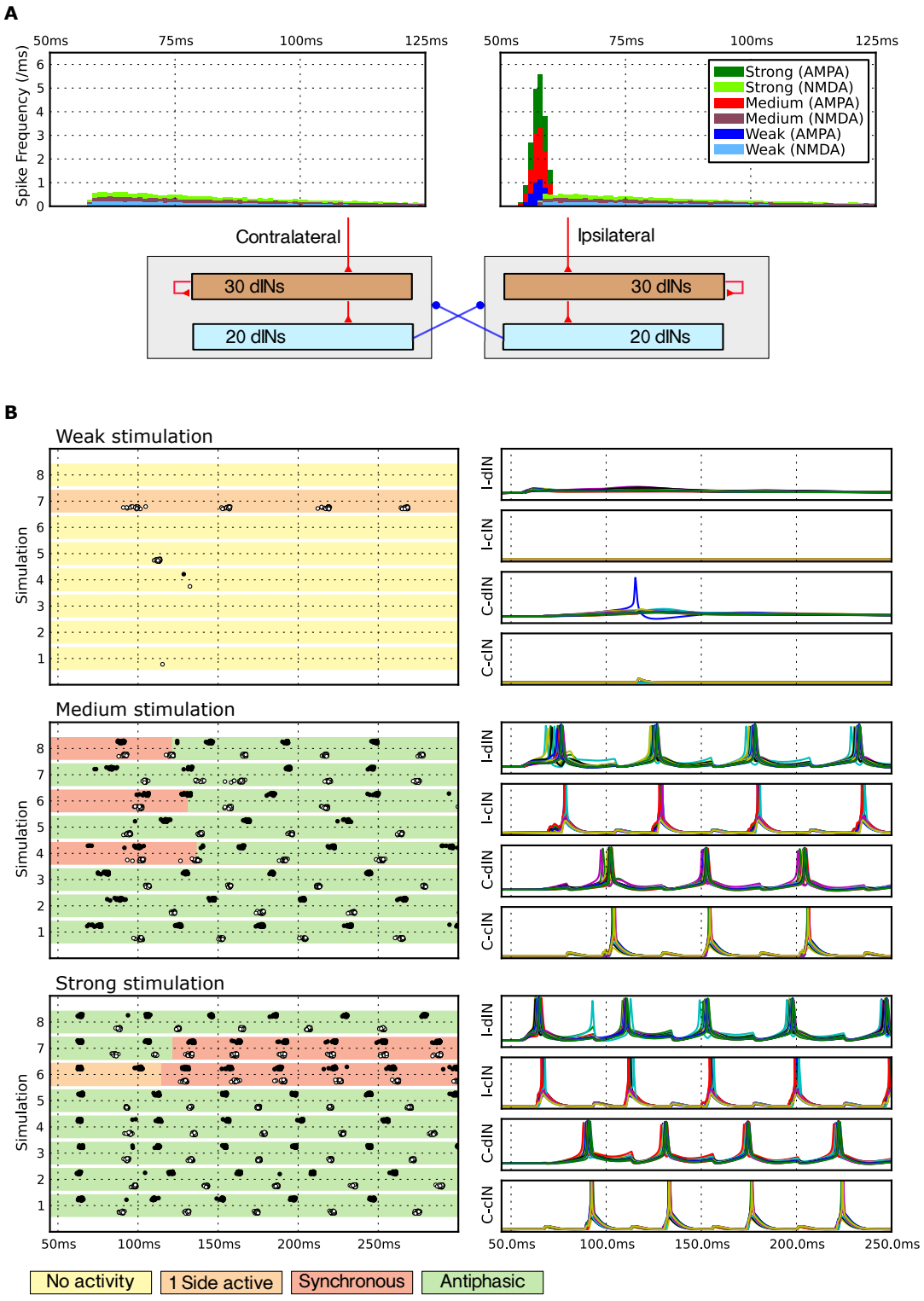


Figure 6.12 – The response of the bilateral network to asymmetrical, AMPAR and NMDAR-mediated excitation. (The layout of the results is the same as Fig. 6.9)

The results of the four simulations are summarised in Fig. 6.13, in which the resulting network activity 250 ms after the stimulus was given, is shown for the 8 simulations at the three levels of stimulation.

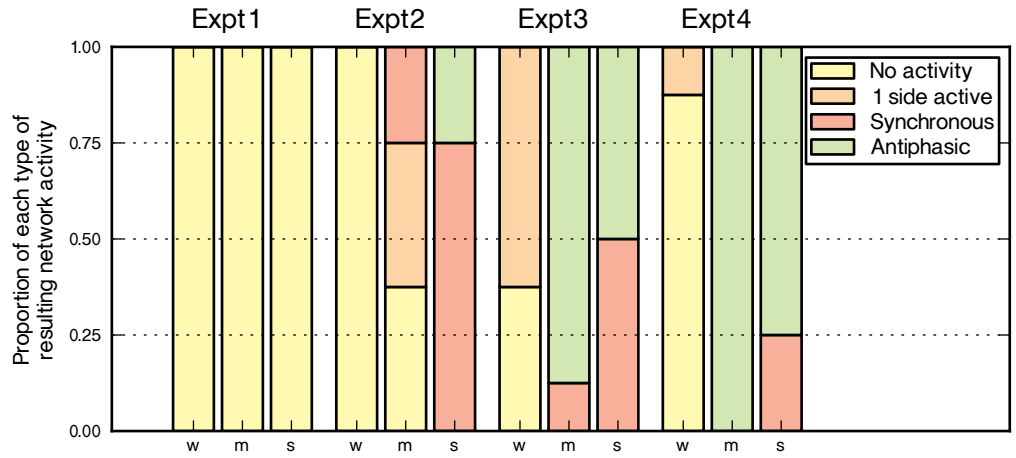


Figure 6.13 – A summary of the activity of the networks after 300 ms in the four bilateral initiation experiments (a-d). For each experiment, the results for weak, medium and strong stimulation are shown (*w*, m, s).

6.5 DISCUSSION

An animal's ability to respond effectively to sensory stimulation is central to its survival. Swimming away from danger is one of the few defence mechanisms for a hatchling tadpole. Robust pathways to initiate swimming are crucial - if a tadpole responds too slowly or too predictably then it may become an easier target for predators, however if it responds too readily, it wastes energy and risks drawing attention to itself [Lambert, 2004]. I have used modelling to investigate aspects of the head-skin sensory pathway at the level of individual neurons which would lead to the robust initiation of swimming.

6.5.1 *The effects of electrical coupling between dINs during initiation*

The modelling suggests that electrical coupling within the populations of dINs plays an important role during initiation of swimming. In simulations of a single popula-

tion of dINs, the electrical coupling between individual dINs, based on physiological evidence [Li et al., 2009], was sufficiently strong that the population would either all fire together, or not at all. Even when contralateral inhibition was introduced in the bilateral experiments, remarkably little *fragmentation* of the firing the populations of dINs was observed, and when it did occur, it was often cleaned up within a few cycles (for example Fig. 6.11 medium-5). Although the input model of tIN spike times is simplistic, in general terms, I would expect that the amount of synaptic input provided to the dINs from the input pathway would not be an unambiguous *swim* or *don't swim* signal, rather it will often fall around the threshold for swimming, i. e. around the threshold for dIN firing. Since there is variation in properties of the dINs within the population and stochasticity in the synaptic input, the effect of electrical coupling within a population of dINs will be central to prevent split in the population activity and reliably convert a continuously varying input into a binary decision. Unfortunately this hypothesis will be difficult to validate experimentally because it is difficult make intracellular recordings from more than two dINs simultaneously.

6.5.2 *The effects of different distributions of EPSPs on the initiation of activity in a bilateral network*

In a series of experiments, I have investigated the effects of different distributions of synaptic input from sensory pathways to the populations of dINs to initiate swimming in a bilateral CPG. There are many unknowns in this system, and care must be taken before interpreting these results too enthusiastically. The choice of unknown parameters, such as the strength of conductances from the xIN pathway, has been estimated, as have the relative strengths of the pathways. In life, the dINs appear to have a limited range of firing frequencies, even in response to high levels of current injection [Li et al., 2010]. In contrast, the dIN model has a broader range of firing frequencies and fires too quickly in response to strong stimuli (discussed in Chapters 5 and 7). This is a problem in this initiation model, because it makes it more likely that the two sides of the network will switch from firing in antiphase to firing synchronously. In order to offset this, the levels of excitation, both from the initiation pathway and feedback excitation were reduced and the closing time of the reciprocal inhibitory synapses

was shortened compared to what has been measured experimentally [Sautois et al., 2007]. With these changes, the modelling has pointed to some interesting issues in the initiation of activity in a bilateral network, described in the next paragraphs.

I found that the distributions of EPSP times to the dINs played a central role in determining whether swimming-like activity was generated by the network. The model was unable to initiate antiphasic activity if the EPSPs were given over a short duration, even if they were mediated by NMDARs (experiments a & b). In these experiments, when the stimulus was around threshold (medium), either one side would fire repetitively, or both sides would fire together in synchrony, and as the stimulus increased, sustained synchronous firing was observed in most cases. When the EPSPs were distributed over a longer duration, (as in experiment c), antiphasic activity started reliably in almost all simulations at medium levels of stimulation, but as the stimulation strength increased the two sides would initially fire together and remain in synchrony. An explanation for this is that when the level of stimulation is medium, there is a longer latency before each side fires the first action potential than when the level of stimulation was stronger. This means that the probability of the two sides firing their first spikes in synchrony is greater when the stimulation is strong rather than medium. These results suggest that symmetrically distributed input to the two sides could initiate swimming but that in order to prevent both side initially firing in synchrony, firstly the delay between the stimulus and the start of swimming would need to be long, and secondly the strength of the synaptic input to the dINs would need to be carefully tuned, and possibly limited by a mechanism to prevent over-excitation.

When I introduced asymmetry into the pathway, I found that the initiation of swimming was much more robust. At medium levels of stimulation, when there was not an early initial spike on the ipsilateral side, it was possible to get synchronisation of the initial firing, (which would often resolve into antiphase). At high levels of stimulation, the situation was different, the initial spike reliably fired at short latency, which started the network in antiphase, but in some cases, the system would begin to fire in phase shortly after. In the simulations that transitioned into in-phase firing, the dINs were firing at a higher frequency (~40 to 50 Hz), than is normally observed in life (~20 to 25 Hz [Li et al., 2010]). These results suggest that for a tadpole to reliably initiate

antiphasic activity from the first cycle of swimming, that is short latency, robust over a range of stimulation strengths and stochastic in the initial starting direction, then the initiation pathways will require more complexity than short-duration excitation. I have tested a hypothesis about how asymmetrical input, as is observed in the real tadpole, could lead to the initiation of swimming-like activity in the tadpole CPG and using the model has suggested why this additional complexity might be necessary for reliable initiation. The results suggest that slow, long duration excitation is important for providing sufficient background excitation to allow sustained activity after the initial spike on a side, and that initial synchrony can be avoided without needing long delays to the start of swimming by using different forms of synaptic input to the two sides.

6.5.3 *Asymmetry in the initiation pathway and the xINs and yINs*

In the animal, it has been observed that when stimulation is given near the threshold for swimming, the initial swimming direction is stochastic and there is a chance of firing starting on each side of the body [Boothby and Roberts, 1995; Buhl et al., 2012]. I found that to achieve this in the asymmetrical model, the long-duration excitation (i. e. from the yINs) to the ipsilateral side had to be slightly less than that to the contralateral side (i. e. from the xINs) (not shown). When input was given from the tIN sensory pathway, that was not strong enough to cause short latency firing in the ipsilateral dINs, it would still depolarise the neurons. Next, when the same amounts of long-duration background synaptic input were given to each side (from the xINs and yINs), this initial imbalance caused firing to often occur first on the ipsilateral side. In the model, I assumed that the animal is symmetrical, and so only considered input to a single side of the head. In reality, there are two sensory initiation pathways, one from each side of the head (Fig. 6.14). In the asymmetrical pathway model, the same distribution of spike times was used ipsilaterally and contralaterally to provide background excitation by the unknown neurons (xINs & yINs), but fewer spikes were generated in the ipsilateral population (xINs). One possibility is that the same populations of unknown neurons (xINs & yINs) could be used by the initiation pathway on each side of the head to provide the background excitation to the dINs (Fig. 6.14).

If the drive from the ipsilateral yINs to the dINs needs to be less than that from the contralateral xINs, then one way this could be achieved is by making the strengths of the connections from the sensory afferent neurons onto the contralateral neurons (Fig. 6.14 SC) stronger than onto the ipsilateral ones (Fig. 6.14 SI).

The modelling also suggests that only a small number of spikes from the xINs and yINs were needed in order to provide sufficient background excitation to the dINs so they could either fire from gradual build-up or on rebound (~10-30 spikes in total were used in experiments c & d). If it is assumed that these spikes were due to a small population of neurons firing repetitively at low frequency, the number of neurons in this population could be very small (<10). Interestingly, it has been difficult to find the neurons in the pathway carrying excitation to the contralateral side experimentally, potentially because they are also so few in reality.

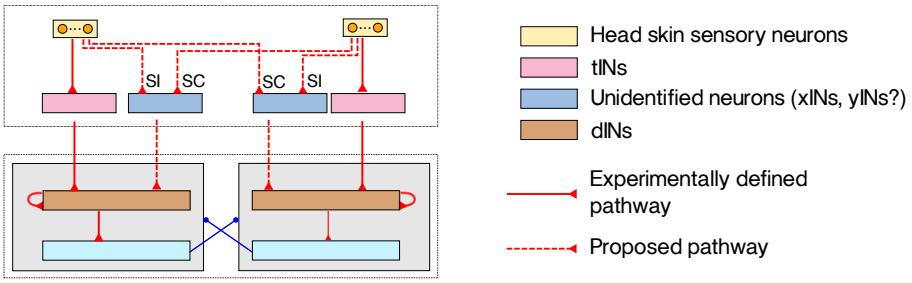


Figure 6.14 – A hypothesis for a two sided initiation pathway. The population of unidentified neurons (which I have referred to as xINs and yINs in the modelling) on each side of the tadpole are used to provide the ipsilateral and contralateral excitation to the dINs, depending on the side of stimulation. Modelling suggests that the background drive to the contralateral side needs to be stronger than that to the ipsilateral side, which could be arranged by having stronger contralateral connections (SC) from the sensory afferent neurons (yellow) to the contralateral neurons (blue) than ipsilateral connections to a symmetric population (SI).

6.5.4 *Further questions raised by modelling*

Two further questions have been raised by this modelling. Firstly, in life, it is rare to see single-sided activity during initiation. In several cases, particularly around threshold values, this is observed in the model. For example, in the voltage traces in Fig. 6.11 - weak stimulation, the ipsilateral side has become active and it is both

exciting itself, due to excitatory feedback connections, and preventing the opposite side from firing because of inhibition. The self-excitation causes the ipsilateral side to fire faster, which causes IPSPs to always arrive to the contralateral dINs before they are able to fire action potentials, leading to a reinforcing cycle. Are there mechanisms in the tadpole that prevent the domination of one side over another? One possibility is that the aINs could act against such situations occurring, because they provide local inhibition to the neurons on the same side and have a longer latency to firing than the cINs [Li et al., 2004b]. This would produce inhibition to the ipsilateral dINs (due to the aINs) that would be slightly later than contralateral dINs (due to the cINs), which might allow the contralateral side to *escape* and become active. Secondly, synchronous firing occurs in the model because of the time delay between the dINs firing and IPSPs arriving in contralateral interneurons, due to the delays in the synaptic pathways and in the firing of the cINs. This allows both populations of dINs to fire at that same time in synchrony and can be a stable firing pattern in the model and in other theoretical modelling studies of reciprocally inhibitory coupled oscillators [Terman et al., 1998]. One explanation, due to Hu et al. [2011], introduces the concept of a 'window-of-opportunity' for a neuron which describes the period of time between an action potential being fired and an IPSP occurring in a connected neuron. They suggest that for a pair of neurons connected with reciprocal inhibition, the overlap of their two windows allows synchronous firing, although why this should be a *stable* state is not clear. In the tadpole it is not yet clear which mechanisms resolve synchronous firing of the two sides into antiphase, as is seen for example in Kahn and Roberts [1982a]; Fig 6A.

6.5.5 *Outstanding issues with the model*

The asymmetrical-initiation–pathway model reproduces many general experimental observations; for example: swimming can start on a random side, there is a difference in the latencies when swimming starts on each side (Fig. 6.3) and higher levels of stimulation tend to cause swimming to start ipsilaterally and at shorter latency.

Where do the model and experimental results diverge? Although this modelling can give us intuition about why an initiation pathway may have a certain structure,

it has some limitations. One issue is the strength of NMDA feedback excitation within the population of dINs, which was reduced in order to limit their firing frequency. In life, the range of dIN firing frequencies are limited, which is likely to be due to an intrinsic, cellular-based mechanism that is absent from the model (see [Li et al., 2009] and discussion in Chapter 5). It is likely that the level of background excitation will have a strong effect on the dynamics of population firing during initiation, which is not properly captured by the model. Another issue is that when the tIN pathway is used to drive a population of 30 dINs, it seems to be having a stronger effect than what would be expected. We might expect that at threshold stimuli for swimming, (i. e. s = 100%), the population would be recruited less than 60%[1] of the time, but instead, in the single sided model, it is always recruited (Fig. 6.7). This suggests that the excitatory input from the tIN pathway to the dINs is too strong and one possibility is that the contact probability used is too high - In an electrically coupled population of dINs, EPSPs seen in one neuron will passively spread to other neurons and for example, in the case of tINs, I would expect that a spike in a single tIN would make synchronous EPSPs occur in ~50% of all the dINs, and therefore lead to substantial passive transmission to surrounding dINs, which didn't receive direct EPSPs. This makes it difficult to differentiate whether the EPSP-like bumps seen in intracellular recordings (Fig. 6.3) are the result of synaptic receptors opening in that neuron directly, or due to the passive propagation of EPSPs in surrounding dINs, and therefore care must be taken in estimating the numbers and synaptic conductances of EPSPs arriving to a dIN.

6.5.6 *Conclusions*

The physiological recordings have shown that there is asymmetry in the initiation pathways to populations of dINs on each side of the animal. Using modelling I have investigated issues surrounding robust initiation in a bilateral network, and suggest that initiation of swimming in the tadpole might be more subtle than simply driving both populations of dINs with short-duration synaptic input. This modelling suggests that the underlying structure of the pathways and distribution of different synaptic

1 since s = 100% is the threshold for starting swimming, and assuming that swimming starts on ~50% of trials on the ipsilateral side

receptors play an important role in effectively starting swimming. Introducing asymmetry into the head-skin pathway would allow an antiphasic drive to be delivered to the dIN populations to setup the two sides of the CPG to fire at short latency, in antiphase in response to a stimulus. What is now needed is more experimental data about the unidentified interneurons.

Part IV

DISCUSSION

7

GENERAL DISCUSSION

7.1 OVERVIEW

> "Philosophy (nature) is written in that great book which ever lies before our eyes - I mean the universe - but we cannot understand it if we do not first learn the language and grasp the symbols, in which it is written."
>
> – Galileo Galilei
> The Assayer

> "The usefulness of modeling has probably been greatly overestimated. Supplied with insufficient, often incorrect information, the model will usually have enough variable parameters for the modeler to produce any rhythm he desires for any circuit he sets out to model."
>
> – Allen Selverston
> Are central pattern generators understandable? [Selverston, 1980]

From the motion of the planets to the diverse range of life around us, science tries to explain the complex phenomena that we observe in the world using a small number of simple rules. Biophysical neuronal modelling falls on the boundary between physics and biology and the scientific approaches taken in these subjects are different [Hillis, 1993]. There is a very close link between physics and mathematics and over the last two centuries it has turned out that relatively simple theoretical assumptions can be used to make surprisingly accurate predictions about the real world. In contrast, biological systems are more messy and using a similar reductive approach is more problematic. There are fewer strong, quantitatively predictive, general theories and instead a broader approach must be taken to understanding systems [Parker,

2006]. Although fantastic progress has been made over the past 150 years in experimentally investigating nervous systems, it remains arguable whether we can point to any nervous system and say that we understand how it works. Nervous systems are complex systems composed of many inhomogeneous components. It is often very difficult to perform the experiments needed to understand the characteristics of individual components, because of their interdependencies. Even if we could perform all the ideal experiments, these systems are dynamic and non-linear. As a result, even in theoretical models, small changes in parameter values can lead to drastic changes in output, which makes making predictions difficult.

A commonly levelled criticism of modelling is that it *always works* and that by suitably adjusting unconstrained parameters any desired outputs can be generated [Selverston, 1980]. If there are not sufficient constraints on our models from experimental data, we will be forced to estimate parameters and it can be hard to conclude that the results of modelling studies provide any meaningful insight into the system rather than a reflection of the modeller's patience to sufficiently tweak parameters. There might be merit in this criticism, but it is important not to throw the baby out with the bathwater. Numbers are central to science, and in general the scope of testable predictions generated from qualitative theories is much more limited than from quantitative ones. Modelling can provide conceptual insight into a problem [Hillis, 1993]. It can highlight ambiguities in experimental data and also give insight into how tightly tuned parameters of a system need to be [Marder et al., 2007]. Modelling can allow investigation of phenomena which cannot be explained by existing hypotheses and allow *in silico* experiments to be performed that are not possible *in vivo*. A model is always an approximation to reality, and the appropriate level of detail will depend on the question being addressed. It is important to appreciate the limitations of a model and be open about problems with models, because areas in which a model is wrong demonstrate which aspects of a system are less understood and highlight directions for further enquiry [Parker, 2006, 2010].

7.2 SUMMARY OF WHAT HAS BEEN ACHIEVED IN THIS THESIS

In this thesis, I have used computational modelling to investigate how a small population of experimentally defined brainstem neurons (dINs) can generate rhythmic activity underlying frog tadpole swimming.

First I studied the electrical coupling between the dINs. I used anatomical and electrophysiological data in conjunction with modelling to place constraints on where the axo-axonic gap junctions could form on the thin unmyelinated axons and found that they need to form close to the soma and that most of the pairs of neurons were indirectly coupled via the axons of a third neuron. I found that axon diameter had a strong effect on the coupling between neurons and that it was not possible to achieve similar coupling coefficients to those observed experimentally when the axons were made thinner than 0.3 µm. I found that the presence of gap junctions in the network would affect the propagation of action potentials and that in order to match the model with experimental observations I had to distribute a higher density of sodium channels in an initial region of the dIN axons in order to allow action propagation whilst not producing a neuron that was too excitable. I found that the electrical coupling between neurons had a dramatic effect on their firing properties, proposed that the firing in the dINs may be modulated by the activity in surrounding neurons and hypothesised that isolated dINs would fire repetitively to step current injections. I found that the electrical coupling measured experimentally between dINs would be sufficient to synchronise their activity and lead to pacemaker firing in the population.

Next, I built on this hypothesis and was able to demonstrate that the excitatory feedback NMDAR synapses in a small one-sided population of dINs would allow the network to sustain activity following brief stimulation. I investigated the current-voltage relationship of the feedback NMDAR synapses between the dINs and found that the non-linearity allows the dINs to produce rhythmic activity when driven over wider range of synaptic conductances. I showed that rhythmic activity in the population could be switched on and off at short latency using biologically realistic synaptic input and that the NMDAR synapse feedback mechanism could be generalised to allow Hodgkin-Huxley type neurons to also sustain activity that could be switched on and off synaptically.

Finally, I investigated the initiation of activity in the dINs in more detail and showed that in response to synaptic input, the electrical coupling between the dINs causes either none or all of the population of dINs to be recruited which would provide a mechanism for converting a continuously varying sensory input into a discrete decision. I built models of the input pathways to the bilateral populations of dINs based on experimental data and found that during initiation the two populations of dINs could start and then remain firing in synchrony which is not observed experimentally. I found that in order to activate both sides in antiphase, simple short-latency excitation to the two sides was not effective and that asymmetrical synaptic input to each side of the nervous system, consistent with what is observed experimentally, was required in order to start swimming over a range of stimulation strengths. Finally, I proposed that a population of unidentified neurons on each side of the nervous system provide long-lasting the NMDAR-mediated excitation to the dINs and that this long-duration excitation is critical to the initiation of swimming in the tadpole in response to head-skin stimulation.

7.3 FEEDBACK FOR EXPERIMENTALISTS

In this thesis, I have produced a model that reproduces many features observed in the tadpole dIN population experimentally. However, there are discrepancies between the behaviour of the models and what is observed physiologically which have not been possible to reconcile and lead to new questions being raised. Moreover some of the modelling results have lead to predictions that can be explored in future experimental work.

CAN AXO-AXONIC GAP JUNCTIONS CAUSE ACTION POTENTIAL PROPAGATION FAILURE? In Chapter 4, I found that action potentials would fail to propagate along the thin axons of the multicompartmental dINs over regions with high densities of gap junctions when the coupled neurons were at rest. When the strength of resistance of the gap junctions was increased, the axon potentials could propagate. This suggests that the failure was due to current being shunted into the coupled neurons that were less depolarised. I also found (not shown) that if instead of increasing gap junction resistance, the coupled neurons were also depolarised, then action potentials

would also propagate along the axons over the gap junctions. This would be the case for example when all the dINs in the the network are rhythmically active. In general, if there were a synapse between neuron A (presynaptic) to neuron B (postsynaptic), and neuron A has a high density of gap junctions more proximal that the presynaptic terminal, then experimental paired-recording experiments of these two neurons might not detect the synapse. This is because paired recordings are normally taken with the rest of the network at rest, and so action potentials would fail to propagate along the axon of neuron A over the region of high gap junctions and so not reach the presynaptic terminal. However, during normal behaviour, when the electrically coupled neurons are also depolarised, the action potentials could successfully propagate along the axons to the presynaptic terminals and lead to Post-Synaptic Potentials (PSPs). This suggests that the apparent absence of postsynaptic potentials in paired recordings of neurons with gap junctions on their axons in experiments when the rest of the network is at rest does not necessarily imply that the synapses do not exist between the two neurons. This is difficult to avoid experimentally since in many cases recordings are intentionally made while the network at rest in order to avoid the barrage of surrounding network activity.

HOW CAN SYNAPSES BE DETECTED WHEN POSTSYNAPTIC NEURONS ARE ELECTRICALLY COUPLED? In the investigations into initiation by the tIN pathway, I assumed that although the population of 30 dINs all receive synchronous synaptic input, the synapses only cause an EPSP in response to an presynaptic action potential in 50% of cases. This was based on paired recordings from tINs and dINs which show that an action potential in a tIN leads to an EPSP in a dIN in approximately 50% of trials [Buhl et al., 2012]. In the model, even when a dIN did not receive a direct EPSP, the membrane voltage often looked as though it had because of the passive propagation of EPSPs from surrounding electrically coupled neurons, in response to synchronous input to 50% of the surrounding dINs. These passively propagated EPSPs were often about half the size of a direct EPSP (not shown). In life, paired dIN-dIN recordings in which sinusoidal currents were injected into one of the neurons show that the coupling coefficient between neurons drops to a third of its maximum at ~100 Hz (10 ms) [Li et al., 2009], which suggests that the RC-filtering properties of the network would not be sufficient to block the passive transmission of an EPSP between neurons [Li

et al., 2009]. The modelling suggests that we would also expect to see the passive transmission of PSPs in response to input from other populations (e. g. cINs, aINs & MHRs), which would be hard to distinguish from a *direct* EPSP, particularly in cases of synchronous input to the population. This suggests that in paired recordings, an observed EPSP in a postsynaptic neuron that is electrically coupled does not necessarily imply that a synapse exists between the two neurons. This phenomena adds to the already documented difficulties of reliably determining synaptic connectivity within a network [Berry and Pentreath, 1976; Parker, 2010].

DO ISOLATED DINS FIRE REPETITIVELY TO STEP CURRENT INJECTION? The relative roles of the network and cellular properties in rhythm generation in the tadpole network remain unclear. This modelling proposes a way in which two seemingly conflicting experimental observations could be reconciled: a) that the dINs only ever fire a single spike to step current injections irrespective of the stimulus strength [Li et al., 2006]; and b) that perfusing the dINs with NMDA causes them to fire as pacemakers [Li et al., 2010]. One possibility is that because intracellular step current injection and excitation by NMDARs are mediated by different ions (probably potassium and calcium respectively) and the flow of these ions is having a more profound effect than simply depolarising the dINs, perhaps activating secondary messenger systems [Hille, 2001]. Another possibility raised in this modelling work is that an isolated dIN will fire repetitively to step current injection and that the observed single-spiking behaviour is an experimental artefact due to the electrical coupling to hyperpolarised neurons, rather than a scenario that would occur during swimming. This suggests that an ideal experimental block of the gap junctions would result in an individual dIN firing repetitively in response to step current injections. Experimental application of gap junction blockers did not show this, but nor did they lead to a large change in input resistance, possibly because they were also having some side-effects (see below). Interestingly, repetitively firing neurons with descending axons that are active during fictive swimming have been identified in another experimental paper, but relatively few numbers of neurons were recorded [Aiken et al., 2003].

DOES AN ISOLATED SIDE OF THE NERVOUS SYSTEM FIRE REPETITIVELY WITHOUT THE NEED FOR INHIBITION? The modelling suggests that a single side of the

nervous system will sustain rhythmic activity following brief stimulation by the mutual self-excitation of NMDAR synapses among dINs, even when the conductance of NMDAR feedback is reduced to less than 50% of its measured value (Fig. 5.4). It has been difficult to demonstrate this definitively experimentally, especially since there is good evidence that lesioned spinal preparations have homeostatic mechanisms that act over the timescales of minutes [Hoffman and Parker, 2010; Moult et al., 2013]. As was discussed in Chapter 5, experimental ventral root recordings showed that one side of the nervous system of the tadpole can sustain activity for many cycles in response to brief electrical stimulation. Applying a pharmacological NMDAR blocker (AP5) abolished rhythm generation in a one-sided nervous system, and application of NMDA to a single side of the nervous system made it more excitable and led to an increase in the duration of episodes [Soffe, 1989]. Blocking the inhibitory synapses does not prevent rhythmic activity in one side of the nervous system, but the duration of activity is generally shorter. It has recently been suggested that contralateral inhibition is necessary for rhythm generation [Moult et al., 2013]. Light-driven outward proton pumps were expressed in cINs in the tadpole spinal cord which allowed them to be inactivated rapidly. It was found that inactivating these neurons during swimming could terminate the episode. This paper also found that the swimming process is affected by homeostatic mechanisms and that within 23 minutes of inactivating the cINs, one side of the nervous system was able to sustain rhythm. Unfortunately, it is not clear whether inactivation of cINs would prevent the initiation of swimming and one possibility is that the network and cellular mechanisms of rhythm generation are active at different times: at the start of swimming, the pacemaking properties of the dINs could play an important role, and later, once swimming is established, network mechanisms become more important.

WHAT IS THE EFFECT OF GAP JUNCTION BLOCKERS? In the model, an isolated dIN has an input resistance of approximately twice that of one in the electrically coupled network. If the experimental gap junction blockers were simply blocking the gap junctions then it would be expected that the input resistances of the dINs would increase by ~100% when a blocker (such as 18-β-glycyrrhetinic acid) is applied, however the input resistances only increase by about ~20% ([Li et al., 2009] Fig. 5).

One explanation would be that the electrical coupling is too strong in the model and interestingly it was difficult produce models of the passive network of 30 dINs with the distribution of coupling coefficients observed experimentally. However, the basic assumptions are that in the 300 µm region of nervous system there are 30 dINs and that 90% of the pairs are electrically coupled with coupling coefficients between 5% and 15% [Li et al., 2009]. Simple calculations with these numbers suggest that when current is injected into a single dIN experimentally, a substantial proportion of the current flows out of this neuron through gap junctions and across the membranes of coupled neurons. Although it is possible that these estimates are too high, it has also been observed that a strong hyperpolarising input into a single dIN during swimming can be sufficient to stop activity, which suggests that the coupling is also strong in the animal [Moult et al., 2013]. A more likely possibility is that the gap junction blockers somehow disrupt the gap junctions and also have side-effects on the cellular properties of the neurons. One idea is that instead of flowing across the gap junction, the blocker is causing the current to leak out instead, which causes lower coupling coefficients to be observed between dINs.

WHAT LIMITS THE PACEMAKER FIRING FREQUENCY OF THE ELECTRICALLY COUPLED DIN POPULATION? The motor neurons are directly driven by the dINs, and the dINs are the first neurons to fire on each side on each cycle during swimming [Soffe et al., 2009]. Rhythm at swimming frequencies can be generated in a single side of the nervous system, without the need for local inhibitory feedback [Soffe, 1989]. In the dIN model, the firing frequency increases with excitation and can reach up to 70 Hz. It is difficult to isolate the dIN population experimentally. It has been found that the frequency of swimming is correlated with the strength of background excitation to the dINs, but the reliability of cIN firing is also correlated with frequency of swimming [Li and Moult, 2012]. In another set of experiments, when the population of dINs has been perfused with NMDA, current injection into a single dIN does not cause its frequency to increase beyond ~35 Hz [Li et al., 2010]. All this suggests that the low firing frequencies observed in the dINs could be the result of their cellular properties. In other systems, inactivating outward currents are proposed to regulate the lengths of cycle periods [Connor and Stevens, 1971; Hille, 2001]. I concluded that an inactivating A-type potassium current was present in other tadpole CPG neurons

in Chapter 2, but the time-course was thought to be too quick to have a significant effect on cycle period (channel activation/inactivation: 4 ms; a swimming frequency of 25 Hz (40 ms period)). Only more direct evidence on dIN currents will resolve this.

7.4 MANAGING SIMULATION COMPLEXITY

An experimentalist requires suitable instruments to effectively address scientific questions. Similarly, a modeller requires suitable software tools to build and experiment on models. In Chapter 3, I presented three prototype pieces of software designed to make it easier to build and manage simulations of multicompartmental neurons. The models in this thesis use a relatively simple and limited set of building blocks (~3 neuron types, ~10 types of channel, ~5 sources of input spikes, ~5 types of synapse). This thesis contains over 50 graphs produced by hundreds of simulations in over ~90 simulation scripts, and in order to be confident the models act as expected I have built tools specifically with the focus of managing them. All the graphs of simulations in this thesis, and the thesis itself, can be built from scratch using a single command. This should allow someone continuing development on the multicompartmental dIN model to quickly understand the work and make their own changes. As a community, if we want to build more complex models and validate them against experimental results, the infrastructure needed to manage the data and simulations will become more complex. The scientific community has limited resources and we need to move towards standardised, flexible platforms that everyone can use and contribute to [Crook et al., 2012]. In biophysical modelling, the problems faced are not from understanding the concepts behind mathematical models but from the difficulties in quickly and reliably converting our ideas into simulations and then managing and communicating them [Wilson, 2006]. To do this, we need to eliminate unnecessary complexity, avoid sources of trivial errors and provide libraries that allow us to succinctly build models without needing to regularly reinvent the wheel. We should develop and standardise libraries, file formats and object models that simplify mundane tasks to allow us to focus on the exciting scientific questions.

7.5 FURTHER DIRECTIONS FOR MODELLING

A model of the dIN population is likely to be central in any future simulation studies of tadpole swimming. Further refinement of the model presented in this thesis will need more experimental data, specifically about the properties of the active dIN channels and more generally on synaptic kinetics. More data would be valuable about the amounts and types of currents flowing across synapses and the synaptic dynamics: whether they saturate, depress or facilitate, as well as further characterisation of the NMDAR voltage dependency. These are particularly important for understanding the mechanisms that limit the firing frequencies of the dINs during sustained activity.

Intracellular recordings provide direct evidence about the electrical activity in a neuron. One problem with fitting models is knowing what to fit against — what is a *typical case* and what is a *good illustration* of a principle. In the tadpole, following years of electrophysiological investigation, there are literally thousands of paired recordings including information about the neuron types, locations and drug perfusions. Often, simple parameters such as input resistance are manually calculated from these recordings and summarised in spreadsheets. However these recordings contain much more information, and although it would take a lot of work to suitably format and index these datasets, doing so would provide an invaluable resource for further modelling. It would allow analysis algorithms to be run on the time series, for example, to ask how firing frequency changes throughout a swimming episode, how many spikes do aINs fire at the start of swimming, or to try and infer how many EPSPs a neuron receives using automated time series analysis (e. g. [Molkov et al., 2012]). This would also allow more detailed analysis into the variability of properties within neuronal classes to better understand the effects of noise on the system.

One limitation of intracellular recordings is that only a limited number of neurons can be recorded simultaneously. Another exciting possibility would be the use of extracellular multi-electrode arrays (MEAs), which can record many extracellular potentials simultaneously in a fixed grid close to the neurons [Liu et al., 2012]. By assuming that these potentials are the result of transmembrane current flows, it is possible to estimate the currents flowing across membranes in different areas of the nervous system [Gold et al., 2006]. MEAs have been used to record from laminar sys-

tems, such as the retina. The layer of neurons in the spinal cord of the tadpole in relatively thin and the classes of neurons form columns. If a viable preparation could be made, such recordings could provide direct information about relative times of activation of neurons in different regions of the nervous system at high temporal resolution, which could be used in conjunction with the intracellular recordings to provide more of the big picture of how activity is initiated and maintained during tadpole swimming.

Part V

APPENDICES

A

DIN MODEL PARAMETERS

The morphology and compartmentalisation of the sections of the dINs is given in Fig. A.1. Adjacent compartments are electrically connected by resistances, calculated as the path integral between the centres of the two compartments of the axial resistance per unit length, $r_a = 4R_i/(\pi d^2)$, where d is the axon diameter and $R_i = 80\,\Omega\mathrm{cm}$ [Koch, 1999; Carnevale and Hines, 2006]. The transmembrane current flow in each compartment is given in Eqn. A.1, where i_{ext} includes currents due to electrical and chemical synapses, as well as external current injections. The current equations are given in Table A.2. The sodium, calcium and potassium channels have gating variables, which evolve according to Eqn. A.2, using the parameters specified in Table A.1.

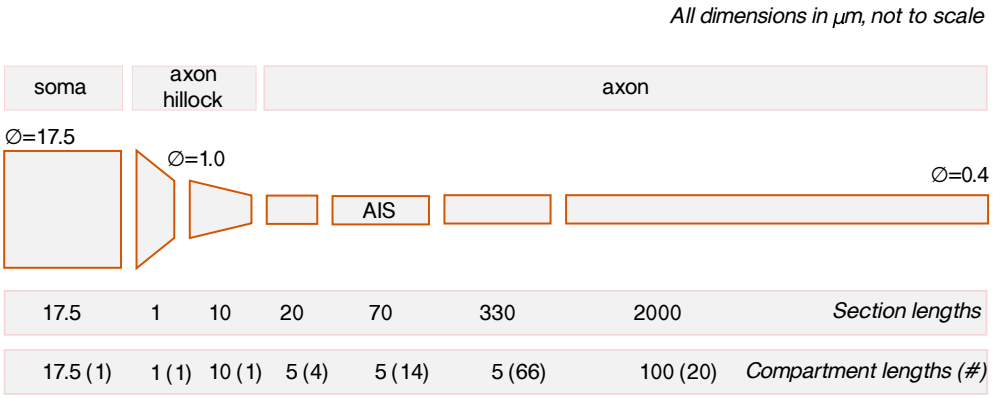


Figure A.1 – Side view of the morphology of the multicompartmental dIN model. The morphology is made of 7 sections, which are then further subdivided into smaller compartments. Each section is a conical frustra, and the diameter measurements at the joins are denoted by ∅. All dimensions are in μm. The Axon Initial Segment (AIS) is marked because it contains a different density of channels than the other regions of the neuron.

$$\frac{dV}{dt} = \frac{1}{C} i_{Lk} + i_{Na} + i_{Ks} + i_{Kf} + i_{Ca} + i_{ext} \tag{A.1}$$

$$\frac{dx}{dt} = \frac{x_\infty(V) - x}{\tau_x(V)} \qquad x_\infty(V) = \frac{\alpha_x(V)}{\alpha_x(V) + \beta_x(V)} \qquad \tau_x(V) = \frac{1.}{\alpha_x(V) + \beta_x(V)} \tag{A.2}$$

$$\alpha(V), \beta(V) \quad = \frac{A + BV}{C + exp\left(\frac{(D+V)}{E}\right)} \tag{A.3}$$

Table A.1 – The parameters of the forward and backward rate constants used in the channel models of different currents

CHANNEL	RATE CONSTANT	A (ms^{-1})	B ($ms^{-1}mV^{-1}$)	C ( )	D (mV)	E (mV)	
Calcium (Ca)	α_m	4.05	0	1.0	-15.32	-13.57	(a)
	β_m (for $V \le -25$ mV)	1.24	0.093	-1.0	10.63	1.0	
	β_m (for $V > -25$ mV)	1.28	0	1.0	5.39	12.11	
Fast Potassium (Kf)	α_n	5.06	0.0666	5.12	-18.396	-25.42	(b)
	β_n	0.505	0	0	28.7	34.6	
Slow-Potassium (Ks)	α_n	0.462	8.204e-3	4.59	-4.21	-11.97	(c)
	β_n	0.0924	-1.353e-3	1.615	2.10e5	3.33e5	
Sodium (Na)	α_m	8.67	0	1.0	-1.01	-12.56	(d)
	β_m	3.82	0	1	9.01	9.69	
	α_h	0.08	0	0	38.88	26.0	
	β_h	4.08	0	1.0	-5.09	-10.21	

(a) from [Dale, 1995a]. See Section 2.2.5 on page 50;
(b) See Section 2.2.2 on page 43;
(c) See Section 2.2.1 on page 38;
(d) from [Dale, 1995a]. See Section 2.2.4 on page 49;

Table A.2 – Final conductances and reversal potentials of the transmembrane currents used in the dIN model

CHANNEL (X)		CURRENT EQUATION i_X	REVERSAL POTENTIAL E_X	CONDUCTANCE DENSITY[(i)] $\bar{g}_X$	(NORMALISED)[(i)] $\bar{g}_X/\bar{g}_{Lk}$	POINT-NEURON EQUIVALENT $(\bar{g}_X \times 1000\,\mu m^2)$
Leak (Lk)	(a)	$\bar{g}_{Lk} \times (V - E_{Lk})$	$-52\,mV$	$0.125\,mS/cm^2$[(g)]	1	$1.25\,nS$
Sodium (Na)	(b)	$\bar{g}_{Na} \times (V - E_{Na}) \times m^3 h$	$50\,mV$	$25\,mS/cm^2$ (AIS : ×6.0)	200	$250.0\,nS$
Fast Potassium (Kf)	(c)	$\bar{g}_{Kf} \times (V - E_{Kf}) \times n^4$	$-81.5\,mV$	$1.25\,mS/cm^2$(AIS : ×1.5)	10	$12.5\,nS$
Slow-Potassium (Ks)	(d)	$\bar{g}_{Ks} \times (V - E_{Ks}) \times n^2$	$-81.5\,mV$	$1.0\,mS/cm^2$(AIS : ×1.5)	8	$10.0\,nS$
Calcium (Ca)	(e)	*(Formulation given in Section 2.2.5)*		$0.016\,cm/s$[(h)]	–	$0.16\,mm^3\,ms^{-1}$
(Capacitance)	(f)	–		$(1.0\,\mu F/cm^2)$	–	$10\,pF$

[(i)] Values used in the soma. Scaling factors for the axon initial segment (AIS) are given in brackets. For each dIN, noise is added to each conductance density, by multiplying $\bar{g}_X$ by $\epsilon\,\mathcal{N}\left(\mu = 1.0, \sigma^2 = 0.05\right)$ (values of ϵ are clipped to zero)

[(a-e)] See a: Section 4.3.2;[Sautois et al., 2007]; b: Section 2.2.4 on page 49; c: Section 2.2.2 on page 43; d: Section 2.2.1 on page 38; e: Section 2.2.5 on page 50;

[(f)] Value taken from the literature [Koch, 1999]

[(g)] An isolated model dIN has an input resistance of ~600 MΩ, but this decreases to the value measured experimentally (~350 MΩ, [Sautois et al., 2007]) when electrically coupled.

[(h)] Permeability, rather than conductance.

B

NEUROUNIT IMPLEMENTATION

I propose a grammar for defining units, quantities and systems for equations involving units and have defined three layers to support increasingly complex use-cases:

LEVEL-1 Simple units and quantities (e. g. `mV`, `10pA/cm2`, `0.1e-3 centimeter second`)

LEVEL-2 Expressions involving quantities (e. g. `(1/{300 MOhm})/{590 um2}`)

LEVEL-3 Systems of equations and libraries (e. g. see Appendix *C*).

It is difficult to write the grammar for neurounit as a single, context-free grammar because of the way in which whitespace needs to be handled and because in many cases the meaning of alphanumeric tokens cannot be ascribed immediately during parsing. (For example, whether an `F` denotes a variable name, Faraday's constant or the unit Farad.) Instead, parsing of expressions is performed in four phases: (*1*) preprocessing to normalise the input string; (*2*) parsing of the normalised string using a Backus-Naur Form (BNF) grammar; (*3*) resolution of the individual unit-symbol; and (*4*) inference of the dimensions of symbols by analysing the ASTs These are described in the following text.

B.1 IMPLEMENTATION DETAILS

B.1.1 *Phase 1: Preprocessing the input*

In life, when writing quantities involving units, care must be taken with the spacing between the characters. For example, `1 m s` (*1 meter second*) is not the same as `1 ms` (*1 millisecond*). In the proposed grammar, this whitespace is also important. In some languages, whitespace between the tokens does not carry meaning except to separate them and can be ignored after the tokenising phase. This simplifies the parsing phase, because whitespace no longer appears in the grammar definition. Unfortunately, since whitespace has meaning in `neurounit` syntax, it cannot simply be ignored. Instead, to reduce the complexity of the BNF grammar, a preprocessing step is used to normalise the string.

Whitespace is only kept if it lies between an alphanumeric character and an alpha character. (Two numbers separated by a whitespace is not valid). The keywords `and` `or` and `not` are converted into the symbols `&`, `|` and `!` respectively, and for LEVEL-3 strings, any empty lines are removed, and lines ending with a backslash are joined with the following line. For LEVEL-3 strings, we also add semicolons to the ends of lines. These simplifications reduce the complexity of the following parsing phase.

B.1.2 *Phase 2: Context-free grammar*

During this stage, the input string is converted into a set of tokens, and then these tokens are parsed by a grammar. A context-free grammar can be defined as a 4-tuple $G = (V_N, V_T, S, P)$ where V_N are the non-terminal symbols, V_T are the terminal symbols, S is the starting symbol and P is the set of production rules [Grune et al., 2000]. In `neurounit`, the three levels all use the same V_N, V_T and P, and only differ by changing the starting symbols, S. The terminal symbols consist of the simple single characters, (`<>()[]{};:+-/=,!&|`) a set of keywords (`if`, `else`, `not`, `and`, `or`, `from`, `import`, `as`, `library` and `eqnset`) and more complex regular-expression-based symbols, given in Table B.1. The full syntax is described by Grammars B.1-B.4.

Table B.1 – Non-trivial terminals symbols used in `neurounit`. The Python regular expression syntax is used.

TERMINAL SYMBOL	DEFINITION
IO_LINE	r"""<=> [^;]*"""
ONEVENT_SYMBOL	r"""==>>"""
INTEGER	r"""[-]?[0-9]+"""
FLOAT	r"""([-]?[0-9]+\.[0-9]*([eE][+-]?[0-9]+)?)| ([-]?[0-9]+([eE][+-]?[0-9]+))"""
WHITESPACE	r"""[\t]+"""
ALPHATOKEN	r"""[a-zA-Z_]+"""
NO_UNIT	r"""NO_UNIT"""
NEWLINE	r"""\n+"""

⟨*alphanumtoken*⟩ ::= ALPHATOKEN>
| ⟨*alphanumtoken*⟩ ALPHATOKEN
| ⟨*alphanumtoken*⟩ INTEGER

⟨*quantity_expr*⟩ ::= ⟨*quantity_expr*⟩ PLUS ⟨*quantity_term*⟩
| ⟨*quantity_expr*⟩ MINUS ⟨*quantity_term*⟩
| ⟨*quantity_term*⟩

⟨*quantity_term*⟩ ::= ⟨*quantity_term*⟩ TIMES ⟨*quantity_factor*⟩
| ⟨*quantity_term*⟩ SLASH ⟨*quantity_factor*⟩
| ⟨*quantity_factor*⟩

⟨*quantity_factor*⟩ ::= ⟨*quantity*⟩
| LBRACKET ⟨*quantity_expr*⟩ RBRACKET

⟨*quantity*⟩ ::= ⟨*magnitude*⟩
| ⟨*magnitude*⟩ ⟨*unit_expr*⟩
| ⟨*magnitude*⟩ WHITESPACE ⟨*unit_expr*⟩

⟨*magnitude*⟩ ::= ⟨*FLOAT*⟩
| ⟨*INTEGER*⟩

⟨*unit_expr*⟩ ::= ⟨*unit_term_grp*⟩
| ⟨*unit_term_grp*⟩ SLASH ⟨*unit_term_grp*⟩
| ⟨*parameterised_unit_term*⟩ SLASH ⟨*parameterised_unit_term*⟩
| ⟨*unit_term_grp*⟩ SLASH ⟨*parameterised_unit_term*⟩
| ⟨*parameterised_unit_term*⟩ SLASH ⟨*unit_term_grp*⟩
| ⟨*parameterised_unit_term*⟩

⟨*parameterised_unit_term*⟩ ::= LBRACKET ⟨*unit_term_grp*⟩ RBRACKET
| ⟨*LBRACKET unit_term_grp*⟩ SLASH ⟨*unit_term_grp*⟩ RBRACKET

⟨*unit_term_grp*⟩ ::= ⟨*unit_term*⟩
| ⟨*unit_term_grp*⟩ WHITESPACE ⟨*unit_term*⟩

⟨*unit_term*⟩ ::= ⟨*unit_term_unpowered*⟩
| ⟨*unit_term_unpowered*⟩ INTEGER

⟨*unit_term_unpowered*⟩ ::= ALPHATOKEN

⟨*empty*⟩ ::= ⟨⟩

⟨*whiteslurp*⟩ ::= ⟨*empty*⟩
| ⟨*WHITESPACE*⟩

⟨*white_or_newline_slurp*⟩ ::= ⟨*empty*⟩
| WHITESPACE
| NEWLINE
| ⟨*white_or_newline_slurp*⟩ WHITESPACE
| ⟨*white_or_newline_slurp*⟩ NEWLINE

Grammar B.1 – The BNF grammar for `neurounits` (Part 1: Units & Quantity terms)

⟨*assignment*⟩ ::= ⟨*lhs_symbol*⟩ EQUALS ⟨*rhs_generic*⟩

⟨*time_derivative*⟩ ::= ⟨*lhs_symbol*⟩ PRIME EQUALS ⟨*rhs_generic*⟩

⟨*rhs_symbol*⟩ ::= ⟨*localsymbol*⟩
| ⟨*externalsymbol*⟩

⟨*lhs_symbol*⟩ ::= ⟨*localsymbol*⟩

⟨*open_funcdef_scope*⟩ ::= ⟨⟩

⟨*function_def*⟩ ::= ⟨*lhs_symbol*⟩ LBRACKET ⟨*function_def_params*⟩ RBRACKET EQUALS ⟨*open_funcdef_scope*⟩ ⟨*rhs_generic*⟩

⟨*function_def_param*⟩ ::= ⟨*localsymbol*⟩
| ⟨*localsymbol*⟩ COLON LCURLYBRACKET RCURLYBRACKET>
| ⟨*localsymbol COLON LCURLYBRACKET <unit_expr*⟩ RCURLYBRACKET>

⟨*function_def_params*⟩ ::= ⟨*whiteslurp*⟩
| ⟨*function_def_param whiteslurp*⟩
| ⟨*function_def_params*⟩ COMMA ⟨*whiteslurp*⟩ ⟨*function_def_param whiteslurp*⟩

⟨*rhs_term*⟩ ::= ⟨*function_call_l3*⟩
| MINUS ⟨*rhs_term*⟩
| LSQUAREBRACKET ⟨*rhs_generic*⟩ RSQUAREBRACKET IF LSQUAREBRACKET ⟨*bool_expr*⟩ RSQUAREBRACKET ELSE LSQUAREBRACKET ⟨*rhs_generic*⟩ RSQUAREBRACKET
| LBRACKET ⟨*rhs_term*⟩ RBRACKET
| ⟨*rhs_term*⟩ PLUS ⟨*rhs_term*⟩
| ⟨*rhs_term*⟩ MINUS ⟨*rhs_term*⟩
| ⟨*rhs_term*⟩ TIMES ⟨*rhs_term*⟩
| ⟨*rhs_term*⟩ TIMESTIMES INTEGER
| ⟨*rhs_term*⟩ SLASH ⟨*rhs_term*⟩
| ⟨*rhs_variable*⟩
| ⟨*rhs_quantity_expr*⟩
| ⟨*quantity*⟩

⟨*function_call_l3*⟩ ::= ⟨*rhs_symbol*⟩ LBRACKET ⟨*func_call_params_l3*⟩ RBRACKET

⟨*func_call_params_l3*⟩ ::= ⟨*rhs_term*⟩
| ⟨*func_call_param_l3*⟩ ⟨*whiteslurp*⟩
| ⟨*func_call_params_l3*⟩ COMMA ⟨*whiteslurp func_call_param_l3*⟩

⟨*func_call_param_l3*⟩ ::= ⟨*alphanumtoken*⟩ EQUALS ⟨*rhs_term*⟩

⟨*rhs_generic*⟩ ::= ⟨*rhs_term*⟩

⟨*bool_term*⟩ ::= ⟨*rhs_term*⟩ LESSTHAN ⟨*rhs_term*⟩
| ⟨*rhs_term*⟩ GREATERTHAN ⟨*rhs_term*⟩

⟨*bool_expr*⟩ ::= ⟨*bool_term*⟩
| ⟨*bool_expr*⟩ AND ⟨*bool_expr*⟩
| ⟨*bool_expr*⟩ OR ⟨*bool_expr*⟩
| NOT ⟨*bool_expr*⟩
| LBRACKET ⟨*bool_expr*⟩ RBRACKET

⟨*rhs_variable*⟩ ::= ⟨*rhs_symbol*⟩

⟨*rhs_quantity_expr*⟩ ::= LCURLYBRACKET ⟨*quantity*⟩ RCURLYBRACKET

Grammar B.2 – The BNF grammar for `neurounits` (Part 2: Expressions & Functions)

⟨*text_block*⟩ ::= ⟨*white_or_newline_slurp*⟩
| ⟨*text_block*⟩ ⟨*block_type*⟩

⟨*block_type*⟩ ::= ⟨*eqnset_def*⟩
| ⟨*library_def*⟩

⟨*open_eqnset*⟩ ::= ⟨*empty*⟩

⟨*eqnset_def*⟩ ::= ⟨*eqnset_def_internal*⟩

⟨*eqnset_def_internal*⟩ ::= EQNSET ⟨*open_eqnset*⟩ WHITESPACE ⟨*namespace*⟩ LCURLYBRACKET ⟨*eqnsetcontents*⟩ ⟨*white_or_newline_slurp*⟩ RCURLYBRACKET ⟨*white_or_newline_slurp*⟩ SEMICOLON ⟨*white_or_newline_slurp*⟩

⟨*complete_eqnset_line*⟩ ::= ⟨*white_or_newline_slurp*⟩ ⟨*eqnsetlinecontents*⟩ ⟨*white_or_newline_slurp*⟩ SEMICOLON

⟨*eqnsetcontents*⟩ ::= ⟨*white_or_newline_slurp*⟩
| ⟨*complete_eqnset_line*⟩
| ⟨*eqnsetcontents*⟩ ⟨*complete_eqnset_line*⟩

⟨*eqnsetlinecontents*⟩ ::= IO_LINE
| ONEVENT_SYMBOL ⟨*event_def*⟩
| ⟨*import*⟩
| ⟨*function_def*⟩
| ⟨*assignment*⟩
| ⟨*time_derivative*⟩

⟨*open_library*⟩ ::= ⟨*empty*⟩

⟨*library_def*⟩ ::= ⟨*library_def_internal*⟩

⟨*library_def_internal*⟩ ::= LIBRARY ⟨*open_library*⟩ WHITESPACE ⟨*namespace*⟩ LCURLYBRACKET ⟨*librarycontents*⟩ ⟨*white_or_newline_slurp*⟩ RCURLYBRACKET ⟨*white_or_newline_slurp*⟩ SEMICOLON ⟨*white_or_newline_slurp*⟩

⟨*complete_library_line*⟩ ::= ⟨*white_or_newline_slurp*⟩ ⟨*librarylinecontents*⟩ ⟨*white_or_newline_slurp*⟩ SEMICOLON

⟨*librarycontents*⟩ ::= ⟨*white_or_newline_slurp*⟩
| ⟨*complete_library_line*⟩
| ⟨*librarycontents*⟩ ⟨*complete_library_line*⟩

⟨*librarylinecontents*⟩ ::= ⟨*import*⟩
| ⟨*function_def*⟩
| ⟨*assignment*⟩

⟨*open_event_def_scope*⟩ ::= ⟨*empty*⟩

⟨*event_def*⟩ ::= ⟨*alphanumtoken*⟩ LBRACKET ⟨*function_def_params*⟩ RBRACKET ⟨*white_or_newline_slurp*⟩ LCURLYBRACKET ⟨*open_event_def_scope*⟩ ⟨*on_event_actions_blk*⟩ RCURLYBRACKET

⟨*on_event_actions_blk*⟩ ::= ⟨*white_or_newline_slurp*⟩ ⟨*on_event_actions*⟩

⟨*on_event_actions*⟩ ::= ⟨*empty*⟩
| ⟨*on_event_action*⟩
| ⟨*on_event_actions*⟩ ⟨*on_event_action*⟩

⟨*on_event_action*⟩ ::= ⟨*empty NEWLINE*⟩
| ⟨*alphanumtoken*⟩ EQUALS ⟨*rhs_term*⟩ ⟨*whiteslurp*⟩ SEMICOLON

Grammar B.3 – The BNF grammar for `neurounits` (Part 3: Eqnsets & Library definitions)

⟨*import*⟩ ::= FROM WHITESPACE ⟨*namespace*⟩ WHITESPACE IMPORT WHITESPACE ⟨*import_target_list*⟩
| ⟨*FROM WHITESPACE <namespace*⟩ WHITESPACE IMPORT WHITESPACE ⟨*localsymbol*⟩ WHITESPACE AS WHITESPACE ⟨*localsymbol*⟩

⟨*import_target_list*⟩ ::= ⟨*localsymbol*⟩ ⟨*whiteslurp*⟩
| ⟨*import_target_list*⟩ COMMA ⟨*whiteslurp*⟩ ⟨*localsymbol*⟩ ⟨*whiteslurp*⟩

⟨*namespace*⟩ ::= ⟨*alphanumtoken*⟩
| ⟨*namespace*⟩ DOT ⟨*alphanumtoken*⟩

⟨*localsymbol*⟩ ::= ⟨*alphanumtoken*⟩

⟨*externalsymbol*⟩ ::= ⟨*namespace*⟩ DOT ⟨*localsymbol*⟩

Grammar B.4 – The BNF grammar for `neurounits` (Part 4: Imports & Namespaces)

B.1.3 *Phase 3: Unit-group symbol resolution*

The next step is to resolve unit-terms strings, such as `'millivolt'` and `'pA'`. In the previous phase, the string has been decomposed into tokens, including `ALPHATOKEN`s (tokens only containing ASCII letters), which might include for example `V` and `F`. Phase-2 provides information to determine whether these tokens represent variable names, constants or units. In Phase-3, the `ALPHATOKEN`s that represent unit terms are resolved. In `neurounit`, this is done with another BNF grammar (not given) and some explicit code to resolve corner-cases. Both short and long forms of values are supported, (e. g. `mV` and `millivolt`). The following supported prefixes and identifiers are given in Table B.2. A unit-term can be either a long or short form of a dimension, a long prefix and a long dimension or a short prefix and a short dimension.

Table B.2 – Basic units available in `neurounit`

LONG-FORM	SHORT-FORM
meter	m
gram	g
second	s
ampere	A
kelvin	K
mole	mol
candela	cd

LONG-FORM	SHORT-FORM
volt	V
siemen	S
farad	F
ohm	Ohm
coulomb	C
hertz	Hz
watt	W
joule	J
newton	N
liter	L
molar	M
watt	W

B.1.4 *Phase-4: Inference of dimensions using ASTs*

Phase-3 provides a set of syntax trees, which define the equations in terms of the symbols. In the Phase-4, dimensional inference occurs, in which `neurounit` inspects the trees and automatically infers the units for symbols which have not had their

Table B.3 – Prefixes support by `neurounit`

LONG-FORM	SHORT-FORM
giga	g
mega	m
kilo	k
centi	c
milli	m
micro	u
nano	n
pico	p

units specified explicitly. For example, in Listing B.1, because we have specified that v is in millivolts, and `g` is in Siemens, `neurounit` will infer that the dimensions of `r` must be Amperes and `g2` must be Siemens. At this stage, `neurounit` does not worry about scaling factors (powers of ten) between quantities, only the base dimensions. Later, when the syntax trees are used for other tasks, for example to generate MODL the relevant scaling factors will be applied accordingly.

In `neurounit`, dimensions are inferred by applying a series of heuristics to the nodes in the AST. For example, at an AST node representing addition, if one of the arguments or the result has a particular dimension, for example Siemens, then all both the arguments and result must also have that dimension, since it does not make sense to add a value in volts to a value in Siemens for example. Different nodes have different heuristics, and the heuristics are also applied to functions and their parameters. In this way, `neurounit` is able to propagate units around the AST trees.

In some cases, it will not be possible to infer all the dimensions. In Listing B.2 for example, there is no way to infer the dimensions of `x` and `y` without further information. The dimensionality of a symbol can be provided with a statement such as the `<=> PARAMETER` statement in Listing B.1. If `neurounit` unit is unable to infer the dimensions of a symbol at the end of Phase-4, an exception is raised.

```
eqnset e1 {
   v = 1 mV
   i = v*(g+g2)
   <=> PARAMETER g : S
   <=> PARAMETER g2
}
```

Listing B.1 – An example equationset in which neurounit is able to infer the dimensions of all the symbols (v, g & i)

```
eqnset e2 {
   v = 1 mV
   x = v * y
}
```

Listing B.2 – An example equationset in which neurounit is unable to infer the dimensions of the symbols x & y

B.2 ISSUES ARISING IN PARSING EXPRESSIONS INVOLVING UNITS

Whilst defining this grammar, several salient issues were raised, which are briefly discussed here.

OPERATOR NOTATION & DIVISION Conventionally, no multiplication sign is written between a magnitude and its dimension, for example, we write $4\,\mathrm{ms}$ rather than $4 \times \mathrm{m} \times \mathrm{s}$ [des Poids et Mesures, 2006]. Similarly, when quantities are written in `neurounit`, no multiplication sign is used, and there is no ambiguity. However, in the case of division of units, the case is more complex and existing tools use different notations. For example, defining the gas constant, `R` = 8.3144 $\frac{\mathrm{J\,mol}}{\mathrm{K}}$. A human, and MODL, would interpret `8.314 J/K mol` as `8.314 J/(K * mol)`, but a `C`-style language[1] as `8.314 (J/K)*mol` because multiplication and division have equal precedence [Kernighan and Ritchie, 1988]. The situation is more ambiguous with multiple division operators, for example `'1m/s/s'`. One approach, as taken by `GNU` units is to introduce a high-priority division operator `|`. `Neurounit` interprets such strings as a human or MODL would, by changing the precedence of space (multiplication) and divide and explicitly disallowing more than one division sign in a units definition, unless parentheses are used. This change in precedence only applies to unit-terms, and C-type precedence is

1 Assuming that space is interpreted as multiplication

used in calculation between quantities, so `{4 m} / {2 s} / {2 s}` is valid and results in $1\,\mathrm{m\,s^{-2}}$.

LIBRARIES & CONSTANTS: `library` allow constant values and functions to be reused between `eqnsets`. A `library` can only contain functions and constants, although one constant can be defined in terms of other constants. It is possible to `import` functions and constants from a `library`, using Python style syntax, for example `from math import pi`.

FUNCTION CALLS It is useful to be able to define functions to avoid duplicating code. `Neurounit` supports the definition of functions within `eqnset` or `library` blocks. Parameters only return a single value, which is denoted by the equals sign in their definition, for example: `area_rect(x,y) = x*y` defines a function called `area_rect` which takes two parameters, `x` & `y` and returns their product. Functions take Python-style named parameters and in function calls, all parameters must be explicitly specified by name, unless the function only takes a single parameter. Functions are defined in `eqnset` or `library` blocks. Briefly, function calls do not have *side-effects*, because functions can only *see* their arguments, other functions and *constants* in the block they are defined in. It is possible to nest function calls, for example `area_square(x)=area_rect(x=x,y=x)`, which would return the square of `x`, but recursive functions are not allowed. To help with unit inference, it is possible to define the expected dimensions of parameters given to functions.

`Neurounit` is a prototype library, and several issues remain to be solved. One issue is that the error reporting is limited when an invalid `eqnset` or `library` is given. Specifically, it can be difficult to diagnose the exact location of either a syntactic mistake, or a mistake due to a units mismatch. The `neurounit` library needs better diagnostics, and isolation of where there is a mismatch in the units expected for a particular symbol, as well as a history of how they have been inferred. Another issue surrounds the flexibility of functions and units. In the current prototype, it is slightly convoluted to write the equation for the electrotonic length of a neuron. This can be calculated from $\lambda = \sqrt{r_m/r_a}$, the dimensions of $r_m/r_a$is in m^2 [Koch, 1999]. Currently, `neurounit` is unable to propagate the dimensions correctly through the square root

function. The `sqrt` function expects a dimensionless arguments and return value, so this expression has to be written as `lambda = sqrt((rm-ra)/{1m2})*{1m}`. This is a symptom of a more general problem with the current implementation, that it is not possible to define 'template' functions, as can be done in C++ (or generics in Java). Fortunately, this does not limit what can be done with the syntax, and only make a few edge case expressions more verbose since it is always possible to explicitly *factor* the dimensions in and out of the arguments and the return value, by multiplying by the appropriate unit constants, as was done in the this case.

C

NEUROUNIT EXAMPLES

C.1 LEVEL-1

```
m == meter
s == second
millisecond == ms
millisecond-1 == ms-1
amp == coulomb/s
m/s2 == m/s s == m/(s s)
volt == amp ohm == A Ohm == W / A == J / A s == N m / A s == kg m2 / C s2 == N m / C == J /C == kg ↙
     ↳ m2/A s3
m / s2 == (m/s)/s == m/(s s)
1.0m == 1meter
100.cm == 1m
101cm != 1m
1 mM == 1 mol/m3
1uM == 1e-3 mol/m3
3 millivolt == 3 kiloamp microohm == 3.0 mA Ohm == 3 W / kA == 3 pJ / A ns == 3e-3 N m / A s == ↙
     ↳ 0.003 kg m2 / C s2 == 0.003 N m / C == 3 J / kilocoulomb == 3 g m2/A s3
3 millivolt != 1 kiloamp microohm
3 millivolt != 3 kiloamp ohm
```

Listing C.1 – Examples of valid neurounit strings for LEVEL-1. The '==' and '!=' are not part of neurounit LEVEL-1 but denote that two expressions are equivalent.

C.2 LEVEL-2

```
{1.0m} / {2s} == {0.5 m/s} == {50 cm/s} == {0.5 mm/ms}
3+4/2==5
3+6/2+1==7
5 m/s2 == 5 m/s s
6 s/m == 6 ms/mm
1/{1s} == 1Hz
1000L == 1m3
({1m/s} - {5m s-1}) / {8ms} == -500. m/s2 == -0.5 km s-1 s-1
{500mN} == {1 kg} * {1 m/s2} / 2.
1 == 1 / 2 * 2
0.125 == 1 / 2 / 4
std.physics.R == 8.314472 J mol-1 K-1
```

Listing C.2 – The '==' and '!=' are not part of neurounit LEVEL-2 but denote that two expressions are equivalent.

'

C.3 LEVEL-3

```
eqnset chlstd_leak {
   i = g * (V-erev)

   <=> PARAMETER g, erev
   <=> OUTPUT i:(mA/cm2) METADATA {"mf":{"role":"TRANSMEMBRANECURRENT"} }
   <=> INPUT V: mV METADATA {"mf":{"role":"MEMBRANEVOLTAGE"} }
}
```

Listing C.3 – An example of neurounit LEVEL-3, used to define a leak channel

```
library my_neuro {
   from std.math import pow
   RateConstant5(V:{V},a1:{s-1} ,a2:{V-1 s-1}, a3:{},a4:{V},a5:{V} ) = (a1 + ↙
        ↳ a2*V)/(a3+std.math.exp( (V+a4)/a5) )
}

eqnset chlstd_hh_na {

   from my_neuro import RateConstant5
   from std.math import exp
   i = g * (v-erev) * m**3*h

   minf = m_alpha_rate / (m_alpha_rate + m_beta_rate)
   mtau = 1.0 / (m_alpha_rate + m_beta_rate)
   m' = (minf-m) / mtau

   hinf = h_alpha_rate / (h_alpha_rate + h_beta_rate)
   htau = 1.0 / (h_alpha_rate + h_beta_rate)
   h' = (hinf-h) / htau

   m_alpha_rate = RateConstant5( V=v,a1=m_a1,a2=m_a2,a3=m_a3,a4=m_a4,a5=m_a5)
   m_beta_rate = RateConstant5( V=v,a1=m_b1,a2=m_b2,a3=m_b3,a4=m_b4,a5=m_b5)
   h_alpha_rate = RateConstant5( V=v,a1=h_a1,a2=h_a2,a3=h_a3,a4=h_a4,a5=h_a5)
   h_beta_rate = RateConstant5( V=v,a1=h_b1,a2=h_b2,a3=h_b3,a4=h_b4,a5=h_b5)

   <=> PARAMETER g, erev
   <=> PARAMETER m_a1, m_a2, m_a3, m_a4, m_a5
   <=> PARAMETER m_b1, m_b2, m_b3, m_b4, m_b5
   <=> PARAMETER h_a1, h_a2, h_a3, h_a4, h_a5
   <=> PARAMETER h_b1, h_b2, h_b3, h_b4, h_b5
   <=> OUTPUT i:(mA/cm2) METADATA {"mf":{"role":"TRANSMEMBRANECURRENT"} }
   <=> INPUT v: mV METADATA {"mf":{"role":"MEMBRANEVOLTAGE"} }
}
```

Listing C.4 – An example of neurounit LEVEL-3, used to define an HH-type voltage-gated sodium channel. A simple library is defined, and the RateConstant5 function imported from it in the channel definition

```
eqnset cachl {
      from std.math import exp
      from std.physics import F,R

      n_alpha = ( {10mV} - V ) / ( 100* exp( ({10mV}-V)/{10mV} ) - {1} ) * (1 ms-1 V-1)
      n_beta = 0.128 * exp( V / {-80mV}) * (1 ms-1)
      n_tau = 1/(n_alpha+n_beta)
      n_inf = n_alpha/(n_alpha+n_beta)
      n'= (n_inf - n) / n_tau

      gca = pca* 2 * up * F * (CAi - CAo*exp(-1.0*up) ) / (1-exp(-1.0*up) )
      ica = gca * n**2
      up = 2 * V * F / (R*T)

      T = 300 K

      <=> INPUT V: mV METADATA {"mf":{"role":"MEMBRANEVOLTAGE"} }
   <=> OUTPUT ica:(mA/cm2) METADATA {"mf":{"role":"TRANSMEMBRANECURRENT"} }

      <=> PARAMETER pca
      <=> PARAMETER CAi:(mole /m3)
      <=> PARAMETER CAo
 }
```

Listing C.5 – An example of neurounit LEVEL-3, used to define a voltage-gated GHK-type calcium channel channel

```
eqnset syn_simple {
   g' = - g/g_tau
   i = gmax * (v-erev) * g

   gmax = 300pS
   erev = 0mV

   g_tau = 20ms
   <=> INPUT v: mV METADATA {"mf":{"role":"MEMBRANEVOLTAGE"} }
   <=> OUTPUT i:(mA) METADATA {"mf":{"role":"TRANSMEMBRANECURRENT"} }

   ==>> on_event() {
      g = g + 1.0
   }
}
```

Listing C.6 – An example of neurounit LEVEL-3, used to define a simple synapse model

D

MORPHFORGE IMPLEMENTATION

D.1 SOFTWARE ARCHITECTURE

D.1.1 *simulation layer*

Using factories (i) - simulator specific object construction

A simulation in morphforge is entirely encapsulated within a `Simulation` object, which is populated with *primitive objects* such as `Cells`, `Channels`, `Synapses` and then executed by calling `Simulation.run()` (e. g. Listing 3.5). In morphforge, in contrast to the PyNEURON interface for example, all interaction with the simulator-backend is deferred until the `run()` method is called. That is, all changes to the `Simulation` object before this are only modifying morphforge's internal object-model. This approach offers several advantages over interacting with the simulator-backends immediately; (i) a set of simulations can be built in a single thread and their executions can be easily distributed over multiple processors, for example using Python's multiprocessing module[1]; (ii) it is simple to cache the results of a simulation run[2]; (iii) this approach allows us to robustly perform multiple NEURON simulations in a single script (iv) it

1 Although interaction with the simulator directly does not preclude parallelisation, it means that aspects of the simulation will be more difficult to query in the original thread. To parallelise a simulation interacting with the backend directly, the code to build the simulation will likely need to be run in the separate process, and for example, if the number of synapses in the model are made randomly, it will require more effort to return connection information back to the parent process.

2 For example, provided all the information for the simulation is contained in the `Simuation` object, computing and comparing a hash-function of the serialised the `Simulation` objects can provide an efficient way to check whether two simulations are identical.

offers the possibility to optimise aspects of the simulation, for example the possibility of sharing of postsynaptic receptors (discussed in Section 3.6.6).

There are different approaches to creating such a `Simulation` object-model that would support different simulator backends, for example NEURON, GENESIS, MOOSE). One possibility would be to create a `Simulation` object and populate it with backend-independent primitives (e. g. `Cell` & `Synapse` objects). When `Simulation.run()` was called, these objects would then be traversed using a backend-specific mapping tool to convert them into appropriate code to run the on the specific backend.

The approach taken by morphforge is to specify the simulator-backend before constructing the `Simulation` object. Then, instead of populating the object-model with *backend-independent primitives* (e. g. `Cell` & `Synapse` objects), it is populated with *backend-specific primitives* (e. g. `NEURONCell` & `NEURONSynapse` objects). These backend-specific primitives are subclasses of the backend-independent primitives (for example, `NEURONSynapse` inherits from `Synapse`), which allows the simulator-backends to add their specialist methods and variables to the primitives, which greatly simplifies the internal code needed when `Simulation.run()` is called (Fig. D.1). Since we are using a dynamically typed language, it is not strictly necessary to use inheritance, but doing so ensures a consistent interface and separates conceptual and implementation details.

How is the object-model populated with these backend-specific primitives such that is easy to switch simulator-backends? In morphforge, all instantiation of primitives occurs through an `Environment` object. The `Environment` is a factory object which produces backend-specific objects, e. g., the `NEURONEnvironment` produces NEURON-specific primitives [Gamma et al., 1994]. For example, in Listing 3.5, all the objects have all been created, possibly indirectly[3], through calls to the `NEURONEnvironment`, which was defined on line 4, which means the resulting object-model is populated by objects such as: `NEURONSimulation`, `NEURONCell`, `NEURONStdChlLeak` & `NEURONCurrentClamp`.

3 In some cases, instantiation methods are provided via the `Simulation` class, for example, `sim.create_cell` on but internally these methods construct objects using the associated `Environment` object.

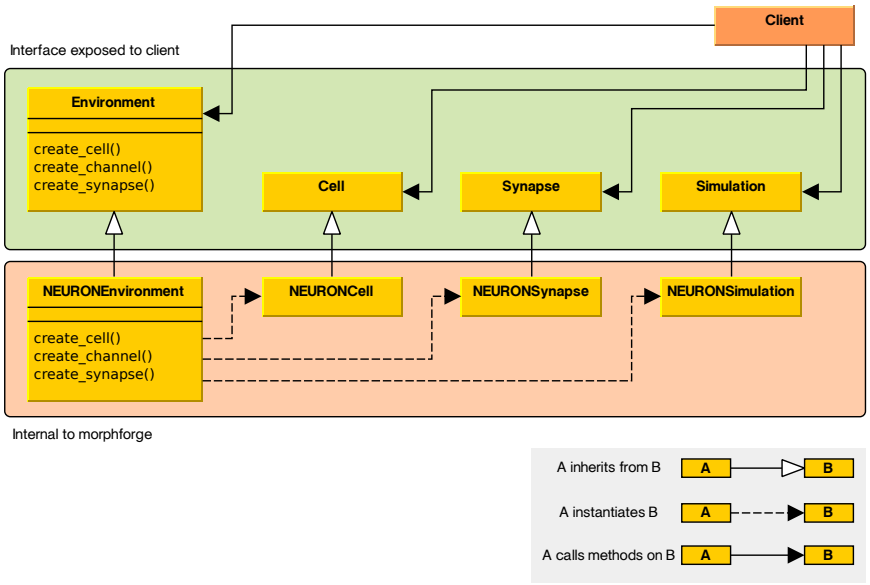


Figure D.1 – Using indirection to create simulation primitives using the `Environment` object. The client first instantiates a subclass of `Environment`, for example `NEURONEnvironment`. By constructing primitives through this object, an internal data structure of backend-specific objects (e. g. `NEURONCell`, `NEURONSynapse` and `NEURONSimulation`) can be created, but the interface only exposes *backend-independent* classes (e. g. `Cell`, `Synapse` and `Simulation`).

Using factories (ii) - channel creation

Several specialist file formats already exist for defining membrane channel and synaptic dynamics and parameters in simulators, for example MODL, NeuroML, NineML, or `neurounit`, and we would also like to be able to define channels and synapses in code directly. Rather than choose a single format for specifying primitive dynamics, morphforge uses Python's dynamic typing to support a flexible model for membrane channels and synapses. We begin by discussing channel models.

Morphforge is agnostic about the underlying format of channel models. We assume that an abstract channel can have a series of parameters, that can change in different areas of the membrane, and there is a set of default values for these parameters. To integrate with the morphforge framework, `Channel` objects are expected to provide a particular interface, some methods of which are general and some of which are simulator-backend specific. All `Channel` objects must implement the meth-

ods `get_variables()` & `get_defaults()` which return a list of parameter names for that `Channel`, (for example: `['g_bar','erev']`) and their default values respectively. These are used by the channel-distribution infrastructure in morphforge when calculating the parameter values which should be applied to each compartment of a `Cell` (this is described in Section D.1.1). Additionally, when the NEURON-backend is used, `Channel` objects must also implement the methods `create_modfile()` and `build_hoc_section()`, which build the MODL code and insert the relevant code into the HOC file.

As with the simulation primitives above, this is achieved by inheritance. For example, `NeuroMLChl` represents NeuroML channel loaded from an XML file, and is subclassed to produce the NEURON- specific `NEURONNeuroMLChl`. `NeuroMLChl`, implements the methods `get_variables()` and `get_defaults()`, which return the names of parameters that can be varied over the neuron, (for example `g_bar`) and their default values which are found by parsing the XML file. `NEURONNeuroMLChl` implements the methods `create_modfile()`, which returns a string of MODL code for the NeuroML file (for example generated using an XSLT) and `build_hoc_section()`, which returns the relevant HOC statements for inserting the `Channel` into a `Section` with a particular set of parameter values.

As with the simulation primitives, `Channel` objects are constructed using indirection through the `Environment` object, which allows morphforge to produce the appropriate backend-specific objects (Listing D.1). This architecture allows a modeller to use any supported formats for a particular simulator-backend. `Channel` types can be mixed and matched within a single `Simulation`, providing a path for incrementally translating their model into newer formats, for example NineML, NeuroML or `neurounit`, while still allowing the use of simulator-specific features.

```
env = NEURONEnvironment()
sim = env.Simulation()
# Create a series of channels of different types:
chl1 = env.Channel( NeuroMLChl, file='my_neuroml_channel.xml')
chl2 = env.Channel( NeuroUnitChl, file='my_neurounit_channel.eqn')
chl3 = env.Channel( NEURONMODChl, file='my_channel.mod')
chl4 = env.Channel( StdChlLeak, reversal_potential=qty('-50mV'), conductance=('0.03mS/cm2') )

# All channels are applied to the neuron in same way:
cell = sim.create_cell(...)
cell.apply_channel(chl1)
cell.apply_channel(chl2)
cell.apply_channel(chl3)
cell.apply_channel(chl4)
```

Listing D.1 – Constructing `Channel` objects via the `Environment`. By using the environment, NEURON-specific channel objects are created: `NEURONNeuroMLChl` (chl1); `NEURONNeuroUnitChl` (chl2); `NEURONMODChl` (chl3) & `NEURONStdChlLeak` (chl4).

Using flyweights to template postsynaptic receptors

In morphforge, as in NEURON [Carnevale and Hines, 2006], a `Synapse` is built from a pair of `Presynaptic` and `Postsynaptic` objects (Fig. D.2). `Presynaptic` objects generate *events*, for example from a list of times or in response to the voltage of a presynaptic neuron crossing a threshold (Fig. D.2). `PostSynaptic` objects represent synaptic receptors in the postsynaptic neuron, which open and close with particular dynamics and change the membrane conductance. `PostSynaptic` consume *events* to produce discrete changes in the receptor state.

Using this scheme allows the presynaptic and postsynaptic components of the synapse to be uncoupled; a particular postsynaptic receptor can be modelled and it is simple to change to the source of spike timing. Another advantage of this scheme is that if the model of the postsynaptic receptors are linearly superposable, then it is possible to replace multiple, identical postsynaptic receptors on a single neuron with a single instance which is driven by multiple event sources (Fig. D.3), which can dramatically reduce the number of equations that need to be solved by the simulator.

Different forms of equations and description languages have already been used to describe postsynaptic receptor dynamics. As with the definition of `Channel` objects (Section D.1.1), morphforge does not require a particular format, but expects that any `PostSynaptic` objects conform to a particular, backend-specific, interface. As

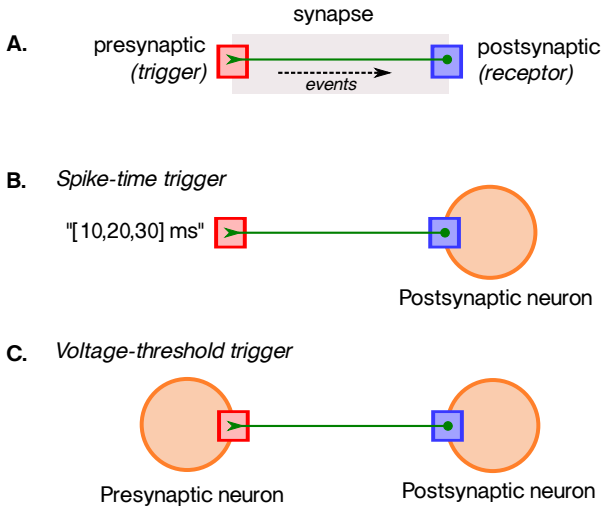


Figure D.2 – Synapses in morphforge. **A.** Synapses are composed of a presynaptic *trigger* and postsynaptic *receptor*. The trigger produces *events*, which cause discrete changes in the state of the receptor. **B.** The trigger can be either a set of event times or **C.** an object that monitors the voltage of a presynaptic neuron and produces an event each time a threshold is crossed.

with the `Channel` objects, `PostSynaptic` object are constructed indirectly, using the `Environment` factory object.

Although morphforge is not designed with efficiency as the first priority, efficiency must be considered with `Synapse` objects. Even in networks with a relatively small number of neurons, the number of synapses between them can quickly become very large. A recent model of the tadpole spinal cord has only 840 neurons, but approximately 180,000 synaptic connections [Borisyuk et al., 2008]. It is important to be able to specify large numbers of synapses, with variation in the parameters, which can be translated to efficient simulator-specific code.

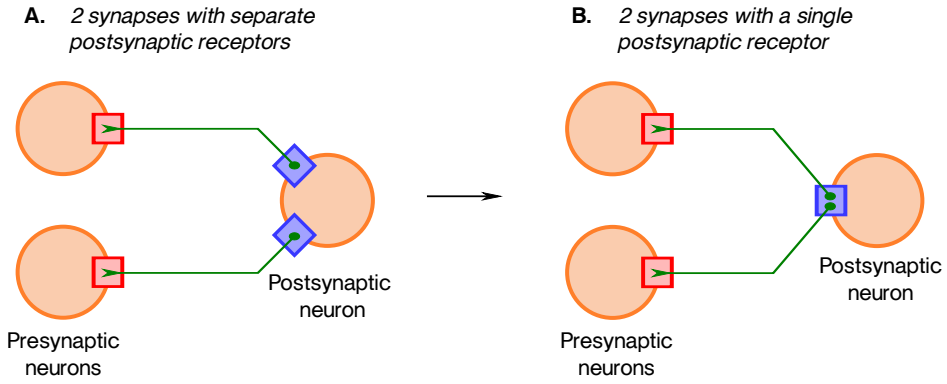


Figure D.3 – Merging postsynaptic receptors. **A.** A single postsynaptic neuron has synapses from two presynaptic neurons. The postsynaptic receptors (blue) are distinct objects. **B.** If the postsynaptic objects are linearly superposable, then we can replace the two postsynaptic objects with a single postsynaptic object, which is driven with events from the two presynaptic triggers.

```
synapse_def = """
eqnset syn_alpha {
   g' = -g/{2ms}
   g2' = -g2/{10ms}
   i = (g-g2) * (v-{0mV})
   gInc = 5pS
   <=> INPUT v: mV METADATA {"mf":{"role":"MEMBRANEVOLTAGE"} }
      <=> OUTPUT i:(mA) METADATA {"mf":{"role":"TRANSMEMBRANECURRENT"} }

      ==>> on_event(){
            g = g + gInc
            g2 = g2 + gInc
      }
}
"""
cell1 = sim.create_cell(...)
cell2 = sim.create_cell(...)
cell3 = sim.create_cell(...)

syn_tmpl = env.PostSynapticMechanismTemplate(NeuroUnitsTemplate, synapse_def)
sim.create_synapse(
            trigger = VoltageThresholdTrigger(cell_location = cell1.soma, v=0*mV),
            post_synaptic = env.PostSynaptic(NeuroUnitSynapse, synapse_def, ↙
                  ↳ cell_location=cell2.soma )
)
sim.create_synapse(
            trigger=VoltageThresholdTrigger(cell_location = cell1.soma, v=0*mV),
            post_synaptic = env.PostSynaptic(NeuroUnitSynapse, synapse_def, ↙
                  ↳ cell_location=cell3.soma )
)
```

Listing D.2 – Motivation for postsynaptic templates. In this example, we create three neurons, and two synapses (cell1 → cell2, cell1 → cell3). If we use the NEURON backend, it is difficult to infer that only a single MODL file needs to be built, which can be used for both of the synapses.

Morphforge takes an explicit approach to constructing multiple synapses of the same type by using the flyweight-pattern [Gamma et al., 1994]. This involves another factory class, `PostSynapticTemplate`, which has a method, `instantiate()`, which returns a new `PostSynapticObject`. This object delegates its backend-specific methods (e. g. `create_modfile()`) back to the parent `PostSynapticTemplate` object, to allow efficient code-generation. Since the `instantiate()` method can take parameters, it is possible to incorporate variation into the parameters of individual synapses. An example of using postsynaptic templates is given in Listing D.3.

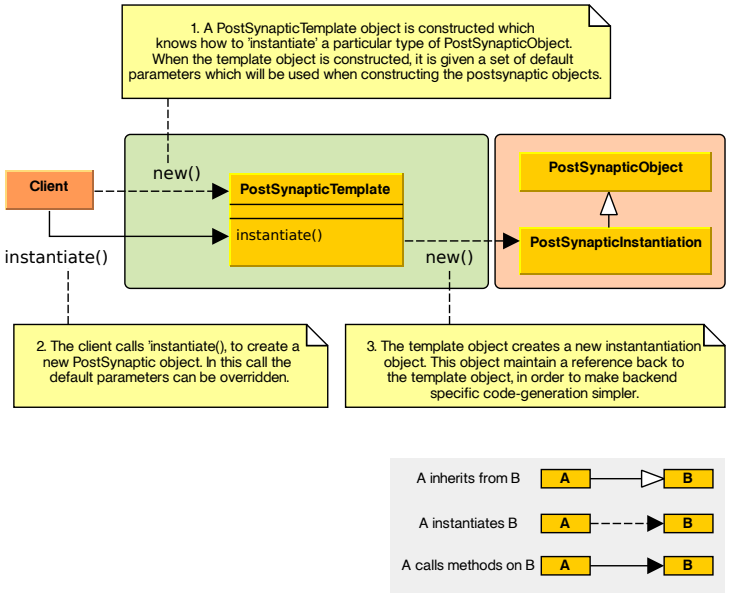


Figure D.4 – Overview of `PostSynapticTemplate` objects in morphforge.

Using strategy patterns to encapsulate concepts as objects

Abstractions are an effective tool for communicating ideas, allowing salient information to be conveyed whilst shielding the listener from less important details. In programming, suitable use of abstractions can make code much easier to understand and many techniques are used from naming constants to functions to make the conceptual aims of code more apparent [McConnell, 2004]. As our models grow in complexity,

```
# As before:
synapse_def = """
eqnset syn_alpha { ...
}
"""

syn_tmpl = env.PostSynapticMechanismTemplate(NeuroUnitsTemplate, synapse_def)
sim.create_synapse(
            trigger = [100*ms],
            post_synaptic = synaptic_tmpl.instantiate(cell_location = cell1.soma)
)
sim.create_synapse(
            trigger=VoltageThresholdTrigger(cell_location = cell1.soma, v=0*mV),
            post_synaptic=syn_tmpl.instantiate(
                              cell_location = cell2.soma,
                              parameter_multipliers={'g':2.0})
)
```

Listing D.3 – Example of defining synapses using a `PostSynapticTemplate` object.

we are likely to spend more time needing to reassimilate the intention of what we previously wrote, so it is important to be able to quickly understand the purpose of our own code. For example, in Listing D.4, although all three blocks of code achieve the same goal, calculating the diameter of sections as distance varies in order to produce a sphere, to the intention of the third version is more readily apparent than the first and second.

```
# Version 1
spine_sections = [
      (0., 0.00), (1., 3.00), (2., 4.00), (3., 4.58), (4., 4.90),
      (5., 5.00), (6., 4.90), (7., 4.58), (8., 4.00), (9., 3.00), (10., 0.00) ]

# Version 2
R = 5.
spine_sections = [(a, sqrt(R**2 - (x-R)**2)) for a in linspace(0.,10.,num=11)]

# Version 3
spine_sections = get_section_radii_for_spherical_spinehead(radius=5, nsections=11)
```

Listing D.4 – Making the intention of code explicit. The three examples given below may perform the same function, but it is much simpler to understand the authors intention from looking at Version 3 than Version 1.

There is a tradeoff in this indirection and defining interfaces which can encapsulate ideas yet still allow flexibility is difficult. In Listing D.4, if we wanted to change the shape of the sections to produce an oblate spheroid instead, then it is more immediately obvious what to change in Version 2 than Version 3. We must try to define modelling libraries that allow concepts to be defined at a suitably high level, so a simple statement like, 'remove all potassium channels' corresponds to a single line modification in the model code, but that these interfaces also offer sufficient flexibility and do not burden the modeller with learning a complex internal API.

One source of problems is that it has been difficult to separate orthogonal abstractions in code in mainstream languages. An example of an orthogonal abstraction is in sorting, where a general sorting algorithm such as quicksort is orthogonal to the comparison function between objects [Hoare, 1961; Press et al., 2007]. One long-standing difficulty in encapsulating data and algorithms - most languages allow data to be passed to functions, but often passing an algorithm to a function has involved either complex class hierarchies, difficult low-level syntax, or dangerous techniques. Modern languages such as Python makes it easy to build treat algorithms as objects, either by using callable objects or through duck typing. This technique is used extensively in morphforge to express concepts; two examples are given in the next sections: first, for defining the segmentation of `Cell`, and second for defining the distribution of `Channels` over a neuron.

(I) `cells` & SEGMENTATION A simple example of using a functor in morphforge is in the compartmentalisation of neurons. In morphforge, a `Cell` contains a `Morphology` object, which contains a tree of `Sections`, corresponding to cylinders. The `Morphology` and `Section` objects represent gross neuronal morphology. In electrical simulation these can be further subdivided, or *segmented*, into a number of smaller compartments for better spatial accuracy during simulation. Rather than specify the number of segments per `Section` directly, instead an object is passed as a parameter when a `Cell` object is constructed. This object implements the method `get_num_segments()`, which returns the number of segments for a given compartment. Listing D.5 shows 2 examples: the `CellSegmenter_MaxCompartmentLength` object on line 4 segments the morphology so that the longest compartment is 5 µm and

the `CellSegmenter_CompartmentLengthToDiameterRatio` object on line 6 makes compartments that are no longer than five times the length of the thinnest diameter of the `Section`. It is also simple to define custom classes, for example to implement the d-lambda rule [Carnevale and Hines, 2006] as illustrated in Listing D.6.

```
morphology = Morphology.load_from_swc('myfile.swc'),
cell = sim.create_cell(morphology = morphology,
      segmenter = CellSegmenter_MaxCompartmentLength(5)
      )

cell = sim.create_cell(morphology = morphology,
      segmenter = CellSegmenter_CompartmentLengthToDiameterRatio(5)
      )
```

Listing D.5 – Defining the segmentation of a morphology for simulation using objects defined in morphforge

```
class MyDLambdaSegmenter(AbstCellSegmenter):

   def __init__(self, d_lambda=0.1, frequency=100*units.Hz):
      self.d_lambda = d_lambda
      self.frequency = frequency

   def get_num_segments(self, section):
      # Query the model (not shown):
      Cm, Ra, = ...
      # Choose the thinnest radius from proximal and distal:
      d = min([section.p_r, section.d_r]) * 2.0 * units.um
      # Calculate d_lambda (see 'The NEURON book', pg. 122)
      lambda_100 = (1./2.) * math.sqrt( d / (pi*self.frequency*Ra*Cm))
      n_sections = int(ceil(sect.length / (lambda_100 * self.d_lambda)))
      # Ensure odd number of segments:
      if n_sections % 2 == 0:
         n_sections = n_sections + 1
      return n_sections

cell = sim.create_cell(morphology=morphology, segmenter=MyDLambdaSegmenter(d_lambda=0.1))
```

Listing D.6 – Defining the segmentation of a morphology for simulation using user-defined objects. Pseudo code for implementing the d-lambda rule is given

(II) CHANNELS & DISTRIBUTIONS In Section D.1.1, we discussed how a `Channel` object is created. We now discuss how it is applied to a neuron's membrane. In many neurons it is known that the distribution density of a particular channel type over

the membrane is not uniform. Often in models, we want to incorporate this, and specify that a `Channel` exists all over particular regions of the neuron, and use specific parameters in specific regions. For example, the conductance density of potassium channels might be $30\,\mathrm{mS/cm^2}$ all over on a model neuron's membrane, except in apical dendrites where it is $50\,\mathrm{mS/cm^2}$. Existing models have used even more complex channel distribution schemes, for example that the density of sodium channels on the initial segment of the axon should vary as the function of distance from the soma [Schmidt-Hieber et al., 2008].

Morphforge supports complex specifications of channel densities over neurons using a high-level notation. This is achieved by passing a triplet of objects to the `apply_channel` method of `Cell` objects: (`Channel`, `Applicator`,`Targeter`). The `Targeter` object defines which `Sections` in the `Cell` this triplet applies to (i. e. a predicate object). The `Applicator` object defines how the parameters of the `Channel` should vary over the specified `Sections`. Listing D.7 shows an example in which twice the density of potassium channels are applied in the "dendrites" as the rest of the `Cell`. In this example, we use two `Targeters`: `TargetEverywhere` and `TargetRegion`, and one `Applicator`: `ApplyUniform`. A `Channel` object has an associated set of default parameters (e. g. `gbar`, see Section D.1.1), which are used by default by `ApplyUniform` (e. g. Listing D.7 line 2), although they can be overridden or scaled (e. g. Listing D.7 line 3).

```
# Apply more potassium in the dendrites:
cell.apply_channel(k_chl, TargetEverywhere(), ApplyUniform() )
cell.apply_channel(k_chl, TargetRegion('dendrites'), ApplyUniform(multiply_parameters={'gbar':2.0}) )
```

Listing D.7 – Example of defining channel distributions in `morphforge`

The `apply_channel` method can be called many times for the same `Channel` on the same `Cell`, with different `Targeters` and `Applicators`. However, in the simulation, a particular `Channel` will only be applied once to any given `Section`. If multiple `Targeters` affect the same `Section`, a system is needed to resolve which parameter values to use. For example, in Listing D.7, which value of `gbar` should be applied to the dendrites — should it be the *default* (since the "dendrites" region will be targeted

by `TargetEverywhere`), or should twice the default (since the "dendrites" region will also be targeted by `TargetRegion`)?

To resolve these conflicts, each `Targeter` object has a *priority level* associated with it. For example `TargeterEverywhere` has a priority of 10, and `TargeterRegion` has a priority of 20. When `Simulation.run()` is called, for every `Channel` applied to every `Section`, morphforge finds the corresponding targeter with the highest priority. The algorithm described in Algorithm 1 in Appendix D.2.1. Therefore in Listing D.7, the dendrites will have twice the value of `g_bar` for `k_chl` in the dendrites than the rest of the neuron.

The mechanism also allows us to define non-uniform distributions of channels over a particular region. For example, we might want to distribute a type of sodium channel along the initial part of the axon, such that the channel density is specified as a function of distance from the soma. This is achieved by creating a new `Applicator` class, and implementing the method `get_variable_value_for_section()`. An example of this is shown in pseudocode in Listing D.8.

```
class CustomApplyDensityAsAFunctionOfDistance(ChannelApplicator):
   def get_variable_value_for_section(self, variable_name, section):
      if variable_name=='g_bar':
         # return a conductance density for 'section', that is some function of its
         # distance to the soma.
      else:
         # return a constant, default value for the other parameters (e.g. reversal potential)

# Apply more sodium non-uniformly in the axon:
cell.apply_channel(na_chl, TargetEverywhere(), ApplyUniform() )
cell.apply_channel(na_chl, TargetRegion('axon'), CustomApplyDensityAsAFunctionOfDistance() )
```

Listing D.8 – Example of defining a custom `ChannelApplicator` for defining a particular channel distribution on a neuron

(III) COMPLEXITY AND CLARITY In the previous sections, we have complicated morphforge by adding more indirection and classes into the framework. What are the benefits of this architecture? For one, once we understand the triplet (`Channel`, `Targeter`, `Applicator`) system, then the intention of Listings D.7 & D.8 becomes immediately clear. We do not need to decipher a complex set of `for`-loops and `if` statements, nor trust in comments in the code in order understand the intention, and we

have reduced the code needed to define a complex concept to a single line of code. In order to ask simple scientific questions, such as, what happens if the distribution of channels is simplified to be uniform, we are making changes at a single well-defined point in the code.

What are the advantages of this system, over simply defining allowing user-defined function that calculates the distribution of channels for each `Section` explicitly? Firstly, orthogonal issues, (such as iterating over the tree and calculating the conductances for each `Section`) are partitioned from each other. Since these have been decoupled, and there is no need to write any code for the iteration over `Section`s in Listing D.7, we reduce the likelihood of introducing bugs into our model. Secondly, we retain this abstraction within our object model. We may want to produce a summary output of the simulation as a pdf document for example. In this case, it will be much more useful to summarise the distribution of channels using the (`Channel`, `Targeter`, `Applicator`) notation, rather than as list of every parameter for every `Channel` for every compartment.

Using interfaces to keeping scientific questions centre-stage

As was touched upon in the previous section, we will need to read parts of our model many times, so it is important this is as easy as possible. Using interfaces allows us ignore implementation details, and focus on scientific questions. Simple and consistent interfaces are easier to understand and allow more mental resources to be dedicated to the conceptual problems at hand. Interfaces are widely used in morphforge where they are used to hide implementation details. I discuss how they are used to solve two particular problems, the first is in hiding the complexity of recording values from a simulation, and the second is a class to represent analogue signals with units.

(I) RECORDING VALUES DURING SIMULATIONS We need to record various values during a simulation, for example voltages and currents. The morphforge object-model agnostic to the underlying formats of particular synapse and channel formats and supports the recording of any values from any `Channel`, `Synapse` or other objects through a consistent interface using the method `Simulation.record()`, as shown in

Listing D.9. Internally, `record(obj,...)` forwards calls to `obj.get_recordable(...)`, which provides a flexible, yet consistent interface to recording what could be specific data from simulation objects (for example the voltage-dependence term of an NMDA synapse).

```
env = NEURONEnvironment()
sim = env.Simulation()

# Create a passive cell:
cell = CellLibrary.create_cell(sim, StandardModels.SingleCompartmentPassive, area=qty('1000:um2'), ↙
    ↳ input_resistance=qty('300:MOhm'))

# Add active channels:
chl = ChannelLibrary.get_channel( modelsrc = StandardModels.HH52, channeltype="Na", ↙
    ↳ env=sim.environment)
cell.apply_channel(chl)

cc = current_clamp = sim.create_currentclamp(amp=qty('100:pA'), dur=qty('100:ms'), ↙
    ↳ delay=qty('100:ms'), cell_location = cell.soma)
#syn = sim.create_synapse()

sim.record(cell, what='Voltage', name='V', cell_location=cell.soma)
sim.record(chl, what='StateVariable', state='m', cell_location=cell.soma)
sim.record(cc, what='Current')
#sim.record(syn, what='Current', user_tags=['mytag'])
```

Listing D.9 – Example of recording from different objects in a simulation

A set of *standard* strings, such as `Voltage`, `Current` and `StateVariable` are defined in morphforge, which are used in the method calls on lines [14-16] in order to specify what to record. The use of strings allows loose coupling across this interface and allows arbitrary variables to be recorded. `sim.record()` can take a variety of parameters, depending on the particular object being recorded from.

In order to provide a clean architecture behind the scenes, another object called `Recordable` is introduced (Fig. D.5). Following a call to `record(obj,...)`, the method `get_recordable()` of `obj` is called which returns a `Recordable` object. This contains the relevant machinery to record a particular value from `obj` using a particular simulator-backend. The `get_recordable()` method must internally determine which type of `Recordable` object to return, depending on the parameters passed. As a more concrete example, if the NEURON simulator backend is being used and if `get_recordable()` is called on a NeuroML channel to record the conductance density at a specific neuron location, then a `Recordable` object is returned that knows the name of the conductance variable in the generated MODL file (see Section D.1.1), and is able to insert

suitable statements into the HOC script to record and return this conductance after the simulation.

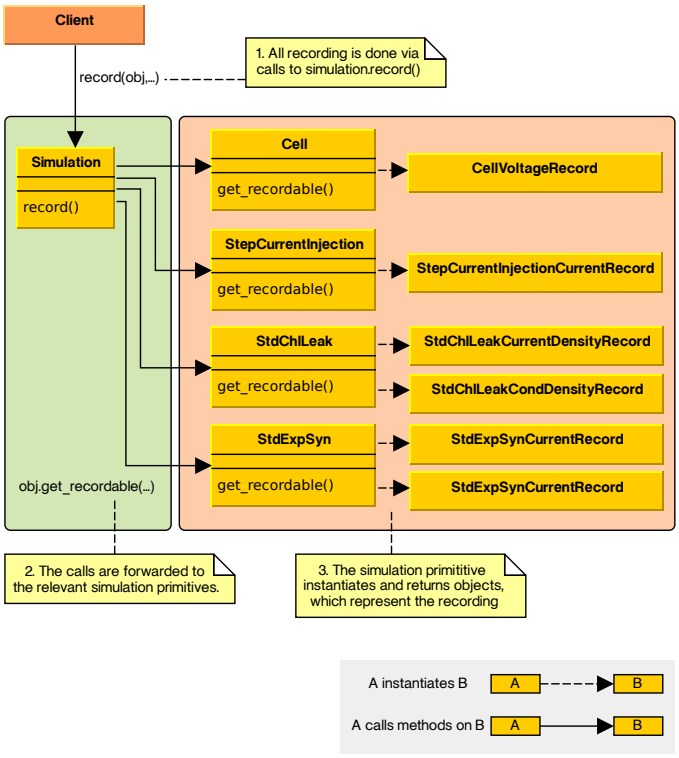


Figure D.5 – A simplified view of the architecture for recording in morphforge. All recording is done via calls to `Simulation.record()`, which forwards the calls onto the appropriate simulation primitives. The primitive objects return `Record` objects, which internally define what should be recorded, and are used by the simulator-backend when `Simulation.run()`is called. At the end of the simulation, the `SimulationResults` object is populated with `Trace` objects corresponding `Record` objects, (i. e. one for each call to `record()`).

A `Recordable` object can have a name (e. g. Listing D.9 line 14), which can be used to access the corresponding `Trace` object after the simulation has been run (see next section). A `Recordable` object can also take a set of additional user-defined tags, which will be attached to the `Trace` object (see Section D.1.2). The units of the recordings are automatically handled, for example, when the user requests the `Trace` object corresponding to the `record()` statement on line 14 of Listing D.9, it will automatically have the units of *millivolts*.

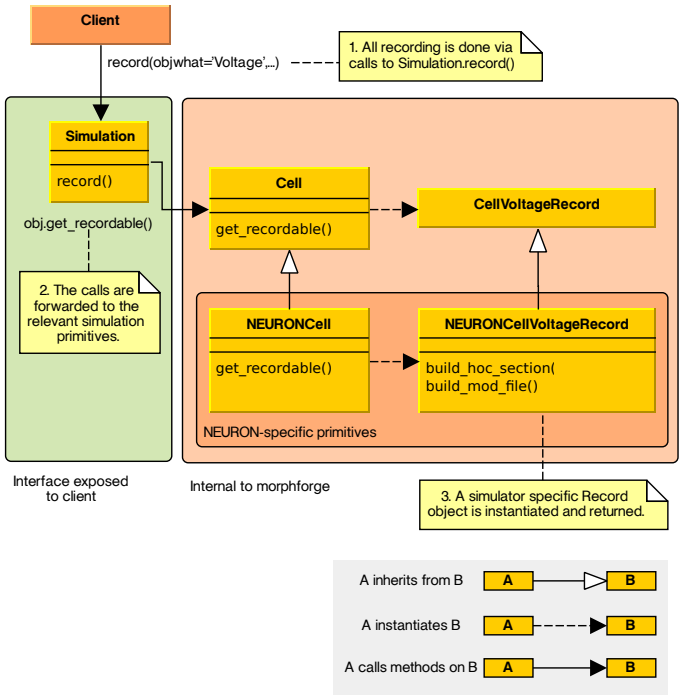


Figure D.6 – Construction of simulator dependant `Record` objects in morphforge. Simulation-backend specific `Record` objects (e. g. `NEURONCellVoltageRecord`) are created by calling `get_recordable()` on the simulation objects (e. g. `NEURONCell`). This allows a simple interface to be exposed to the client, through calls to `Simulation.record()`, but allows simulator-backend specific actions to be executed (e. g. `build_hoc_section()` and `build_mod_file()` methods for NEURON objects).

(II) `traces` After simulations have been run, we might want to perform a variety of analyses on the output. A common task is to analyse part of a recorded value, for example for plotting or finding spike times from a membrane voltage. The analysis might be more complex, for example calculating coupling coefficients between two electrically coupled neurons using their deviations from resting potential in a particular time window following a step current injection to one of the neurons . We might also want to compare a simulated membrane voltage trace against an experimentally recorded one for example to find the difference as was done in Chapter 2.

These analogue signals are often represented as an array of times and an array of values. In many cases, DEs can be solved efficiently using variable time-step integrat-

ors [Hindmarsh et al., 2005], which return values of the signal at irregularly spaced times. This means that even simple operations such as adding two signals and may require interpolation of values so that both signals use the same time values. Since the time and data points for a single recording are coupled, it is useful to encapsulate them together as a single object and define functions that operate on both together.

Morphforge builds a `Trace` class on top of the python-quantities and `neurounit` libraries and encapsulates a time and signal-value array. The design of the `Trace` class in morphforge supports both regular and irregularly spaced time-bases transparently. The class has basic methods such as `mean()` and `max()` for simple analysis, and additional methods can be added dynamically. Operators, such as `+-/*` are suitably overloaded and return new `Trace` objects. The `Trace` objects transparently handle units.

Listing D.10 illustrates how the input resistance of a neuron can be calculated. The simulation code is omitted, but we assume that the voltage, `V`, is recorded from a single neuron, which is given a step current injection of 30 pA from 100 ms to 200 ms. We assume that the resting potential is reached after 50 ms and the steady state after the current injection after 150 ms. This code will work regardless of whether variable time steps are used in the simulation, and the resulting object will automatically contain the correct units.

```
# <code to create, and run a simulation omitted>
v = res.get_trace('v')
result=(v.window(150, 200)*ms - v.window(50, 90)*ms).mean() / (-30*pA)
```

Listing D.10 – Calcuating input resistance using `Trace` objects

D.1.2 *`simulationanalysis` layer*

After a simulation has been run, a `SimulationResult` object is returned which contains a set of `Trace` objects (see Section D.1.1). Most of the tools in the `simulationanalysis` layer operate on these `SimulationResult` objects. I next discuss three features of the `simulationanalysis` layer: (i) how a system of *tags* can be used to quickly select

simulation records, (ii) how a high-level plotting library makes it simple to investigate simulation traces (iii) how a summary can be automatically generated from an object-model of a `Simulation` as a pdf-document.

Facilitating loose coupling by using a simple system of tags

In a small network simulation, we may want to visualise the internal states of many neurons and synapses — how do we effectively choose which values to plot or use in other forms of analysis? Imagine we have two populations of neurons, `P1` & `P2`, synapses form stochastically between each pair of neurons in `P1` and `P2` with a probability of 0.3, and we want to plot the conductances of the synapses.

One option would be to store a *handle* from each call to `record()` for the conductance of each synapse. After simulation had `run()`, these handles could be used to look up the corresponding `Trace` object in the `SimulationResult` object. Alternatively, each call to `record()` could be passed an explicit name, as is done on line 29 in Listing 3.5, and later use this string to retrieve the relevant results. However both mechanisms require adding complex code to the simulation script: if handles are used, it is necessary to track which handle refers to which synapse recording and if explicit string names are used, a suitable naming system will be required that can cope with the stochasticity in the number of synapses. The situation quickly becomes more complex, for example, to plot the conductance traces of all synapses which have postsynaptic receptors on a particular neuron from a particular source population.

To solve this problem, morphforge introduces a system of *tags* in order to quickly find `Trace` objects recorded in a simulation. Each `Trace` object contains a set of strings called *tags*, which are used to attach contextual information about the `Trace`. The tags can be specified by the user explicitly during the call to `Simulation.record()`, (for example Listing D.9) and morphforge will also add certain tags automatically. For example, `'Voltage'` or `'CurrentDensity'` will be automatically added if the `Trace` object represents a voltage or current density recording and when recording from a `Synapse` object, morphforge will automatically add the tags `'PRE:cell1'` and `'POST:cell2'` where `cell1` and `cell2` are the names of the presynaptic and postsynaptic neurons respectively (more examples are given in the documentation).

A simple string-based language has been designed, for selecting specific sets of `Trace` objects after a simulation has run. The language uses the keywords: `ALL`, `ANY`, `AND`, `OR` and `NOT`. The terms `ALL{A,B,..,C}` and `ANY{X,Y,...,Z}` are matching predicates which take comma separated arguments. `ALL{A,B,..,C}` returns whether a particular `Trace` contains *all* the tags specified (i. e. A, B, C) and `ANY{X,Y,...,Z}` returns whether a `Trace` contains any of the tags specified (i. e. X, Y, Z). These match predicates can be joined with the `AND`, `OR` and `NOT` operators as well as brackets to allow more complex queries. For example, `ALL{Voltage}` will return all the voltages recorded in the simulation and `ALL{CONDUCTANCE,SYNAPTIC,PRE:cell1,POST:cell2} AND ANY{NMDA,AMPA}` could be used to retrieve all `Trace` objects representing conductances in AMPAR and NMDAR synapses from *cell1* to *cell2*.

This system of tagging, and the use of conventions such as *voltage traces always have a 'Voltage' tag* allows looser coupling between different parts of the code and allows more scripts to be more succinct.

Using conventions to simplify plotting

The `Trace` object contains methods, `time_pts_in()` & `data_pts_in()` that return `numpy` arrays of their time and data converted to specific units. These arrays can then be used for analysis with other scientific Python libraries. Listing D.11 shows how a `Trace` object can be plotted using the library matplotlib. Although this is one way of plotting results, morphforge provides another class `TagViewer` which makes it much less verbose to plot a selection of `Trace` objects from a simulation.

```
my_voltage_trace = result.get_trace(...)
pylab.plot(my_voltage_trace.time_pts_in('ms'), my_voltage_trace.data_pts_in('mV') )
```

Listing D.11 – Plotting a `Trace` object with matplotlib

The output of the `TagViewer` is a single figure, containing a series of axes with the same time base. The details of each axis, such as the y-label, the appropriate display range and unit are specified by `PlotSpec` objects. The `PlotSpec` object also takes a tag-selection string, to define which `Traces` should be plot on that axis, and rather than needing to explicitly specify which traces should be plot, the `TagViewer` object

directly queries the `SimulationResults` object. An example is given in Listing D.12, in which two axes will be displayed, one in which contains all `Traces` containing the tag 'Voltage', and another which contains `Traces` with both *Voltage* and *cell47* as tags. `TagViewer` objects have a set of `PlotSpecs` that are used by default and will automatically plot `'Voltage'`, `'Current'`, `'Conductance'` and other *standard-tags*. More examples of the figures generated for different simulations by `TagViewer` are given in Appendix E.

```
results = simulation.run()
TagViewer(results,
      specs = [
            PlotSpec('Voltage',....)
            PlotSpec('ALL{Voltage,cell47}', ....)
])
```

Listing D.12 – Plotting the results of a simulation with `TagViewer`

Producing self-documenting models

Models in computational neuroscience involve complex equations, many units and have large numbers of parameters; a typical HH-type sodium channel has 7 equations involving 12 parameters. Often modelling involves adjusting parameters. How do we keep track of which parameters produce which results? Experimentalists use lab notebooks as a way to record protocol setups but manually noting all the details of a complex simulation is unfeasible. One approach is to use version control, for example Sumatra [Davison, 2012]. An alternative approach is to generate summaries of a simulation from the internal object-model to produce a human readable output directly. The need for standard presentation formats for models has been recognised, even if exact formats have not yet been defined (e. g. [Nordlie et al., 2009; Nordlie and Plesser, 2010; Crook et al., 2012]).

Morphforge supports the production of html and pdf-document summaries from `Simulation` objects directly using `mredoc`, (Modular Reduced Documentation) library. This library is a high-level interface for producing documents containing images, tables, code-snippets and equations for documenting mathematical models. After the `Simulation` object has been populated, it can be summarised as shown in Listing D.13

```
sim = env.Simulation()

# Populate the simulation ...
sim.create_cell(...)
sim.create_synapse(...)

# Summarise the object
SummaryManager.summarise(simulation).to_pdf('~/mysimulation.pdf')
```

Listing D.13 – Building summaries of simulations in morphforge

Simulations in morphforge can be populated with `Synapse` and `Channel` objects of different types, for example NineML, NeuroML & `neurounit`. The summary architecture allows these objects to create summaries of themselves. An example of summarising a simulation and the resulting pdf document are given in Appendix E.

D.2 IMPLEMENTATION DETAILS

D.2.1 *Defining channel distributions*

The distribution of channels on a neuron is specified by calls to `Cell.apply_channel( chl, applicator, targeter)`, abbreviated `C,A,T`, described in Section D.1.1. Because multiple calls to `apply_channel()` are allowed, in the case that for a single `Channel` object, different parameters are specified to be specified to the same `Section`, a system of priorities is used to determine which parameters should be used. The targeter object has a method `get_priority_level()` which returns an integer. For each `Section`, for each `Channel`, the relevant triplet,`(c,a,t)` in which the targeter object has the highest priority is chosen. The pseudocode for this process is given in Algorithm 1.

Algorithm 1: The algorithm used by morphforge to resolve which parameters are used by a channels in different `Sections` of a neuron

```
Data: all_sections, cat_list
for s in all_sections do
    chls ⟵ ∅ ;
    cat_target ⟵ ∅ ;
    for (c, a, t) in cat_list do
        if t.does_target(s) then
            if c not in chls then
                chls.add(c) ;
            cat_target.add((c, a, t)) ;
    for chl in chls do
        (hp, hpa) ⟵ (0, None) ;
        hp_ambiguous ⟵ False ;
        for (c, a, t) in cat_targets do
            if t.priority = hp then
                hp_ambiguous ⟵ True ;
            if t.priority > hp then
                hp_ambiguous ⟵ False ;
                hp, hpa ⟵ (t.priority, a) ;
        if hp_ambiguous == False then
            Apply chl to s using hpa ;
        else
            Raise an exception. ;
```

D.2.2 *Tag selection grammar*

`Trace` objects in morphforge can be selected using *tag-selection* strings, which specify which tags an object must have to be selected (see Section D.1.2). Examples are given in Listing D.14. A simple BNF grammar to parse these strings in Grammar D.1. The starting symbol for the grammar is `expr`. A `<TAG>` token must start with letter, followed by any number of alphanumeric characters, underscore, dash, period and colon. (defined by the Python regular expression `r"'[a-zA-Z_][a-zA-Z0-9_.:-]*"'`). The precedence rules for `and`, `or` and `not` operators are the same as those in C [Kernighan and Ritchie, 1988].

```
TagSelect('ALL{Voltage}')
TagSelect('ALL{Voltage} AND ANY{cell1,cell2,cell3}')
TagSelect('{ALL{Current,cell1} OR ( ALL{Synaptic,Current,POST:cell1} AND ANY {NMDA,AMPA} ) }')
```

Listing D.14 – Example tag-selection strings, which could be used to select particular `Trace` objects after a simulation

⟨*expr*⟩ ::= ⟨*tag_term_factor*⟩
| ⟨*tag_line_simple*⟩
| '(' ⟨*expr*⟩ ')'
| ⟨*expr*⟩ 'AND' ⟨*expr*⟩
| ⟨*expr*⟩ 'OR' ⟨*expr*⟩
| 'NOT' ⟨*expr*⟩

⟨*tag_item_simple*⟩ ::= ⟨*TAG*⟩

⟨*tag_line_simple*⟩ ::= ⟨*TAG*⟩
| ⟨*tag_line_simple*⟩ ',' ⟨*TAG*⟩

⟨*tag_term_factor*⟩ ::= ⟨*tag_item_simple*⟩
| ⟨*tag_group_factor_all*⟩
| ⟨*tag_group_factor_any*⟩

⟨*tag_group_factor_all*⟩ ::= 'ALL' ⟨*tag_group_bracketed*⟩

⟨*tag_group_factor_any*⟩ ::= 'ANY' ⟨*tag_group_bracketed*⟩

⟨*tag_group_bracketed*⟩ ::= '' ⟨*tag_group*⟩ ''

⟨*tag_group*⟩ ::= ⟨*empty*⟩
| ⟨*TAG*⟩
| ⟨*tag_group*⟩ ',' ⟨*TAG*⟩

Grammar D.1 – The BNF grammar used to define a syntax for selecting objects based on a system of tags

E

MORPHFORGE EXAMPLE SIMULATIONS

E.1 CURRENT INJECTION INTO A SINGLE COMPARTMENT NEURON WITH HH-TYPE CHANNELS

In this example, a single-compartment neuron containing HH-type leak, sodium and potassium channels is stimulated with a step current injection (Listing E.1). This produces the graphs shown in Figure E.1.

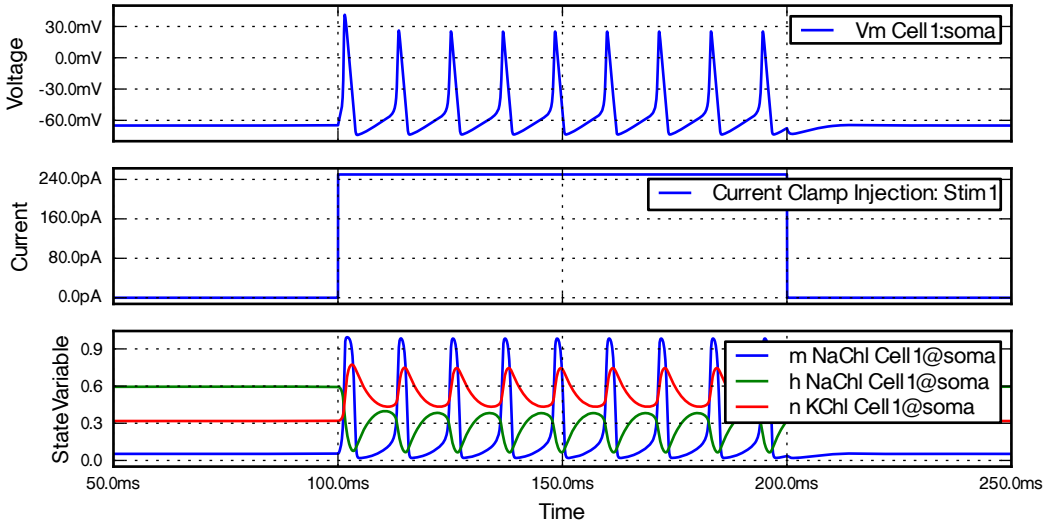


Figure E.1 – The output figure produced from running Listing E.1. The graphs show the membrane voltage, the injected current, and the state variables of the channels during simulation.

```
from morphforge.stdimports import *
from morphforgecontrib.stdimports import *

#from morphforgecontrib.simulation.channels.hh_style.core.mmleak import StdChlLeak
#from morphforgecontrib.simulation.channels.hh_style.core.mmalphabeta import StdChlAlphaBeta

# Create the environment & simulation:
env = NEURONEnvironment()
sim = env.Simulation()

# Create a cell:
cell = sim.create_cell(name="Cell1", morphology=MorphologyBuilder.get_single_section_soma(rad=10))

lk_chl = env.Channel(StdChlLeak, name="LkChl",
                conductance=qty("0.3:mS/cm2"), reversalpotential=qty("-54.3:mV"))

na_state_vars = { "m": {
                  "alpha":[-4.00,-0.10,-1.00,40.00,-10.00],
                  "beta": [4.00, 0.00, 0.00,65.00, 18.00]},
                 "h": {
                    "alpha":[0.07,0.00,0.00,65.00,20.00] ,
                    "beta": [1.00,0.00,1.00,35.00,-10.00]}}

na_chl = env.Channel(StdChlAlphaBeta,
                name="NaChl", equation="m*m*m*h",
                conductance=qty("120:mS/cm2"), reversalpotential=qty("50:mV"),
                statevars=na_state_vars)

k_state_vars = { "n": {
                  "alpha":[-0.55,-0.01,-1.0,55.0,-10.0],
                  "beta": [0.125,0,0,65,80]}}

k_chl = env.Channel(StdChlAlphaBeta,
                name="KChl", equation="n*n*n*n",
                conductance=qty("36:mS/cm2"), reversalpotential=qty("-77:mV"),
                statevars=k_state_vars)

# Apply the channels uniformly over the cell, and set capacitance
cell.apply_channel(lk_chl)
cell.apply_channel(na_chl)
cell.apply_channel(k_chl)
cell.set_passive( PassiveProperty.SpecificCapacitance, qty('1.0:uF/cm2'))

# Create the stimulus and record the injected current:
cc = sim.create_currentclamp(
         name="Stim1", cell_location=cell.soma,
         amp=qty("250:pA"), dur=qty("100:ms"), delay=qty("100:ms"))

# Define what to record:
sim.record(cc, what=StandardTags.Current)
sim.record(cell, what=StandardTags.Voltage, name="SomaVoltage", cell_location = cell.soma)
sim.record(na_chl, what=StandardTags.StateVariable, state='m', cell_location = cell.soma)
sim.record(na_chl, what=StandardTags.StateVariable, state='h', cell_location = cell.soma)
sim.record(k_chl, what=StandardTags.StateVariable, state='n', cell_location = cell.soma)

# Run the simulation & display the results:
results = sim.run()
TagViewer(results, timerange=(50, 250)*units.ms, show=True)
```

Listing E.1 – An example simulation using morphforge, in which an HH-type neuron is stimulated with a step current injection.

E.2 THE EFFECT OF AXONAL RADIUS ON THE SPEED OF ACTION POTENTIAL PROPAGATION.

In this example, a neuron consisting of a soma with a long axon, which containing HH-type leak, sodium and potassium channels is stimulated in the soma with a short step current injection to initiate an action potential (Listing E.2). The voltage is recorded at points along the axon to show the action potential propagation. Three simulations are run, in which the radius of the axon is 0.2, 0.4 and 0.6 µm. This produces the graphs shown in Figure E.2.

```
from morphforge.stdimports import *
from morphforgecontrib.data_library.stdmodels import StandardModels


def sim(axon_radius, tag):
    # Create the environment:
    env = NEURONEnvironment()

    # Create the simulation:
    sim = env.Simulation()

    # Create a cell:
    morph = MorphologyBuilder.get_soma_axon_morph(axon_length=3000.0, axon_radius=axon_radius, ↙
        ↳ soma_radius=9.0, axon_sections=20)
    cell = sim.create_cell(name="Cell1", morphology=morph)
    lk_chl = ChannelLibrary.get_channel(modelsrc=StandardModels.HH52, channeltype="Lk", env=env)
    na_chl = ChannelLibrary.get_channel(modelsrc=StandardModels.HH52, channeltype="Na", env=env)
    k_chl = ChannelLibrary.get_channel(modelsrc=StandardModels.HH52, channeltype="K", env=env)

    # Apply the channels uniformly over the cell
    cell.apply_channel(lk_chl)
    cell.apply_channel(na_chl)
    cell.apply_channel(k_chl)

    # Over-ride the parameters in the axon:
    cell.apply_channel(channel=lk_chl, where="axon", parameter_multipliers={'gScale':1.2})
    cell.apply_channel(channel=na_chl, where="axon", parameter_multipliers={'gScale':1.2})

    # Record at 100um intervals along the axon
    for cell_location in CellLocator.get_locations_at_distances_away_from_dummy(cell=cell, ↙
        ↳ distances=range(9, 3000, 100)):
        sim.record(cell, what=StandardTags.Voltage, cell_location=cell_location, user_tags=[tag])
```

```
    # Create the stimulus and record the injected current:
    cc = sim.create_currentclamp(name="Stim1", amp=qty("250:pA"), dur=qty("5:ms"), ↙
        ↳ delay=qty("100:ms"), cell_location=cell.soma)
    sim.record(cc, what=StandardTags.Current,user_tags=[tag])

    # run the simulation
    return sim.run()



results = [
    sim(axon_radius=0.2, tag="SIM1"),
    sim(axon_radius=0.4, tag="SIM2"),
    sim(axon_radius=0.6, tag="SIM3"),
]

kw = { 'yrange':(-80, 50)*units.mV, 'legend_labeller':None, 'yticks':3}
TagViewer(results, timerange=(97.5, 120)*units.ms, show=False,
    plots = [
        TagPlot("ALL{Current,SIM1}", ylabel='IInj', legend_labeller=None),
        TagPlot("ALL{Voltage,SIM1}", ylabel='R:0.2um\nVoltage', **kw),
        TagPlot("ALL{Voltage,SIM2}", ylabel='R:0.4um\nVoltage', **kw),
        TagPlot("ALL{Voltage,SIM3}", ylabel='R:0.6um\nVoltage', **kw),
    ])
```

Listing E.2 – An example of running three simulations in a single script using morphforge. An HH-type multicompartmental neuron with an axon is stimulated with a step current injection. The radius of the axon is set to 0.2, 0.4 and 0.6 µm in the three simulations. The current injected into the first stimulation is shown on the top graph, and the voltages recorded along the neurons' axons in the three experiments are shown in the graphs below.

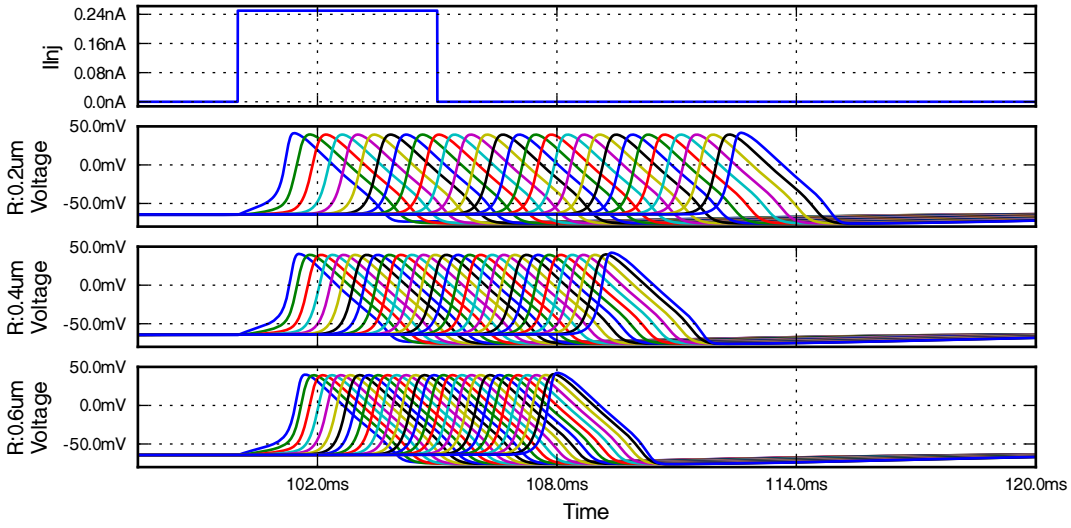


Figure E.2 – The output figure produce from running Listing E.2. The graphs show the effect of axon diameter on action potential propagation velocity.

E.3 SIMULATION OF THREE NEURONS USING NEUROUNIT TO DEFINE SYNAPSES

In this example, three neurons are created (cell1, cell2 & cell3), each of which have a soma and a long axon. A PostSynaptic template is created from a neurounit string (Listing E.1). Two synapses are instantiated using this template, from cell1 → cell2 and cell1 → cell3, in which the presynaptic trigger is at different points on the axon of cell1. The conductance of the second synapse is twice that of the first. This produces the graphs shown in Figure E.3. This simulation also produces a summary pdf document, shown in Figure E.4.

```
from morphforge.stdimports import *
from morphforgecontrib.stdimports import *

# Create a cell:
def build_cell(name, sim):
    morph = MorphologyBuilder.get_soma_axon_morph(axon_length=1500., axon_radius=0.3, ↙
        ↳ soma_radius=10.0)
    cell = sim.create_cell(name=name, morphology=morph)
    model = StandardModels.HH52
    na_chls = ChannelLibrary.get_channel(modelsrc=model, channeltype="Na", env=sim.environment)
    k_chls = ChannelLibrary.get_channel(modelsrc=model, channeltype="K", env=sim.environment)
    lk_chls = ChannelLibrary.get_channel(modelsrc=model, channeltype="Lk", env=sim.environment)
```

```
    cell.apply_channel(lk_chls)
    cell.apply_channel(k_chls)
    cell.apply_channel(na_chls)
    cell.apply_channel(na_chls, where="axon", parameter_multipliers={'gScale':1.2})
    return cell


# Create a simulation:
env = NEURONEnvironment()
sim = env.Simulation(name='Example3_Test_Neurounit_Synapse')

# Create three cells
cell1 = build_cell(name="cell1", sim=sim)
cell2 = build_cell(name="cell2", sim=sim)
cell3 = build_cell(name="cell3", sim=sim)

# Define a synapse in NeuroUnit format, and use it to build a template:
simple_ampa_syn = """
eqnset syn_simple {
    g' = - g/g_tau
    i = gmax * (v-erev) * g

    gmax = 300pS * scale
    erev = 0mV
    g_tau = 10ms

    ==>> on_event() {
        g = g + 1.0
    }

    <=> INPUT v: mV METADATA {"mf":{"role":"MEMBRANEVOLTAGE"} }
    <=> OUTPUT i:(mA) METADATA {"mf":{"role":"TRANSMEMBRANECURRENT"} }
    <=> PARAMETER scale:()
}"""

post_syn_tmpl = env.PostSynapticMechTemplate(NeuroUnitEqnsetPostSynaptic,
      name = 'MyTemplate', eqnset = simple_ampa_syn, default_parameters = { 'scale':1.0} )

# Create two instances of the synaptic template: cell1->cell2 and cell1-> cell3
syn1 = sim.create_synapse(
      name='Syn1',
      trigger = env.SynapticTrigger(
          SynapticTriggerByVoltageThreshold,
          cell_location = CellLocator.get_location_at_distance_away_from_dummy(cell1, 300),
          voltage_threshold = qty("0:mV"), delay=qty("0:ms") ),
      postsynaptic_mech = post_syn_tmpl.instantiate(
```

```
        cell_location = cell2.soma)
    )

syn2 = sim.create_synapse(
    name='Syn2',
    trigger = env.SynapticTrigger(
        SynapticTriggerByVoltageThreshold,
        cell_location = CellLocator.get_location_at_distance_away_from_dummy(cell1, 700),
        voltage_threshold = qty("0:mV"), delay = qty("0:ms") ),
    postsynaptic_mech = post_syn_tmpl.instantiate(
        cell_location = cell3.soma,
        parameter_overrides={'scale':2.0})
    )

# Record Voltages from axons:
record_locs = CellLocator.get_locations_at_distances_away_from_dummy(cell1, range(0, 1000, 50))
for loc in record_locs:
   sim.record( what=StandardTags.Voltage, cell_location = loc, user_tags=['cell1'])
sim.record(what=StandardTags.Voltage, cell_location=cell2.soma, user_tags=['cell2'])
sim.record(what=StandardTags.Voltage, cell_location=cell3.soma, user_tags=['cell3'])

# Create the stimulus and record the injected current:
cc = sim.create_currentclamp(name="CC1", amp=qty("200:pA"), dur=qty("1:ms"), delay=qty("100:ms"), ↙
    ↳ cell_location=cell1.soma)
sim.record(cc, what=StandardTags.Current)

results = sim.run()
mV = units.mV
t = TagViewer(results,
   timerange=(98, 120)*units.ms,
   plots = [
      TagPlot('Current', yunit=units.picoamp),
      TagPlot('Voltage,cell1', ylabel='Presynaptic\nVoltages', yrange=(-80, 50)*mV, yunit=mV, ↙
          ↳ legend_labeller=False),
      TagPlot('Voltage AND ANY{cell2,cell3}', ylabel='Postsynaptic\nVoltages',yrange=(-70, -55)*mV, ↙
          ↳ yunit=units.mV),
        ],
      )
# Build the summary:
options = SummariserOptions()
options.include_details_indvidual_neuron_morphology_mpl = False
SimulationMRedoc.build(sim, options=options).to_pdf('app_ex3_summary.pdf')
```

Listing E.3 – An example simulation containing three neurons and two synapses. The neurons are all multicompartmental and have a soma and a long axon. The neurons are connected via two synapses, (cell1 → cell2 and (cell1 → cell3) and the kinetics of the synapses are defined using neurounit.

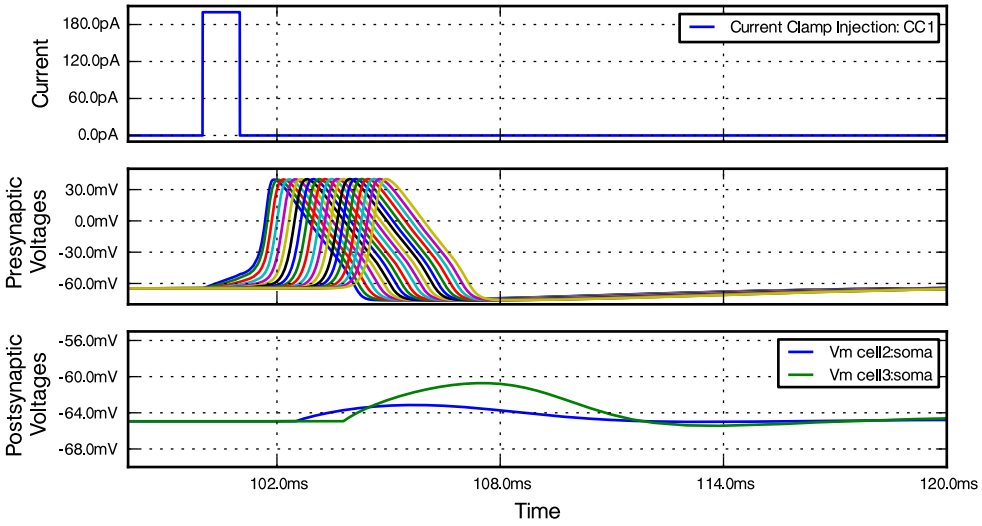
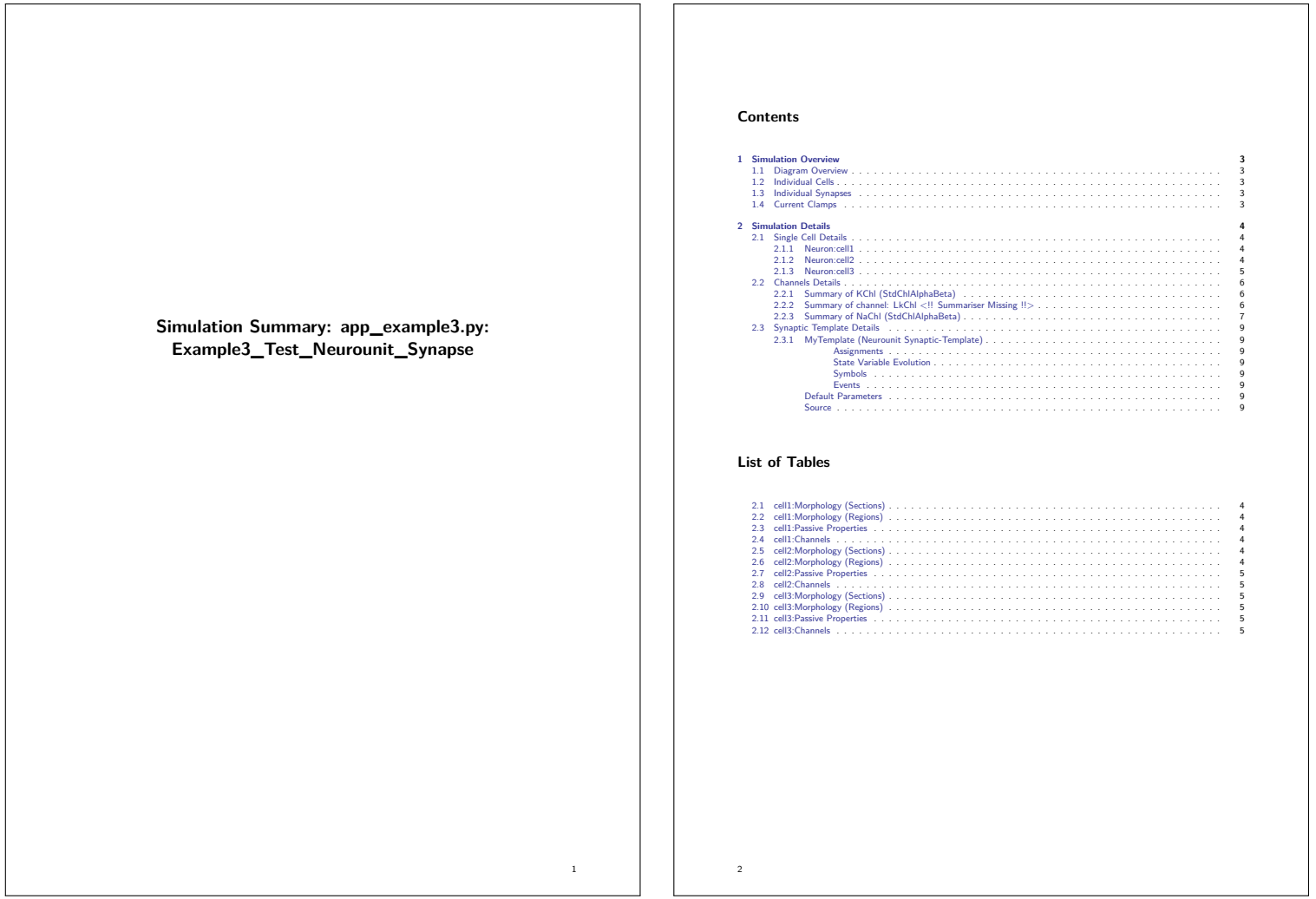


Figure E.3 – The output figure produce from running Listing E.3



Simulation Summary: app_example3.py: Example3_Test_Neurounit_Synapse

1

Contents

List of Tables

2

Figure E.4 – The summary pdf document produced by Listing E.3

1 Simulation Overview

1.1 Diagram Overview

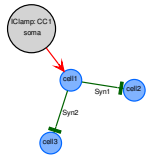


Figure 1.1

1.2 Individual Cells

Name	Type	SA(um2)	#sections/segments	Regions(SA(um2):nseg)	#Pre/Post-Synapse	#GapJunctions	Chls
cell1	<?>	8382	10:284	axon(7125:279) soma(1256:5)	2 0	0	LkChl NaChl KChl
cell2	<?>	8382	10:284	axon(7125:279) soma(1256:5)	0 1	0	LkChl NaChl KChl
cell3	<?>	8382	10:284	axon(7125:279) soma(1256:5)	0 1	0	LkChl NaChl KChl

1.3 Individual Synapses

Name	Type	Trigger	PostSynaptic Cell	Receptor
Syn1	<?>	cell1@axon_2: [threshold: 0.0 mV]	cell2@soma	<Defined through NeuroUnit: 'MyTemplate' >
Syn2	<?>	cell1@axon_5: [threshold: 0.0 mV]	cell3@soma	<Defined through NeuroUnit: 'MyTemplate' Overrides: 'scale': 2.0>

1.4 Current Clamps

Name	Location	Description
CC1	cell1 (0um from soma)	Step-Change: amp=2e-10 A dur=0.001 s delay=0.1 s

2 Simulation Details

2.1 Single Cell Details

2.1.1 Neuron:cell1

ID	Tags	Lateral Surface Area (um2)	Region	nseg	L	diam (prox/dist)
0	soma	1257	soma	5	20.0	20.0/20.0
1	axon_1	4864	axon	31	150.0	20.0/0.6
2	axon_2	283	axon	31	150.0	0.6/0.6
3	axon_3	283	axon	31	150.0	0.6/0.6
4	axon_4	283	axon	31	150.0	0.6/0.6
5	axon_5	283	axon	31	150.0	0.6/0.6
6	axon_6	283	axon	31	150.0	0.6/0.6
7	axon_7	283	axon	31	150.0	0.6/0.6
8	axon_8	283	axon	31	150.0	0.6/0.6
9	axon_9	283	axon	31	150.0	0.6/0.6

Table 2.1 – cell1:Morphology (Sections)

Region	Surface Area	#Sections
axon	7125.84544684	9
soma	1256.63706144	1

Table 2.2 – cell1:Morphology (Regions)

PassiveProp	Priority	Targetter	Value
AxialResistance	0	Default	80.0 ohmcm
SpecificCapacitance	0	Default	1.0 uF/cm2

Table 2.3 – cell1:Passive Properties

Mechanism	Priority	Targetter	Applicator
LkChl	10	Everywhere	Uniform Applicator:
KChl	10	Everywhere	Uniform Applicator:
NaChl	10	Everywhere	Uniform Applicator:
NaChl	20	Region: axon	Uniform Applicator: Multipliers:gScale=1.2

Table 2.4 – cell1:Channels

2.1.2 Neuron:cell2

ID	Tags	Lateral Surface Area (um2)	Region	nseg	L	diam (prox/dist)
0	soma	1257	soma	5	20.0	20.0/20.0
1	axon_1	4864	axon	31	150.0	20.0/0.6
2	axon_2	283	axon	31	150.0	0.6/0.6
3	axon_3	283	axon	31	150.0	0.6/0.6
4	axon_4	283	axon	31	150.0	0.6/0.6
5	axon_5	283	axon	31	150.0	0.6/0.6
6	axon_6	283	axon	31	150.0	0.6/0.6
7	axon_7	283	axon	31	150.0	0.6/0.6
8	axon_8	283	axon	31	150.0	0.6/0.6
9	axon_9	283	axon	31	150.0	0.6/0.6

Table 2.5 – cell2:Morphology (Sections)

Region	Surface Area	#Sections
axon	7125.84544684	9
soma	1256.63706144	1

Table 2.6 – cell2:Morphology (Regions)

PassiveProp	Priority	Targetter	Value
AxialResistance	0	Default	80.0 ohmcm
SpecificCapacitance	0	Default	1.0 uF/cm2

Table 2.7 – cell2:Passive Properties

Mechanism	Priority	Targetter	Applicator
LkChl	10	Everywhere	Uniform Applicator:
KChl	10	Everywhere	Uniform Applicator:
NaChl	10	Everywhere	Uniform Applicator:
NaChl	20	Region: axon	Uniform Applicator: Multipliers:gScale=1.2

Table 2.8 – cell2:Channels

2.1.3 Neuron:cell3

ID	Tags	Lateral Surface Area (um2)	Region	nseg	L	diam (prox/dist)
0	soma	1257	soma	5	20.0	20.0/20.0
1	axon_1	4864	axon	31	150.0	20.0/0.6
2	axon_2	283	axon	31	150.0	0.6/0.6
3	axon_3	283	axon	31	150.0	0.6/0.6
4	axon_4	283	axon	31	150.0	0.6/0.6
5	axon_5	283	axon	31	150.0	0.6/0.6
6	axon_6	283	axon	31	150.0	0.6/0.6
7	axon_7	283	axon	31	150.0	0.6/0.6
8	axon_8	283	axon	31	150.0	0.6/0.6
9	axon_9	283	axon	31	150.0	0.6/0.6

Table 2.9 – cell3:Morphology (Sections)

Region	Surface Area	#Sections
axon	7125.84544684	9
soma	1256.63706144	1

Table 2.10 – cell3:Morphology (Regions)

PassiveProp	Priority	Targetter	Value
AxialResistance	0	Default	80.0 ohmcm
SpecificCapacitance	0	Default	1.0 uF/cm2

Table 2.11 – cell3:Passive Properties

Mechanism	Priority	Targetter	Applicator
LkChl	10	Everywhere	Uniform Applicator:
KChl	10	Everywhere	Uniform Applicator:
NaChl	10	Everywhere	Uniform Applicator:
NaChl	20	Region: axon	Uniform Applicator: Multipliers:gScale=1.2

Table 2.12 – cell3:Channels

2.2 Channels Details

2.2.1 Summary of KChl (StdChlAlphaBeta)

$$g = gmax * n * n * n * n$$
$$i = g * (erev - V)$$

$$\frac{d}{dt} n = \frac{n_\infty (V) - n}{\tau_n (V)}$$
$$n_\infty (V) = \frac{\alpha_n (V)}{\alpha_n (V) + \beta_n (V)}$$
$$\tau_n (V) = \frac{1.0}{\alpha_n (V) + \beta_n (V)}$$
$$\alpha_n (V), \beta_n (V) = \frac{A + BV}{C + exp\left(\frac{D+V}{E}\right)}$$

Parameter	Value
Conductance (gmax)	36.0 mS/cm2
Reversal Potential	-77.0 mV

n	A	B	C	D	E
Alpha	-0.55	-0.01	-1.0	55.0	-10.0
Beta	0.125	0	0	65	80

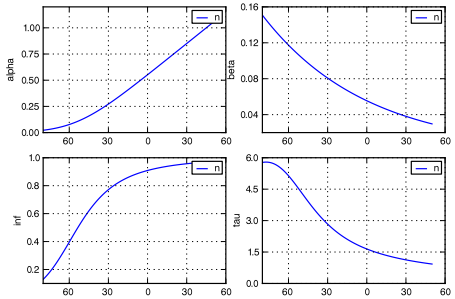


Figure 2.1 – The rate constants and resulting steady-state activation and time constants for n

2.2.2 Summary of channel: LkChl <!! Summariser Missing !!>

<Summariser Missing for type: <class 'morphforgecontrib.simulation.channels.hh_style.neuron.mm_neuron_leak.NEURONChl_Leak'»

2.2.3 Summary of NaChl (StdChlAlphaBeta)

$$g = gmax * m * m * m * h$$
$$i = g * (erev - V)$$

$$\frac{d}{dt}h = \frac{h_\infty(V) - h}{\tau_h(V)}$$
$$h_\infty(V) = \frac{\alpha_h(V)}{\alpha_h(V) + \beta_h(V)}$$
$$\tau_h(V) = \frac{1.0}{\alpha_h(V) + \beta_h(V)}$$
$$\alpha_h(V), \beta_h(V) = \frac{A + BV}{C + exp\left(\frac{D+V}{E}\right)}$$
$$\frac{d}{dt}m = \frac{m_\infty(V) - m}{\tau_m(V)}$$
$$m_\infty(V) = \frac{\alpha_m(V)}{\alpha_m(V) + \beta_m(V)}$$
$$\tau_m(V) = \frac{1.0}{\alpha_m(V) + \beta_m(V)}$$
$$\alpha_m(V), \beta_m(V) = \frac{A + BV}{C + exp\left(\frac{D+V}{E}\right)}$$

Parameter	Value
Conductance (gmax)	120.0 mS/cm2
Reversal Potential	50.0 mV

h	A	B	C	D	E
Alpha	0.07	0.0	0.0	65.0	20.0
Beta	1.0	0.0	1.0	35.0	-10.0

m	A	B	C	D	E
Alpha	-4.0	-0.1	-1.0	40.0	-10.0
Beta	4.0	0.0	0.0	65.0	18.0

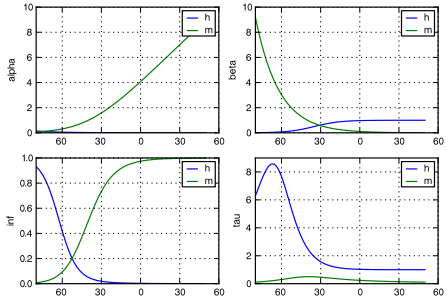


Figure 2.2 – The rate constants and resulting steady-state activation and time constants for h,m

2.3 Synaptic Template Details

2.3.1 MyTemplate (Neurounit Synaptic-Template)

Summary of 'syn_simple'

Assignments

$$\boldsymbol{gmax} = \langle 300.0 \left(10^{-12}\right) \mathrm{m}^{-2} \cdot \mathrm{kg}^{-1} \cdot \mathrm{s}^{3} \cdot \mathrm{A}^{2} \rangle \cdot \boldsymbol{scale}$$
$$\boldsymbol{i} = \boldsymbol{gmax} \cdot (\boldsymbol{v} - \boldsymbol{erev}) \cdot \boldsymbol{g}$$

State Variable Evolution

$$\frac{d}{dt}\boldsymbol{g} = -1.0 \cdot \frac{\boldsymbol{g}}{\boldsymbol{g_\tau}}$$

Symbol	Type	Value	Dimensions	Dependancies	Metadata
$\boldsymbol{scale}$	Param	-	-	-	-
$\boldsymbol{v}$	Supp	-	V	-	u'mf': u'role': u'MEMBRANEVOLTAGE'
$\boldsymbol{erev}$	Const	0.0 (10e-3) m 2 kg 1 s -3 A -1	V	-	-
$\boldsymbol{g_\tau}$	Const	10.0 (10e-3) s 1	s^1	-	-
$\boldsymbol{gmax}$	Assd	-	S	$\{\boldsymbol{scale}\}$	-
$\boldsymbol{i}$	Assd	-	A	$\{\boldsymbol{g}, \boldsymbol{scale}, \boldsymbol{v}\}$	u'mf': u'role': u'TRANSMEMBRANECURRENT'
$\boldsymbol{g}$	State	-	-	$\{\}$	-

Symbols

Events

$$on\_event\ () \rightarrow \left\{\boldsymbol{g} = (\boldsymbol{g} + 1.0)\right.$$

Default Parameters

Parameter	Default Value
scale	1.0

Source

Listing 2.1 – Source code for template: MyTemplate

```
eqnset syn_simple{
g'=-g/g_tau;
i=gmax*(v-erev)*g;
gmax=300pS*scale;
erev=0mV;
g_tau=10ms;
==>>on_event(){
g=g+1.0;
};
<=>INPUT v:mV METADATA{"mf":{"role":"MEMBRANEVOLTAGE"}};
<=>OUTPUT i:(mA)METADATA{"mf":{"role":"TRANSMEMBRANECURRENT"}};
<=>PARAMETER scale:();
};
```

F

SIMULATOR TEST DATA REPOSITORY

Morphforge has been built iteratively according to the needs of the modelling in this thesis. In order to ensure that bugs are not accidentally introduced into the morphforge library, a testsuite was written to allow morphforge to be tested (Section 3.6.5). The Simulator-TestData repository defines a set of *scenarios* in a consistent, human and machine-readable text file.

Each scenario file describes the setup for a simulation. A scenario designed to test the implementation of an alpha-type synapse is shown in Listing F.1. The description is given as a text string [lines: 4-13]. The specification allows parameters to be used in the description (e. g. `<GLEAK>`, `<GSYN>`, `<ESYN>` & `<TCLOSE>`), and defines what should be recorded (e. g. the voltage of *cell1* as `V` and the conductance and currents of the synapse as `SYN_G` & `SYN_I`). The file defines which units to use for all the parameters and recorded values [lines: 20-28], and defines the values that should be used for each parameter in a parameter sweep [lines: 30-34]. The format also describes the columns that should be used for output results in CSV format [line: 37] and the output filename which is defined by the parameters used for that particular simulation [lines: 38]. The repository allows validation of simulation results in two ways. Firstly, results can be compared against a set of hand-calculated results [lines: 43-77], and secondly, results from different simulators can be compared against each other. The repository supports the parametrisation of simulations. To define hand-calculated responses for particular parameters in the text file concisely, *ASCII-table* syntax is used. These tables have two sets of columns: 1 set specifies the particular parameters used [line: 47 `<GLEAK>`, `<GSYN>`, `<ESYN>` & `<TCLOSE>`] another set defines the expected output [line 47: `SYN_G[100.5:290].max`, `SYN_I[100.5:290].min` & `V[100:290].max`]. Values of output

at particular point in time [line 72: `SYN_G[50]`], can be used for comparison, or simple numerical methods such as `mean`, `min` and `max` [line 47] can be used with a *slice-like* notation for specifying periods of time, for example, `SYN_G[100.5:290].max` gives the maximum value of `SYN_G` between 100 to 349 ms. It is possible to set tolerances per column [line: 47] , per table, or per scenario [line: 41]. The repository contains 10 scenario files to test the objects used in this thesis: neurons, channels, synapses & gap junctions. Implementations of the scenarios have been written in NEURON for comparison with morphforge.

```
scenario_short= scenario020
title = Response of a passive cell to an alpha-synapse driven with events

description = """
In a simulation lasting 350ms
Create a single compartment neuron 'cell1' with area 10000um2 and initialvoltage -50mV and capacitance 1e-3 uF/cm2
Add Leak channels to cell1 with conductance <GLEAK> and reversalpotential -50mV
Create a SingleExponential synapse 'SYN' onto cell1 with conductance <GSYN> and reversalpotential <ESYN> and closing-time <TCLOSE>
    driven with spike-times at [100, 300, 300]ms
Record cell1.V as $V
Record SYN.Conductance as $SYN_G
Record SYN.Current as $SYN_I
Run the simulation
"""

[Sampling]
    stop = 350
    dt = 0.1

[Units]
    t = ms
    GLEAK=mS/cm2
    TCLOSE=ms
    ESYN=mV
    GSYN=pS
    V=mV
    SYN_G=pS
    SYN_I=pA

[Parameter Values]
    GSYN=  500, 1000
    ESYN=  0, -20
    GLEAK= 0.03333, 0.014286
    TCLOSE= 5, 20, 100000

[Output Format]
    columns = t, V, SYN_G, SYN_I
    base_filename = scenario020_ESYN<ESYN>_GSYN<GSYN>_GLEAK<GLEAK>_TCLOSE<TCLOSE>_result_

[Check Values]
    eps = 0.05

expectations_i = """
# 1. Use a long decaying syanpse to check that the steady state voltages seem right
# for different values of input resistance, synaptic conductance and reveral potential
| GLEAK    | GSYN    | ESYN | TCLOSE  | SYN_G[100.5:290].max (eps=0.5) | SYN_I[100.5:290].min | V[100:290].max |
|----------|---------|------|---------|--------------------------------|----------------------|----------------|
| 0.03333  | 1000.00 | 0    | 100000  | 1000                           | -38.46               | -38.4615       |
| 0.014286 | 1000.00 | 0    | 100000  | 1000                           | ?                    | -29.4118       |
| 0.03333  |  500.00 | 0    | 100000  | 500                            | ?                    | -43.4769       |
| 0.014286 |  500.00 | 0    | 100000  | 500                            | ?                    | -37.0370       |
| 0.03333  | 1000.00 | -20  | 100000  | 1000                           | ?                    | -43.0769       |
| 0.014286 | 1000.00 | -20  | 100000  | 1000                           | ?                    | -37.6471       |
| 0.03333  |  500.00 | -20  | 100000  | 500                            | ?                    | -46.0869       |
| 0.014286 |  500.00 | -20  | 100000  | 500                            | ?                    | -42.2222       |
"""
expectations_ii = """
# 2. Check that the synapse decays as expected over time:
| GLEAK       | GSYN     | ESYN   | TCLOSE    | SYN_G[102](eps=0.005) |
|-------------|----------|--------|-----------|-----------------------|
| 0.03333     | 1000.00  | 0      | 5         | 670.32                |
| 0.03333     |  500.00  | 0      | 5         | 335.16                |
| 0.03333     | 1000.00  | 0      | 20        | 904.84                |
| 0.03333     |  500.00  | 0      | 20        | 452.42                |
"""

expectations_iii = """
# 3. Check that multiple events work as expected:
| GLEAK       | GSYN     | ESYN   | TCLOSE    | SYN_G[302] (eps=0.5) |
|-------------|----------|--------|-----------|----------------------|
| 0.03333     | 1000.00  | 0      | 5         | 1340.64              |
| 0.03333     |  500.00  | 0      | 5         |  670.32              |
| 0.03333     | 1000.00  | 0      | 20        | 1809.68              |
| 0.03333     |  500.00  | 0      | 20        |  904.84              |
"""
```

Listing F.1 – An example scenario file from the Simulator-TestData repository. The scenario is designed to test an alpha-type synapse.

G

SIMULATION PLATFORM

All simulations were run on a quad-core Intel i5 desktop processor, running Ubuntu Linux, 12.10, kernel `3.2.0-38-generic-pae`. The versions of software used for the simulations are given in Table G.1. Several additional packages were developed through the course of this thesis, details are given in Table G.2.

SOFTWARE	VERSION	URL
Python	2.7.3	http://www.python.org/
numpy	1.6.1	http://www.numpy.org/
scipy	0.9.0	http://www.scipy.org/
matplotlib	1.1.1rc	http://www.matplotlib.org/
quantities	0.10.1	http://pythonhosted.org/quantities/
NEURON	7.1 (359:7f113b76a94b)	http://www.neuron.yale.edu/neuron/

Table G.1 – The versions of software used in for simulations

SOFTWARE	URLS
morphforge	DOC: https://morphforge.rtfd.org SRC: https://github.com/mikehulluk/morphforge
neurounit	DOC: https://neurounit.rtfd.org SRC: https://github.com/mikehulluk/neurounit
mreorg	DOC: https://mredoc.rtfd.org SRC: https://github.com/mikehulluk/mreorg
mredoc	DOC: https://mredoc.rtfd.org SRC: https://github.com/mikehulluk/mredoc
TestData Repository	SRC: https://github.com/mikehulluk/simulator-test-data

Table G.2 – Packages written during this thesis.

BIBLIOGRAPHY

Aiken, S. P., Kuenzi, F. M., and Dale, N. (2003). *Xenopus* embryonic spinal neurons recorded *in situ* with patch-clamp electrodes - conditional oscillators after all? *Eur J Neurosci*, 18(2):333–343. (Cited on pages 129, 145, and 186.)

Alexandrescu, A., Meyers, S., and Vlissides, J. (2001). *Modern C++ Design: Applied Generic and Design Patterns (C++ in Depth)*. Addison Wesley. (Cited on page 77.)

Angstadt, J. D., Grassmann, J. L., Theriault, K. M., and Levasseur, S. M. (2005). Mechanisms of postinhibitory rebound and its modulation by serotonin in excitatory swim motor neurons of the medicinal leech. *J Comp Physiol*, 191(8):715–32. (Cited on pages 9 and 128.)

Arbas, E. and Calabrese, R. L. (1987). Slow oscillations of membrane potential in interneurons that control heartbeat in the medicinal leech. *J Neurosci*, 12(7):3953–60. (Cited on page 128.)

Arshavsky, Y. I. (2003). Cellular and network properties in the functioning of the nervous system: from central pattern generators to cognition. *Brain Res Rev*, 41(2-3):229–267. (Cited on page 128.)

Ayers, J., Carpenter, G., Currie, S., and Kinch, J. (1983). Which behavior does the lamprey central motor program mediate? *SCIENCE*, 221:1313–4. (Cited on page 4.)

Barry, M. J. and O'Donovan, M. J. (1987). The effects of excitatory amino acids and their antagonists on the generation of motor activity in the isolated chick spinal cord. *Brain Res*, 433(2):271–6. (Cited on page 10.)

Beck, K. (2002). *Test Driven Development: By Example*. Addison-Wesley Professional. (Cited on pages 62 and 65.)

Belousov, A. B. (2011). The regulation and role of neuronal gap junctions during development. *Communicative & Integrative Biology*, 4(5):579–81. (Cited on page 97.)

Bennett, M. V., Pappas, G. D., Giménez, M., and Nakajima, Y. (1967). Physiology and ultrastructure of electrotonic junctions. IV. Medullary electromotor nuclei in gymnotid fish. *J Physiol*, 30(2):236–300. (Cited on page 96.)

Bennett, M. V. L. (1966). Physiology of electrotonic junctions. *Ann NY Acad Sci*, 137(2):509–539. (Cited on pages 30 and 58.)

Bennett, M. V. L. and Zukin, R. S. (2004). Neuronal synchronization in the mammalian brain. *Neuron*, 41(4):495–511. (Cited on pages 19, 96, and 118.)

Berkowitz, A., Roberts, A., and Soffe, S. R. (2010). Roles for multifunctional and specialized spinal interneurons during motor pattern generation in tadpoles, zebrafish larvae, and turtles. *Front Behav Neurosci*, 4(36):36. (Cited on page 4.)

Berry, M. and Pentreath, V. (1976). Criteria for distinguishing between monosynaptic and polysynaptic transmission. *Brain Research*, 105(1):1 – 20. (Cited on pages 158 and 186.)

Bertrand, S. (1998). Postinhibitory rebound during locomotor-like activity in neonatal rat motoneurons *in vitro*. *J Neurophysiol*, pages 342–351. (Cited on pages 10 and 128.)

Boiko, T., Van Wart, A., Caldwell, J. H., Levinson, S. R., Trimmer, J. S., and Matthews, G. (2003). Functional specialization of the axon initial segment by isoform-specific sodium channel targeting. *J Neurosci*, 23(6):2306–13. (Cited on pages 119 and 120.)

Boothby, K. M. and Roberts, A. (1995). Effects of site of tactile stimulation on the escape swimming responses of hatchling *Xenopus laevis* embryos. *J Zool*, 235(1):113–125. (Cited on pages 152 and 174.)

Borisyuk, R., Al Azad, A. K., Conte, D., Roberts, A., and Soffe, S. R. (2011). Modeling the connectome of a simple spinal cord. *Front Neuroinform*, 5(September):20. (Cited on page 21.)

Borisyuk, R., Cooke, T., and Roberts, A. (2008). Stochasticity and functionality of neural systems: mathematical modelling of axon growth in the spinal cord of tadpole. *Bio Systems*, 93(1-2):101–14. (Cited on pages 21 and 220.)

Bostock, H., Sherratt, R., and Sears, T. (1978). Overcoming conduction failure in demyelinated nerve fibres by prolonging action potentials. *Nature*, 274(July):385–387. (Cited on page 120.)

Bower, J. M. and Beeman, D. (1998). *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System*. Springer. (Cited on page 67.)

Boyan, G. S., Ashman, S., and Ball, E. E. (1986). Initiation and modulation of flight by a single giant interneuron in the cercal system of the locust. *Naturwissenschaften*, 73(5):272–274. (Cited on page 5.)

Boyle, J. H. and Cohen, N. (2008). *Caenorhabditis elegans* body wall muscles are simple actuators. *Bio Systems*, 94(1-2):170–81. (Cited on page 11.)

Boyle, M. B., Cohen, L. B., Macagno, E. R., and Orbach, H. (1983). The number and size of neurons in the CNS of gastropod molluscs and their suitability for optical recording of activity. *Brain Res*, 266(2):305–317. (Cited on page 8.)

Bresadola, M. (1998). Medicine and science in the life of Luigi Galvani (1737-1798). *Brain Res Bull*, 46(5):367–80. (Cited on page 29.)

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., Diesmann, M., Morrison, A., Goodman, P. H., Harris, F. C., Zirpe, M., Natschläger, T., Pecevski, D., Ermentrout, B., Djurfeldt, M., Lansner, A., Rochel, O., Vieville, T., Muller, E., Davison, A. P., El Boustani, S., and Destexhe, A. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *J Comp Neuro*, 23(3):349–98. (Cited on page 67.)

Brooks, F. P. (1995). *The Mythical Man Month and Other Essays on Software Engineering*. Addison Wesley. (Cited on page 63.)

Brown, T. (1914). On the nature of the fundamental activity of the nervous centres; together with an analysis of the conditioning of rhythmic activity in progression, and a theory of the evolution of function in the nervous system. *J Physiol*, 48(1):18. (Cited on pages 4, 6, 9, and 126.)

Brown, T. G. (1911). The intrinsic factors in the act of progression in the mammal. *P Roy Soc B-Biol Sci*, 84(572):308–319. (Cited on pages 4, 6, and 9.)

Bruederle, D., Davison, A., Kremkow, J., Mueller, E., Muller, E., Nawrot, M., Pereira, M., Perrinet, L., Schmuker, M., and Yger, P. (2010). NeuroTools: Analysis, visualization and management of real and simulated neuroscience data. (Cited on page 66.)

Bryden, J. and Cohen, N. (2004). A simulation model of the locomotion controllers for the nematode *Caenorhabditis elegans*. In Schaal, S., J, I. A., S, B. A. V., J, H., and Meyer, J.-A., editors, *From Animals to Animats 8: Proceedings of the Eighth International Conference on the Simulation of Adaptive Behavior*, number July, pages 183–192. MIT Press. (Cited on pages 9 and 11.)

Buchanan, J. T. (2001). Contributions of identifiable neurons and neuron classes to lamprey vertebrate neurobiology. *Prog Neurobiol*, 63(4):441–66. (Cited on page 7.)

Buhl, E., Roberts, A., and Soffe, S. R. (2012). The role of a trigeminal sensory nucleus in the initiation of locomotion. *J Physiol*, 590(Pt 10):2453–2469. (Cited on pages 16, 17, 22, 25, 53, 56, 128, 132, 134, 152, 153, 156, 157, 158, 174, and 185.)

Calabrese, R. and Feldman, J. (1997). Intrinsic membrane properties and synaptic mechanisms in motor rhythm generators. In Stein, P. S. G., Grillner, S., Selverston, A. I., and Stuart, D. G., editors, *Neurons, Networks and Motor Behaviour*, pages 119–130. A Bradford Book. (Cited on page 128.)

Calabrese, R. L. (1998). Cellular, synaptic, network, and modulatory mechanisms involved in rhythm generation. *Curr Opin Neurobiol*, 8(6):710–7. (Cited on page 10.)

Calì, C., Berger, T., Pignatelli, M., Carleton, A., Markram, H., and Giugliano, M. (2008). Inferring connection proximity in networks of electrically coupled cells by subthreshold frequency response analysis. *J Comp Neuro*, 24:330–345. (Cited on page 97.)

Calin-Jageman, R. J., Tunstall, M. J., Mensh, B. D., Katz, P. S., and Frost, W. N. (2007). Parameter space analysis suggests multi-site plasticity contributes to motor pattern initiation in *Tritonia*. *J Neurophysiol*, 98(4):2382–98. (Cited on pages 11 and 126.)

Cannon, R., Gewaltig, M., and Gleeson, P. (2007). Interoperability of neuroscience modeling software: current status and future directions. *Neuroinformatics*, 5(2):127–138. (Cited on page 68.)

Carnevale, N. T. and Hines, M. L. (2006). *The NEURON Book*. Cambridge University Press, Cambridge. (Cited on pages 35, 67, 68, 84, 195, 219, and 225.)

Catterall, W. (1981). Localization of sodium channels in cultured neural cells. *J Neurosci*, 1(7):777–783. (Cited on pages 119 and 120.)

Chalfie, M., Sulston, J., White, J., Southgate, E., Thomson, J., and Brenner, S. (1985). The neural circuit for touch sensitivity in *Caenorhabditis elegans*. *J Neurosci*, 5(4):956–964. (Cited on page 8.)

Clarac, F. and Pearlstein, E. (2007). Invertebrate preparations and their contribution to neurobiology in the second half of the 20th century. *Brain Res Rev*, 54(1):113–161. (Cited on page 4.)

Clarke, J., Hayes, B., Hunt, S., and Roberts, A. (1984). Sensory physiology, anatomy and immunohistochemistry of Rohon-Beard neurones in embryos of *Xenopus laevis*. *J Physiol.*, 384(March):511–525. (Cited on pages 16 and 109.)

Coghill, G. E. (1929). *Anatomy and the problem of behavior*. The University Press. (Cited on page 12.)

Connor, J. and Stevens, C. (1971). Prediction of repetitive firing behaviour from voltage clamp data on an isolated neurone soma. *J Physiol*, 213:31–53. (Cited on pages 34, 43, 46, 144, and 188.)

Connors, B. W. and Long, M. A. (2004). Electrical synapses in the mammalian brain. *Ann Rev Neuro*, 27:393–418. (Cited on pages 19 and 97.)

Cornelis, H., Rodriguez, A. L., Coop, A. D., and Bower, J. M. (2012). Python as a federation tool for GENESIS 3.0. *PLoS One*, 7(1):e29018. (Cited on page 68.)

Crawley, M. J. (2007). *The R Book*. Wiley-Blackwell. (Cited on page 159.)

Crook, S., Gleeson, P., Howell, F., Svitak, J., and Silver, R. A. (2007). MorphML: level 1 of the NeuroML standards for neuronal morphology data and model specification. *Neuroinformatics*, 5(2):96–104. (Cited on pages 71 and 84.)

Crook, S. M., Bednar, J. A., Berger, S., Cannon, R., Davison, A. P., Djurfeldt, M., Eppler, J., Kriener, B., Furber, S., Graham, B., Plesser, H. E., Schwabe, L., Smith, L., Steuber, V., and van Albada, S. (2012). Creating, documenting and sharing network models. *Network*, 23(4):1–19. (Cited on pages 189 and 235.)

Currie, S. N. (1999). Fictive hindlimb motor patterns evoked by AMPA and NMDA in turtle spinal cord-hindlimb nerve preparations. *J Physiology-Paris*, 93(3):199–211. (Cited on page 10.)

Dale, D. (2011). Python quantities. http://pythonhosted.org/quantities/. Accessed: 2013-06-27. (Cited on page 77.)

Dale, N. (1985). Reciprocal inhibitory interneurones in the *Xenopus* embryo spinal cord. *J Physiol*, 363:61–70. (Cited on page 53.)

Dale, N. (1993). A large, sustained Na (+)-and voltage-dependent K+ current in spinal neurons of the frog embryo. *J Physiol*, 462(1):349. (Cited on page 36.)

Dale, N. (1995a). Experimentally derived model for the locomotor pattern generator in the *Xenopus* embryo. *J Physiol*, 489(1995):489–510. (Cited on pages 12, 19, 23, 37, 47, 49, 50, 122, 128, 129, 142, 143, and 196.)

Dale, N. (1995b). Kinetic characterization of the voltage-gated currents possessed by *Xenopus* embryo spinal neurons. *J Physiol*, 489(2):473–88. (Cited on pages 23, 36, 37, 47, 49, 50, 51, 59, 107, and 121.)

Dale, N. (2003). Coordinated motor activity in simulated spinal networks emerges from simple biologically plausible rules of connectivity. *J Comp Neuro*, 14(1):55–70. (Cited on pages 122 and 128.)

Dale, N., Ottersen, O., and Roberts, A. (1986). Inhibitory neurones of a motor pattern generator in *Xenopus* revealed by antibodies to glycine. *Nature*. (Cited on page 53.)

Dale, N. and Roberts, A. (1985). Dual-component amino-acid-mediated synaptic potentials: excitatory drive for swimming in *Xenopus* embryos. *J Physiol*, 363(1):35. (Cited on pages 10, 18, 53, and 133.)

Davison, A. (2012). Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science & Engineering*, 14(4):48–56. (Cited on pages 65, 72, 73, and 235.)

Davison, A. P., Hines, M. L., and Muller, E. (2009). Trends in programming languages for neuroscience simulations. *Front Neurosci*, 3(3):374–80. (Cited on page 68.)

de Rijk, M. C., Tzourio, C., Breteler, M. M., Dartigues, J. F., Amaducci, L., Lopez-Pousa, S., Manubens-Bertran, J. M., Alpérovitch, A., and Rocca, W. A. (1997). Prevalence of parkinsonism and Parkinson's disease in Europe: the EUROPARKINSON collaborative study. *J Neurol Neurosur Psychiatry*, 62(1):10–5. (Cited on page 150.)

Delcomyn, F. (1980). Neural basis of rhythmic behavior in animals. *Science*, 210(4469):492. (Cited on page 4.)

Delcomyn, F. (1998). *Foundations of Neurobiology*. W. H. Freeman. (Cited on pages 30 and 53.)

des Poids et Mesures, B. I. (2006). *International System of Units (SI)*. Bureau International des Poids et Mesures. (Cited on page 208.)

Djurfeldt, M. (2012). The connection-set algebra–a novel formalism for the representation of connectivity structure in neuronal network models. *Neuroinformatics*, 10(3):287–304. (Cited on page 70.)

Domenici, P., Blagburn, J. M., and Bacon, J. P. (2011). Animal escapology I: theoretical issues and emerging trends in escape trajectories. *J Exp Biol*, 214(Pt 15):2463–73. (Cited on page 152.)

Draguhn, A., Traub, R. D., Schmitz, D., and Jefferys, J. G. (1998). Electrical coupling underlies high-frequency oscillations in the hippocampus *in vitro*. *Nature*, 394(6689):189–92. (Cited on page 97.)

Dubuc, R., Brocard, F., Antri, M., Fénelon, K., Gariépy, J.-F., Smetana, R., Ménard, A., Le Ray, D., Viana Di Prisco, G., Pearlstein, E., Sirota, M. G., Derjean, D., St-Pierre, M., Zielinski, B., Auclair, F., and Veilleux, D. (2008). Initiation of locomotion in lampreys. *Brain Res Rev*, 57(1):172–82. (Cited on page 150.)

Edwards, D., Yeh, S.-R., and Krasne, F. B. (1998). Neuronal coincidence detection by voltage-sensitive electrical synapses. *Proc Natl Acad Sci*, 95(June):7145–7150. (Cited on page 96.)

Edwards, D. H., Heitler, W. J., and Krasne, F. B. (1999). Fifty years of a command neuron: the neurobiology of escape behavior in the crayfish. *Trends Neurosci*, 22(4):153–61. (Cited on page 154.)

Eisen, J. and Marder, E. (1982). Mechanisms underlying pattern generation in lobster stomatogastric ganglion as determined by selective inactivation of identified neurons. III. Synaptic connections of electrically coupled pyloric neurons. *J Neurophysiol*, 48:1392–1415. (Cited on page 96.)

Ekeberg, O. (1994). An integrated neuronal and mechanical model of fish swimming. In Eeckman, F. H., editor, *Computation in Neurons and Neural Systems*, pages 217–222. Springer, Washington, DC. (Cited on page 11.)

Eppler, J. M., Helias, M., Muller, E., Diesmann, M., and Gewaltig, M.-O. (2008). PyNEST: A convenient interface to the NEST simulator. *Front Neuroinform*, 2(January):12. (Cited on page 68.)

Faber, J. and Nieuwkoop, P. D. (1956). *Normal table of Xenopus laevis (Daudin)*. Amsterdam, North-Holland. (Cited on page 13.)

Faisal, A. A. and Laughlin, S. B. (2007). Stochastic simulations on the reliability of action potential propagation in thin axons. *PLoS Comput Biol*, 3(5):783–795. (Cited on pages 105 and 119.)

Faisal, A. A., White, J. A., and Laughlin, S. B. (2005). Ion-channel noise places limits on the miniaturization of the brain's wiring. *Curr Biol*, 15(12):1143–9. (Cited on page 119.)

Feldman, J. L., Mitchell, G. S., and Nattie, E. E. (2003). Breathing: rhythmicity, plasticity, chemosensitivity. *Ann Rev Neuro*, 26:239–66. (Cited on page 126.)

Fetcho, J. R., Higashijima, S.-i., and McLean, D. L. (2008). Zebrafish and motor control over the last decade. *Brain Res Rev*, 57(1):86–93. (Cited on page 7.)

Fetcho, J. R. and Liu, K. S. (1998). Zebrafish as a model system for studying neuronal circuits and behavior. *Ann NY Acad Sci*, 860(1):333–345. (Cited on page 7.)

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code (Object Technology Series)*. Addison Wesley. (Cited on page 65.)

Friesen, W. and Kristan, W. (2007). Leech locomotion: swimming, crawling, and decisions. *Curr Opin Neurobiol*, 17(6):704–711. (Cited on page 8.)

Frost, W., Hoppe, T., Wang, J., and Tian, L. (2001). Swim initiation neurons in *Tritonia diomedea*. *Am Zool*, 41(4):952. (Cited on page 8.)

Fukuda, T. and Kosaka, T. (2000). Gap junctions linking the dendritic network of GABAergic interneurons in the hippocampus. *J Neurosci*, 20(4):1519–28. (Cited on page 97.)

Furshpan, E. J. and Potter, D. D. (1959). Transmission at the giant motor synapses of the crayfish. *J Physiol.*, 145:289–325. (Cited on pages 19, 96, and 117.)

Gabriel, J. P., Ausborn, J., Ampatzis, K., Mahmood, R., Eklöf-Ljunggren, E., and El Manira, A. (2010). Principles governing recruitment of motoneurons during swimming in zebrafish. *Nat Neuro*, 14(1):93–99. (Cited on page 10.)

Galarreta, M. and Hestrin, S. (2002). Electrical and chemical synapses among parvalbumin fast-spiking GABAergic interneurons in adult mouse neocortex. *P Natl Acad Sci USA*, 99(19):12438. (Cited on page 97.)

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns : elements of reusable object-oriented software*. Addison Wesley. (Cited on pages 70, 216, and 222.)

Getting, P. a. (1983). Mechanisms of pattern generation underlying swimming in *Tritonia*. II. Network reconstruction. *J Neurophysiol*, 49(4):1017–35. (Cited on pages 11 and 126.)

Gewaltig, M.-O. and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia*, 2(4):1430. (Cited on page 67.)

Gibson, J. R., Beierlein, M., and Connors, B. W. (1999). Two networks of electrically coupled inhibitory neurons in neocortex. *Nature*, 402(6757):75–9. (Cited on page 97.)

Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L., and Myers, J. (2007). Examining the challenges of scientific workflows. *Computer*, 40(12):24–32. (Cited on page 64.)

Gleeson, P., Crook, S., Cannon, R. C., Hines, M. L., Billings, G. O., Farinella, M., Morse, T. M., Davison, A. P., Ray, S., Bhalla, U. S., Barnes, S. R., Dimitrova, Y. D., and Silver, R. A. (2010). NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. *PloS Comput Biol*, 6(6):e1000815. (Cited on page 69.)

Goddard, N. H., Hucka, M., Howell, F., Cornelis, H., Shankar, K., and Beeman, D. (2001). Towards NeuroML: model description methods for collaborative modelling in neuroscience. *Philos T R Soc B*, 356(1412):1209–28. (Cited on page 69.)

Gold, C., Henze, D. a., Koch, C., and Buzsáki, G. (2006). On the origin of the extracellular action potential waveform: A modeling study. *J Neurophysiol*, 95(5):3113–28. (Cited on page 190.)

Goodman, D. and Brette, R. (2008). Brian: a simulator for spiking neural networks in python. *Front Neuroinform*, 2(November):5. (Cited on pages 67 and 77.)

Goulding, M. (2009). Circuits controlling vertebrate locomotion: moving in a new direction. *Nat Rev Neuro*, 10(7):507–518. (Cited on page 126.)

Grillner, S. (1975). Locomotion in vertebrates: central mechanisms and reflex interaction. *Physiol Rev*, 55(2):247–304. (Cited on page 4.)

Grillner, S. (1999). Bridging the gap–from ion channels to networks and behaviour. *Curr Opin Neurobiol*, pages 663–669. (Cited on page 128.)

Grillner, S., McClellan, A., Sigvardt, K., Wallén, P., and Wilén, M. (1981). Activation of NMDA-receptors elicits "fictive locomotion" in lamprey spinal cord *in vitro*. *Acta Physiol Scand*, 113(4):549–51. (Cited on page 10.)

Grillner, S. and Wallen, P. (1985). Central pattern generators for locomotion, with special reference to vertebrates. *Ann Rev Neuro*, 8(1):233–261. (Cited on page 4.)

Gronenschild, E. H. B. M., Habets, P., Jacobs, H. I. L., Mengelers, R., Rozendaal, N., van Os, J., and Marcelis, M. (2012). The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements. *PLoS One*, 7(6):e38234. (Cited on page 63.)

Groth, P. T. and Gil, Y. (2008). A scientific workflow construction command line. In *Proceedings of the 13th International Conference on Intelligent User Interfaces - IUI '09*, page 445, New York, New York, USA. ACM Press. (Cited on page 65.)

Grubb, M. S. and Burrone, J. (2010). Building and maintaining the axon initial segment. *Curr Opin Neurobiol*, 20(4):481–488. (Cited on page 108.)

Grundfest, H. (1968). Invertebrate nervous systems: their significance for mammalian neurophysiology. *Arch Neurol-Chicago*, 18(2):222–222. (Cited on page 9.)

Grune, D., Bal, H., Jacobs, C., and Langendoen, K. (2000). *Modern Compiler Design*. Wiley. (Cited on page 200.)

Halliday, D., Resnick, R., and Walker, J. (2004). *Fundamentals of Physics*. John Wiley & Sons. (Cited on page 33.)

Hille, B. (2001). *Ion channels of excitable membranes*, volume 3rd. Sinauer Associates. (Cited on pages 9, 23, 29, 32, 34, 35, 43, 46, 50, 107, 109, 144, 186, and 188.)

Hillis, W. D. (1993). Why physicists like models and why biologists should. *Curr Biol*, 3(2):79–81. (Cited on pages 181 and 182.)

Hindmarsh, A., Brown, P., and Grant, K. (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM T Math Software*, 31(3):363–396. (Cited on pages 67 and 232.)

Hines, M. (1984). Efficient computation of branched nerve equations. *Int J Biomed Comput*, 15(1):69–76. (Cited on page 67.)

Hines, M. L. and Carnevale, N. T. (2000). Expanding NEURON's Repertoire of Mechanisms with NMODL. *Neural Comput*, 12(5):995–1007. (Cited on page 68.)

Hines, M. L., Davison, A. P., and Muller, E. (2009). NEURON and Python. *Front Neuroinform*, 3:1. (Cited on page 68.)

Hnasko, T. S. and Edwards, R. H. (2012). Neurotransmitter corelease: mechanism and physiological role. *Annu Rev Physiol*, 74:225–43. (Cited on page 53.)

Hoare, C. A. R. (1961). Algorithm 64: Quicksort. *Commun ACM*, 4(7):321. (Cited on page 224.)

Hodgkin, A. and Huxley, A. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J Physiol*, 117:500–544. (Cited on pages 29, 30, 32, 59, and 139.)

Hoffman, N. and Parker, D. (2010). Lesioning alters functional properties in isolated spinal cord hemisegmental networks. *Neuroscience*, 168(3):732–43. (Cited on pages 10, 145, and 187.)

Hoge, G. J., Davidson, K. G. V., Yasumura, T., Castillo, P. E., Rash, J. E., and Pereda, A. E. (2011). The extent and strength of electrical coupling between inferior olivary neurons is heterogeneous. *J Neurophysiol*, 105(3):1089–101. (Cited on page 97.)

Hormuzdi, S. G., Filippov, M. a., Mitropoulou, G., Monyer, H., and Bruzzone, R. (2004). Electrical synapses: a dynamic signaling system that shapes the activity of neuronal networks. *Biochim Biophys Acta*, 1662(1-2):113–37. (Cited on page 97.)

Hu, H., Ma, Y., and Agmon, A. (2011). Precise Synchrony by Mutual Inhibition: Experiments Vindicate Theory. *Molecular and Cellular Pharmacology*, 3(2):59–65. (Cited on page 176.)

Hudson, R., Norris, J., and Reid, L. (2011). Data-intensive management and analysis for scientific simulations. In *Ninth Australasian Symposium on Parallel and Distributed Computing*, volume 118. (Cited on page 64.)

Humphries, D. A. and Driver, P. M. (1970). Protean defence by prey animals. *Oecologia*, 5(4):285–302. (Cited on page 152.)

Hunt, A. and Thomas, D. (1999). *The Pragmatic Programmer*. Addison Wesley. (Cited on page 65.)

Ijspeert, A. (2001). A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biol Cybern*, 84(5):331–48. (Cited on page 126.)

Izhikevich, E. (2007). *Dynamical systems in neuroscience: the geometry of excitability and bursting*. The MIT press. (Cited on page 35.)

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE T Neural Networ*, 14(6):1569–72. (Cited on page 35.)

Jamieson, D. and Roberts, a. (2000). Responses of young *Xenopus laevis* tadpoles to light dimming: possible roles for the pineal eye. *J Exp Biol*, 203(Pt 12):1857–67. (Cited on page 13.)

Jordan, L. M. (1998). Initiation of locomotion in mammals. *Ann NY Acad Sci*, 860:83–93. (Cited on pages 6 and 150.)

Jordan, L. M., Liu, J., Hedlund, P. B., Akay, T., and Pearson, K. G. (2008). Descending command systems for the initiation of locomotion in mammals. *Brain Res Rev*, 57(1):183–91. (Cited on page 150.)

Juszczak, G. R. and Swiergiel, A. H. (2009). Properties of gap junction blockers and their behavioural, cognitive and electrophysiological effects: Animal and human studies. *Progress in Neuro-Psychopharmacology and Biological Psychiatry*, 33(2):181 – 198. (Cited on page 119.)

Kahn, J. and Roberts, A. (1982a). The central nervous origin of the swimming motor pattern in embryos of *Xenopus laevis*. *J. Exp. Biol.*, 99(1):185–196. (Cited on page 176.)

Kahn, J. and Roberts, A. (1982b). The neuromuscular basis of rhythmic struggling movements in embryos of *Xenopus laevis*. *J Exp Biol*, 99(1):197. (Cited on page 13.)

Kahn, J., Roberts, A., and Kashin, S. (1982). The neuromuscular basis of swimming movements in embryos of the amphibian *Xenopus laevis*. *J Exp Biol*, 99(1):175. (Cited on pages 13, 14, 15, and 127.)

Karlsson, B. (2005). *Beyond the C++ Standard Library: An Introduction to Boost*. Addison Wesley. (Cited on page 77.)

Kernighan, B. W. and Ritchie, D. (1988). *The C Programming Language (2nd Edition)*. Prentice Hall. (Cited on pages 65, 70, 208, and 237.)

Kiehn, O. and Tresch, M. C. (2002). Gap junctions and motor behavior. *Trends Neurosci*, 25(2):108–15. (Cited on pages 117 and 154.)

Knudsen, D., Arsenault, J., Hill, S., Omalley, D., and Jose, J. (2006). Locomotor network modeling based on identified zebrafish neurons. *Neurocomputing*, 69(10-12):1169–1174. (Cited on page 12.)

Knuth, D. E. (1992). *Literate Programming (Center for the Study of Language and Information Publication Lecture Notes)*. Cambridge University Press. (Cited on page 66.)

Koch, C. (1999). *Biophysics of Computation: Information Processing in Single Neurons*. Oxford University Press, New York. (Cited on pages 11, 32, 35, 50, 56, 59, 100, 136, 195, 197, and 209.)

Kole, M. H. P., Ilschner, S. U., Kampa, B. M., Williams, S. R., Ruben, P. C., and Stuart, G. J. (2008). Action potential generation requires a high sodium channel density in the axon initial segment. *Nat Neurosci*, 11(2):178–86. (Cited on page 108.)

Korn, H. and Faber, D. S. (2005). The Mauthner cell half a century later: a neurobiological model for decision-making? *Neuron*, 47(1):13–28. (Cited on page 5.)

Krasne, F. B. (1969). Excitation and habituation of the crayfish escape reflex: the depolarizing response in lateral giant fibres of the isolated abdomen. *J Exp Biol*, 50(1):29–46. (Cited on page 5.)

Kress, G. J., Dowling, M. J., Meeks, J. P., and Mennerick, S. (2008). High threshold, proximal initiation, and slow conduction velocity of action potentials in dentate granule neuron mossy fibers. *J Neurophysiol*, 100(1):281–91. (Cited on pages 108, 119, and 120.)

Kristan, W. B., Calabrese, R. L., and Friesen, W. O. (2005). Neuronal control of leech behavior. *Prog Neurobiol*, 76(5):279–327. (Cited on pages 8 and 126.)

Krug, E. and Kresnow, M. (1999). Retraction: Suicide after natural disasters. *New Engl J Med*, 340(2):148–149. (Cited on page 63.)

Kyriakatos, A., Mahmood, R., Ausborn, J., Porres, C. P., Büschges, A., and El Manira, A. (2011). Initiation of locomotion in adult zebrafish. *J Neurosci*, 31(23):8422–31. (Cited on page 7.)

Lamb, D. G. and Calabrese, R. L. (2011). Neural circuits controlling behavior and autonomic functions in medicinal leeches. *Neural Systems & Circuits*, 1(1):13. (Cited on page 8.)

Lambert, T. D. (2004). Mechanisms and significance of reduced activity and responsiveness in resting frog tadpoles. *J Exp Biol*, 207(7):1113–1125. (Cited on pages 13, 21, and 171.)

Langer, J., Stephan, J., Theis, M., and Rose, C. (2012). Gap junctions mediate intercellular spread of sodium between hippocampal astrocytes *in situ*. *Glia*, 252(March 2011):239–252. (Cited on page 117.)

Leifer, A. M., Fang-Yen, C., Gershow, M., Alkema, M. J., and Samuel, A. D. T. (2011). Optogenetic manipulation of neural activity in freely moving *Caenorhabditis elegans*. *Nat Methods*, 8(2):147–52. (Cited on page 9.)

Leisch, F. (2002). Sweave: dynamic generation of statistical reports using literate data analysis. In *Compstat 2002 Proceedings in Computational Statistics*, volume CompStat 2, pages 575–580. Physica Verlag, Heidelberg. (Cited on page 66.)

Li, W. and Moult, P. (2012). The control of locomotor frequency by excitation and inhibition. *J Neurosci*. (Cited on page 188.)

Li, W.-C., Cooke, T., Sautois, B., Soffe, S., Borisyuk, R., and Roberts, A. (2007a). Axon and dendrite geography predict the specificity of synaptic connections in a functioning spinal cord network. *Neural Dev*, 2:17. (Cited on pages 20, 21, and 163.)

Li, W.-C., Perrins, R., Soffe, S., Yoshida, M., Walford, A., and Roberts, A. (2001). Defining classes of spinal interneuron and their axonal projections in hatchling *Xenopus laevis* tadpoles. *J Comp Neurol*, 441:248–265. (Cited on pages 100 and 154.)

Li, W.-C., Roberts, A., and Soffe, S. R. (2009). Locomotor rhythm maintenance: electrical coupling among premotor excitatory interneurons in the brainstem and

spinal cord of young *Xenopus* tadpoles. *J Physiol*, 587(Pt 8):1677–93. (Cited on pages 19, 23, 37, 57, 97, 98, 100, 101, 118, 119, 154, 172, 177, 185, 187, and 188.)

Li, W.-C., Roberts, A., and Soffe, S. R. (2010). Specific brainstem neurons switch each other into pacemaker mode to drive movement by activating NMDA receptors. *J Neurosci*, 30(49):16609–20. (Cited on pages 18, 19, 98, 107, 121, 122, 128, 130, 132, 133, 134, 139, 142, 144, 145, 147, 151, 172, 173, 186, and 188.)

Li, W.-C., Sautois, B., Roberts, A., and Soffe, S. R. (2007b). Reconfiguration of a vertebrate motor network: specific neuron recruitment and context-dependent synaptic plasticity. *J Neurosci*, 27(45):12267–76. (Cited on pages 16, 19, 20, 53, and 145.)

Li, W.-C., Soffe, S. R., and Roberts, A. (2003). The spinal interneurons and properties of glutamatergic synapses in a primitive vertebrate cutaneous flexion reflex. *J Neurosci*, 23(27):9068–77. (Cited on pages 16, 121, and 156.)

Li, W.-C., Soffe, S. R., and Roberts, A. (2004a). A direct comparison of whole cell patch and sharp electrodes by simultaneous recording from single spinal neurons in frog tadpoles. *Journal of Neurophysiology*, 92(1):380–386. (Cited on pages 122 and 154.)

Li, W.-C., Soffe, S. R., and Roberts, A. (2004b). Dorsal spinal interneurons forming a primitive, cutaneous sensory pathway. *J Neurophysiol*, 92(2):895–904. (Cited on pages 15, 16, 53, 146, and 176.)

Li, W.-C., Soffe, S. R., Wolf, E., and Roberts, A. (2006). Persistent responses to brief stimuli: feedback excitation among brainstem neurons. *J Neurosci*, 26(15):4026–35. (Cited on pages 17, 18, 98, 100, 107, 109, 120, 121, 127, 128, 129, 132, 133, 142, 146, 152, 153, and 186.)

Lin, J. W. and Faber, D. S. (1988). Synaptic transmission mediated by single club endings on the goldfish Mauthner cell. I. Characteristics of electrotonic and chemical postsynaptic potentials. *J Neurosci*, 8(4):1302–12. (Cited on pages 19 and 96.)

Liu, M.-G., Chen, X.-F., He, T., Li, Z., and Chen, J. (2012). Use of multi-electrode array recordings in studies of network synaptic plasticity in both time and space. *Neuroscience bulletin*, 28(4):409–22. (Cited on page 190.)

Llinas, R., Baker, R., and Sotelo, C. (1974). Electrotonic coupling between neurons in cat inferior olive. *J Neurophysiol*. (Cited on page 97.)

Lockery, S., Goodman, M., and Faumont, S. (2009). First report of action potentials in a C. elegans neuron is premature. *Nat Neuro*, 12(4):365–366. (Cited on page 9.)

López-Muñoz, F. and Alamo, C. (2009). Historical evolution of the neurotransmission concept. *J Neural Transm (Vienna, Austria : 1996)*, 116(5):515–33. (Cited on pages 29 and 30.)

Luthi, A. (1998). H-current: properties of a neuronal and network pacemaker. *Neuron*, 21:9–12. (Cited on page 34.)

Maex, R. and De Schutter, E. (2007). Mechanism of spontaneous and self-sustained oscillations in networks connected through axo-axonal gap junctions. *Eur J Neurosci*, 25(11):3347–58. (Cited on page 97.)

Mann-Metzer, P. and Yarom, Y. (1999). Electrotonic coupling interacts with intrinsic properties to generate synchronized activity in cerebellar networks of inhibitory interneurons. *J Neurosci*, 19(9):3298–306. (Cited on page 97.)

Marder, E. (1998). Electrical synapses: beyond speed and synchrony to computation. *Curr Biol*, 8(22):R795–7. (Cited on pages 19 and 117.)

Marder, E. (2000). Motor pattern generation. *Curr Opin Neurobiol*, 10(6):691–8. (Cited on page 4.)

Marder, E. and Bucher, D. (2007). Understanding circuit dynamics using the stomatogastric nervous system of lobsters and crabs. *Annu. Rev. Physiol.*, 69:291–316. (Cited on pages 18, 96, and 126.)

Marder, E. and Calabrese, R. L. (1996). Principles of rhythmic motor pattern generation. *Physiol Rev*, 76(3):687–717. (Cited on page 128.)

Marder, E., Tobin, A.-E., and Grashow, R. (2007). How tightly tuned are network parameters? Insight from computational and experimental studies in small rhythmic motor networks. *Prog Brain Res*, 165:193–200. (Cited on pages 12 and 182.)

Matsushima, T., Tegnér, J., Hill, R. H., and Grillner, S. (1993). GABA-B receptor activation causes a depression of low- and high-voltage activated Ca2+ currents, postinhibitory rebound, and postspike afterhyperpolarization in lamprey neurons. *J Neurophysiol*, 70(6):2606–19. (Cited on page 10.)

McCabe, T. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*, SE-2(4):308–320. (Cited on page 63.)

McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press. (Cited on pages 63, 90, 92, and 222.)

McLean, D. L., Fan, J., Higashijima, S.-i., Hale, M. E., and Fetcho, J. R. (2007). A topographic map of recruitment in spinal cord. *Nature*, 446(7131):71–5. (Cited on page 7.)

Mecklenburg, R. (2004). *Managing Projects with GNU Make (Nutshell Handbooks)*. O'Reilly Media. (Cited on page 65.)

Merali, Z. (2010). Why scientific programming does not compute. *Nature*, pages 6–8. (Cited on page 61.)

Mercer, A., Bannister, A. P., and Thomson, A. M. (2006). Electrical coupling between pyramidal cells in adult cortical regions. *Brain Cell Biol*, 35(1):13–27. (Cited on page 97.)

Miller, G. (2006). Scientific publishing. A scientist's nightmare: software problem leads to five retractions. *Science*, 314(5807):1856–7. (Cited on page 63.)

Mills, S. and Massey, S. (1998). The kinetics of tracer movement through homologous gap junctions in the rabbit retina. *Vis Neurosci*, 15(4):765–777. (Cited on page 97.)

Molkov, Y. I., Loskutov, E. M., Mukhin, D. N., and Feigin, A. M. (2012). Random dynamical models from time series. *Phys Rev E*, 85(3):036216. (Cited on page 190.)

Morris, C. and Lecar, H. (1981). Voltage oscillations in the barnacle giant muscle fiber. *Biophys J*, 35(1):193–213. (Cited on pages 21 and 35.)

Moult, P. R., Cottrell, G. A., and Li, W.-C. (2013). Fast silencing reveals a lost role for reciprocal inhibition in locomotion. *Neuron*, 77(1):129–40. (Cited on pages 145, 187, and 188.)

Mullins, O., Hackett, J., and Buchanan, J. (2010). Neuronal control of swimming behavior: comparison of vertebrate and invertebrate model systems. *Prog Neurobiol*, 93(2):244–269. (Cited on page 7.)

Nakamura, Y. and Nobuo, K. (1995). Generation of masticatory rhythm in the brainstem. *Neurosci Res*, 23:1–19. (Cited on page 126.)

Noble, D. (1962). A modification of the Hodgkin-Huxley equations applicable to Purkinje fibre action and pacemaker potentials. *J Physiol*, pages 317–352. (Cited on page 9.)

Noble, D. (2004). Modeling the heart. *Physiology*, 19:191–7. (Cited on page 34.)

Nordlie, E., Gewaltig, M.-O., and Plesser, H. E. (2009). Towards reproducible descriptions of neuronal network models. *PloS Comput Biol*, 5(8). (Cited on pages 66 and 235.)

Nordlie, E. and Plesser, H. E. (2010). Visualizing neuronal network connectivity with connectivity pattern tables. *Front Neuroinform*, 3(January):39. (Cited on pages 66 and 235.)

Nowak, L., Bregestovski, P., Ascher, P., Herbet, A., and Prochiantz, A. (1984). Magnesium gates glutamate-activated channels in mouse central neurones. *Nature*, 307(5950):462–465. (Cited on pages 55 and 130.)

Olague, H. and Etzkorn, L. (2008). An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: a case study. *Journal of Software*, 20(February):171–197. (Cited on page 64.)

Oliphant, T. E. (2007). Python for Scientific Computing. *Computing in Science & Engineering*, 9(3):10–20. (Cited on page 68.)

Orlovsky, G., Deliagina, T. G., and Grillner, S. (1999). *Neuronal Control of Locomotion: From Mollusc to Man*. Oxford University Press, Oxford. (Cited on pages 6, 10, and 150.)

Pan, F., Mills, S. L., and Massey, S. C. (2007). Screening of gap junction antagonists on dye coupling in the rabbit retina. *Vis Neurosci*, 24(4):609–18. (Cited on page 97.)

Pangratz-Fuehrer, S. and Hestrin, S. (2011). Synaptogenesis of electrical and GABAergic synapses of fast-spiking inhibitory neurons in the neocortex. *J Neurosci*, 31(30):10767–75. (Cited on page 97.)

Pappas, G. D. G. and Bennett, M. M. V. L. (1966). Specialized junctions involved in electrical transmission between neurons. *Ann NY Acad Sci*, 137(2):495–508. (Cited on page 57.)

Parker, D. (2006). Complexities and uncertainties of neuronal network function. *Philos T R Soc B*, 361(1465):81–99. (Cited on pages 181 and 182.)

Parker, D. (2009). Exciting times in the tadpole spinal cord. *J Physiol*, 587(Pt 8):1635. (Cited on page 21.)

Parker, D. (2010). Neuronal network analyses: premises, promises and uncertainties. *Philos T R Soc B*, 365(1551):2315–28. (Cited on pages 158, 182, and 186.)

Parnin, C. and Rugaber, S. (2012). Programmer information needs after memory failure. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 123–132. (Cited on page 63.)

Perge, J. A., Niven, J. E., Mugnaini, E., Balasubramanian, V., and Sterling, P. (2012). Why do axons differ in caliber? *J Neurosci*, 32(2):626–38. (Cited on page 119.)

Perkel, D. and Mulloney, B. (1974). Motor pattern production in reciprocally inhibitory neurons exhibiting postinhibitory rebound. *Science*, 185(4146):181. (Cited on pages 11 and 128.)

Perrins, R. and Roberts, A. (1995). Cholinergic and electrical synapses between synergistic spinal motoneurones in the *Xenopus laevis* embryo. *J Physiol*, 485 (Pt 1:135–44. (Cited on pages 19 and 53.)

Perrins, R., Walford, A., and Roberts, A. (2002). Sensory activation and role of inhibitory reticulospinal neurons that stop swimming in hatchling frog tadpoles. *J Neurosci*, 22(10):4229–40. (Cited on pages 53, 56, 128, 136, 137, and 138.)

Peterson, B. and Healy, M. (1996). ModelDB: an environment for running and storing computational models and their results applied to neuroscience. *J Am Med Inform Assn*, 3(6):389–98. (Cited on page 68.)

Petty, G. W. (2001). Automated computation and consistency checking of physical dimensions and units in scientific programs. *Software: Practice and Experience*, 31(11):1067–1076. (Cited on page 77.)

Pirtle, T. J. and Satterlie, R. a. (2007). The role of postinhibitory rebound in the locomotor central-pattern generator of *Clione limacina*. *Integr Comp Biol*, 47(4):451–6. (Cited on pages 9 and 10.)

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing*. Foundation Books, 3rd editio edition. (Cited on pages 62 and 224.)

Prinz, A. a. (2010). Computational approaches to neuronal network analysis. *Philos T R Soc B*, 365(1551):2397–405. (Cited on page 12.)

Qu, Y. and Dahl, G. (2002). Function of the voltage gate of gap junction channels: selective exclusion of molecules. *P Natl Acad Sci USA*, 99(2):697–702. (Cited on page 56.)

Ramón y Cajal, S. (1909). *Histologie du système nerveux de l'homme & des vertébrés.* Maloine, Paris, ed. França edition. (Cited on page 29.)

Ray, S. and Bhalla, U. S. (2008). PyMOOSE: interoperable scripting in Python for MOOSE. *Front Neuroinform*, 2(December):6. (Cited on page 68.)

Ritchie, J. M. and Rogart, R. B. (1977). Density of sodium channels in mammalian myelinated nerve fibers and nature of the axonal membrane under the myelin sheath. *Proc Natl Acad Sci*, 74(1):211–5. (Cited on page 120.)

Roberts, A. (1980). The function and role of two types of mechanoreceptive 'free' nerve endings in the head skin of amphibian embryos. *J Comp Physiol*, 135(4):341–348. (Cited on pages 14 and 16.)

Roberts, A. (2000). Early functional organization of spinal neurons in developing lower vertebrates. *Brain Res Bull*, 53(5):585–93. (Cited on page 15.)

Roberts, A. and Alford, S. T. (1986). Descending projections and excitation during fictive swimming in *Xenopus* embryos: neuroanatomy and lesion experiments. *J Comp Neurol*, 250(2):253–61. (Cited on page 18.)

Roberts, A., Dale, N., Ottersen, O., and Storm-Mathisen, J. (1988). Development and characterization of commissural interneurones in the spinal cord of *Xenopus laevis* embryos revealed by antibodies to glycine. *Development*, 103(3):447–461. (Cited on pages 18 and 53.)

Roberts, A., Dale, N., Ottersen, O. P., and Storm-Mathisen, J. (1987). The early development of neurons with GABA immunoreactivity in the CNS of *Xenopus laevis* embryos. *J Comp Neurol*, 261(3):435–49. (Cited on page 53.)

Roberts, A., Li, W.-C., and Soffe, S. (2010). How neurons generate behavior in a hatchling amphibian tadpole: an outline. *Front Behav Neurosci*, 4(June):Article 16 (1–11). (Cited on pages 4, 7, 10, 13, 15, 16, 18, 98, 107, 128, 132, 145, 147, 151, 152, and 163.)

Roberts, A., Li, W.-C., Soffe, S. R., and Wolf, E. (2008). Origin of excitatory drive to a spinal locomotor network. *Brain Res Rev*, 57(1):22–8. (Cited on pages 10, 18, 53, 128, and 145.)

Roberts, A. and Tunstall, M. J. (1990). Mutual re-excitation with post-inhibitory rebound: a simulation study on the mechanisms for locomotor rhythm generation in the spinal cord of *Xenopus* embryos. *Eur J Neurosci*, 2(1):11–23. (Cited on pages 19, 20, 122, 128, and 147.)

Roberts, A., Walford, A., Soffe, S. R., and Yoshida, M. (1999). Motoneurons of the axial swimming muscles in hatchling *Xenopus* tadpoles: features, distribution, and central synapses. *J Comp Neurol*, 411(3):472–86. (Cited on page 14.)

Saffrey, P. and Orton, R. (2009). Version control of pathway models using XML patches. *BMC Syst Biol*, 3:34. (Cited on page 69.)

Safronov, B. V., Wolff, M., and Vogel, W. (1997). Functional distribution of three types of Na+ channel on soma and processes of dorsal horn neurones of rat spinal cord. *J Physiol*, 503 (Pt 2(1997):371–85. (Cited on pages 119 and 120.)

Sakmann, B. and Neher, E. (2010). *Single-Channel Recording*. Springer. (Cited on page 29.)

Sanes, D. H., Reh, T. A., and Harris, W. A. (2006). *Development of the nervous system*. Neuroscience Textbook Set. Elsevier Academic Press. (Cited on pages 12 and 20.)

Sattelle, D. B. and Buckingham, S. D. (2006). Invertebrate studies and their ongoing contributions to neuroscience. *Invert Neurosci*, 6(1):1–3. (Cited on page 7.)

Satterlie, R. (1985). Reciprocal inhibition and postinhibitory rebound produce reverberation in a locomotor pattern generator. *Science*, 229(4711):402–404. (Cited on pages 9 and 128.)

Sautois, B., Soffe, S. R., and Roberts, A. (2007). Role of type-specific neuron properties in a spinal cord motor network. *J Comp Neuro*, pages 59–77. (Cited on pages 12, 19, 20, 23, 53, 55, 56, 59, 98, 109, 112, 120, 121, 122, 126, 128, 130, 133, 144, 146, 147, 154, 173, and 197.)

Schmidt-Hieber, C. and Bischofberger, J. (2010). Fast sodium channel gating supports localized and efficient axonal action potential initiation. *J Neurosci*, 30(30):10233–42. (Cited on page 34.)

Schmidt-Hieber, C., Jonas, P., and Bischofberger, J. (2008). Action potential initiation and propagation in hippocampal mossy fibre axons. *J Physiol*, 586(7):1849–57. (Cited on pages 71, 119, and 226.)

Selverston, A. I. (1980). Are central pattern generators understandable? *Behav Brain Sci*, 3(04):535. (Cited on pages 181 and 182.)

Selverston, A. I. (2010). Invertebrate central pattern generator circuits. *Philos T R Soc B*, 365(1551):2329–45. (Cited on pages 4 and 126.)

Shik, M. L., Severin, F. V., and Orlovskii, G. N. (1966). Control of walking and running by means of electric stimulation of the midbrain. *Biofizika+*, 11(4):659–66. (Cited on page 150.)

Simon, A. (1999). Gap junctions: more roles and new structural data. *Trends Cell Biol*, 9(5):169. (Cited on page 19.)

Simon, A. M. and Goodenough, D. A. (1998). Diverse functions of vertebrate gap junctions. *Cell*, 8924(98):477–483. (Cited on pages 19, 56, and 97.)

Soffe, S., Roberts, A., and Li, W.-C. (2009). Defining the excitatory neurons that drive the locomotor rhythm in a simple vertebrate: insights into the origin of reticulospinal control. *J Physiol*, 587(Pt 20):4829–44. (Cited on pages 18, 53, 98, 107, 109, 127, 142, and 188.)

Soffe, S. R. (1989). Roles of glycinergic inhibition and N-Methyl-D-Aspartate receptor mediated excitation in the locomotor rhythmicity of one half of the *Xenopus* embryo central nervous system. *Eur J Neurosci*, 1(6):561–571. (Cited on pages 144, 145, 187, and 188.)

Soffe, S. R., Zhao, F. Y., and Roberts, a. (2001). Functional projection distances of spinal interneurons mediating reciprocal inhibition during swimming in *Xenopus* tadpoles. *Eur J Neurosci*, 13(3):617–27. (Cited on page 107.)

Sommerville, I. (2004). *Software Engineering, 7th Edition*. Addison Wesley. (Cited on pages 63 and 65.)

Squire, L., Berg, D., Bloom, F. E., du Lac, S., Ghosh, A., and Spitzer, N. C. (2012). *Fundamental Neuroscience*. Academic Press. (Cited on pages 30, 52, and 53.)

Srinivas, M., Rozental, R., Kojima, T., Dermietzel, R., Mehler, M., Condorelli, D. F., Kessler, J. a., and Spray, D. C. (1999). Functional properties of channels formed by the neuronal gap junction protein connexin36. *J Neurosci*, 19(22):9848–55. (Cited on page 119.)

Stein, P. S. G., Grillner, S., Selverston, A. I., and Stuart, D. G. (1997). *Neurons, Networks, and Motor Behavior*. A Bradford Book. (Cited on pages 6 and 7.)

Stephenson, A. G., LaPiana, L. S., Mulville, D. R., Rutledge, P. J., Bauer, F. H., Folta, D., Dukeman, G. A., and Sackheim, R. (1999). Mars Climate Orbiter Mishap Investigation Board Phase I Report. Technical report, NASA. (Cited on page 76.)

Stroustrup, B. (1997). *The C++ Programming Language: Third Edition*. Addison Wesley. (Cited on page 70.)

Sulston, J. E. and White, J. G. (1980). Regulation and cell autonomy during postembryonic development of *Caenorhabditis elegans*. *Dev Biol*, 78(2):577–597. (Cited on page 8.)

Szabo, T. M. and Zoran, M. J. (2007). Transient electrical coupling regulates formation of neuronal networks. *Brain Res*, 1129(1):63–71. (Cited on page 19.)

Tabak, J. and Moore, L. E. (1998). Simulation and parameter estimation study of a simple neuronal model of rhythm generation: role of NMDA and non-NMDA receptors. *J Comp Neuro*, 5(2):209–35. (Cited on page 143.)

Taylor, A., Cottrell, G., and Kristan, W. B. (2000). A model of the leech segmental swim central pattern generator. *Neurocomputing*, 33:573–584. (Cited on page 126.)

Taylor, A. L., Goaillard, J.-M., and Marder, E. (2009). How multiple conductances determine electrophysiological properties in a multicompartment model. *J Neurosci*, 29(17):5573–86. (Cited on page 12.)

Terman, D., Kopell, N., and Bose, A. (1998). Dynamics of two mutually coupled slow inhibitory neurons. *Physica D*. (Cited on page 176.)

Traub, R. D. and Bibbig, A. (2000). A model of high-frequency ripples in the hippocampus based on synaptic coupling plus axon-axon gap junctions between pyramidal neurons. *J Neurosci*, 20(6):2086–93. (Cited on page 97.)

Traub, R. D., Middleton, S. J., Knöpfel, T., and Whittington, M. a. (2008). Model of very fast (> 75 Hz) network oscillations generated by electrical coupling between the proximal axons of cerebellar Purkinje cells. *Eur J Neurosci*, 28(8):1603–16. (Cited on page 97.)

Tunstall, M. J., Roberts, A., and Soffe, S. R. (2002). Modelling inter-segmental coordination of neuronal oscillators: synaptic mechanisms for uni-directional coupling during swimming in *Xenopus* tadpoles. *J Comp Neuro*, 13(2):143–58. (Cited on pages 122, 128, and 147.)

van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The NumPy array : a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):1–8. (Cited on page 68.)

van Heesch, D. (1997). Doxygen. http://www.stack.nl/ dimitri/doxygen/index.html. Accessed: 2013-06-27. (Cited on page 66.)

Várkonyi, P. L., Kiemel, T., Hoffman, K., Cohen, A. H., and Holmes, P. (2008). On the derivation and tuning of phase oscillator models for lamprey central pattern generators. *J Comp Neuro*, 25(2):245–61. (Cited on pages 11 and 12.)

Vivar, C., Traub, R. D., and Gutiérrez, R. (2012). Mixed electrical-chemical transmission between hippocampal mossy fibers and pyramidal cells. *Eur J Neurosci*, 35(1):76–82. (Cited on page 97.)

Volman, V., Perc, M., and Bazhenov, M. (2011). Gap junctions and epileptic seizures–two sides of the same coin? *PloS One*, 6(5):e20572. (Cited on page 97.)

Weeks, J. and Kristan Jr, W. (1978). Initiation, maintenance and modulation of swimming in the medicinal leech by the activity of a single neurone. *J Exp Biol*, 77(1):71. (Cited on page 8.)

West, J., Hoesen, G., and Kosel, K. (1982). A demonstration of hippocampal mossy fiber axon morphology using the anterograde transport of horseradish peroxidase. *Exp Brain Res*, 48:209–216. (Cited on page 119.)

Wheatley, M. and Stein, R. B. (1992). An *in vitro* preparation of the mudpuppy for simultaneous intracellular and electromyographic recording during locomotion. *J Neurosci Meth*, 42(1-2):129–37. (Cited on page 10.)

Whelan, P. (1996). Control of locomotion in the decerebrate cat. *Prog Neurobiol*, 49(5):481–515. (Cited on page 150.)

White, J. G., Southgate, E., Thomson, J. N., and Brenner, S. (1986). The structure of the nervous system of the nematode C. elegans. *Philos T R Soc B*, 314:1–340. (Cited on pages 8 and 13.)

Willows, A. O. D. (1967). Behavioral acts elicited by stimulation of single, identifiable brain cells. *Science*, 157(3788):570–574. (Cited on page 7.)

Wilson, D. and Waldron, I. (1968). Models for the generation of the motor output pattern in flying locusts. *P IEEE*, 169:1058–1064. (Cited on page 126.)

Wilson, D. M. (1961). The central nervous control of flight in a locust. *J Exp Biol*, 38(2):471–490. (Cited on page 4.)

Wilson, G. (2006). Where's the real bottleneck in scientific computing? *American Scientist*. (Cited on pages 61 and 189.)

Winlove, C. I. P. and Roberts, A. (2011). Pharmacology of currents underlying the different firing patterns of spinal sensory neurons and interneurons identified *in vivo* using multivariate analysis. *J Neurophysiol*, 105(5):2487–500. (Cited on pages 36, 37, 107, 142, 143, and 145.)

Winlove, C. I. P. and Roberts, A. (2012). The firing patterns of spinal neurons: *in situ* patch-clamp recordings reveal a key role for potassium currents. *Eur J Neurosci*, 36(7):2926–40. (Cited on pages 23, 37, 38, 39, 43, and 59.)

Wolf, E., Zhao, F. Y., and Roberts, A. (1998). Non-linear summation of excitatory synaptic inputs to small neurones: a case study in spinal motoneurones of the young *Xenopus* tadpole. *J Physiol*, 511 (Pt 3:871–86. (Cited on pages 36 and 100.)

Wollner, D. A. and Catterall, W. A. (1986). Localization of sodium channels in axon hillocks and initial segments of retinal ganglion cells. *P Natl Acad Sci USA*, 83(21):8424–8. (Cited on page 108.)

Woodfield, S., Dunsmore, H., and Shen, V. (1981). The effect of modularization and comments on program comprehension. In *5th international conference on Software engineering*, pages 215–223. (Cited on page 63.)

Zaytsev, Y. V. and Morrison, A. (2012). Increasing quality and managing complexity in neuroinformatics software development with continuous integration. *Front Neuroinform*, 6:31. (Cited on page 65.)

Zeng, S. and Tang, Y. (2009). Effect of clustered ion channels along an unmyelinated axon. *Phys Rev E*, 80(2):1–9. (Cited on page 120.)

Zheng, M., Iwasaki, T., and Friesen, W. O. (2004). Systems approach to modeling the neuronal CPG for leech swimming. In *Eng Med Biol Soc Ann*, volume 1, pages 703–6. (Cited on page 12.)