Chapter 3
History of Neural Simulation Software

David Beeman

Abstract This chapter provides a brief history of the development of software for simulating biologically realistic neurons and their networks, beginning with the pioneering work of Hodgkin and Huxley and others who developed the computational models and tools that are used today. I also present a personal and subjective view of some of the issues that came up during the development of GENESIS, NEURON, and other general platforms for neural simulation. This is with the hope that developers and users of the next generation of simulators can learn from some of the good and bad design elements of the last generation. New simulator architectures such as GENESIS 3 allow the use of standard well-supported external modules or specialized tools for neural modeling that are implemented independently from the means of the running the model simulation. This allows not only sharing of models but also sharing of research tools. Other promising recent developments during the past few years include standard simulator-independent declarative representations for neural models, the use of modern scripting languages such as Python in place of simulator-specific ones and the increasing use of open-source software solutions.

Introduction

When Jim Bower first asked me if I would write a chapter on the history of realistic neural simulators, I refused. I reminded him that although I am at an age when scientists wrap up their long careers with a historical account full of advice for young researchers, I have only been involved with computational neuroscience for a little more than 20 years, and I am just getting started with serious cortical modeling.

D. Beeman (✉)
Department of Electrical, Computer, and Energy Engineering,
University of Colorado at Boulder, Boulder, CO 80309-0425, USA
e-mail: dbeeman@colorado.edu

Furthermore, my experience has been almost entirely as a developer of tutorials and documentation for the GEneral NEural SImulation System (GENESIS). Not only that, but I missed out on the crucial first 2 years of GENESIS development.

However, Jim can be very persuasive, and I gave in after he told me that I could tell the story my way as a personal history of what I have learned during the past 22 years. Of course, the task was made easier by the fact that I share Jim's definition, considered very narrow by many, of what constitutes a "realistic neural simulator" (Bower 1992, 2005). What I offer is a somewhat GENESIS-centric and subjective view of some of the issues that came up during the last 20 years of development of GENESIS, NEURON, and other platforms for structurally realistic simulations of neurons and their networks.

More specifically, I finally agreed to write this chapter because it provides an opportunity to use the last 20 year history of the development of neural simulators to offer a few, hopefully useful, opinions on what the developers and users of the next generation of simulators can learn from some of the good and bad design elements of those from the last generation. My own current view is that it is time for a new generation of simulators that addresses a large range of issues beyond merely creating numerical solutions to mathematical models of neural systems. As will be discussed later, new simulation architectures such as that being developed for GENESIS 3 (G-3) offer options for extensibility, interoperability, and model sharing that will significantly extend the capacity and value of simulation technology, providing a foundation for the next 20 years and beyond (Cornelis et al. 2012a). My hope is that this chapter will help motivate and inform these efforts going forward.

My Seduction by Neuroscience

This chapter is in large part a personal recounting of my own experience in the design and development of GENESIS and it is therefore appropriate, I think, to provide the reader some background information on my own path to computational neuroscience. In the spring of 1989 I was 51 years old, and a happily tenured Professor of Physics at Harvey Mudd College, an undergraduate science and engineering college about 30 miles from Caltech. I enjoyed the atmosphere at Harvey Mudd because the students are very intelligent, self-motivated, and creative. In addition, without graduate students, I was able to concentrate on undergraduate education, and to take advantage of the great freedom I had to create and teach interdisciplinary special topics courses on nearly anything that interested me.

Although it might seem strange that, 20+ years later, I would have abandoned physics, given up my tenured faculty position at Harvey Mudd, and now be deeply involved in building models of the mammalian auditory cortex, in fact, looking back, the transition from teaching undergraduate physics to working on GENESIS makes some sense.

A core component of the GENESIS project from the outset, and one that I have been particularly involved with, is the development and application of simulation-based tutorials to engage students in learning. At Harvey Mudd, I had already developed several simulation-based tutorials for teaching concepts in upper division physics courses (e.g., Beeman and Boswell 1977). My transition to computational neuroscience as a subject of study was also linked to my teaching through my research interests, which at an undergraduate college, need to be tied closely together. That research involved the computer modeling of amorphous solids or disordered spin systems which served as a very good source of student projects, and a few publications (e.g., Maley et al. 1988; Thorpe and Beeman 1976; Alben et al. 1977).

Because of the spin system modeling I had been doing, I became aware of the publication by Hopfield (1982) applying spin glass models as a possible model for how networks of neurons might operate. As a result, I volunteered to give some lectures in an interdisciplinary course that was to be taught during my sabbatical in spring 1990. For that course, a biologist was to lecture on basic properties of neurons, a mathematician would present a mathematical approach to artificial neural networks, and I would give a more engineering-oriented approach to artificial neural networks, aided with tutorial simulations.

Of course I was also aware that John Hopfield was at that time a professor in the new Computation and Neural Systems program at Caltech just down the 210 freeway from Harvey Mudd. My first thought was to ask Dr. Hopfield about spending some time with his group at Caltech during my sabbatical year. However, our department received a weekly list of seminars at Caltech, which was posted outside the Physics Department office, and given my interest in simulation technology, I was intrigued by the description of a talk to be given by Matt Wilson on a simulator for biologically realistic neurons and neural networks called GENESIS. I went to the talk and became fascinated with the prospect of actually building models of real neurons. Although Matt's seminar emphasized the use of GENESIS as a research tool, I thought that I might also be able to use it in the course I was planning to teach at Harvey Mudd. I also thought that, through the simulator, I might be able to learn something about biological neurons myself. I had no idea at the time that this was the first step on the long slippery slope that has led so many physicists into neuroscience.

After the talk, as I spoke to Matt, Jim introduced himself and we talked about the possibility of using GENESIS as a basis for a simple tutorial on the properties of biological neurons. Jim told me of his strong commitment to developing tools for education, and we discussed possible collaborations. As a result, I spent my sabbatical year in Jim's laboratory, learning about GENESIS, and far more about neuroscience than I had ever intended.

History of Neural Modeling Prior to Fall 1989

Before I could write tutorial simulations about neuroscience, I needed to learn something about the subject. Of course, I was familiar with the history of artificial "neuron-like" networks, beginning with the Mcullough and Pitts (1943) model, but had no knowledge of modeling spiking neurons, nor very much about their physiology. I sat in on David van Essen's introductory neuroscience course, along with the first year graduate students in neuroscience, and also began learning how to use GENESIS. As always, I learn best by doing, so I started writing my first tutorial based on a simulation of a simple neuron model, as I learned about the Hodgkin–Huxley model, and about compartmental modeling of dendrites.

Of course, I read the classic paper by Hodgkin and Huxley (1952) to understand the modeling of action potentials. Their work, carried out in the early 1950s and described in a series of 1952 papers, won them the Nobel Prize in 1963 and exemplifies, in my view, the ideal combination of modeling and experimental measurements. I can't emphasize enough the importance of this connection between modeling and experimental work. Too many theoretically inclined people get lost in a world of their own when doing computer modeling and lose touch with the world of experiment. This is a common pitfall for theoretical physicists who hope to apply their expertise to the study of the brain. Likewise, experimentalists may end up mindlessly gathering data without having a clear idea of how it will advance theoretical understanding.

The importance of this connection is also clear in computational neuroscience and was essential to what I regard as the first great success of computational neuroscience, the Hodgkin–Huxley model (now 60 years old), which still stands as the basis for most neuronal cell models. Most neurobiologists recognize the importance of Hodgkin and Huxley's work and their development of the voltage clamp technique without realizing how important the modeling was to the work. Essentially, the model was what made them throw out their old way of looking at the changes in the membrane and introduce a new viewpoint. It is important to remember that at the time of their experiments, the modern concept of ion-selective channels controlling the flow of current through the membrane was only one of the several competing hypotheses. It was the model that ruled out these alternative ideas, and also predicted the results of experiments that were not used in formulating the model. The fascinating history of this pioneering synthesis of experiment and modeling has been told in reviews by Cole (1968), Rinzel (1990), Nelson and Rinzel (1998), and many others.

However, it is important to mention how Hodgkin and Huxley quantitatively explained the process by which action potentials are formed by the voltage-dependent activation and subsequent inactivation of sodium channels, terminated by a delayed activation of potassium channels. They did this not by fitting model parameters to those needed to produce action potentials, but by fitting them to an entirely different set of experimental data, obtained using the voltage clamp. Then, with no further changes in parameters, they were able to reproduce the action potential, correctly calculate the velocity of propagation, analyze the refractory period, and account for the phenomenon of post-inhibitory rebound or "anode break." All of this modeling was performed by integrating the coupled differential equations, step by step, on mechanical "hand-crank" calculators, following the method used 20 years before by the physicist Hartree (1932) to calculate atomic wave functions.

As a perhaps ironic aside, a few years after I had taken charge of the GENESIS Users Group (BABEL), I received an email from a postdoctoral student who pointed out what he claimed to be a serious bug in GENESIS. He found that using a hyperpolarizing current injection pulse in one of the GENESIS tutorial simulations produced the obviously impossible result of producing an action potential! I tactfully suggested that he read the original Hodgkin–Huxley papers. At that time, I had been extending Mark Nelson's "Squid" tutorial for use with the chapter that he and John Rinzel were writing for "The Book of GENESIS" (Bower and Beeman 1998), familiarly called "The BoG." It was only after I added the ability to plot the channel activation variables during a current pulse that I fully understood myself the action potential refractory period and the biological phenomenon of post-inhibitory rebound.

In my own efforts to understand the existing techniques for simulating real neurons, the next step was to understand why GENESIS broke a neuron into "compartments." I quickly learned of the other crucial development in the history of neural modeling, which was the introduction of compartmental modeling by Rall (1964). Rall had previously contributed a great deal to the understanding of postsynaptic potential (PSP) propagation in dendrites by applying the mathematical analysis of the attenuation of signals in the transatlantic telephone cable by William Thompson (Lord Kelvin). For example, neural modeling has benefited from the simplifications introduced by the "trees equivalent to a cylinder" transformation, in which Rall (1959, 1962a) demonstrated the conditions under which a branched dendritic tree can be collapsed into a linear cable. The "cable" theory of propagation in dendrites has been reviewed by Rall and Agmon-Smir (1998).

By using a lumped parameter model, dividing a branched dendritic tree into coupled chains of approximately equipotential compartments, Rall's method made it possible to explore realistic dendritic morphologies that could only be analyzed by using numerical methods and computer simulations. In one of the first applications of this method Rall (1967) modeled a linear chain model of a motor neuron with a soma and nine dendritic compartments, activated with an "alpha function" form of synaptic conductance having a linear rise and exponential decay with time. It had no voltage-activated conductances.

Rall and Shepherd (1968) created the first model to combine compartmental modeling of dendrites into a cell model that generated action potentials. Their model with a soma and ten dendritic compartments used parameters taken from rabbit olfactory bulb mitral and granule cells. Because the Hodgkin–Huxley model parameters for the neurons being simulated were not yet known, a simpler and less computationally intensive model was used for the generation of action potentials with active conductances. These simulations were carried out on a Honeywell 800 computer during 1963 and 1964 at the NIH, at a time when realistic simulation of ionic currents was considered to be very time-consuming. Personally, I believe that Rall's pioneering modeling efforts have been on a par with those of Hodgkin and Huxley, and perhaps the only reason that he has not received a Nobel Prize is due to the sheer complexity of dendrites themselves, whose function we still don't understand. Certainly Rall's technical contribution to modeling is on the same level as that of Hodgkin and Huxley.

Around the time that Rall and Shepherd were building their first models of neurons, others were applying the Hodgkin–Huxley equations to single compartment neuron models. For example, Connor and Stevens (1971) performed one of the first computer simulations of the ionic currents and the resulting action potentials in giant molluscan neurons. This model added a transient potassium conductance ("A-current") to modified Hodgkin–Huxley fast sodium and delayed potassium currents, using parameters fitted to experiments.

Dodge and Cooley (1973) were the first to publish a description of a model that combined compartmental modeling with the Hodgkin–Huxley equations. They collapsed a large spinal motor neuron into a compartmentalized nonuniform equivalent cylinder by the method of Rall (1962b) and used fast sodium and delayed rectifier potassium channels with parameters modified to fit motor neuron voltage clamp data.

This model became the basis for later models by Traub (1977), who included calcium-dependent potassium channels in the dendrites. This led to a series of increasingly realistic hippocampal pyramidal cell models (Traub and Llinás 1979; Traub 1982; Traub et al. 1991, 1994) with active conductances in the dendrites. These were run on IBM mainframe computers, with programs initially written in PL/1 and later in FORTRAN.

The earliest network model with multi-compartmental spiking neurons of which I am aware was a simplified model of the cerebellar cortex of the frog by Pellionisz et al. (1977). These used 62-compartment Purkinje cell models having modified Hodgkin–Huxley conductances.

The Introduction of Neural Simulation Systems

In each of the cases mentioned to this point, the computational modeling was done with specific code written by the individual researchers. To my knowledge, there was no effort made to provide that code to anyone else or to generalize it beyond a particular model. There was also no explicit effort to use these simulations as a tool in neuroscience education. Their intended purpose was purely research and was based only in the individual research labs. Yet, the nervous system itself is made up of neurons that share many common components (e.g., ion-selective channels), raising the distinct possibility that a modeling system with common code and a common set of libraries could allow the sharing of components between different laboratories. In effect, this kind of sharing in physics has occurred for hundreds of years, in part because it is easier to share equations than complex biological models dependent on a whole system of equations. In principle, however, a common modeling platform for neural simulations could not only support the sharing of components, but also start to build a common set of models, which we have called "community models," supporting communication between different laboratories and research projects. Jim Bower's chapter in this volume talks about what may be the first such model, of the cerebellar Purkinje cell, originally developed in GENESIS and now implemented in numerous other simulation systems (including

NEURON) and also now being used as the basis for research in a growing number of laboratories. Of particular interest to me, such a general neural simulation system could also, in principle, be used to generate simulation-based tutorials for education in the tools of computational neuroscience, as well as neuroscience itself.

The use of an electrical network simulator was first suggested by Shepherd and Brayton (1979) for a simulation of a dendro-dendritic synapse circuit in the olfactory bulb. However, the first software that could be called a general simulator specifically for realistic neural models was a suite of FORTRAN programs called MANUEL developed by Don Perkel in 1981 (Perkel and Watt 1981). MANUEL generalized some of his earlier custom made programs into a package for constructing multi-compartmental neurons and small circuits. Jim Bower tells me that one of the first things he did after moving to Caltech was to visit Don Perkel in his research trailer parked behind a research building at the University of Irvine. Don had recently been fired as a professor by Stanford, but had used grant money to purchase a trailer, outfitted with computer equipment, and had made a temporary arrangement with UC Irvine renting space in their parking lot. Don was well ahead of his time. The MANUEL programs allowed a wide variety of physiological and anatomical properties to be specified and provided a versatile set of utilities for providing stimuli and recording the results. As a way to provide support for his pioneering efforts, MANUEL was available for a substantial fee, and was written in a Digital Equipment Corporation (DEC) variant of FORTRAN IV, and did not run on Unix systems. MANUEL was used by Peter Getting to make what was probably the first network model made out of realistic neurons to study the swim central pattern generator circuit of the mollusc *Tritonia diomedea* (Getting 1989). Ironically enough, Peter Getting had also not been given tenure at Stanford and ended up at the University of Iowa, where he built his model accurately reproducing the swim pattern and explaining the role of the various ionic conductances in determining the behavior of this small network. Sadly, both Don Perkel and Peter Getting shortly thereafter developed serious health problems that ended their research careers.

There was also an effort in the 1980s to use simulators built for modeling electric circuits, such as SPICE (Segev et al. 1985) and SABER (Carenvale et al. 1990) to construct models of neurons. In principle these simulation systems had the built-in tools needed for simulating the circuits used in compartmental models. They also had the advantage that they were widely used by electrical engineers and also had the advantage of being available for a wide range of computer operating systems. In the end, however, most of the development of these systems was focused on electrical circuit simulation and they were not further optimized for building neuronal models. Similarly, the growth in interest in neural networks for engineering purposes also resulted in the construction of several "neural network" simulation systems such as the Rochester Connectionist Simulator (Goddard et al. 1987) that were also promoted for their possible use in biological network simulations. These also turned out to be too specialized and restricted in their capacity to support full realistic models.

Early History of GENESIS and NEURON

It was during this same period of time, in the middle 1980s, that the development of both GENESIS and NEURON started as dedicated systems specifically for building realistic simulations of the nervous system. By 1989, when I first learned of GENESIS, both systems were well into the early phase of their development—each, however, starting from a different point of view and with somewhat different objectives.

The NEURON simulator had its beginnings in the laboratory of John W. Moore at Duke University, where Michael Hines developed an efficient implicit numerical integration algorithm for use in branched compartmental dendritic models (Hines 1984). The Hines method was initially implemented in CABLE, a simulator developed for modeling propagation of PSPs in dendrites (Hines 1989).

Although it was primarily being used for modeling dendritic structures at this time, it already had the capability of making multi-compartmental models of single cells. In addition to the standard voltage-activated Hodgkin–Huxley sodium and potassium channels, it could model basic mechanisms for calcium dynamics and calcium-dependent potassium channels. By 1990, the name had changed to NEURON, and its single cell modeling capabilities began to expand. Over the next 2 years, it gained a scriptable GUI, the ability to model small networks, and a method of loading and compiling user-specified channel kinetics.

A Personal History of GENESIS

Of course, my personal knowledge of the development of GENESIS is much more detailed than that of NEURON.

GENESIS had its origins during 1984, when Matt Wilson was studying for a Master's degree in electrical engineering at the University of Wisconsin. During this time, after having done postdoctoral studies in the laboratory of Rudolfo Llinás at New York University, Bower was finishing up his postdoctoral work in Lewis Haberly's lab at the University of Wisconsin, studying the olfactory cortex.

As Bower recalls, Matt had been hired to program data acquisition software for a new brain slice preparation he was setting up in the Haberly Laboratory. Jim had also brought to Wisconsin the data he had recorded from many cerebellar Purkinje cells at once while a postdoc at NYU. This data was unusual and complex as it was one of the first sets of multi-single neuron data ever obtained, consisting of recordings of 16–32 signals at once (Sasaki et al. 1989). Bower was interested in finding some means of analyzing the data other than cross-correlation analysis. After approaching Josh Chover, head of the Math Department at the University of Wisconsin, Chover and Bower decided to co-teach a course on statistical analysis of multiunit recording data, for which Matt became the teaching assistant.

During that course, Jim realized that understanding complex neurobiological data would eventually require a tight coupling between experimental and model-based studies (Bower 1991). As an experimentalist, he believed that the types of model that would be most useful were ones that as closely as possible approximated the actual morphological and physiological properties of the brain structures being studied. While it was several years before his laboratory began developing models of the cerebellum (e.g., Santamaria et al. 2007), he decided to see if a model of the olfactory cortex might help explain the pattern of oscillations he was studying in the Haberly laboratory. Working together, Jim and Matt generated the first network model of the olfactory cortex constructed on an IBM XT computer. The model consisted of a linear chain of 75 5-compartment neurons which almost as soon as it was constructed began oscillating with 40 Hz bursts at Theta frequency. Jim's recollection is that it took several weeks to figure out how the bursts were generated in the model.

When Jim came to Caltech in early 1985, as one of the cofounders of the interdisciplinary graduate degree program "Computation and Neural Systems," he encouraged Matt to apply as a doctoral student. After arriving from Wisconsin, Matt continued to elaborate the olfactory cortex model, and published a thesis in 1990 predicting the neural mechanisms underlying the 40 Hz and theta frequency oscillations in cerebral cortex (Gray et al. 1989; Wilson and Bower 1991, 1992).

According to Jim, while Matt was working on his own modeling studies, he asked Matt to generalize his simulation software so that it could be used as a general purpose simulator, rather than as a stand-alone single purpose model. Matt resisted at first, feeling that no computational modeler would ever want to use software that was written by someone other than themselves. Fortunately, Matt relented and began work on GENESIS (Wilson et al. 1989).

Later, as I attended the annual Computational Neuroscience meetings in the early to mid 1990s, and collected data to satisfy funding agencies that GENESIS indeed was being widely used outside the Bower laboratory, Matt's prediction that real programmers would write their own code seemed to be born out. During that decade, the number of poster presentations using GENESIS or NEURON was generally outnumbered by those that used custom software written for a particular simulation or category of simulation. However, today, the number of scientists using simulation systems continues to rise, and importantly, the simulators are increasingly providing an opportunity for nonprogrammers to engage in computational studies. They are also increasingly being used in graduate and even undergraduate education, replacing textbooks with dynamic simulation tutorials.

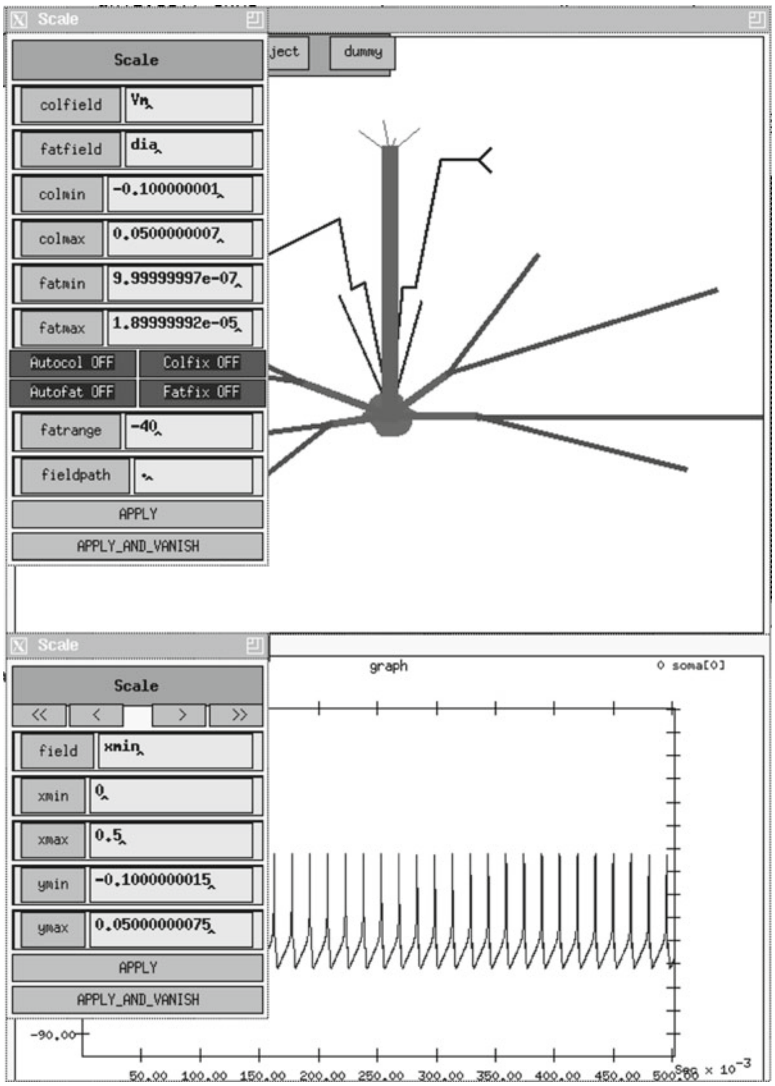
As a side note, as he continued to develop GENESIS, Matt became increasingly interested in the experimental side of neuroscience research, and the studies of the hippocampus being carried out by Bruce McNaughton. After he received his Ph.D. in 1990, he went to the McNaughton laboratory at the University of Arizona for his postdoctoral studies, bringing the multielectrode array design with him. Legend has it that he said that he would come only if he were not required to do further modeling. Matt is now Sherman Fairchild Professor in Neurobiology, Departments of Brain and Cognitive Sciences and Biology at MIT, researching the role of sleep in learning and memory.

With respect to the early history of GENESIS, it is also important to mention the contributions of Upinder S. Bhalla who developed the first GUI for GENESIS, XODUS (the X Oriented Display Utility for Simulations) (Bhalla 1998). Entering the Bower laboratory as a doctoral student in Neuroscience in 1986, "Upi's" principal focus was on multielectrode recording in the olfactory bulb of awake behaving animals, where, along with Matt Wilson, he designed a unique multielectrode array that has subsequently became the basis for many multi-single unit recording experiments in many laboratories. However, Upi was also using GENESIS to model olfactory bulb mitral and granule cells, and started adding graphical capabilities around 1987. The initial version of XODUS was based on the Unix X Window System, and added scripting commands to Matt's Script Language Interpreter (SLI).

Although it is getting somewhat ahead of the story, it is appropriate to mention that after completing his Ph.D. in April 1993, Upi began his postdoctoral studies with Ravi Iyengar at Mount Sinai School of Medicine in New York, contributing to GENESIS development over the Internet. By November 1995 he had added the kinetics library and Kinetikit GUI for modeling chemical kinetics and signaling pathways (Bhalla and Iyengar 1999). After assuming his faculty position at the National Centre for Biological Sciences in Bangalore in January 1996, he continued to study the systems biology of olfaction and memory, and to extend the capabilities of the GENESIS kinetics library (Bhalla 2000). In order to exchange biochemical signaling models, he established the Database of Quantitative Cellular Signaling (DOQCS), one of the first databases of models of signaling pathways in the brain (Bhalla 2003). DOQCS (http://doqcs.ncbs.res.in/) currently contains 76 models contributed by users world-wide. The model representation format used in DOQCS is based on GENESIS 2 SLI commands using the kinetics library; however, it is now being extended to include SBML (Hucka et al. 2003) and Matlab formats. Later, as the limitations of the 1980s and 1990s simulator architectures became apparent, Upi began a major reimplementation of GENESIS 2 as MOOSE (Ray and Bhalla 2008).

Because the technical basis for GENESIS was a model of the olfactory cortex, GENESIS differed from the design of NEURON in that, from the outset, it was designed to simulate neural structures at multiple levels of scale (Wilson et al. 1989). Due to the influence of the parallel computing group at Caltech headed by Geofry Fox, GENESIS was also from the outset, designed to be implemented on parallel computers (Nelson et al. 1989). Although NEURON had its origins as a simulator for single cell models, it acquired improved network modeling capabilities and a parallel implementation in the following years.

By the July 1990 public release of GENESIS version 1.1 with full source code, Upi had added the "Neurokit" graphical environment for editing and running single cell models to GENESIS. Neurokit was written entirely in the GENESIS scripting language, using XODUS. Figure 3.1 shows this first version of Neurokit running a mitral cell model, with the menus, cell view, and graph displayed. By 1991, Upi had added the Hines (1984) integration method to GENESIS and added a cell reader to read in cell model specifications from a GENESIS cell parameter (".p") file, greatly increasing its capabilities for large single cell models.



```
colfield   Vm
fatfield   dia
colmin     -0.100000001
colmax     0.0500000007
fatmin     9.99999997e-07
fatmax     1.89999992e-05
fatrange   -40
fieldpath  *
```

```
field  xmin
xmin   0
xmax   0.5
ymin   -0.1000000015
ymax   0.05000000075
```

Fig. 3.1 An early GENESIS 1 version of Neurokit used to run and edit a mitral cell model. This used scripted XODUS objects to create menus, an animated cell view, and graph

GENESIS and NEURON Go Public

Returning to the chronology, the next major step for GENESIS and NEURON, and the initial availability for the use of both systems by those outside the founding laboratories, came as a result of the establishment of the Summer Course in Methods in Computational Neuroscience at the Marine Biological Laboratory (MBL) in

Woods Hole. As Jim Bower recounts the history, he and his Caltech colleague Christof Koch were sitting in Christof's backyard on a particularly hot and smoggy day in the summer of 1986 in Pasadena California, discussing what they could do the following year to be somewhere more pleasant with their families. Jim, who has spent time as a postdoctoral fellow doing summer research at the MBL, suggested that they propose offering a course in computational neuroscience, and spend the late summer in Woods Hole. That fall, the MBL accepted the course and the first in what has now become a series of courses around the world was offered. The first course of its kind, Jim's strong bias towards "hands on science learning" meant that the course was designed so that its central focus was on student projects based in a computer laboratory. For the first course, Jim and his feisty laboratory systems administrator John Uhley manually pulled the first Internet lines from the Woods Hole Oceanographic Institute in the tunnels under Water Street in Woods Hole to the MBL. Uhley installed 20 brand-new graphics workstations that were donated by the now defunct DEC in the laboratory, and literally the night before the opening day of the course, a public version of GENESIS was installed for the first time. Although Matt regarded this as version 0.001 of GENESIS, it already included a graphical user interface (thanks to Upi), powerful network creation commands, and an efficient method of summing spike events from multiple connections to a synaptically activated channel (Wilson and Bower 1989). The official release of GENESIS 1.0 coincided with the second Woods Hole course in July 1989 with usability greatly increased, due to continued work by Matt, Upi, Dave Bilitch, and John Uhley.

However, the first course in Woods Hole was not only the introduction of GENESIS but also of NEURON. As Jim recounts the story, on the very first day of the course, Michael Hines approached Jim and Christof about the possibility of installing CABLE on the laboratories computers as well, and giving students the option to use either GENESIS or CABLE (soon to be NEURON). Both Jim and Christof thought that this was a wonderful idea, and Michael Hines and NEURON became a regular part of the course from then on.

Federation and User Support

While I missed the first two Woods Hole courses, in the spring of 1990, as Matt was finishing his work at Caltech, my wife received a job offer that was too good to pass up in Boulder, Colorado. I then said goodbye to Harvey Mudd and came to the University of Colorado, supported as a consultant on GENESIS grants. This began the "federalization" of GENESIS development via the Internet, and a collaboration between GENESIS developers that still continues to this day. With the departure of Matt, Dave Bilitch gradually took over as the lead GENESIS software developer, coordinating our efforts with the crude Internet tools of the time: text-based email without attachments, ftp, and remote logins to the server "smaug" at Caltech via telnet. During the 1990s, as former members of the Bower laboratory formed research groups using GENESIS, and the expanding number of GENESIS users contributed to GENESIS capabilities, GENESIS development became increasingly distributed. To facilitate communication between users and developers, an email newsletter to the GENESIS users group was established in April 1991, and the GENESIS web site was established in June 1994.

When I entered the Bower laboratory in fall 1989, GENESIS development seemed to "just happen" by a small group interacting closely without a lot of formal organization. I would tell Upi "it would be nice if GENESIS could do …." A couple of days later he would casually mention that GENESIS could now do it. I noticed this among the Bower lab group at Woods Hole. Someone noticed that something needed to be done, and just did it. This may not be a scalable plan for a major software development project, but it worked extremely well in those days. As GENESIS development has spread from a single laboratory to many, we have had to face the challenge of "federalizing" a large software development project among many user-developers.

From the beginning, GENESIS had the framework for interactive help, but not yet a lot of content. The "man page" was often a shout down the hall to Matt's office "Hey, man. I have a question." Gradually his answers evolved into additions that I made to the documentation as I saw the need for it. The interactive help was invoked in a terminal window at the "genesis >" prompt, and was plain text, formatted similarly to a Unix "man page." Upi added a printed manual with LaTeX source that covered basic syntax for the SLI, and the main GENESIS and XODUS objects and commands.

The GENESIS 1.0 release came with two tutorials that were created in April 1989. Mark Nelson contributed the "Squid" tutorial on the Hodgkin–Huxley model that is still in use today after years of enhancements by GENESIS users. The "MultiCell" tutorial was a simulation of two neurons having a soma and dendrite compartment with synaptic connections, with one being excited and the other being inhibited. The GUI had a control panel, graphs of membrane potential and channel conductances, and labeled text fields (called "dialogs" in XODUS) for changing the synaptic channel parameters. The extensive documentation with reference to a line-numbered version of the main scripts was the most useful early GENESIS documentation.

At an early GENESIS developers meeting, around the time that we launched the GENESIS web site in June 1994, we discussed ways to use a common source for the generation of plain text "help," a printed manual, and an Hypertext Markup Language (HTML) version for the web. I had been looking at an open-source package called "linuxdoc-sgml" that was then being used by The Linux Documentation Project (http://www.tldp.org) for doing this by writing the documentation, not in HTML, but in the much richer Standard Generalized Markup Language (SGML), that was the basis of HTML, and a few years later would become the basis for the now-popular eXtensible Markup Language (XML).

The near-unanimous decision was that I should use a well-supported commercial tool, FrameMaker to generate the documentation. As I was the one writing the documentation and I lived in Colorado, not Pasadena, I went home and wrote the documentation for the GENESIS 2.0 release in SGML. Now, FrameMaker no longer exists, but SGML lives on in the form of an open standard, XML. Actually, this move from commercial software to an open standard was only the first in a series of similar events in computational neuroscience.

Discussions of a major rewrite of GENESIS to create version 2.0 began in early 1993. After more than 2 years of development and several months of beta testing, GENESIS version 2.0 was finally released in August 1995. In addition to having a detailed reference manual in all three formats, it now ran under the Linux and FreeBSD operating systems, and with programming help from Maneesh Sahani, had a completely rewritten version of the XODUS graphical interface. The add-on library for Parallel GENESIS (PGENESIS) was released at the same time, allowing simulations to be spread over multiple processors or networks of workstations on a variety of hardware and software platforms.

In addition to their own research, one of the motivations for the further development of GENESIS was provided by its potential use as a tool in neuroscience education. "The BoG" mentioned previously, was written and edited by Jim Bower and myself as a step-by-step tutorial and interactive self-study for professionals, researchers, and students working in neuroscience. The free Internet edition (http://www.genesis-sim.org/GENESIS/bog/bog.html) and the printed version (Bower and Beeman 1998) use exercises and hands-on tutorials developed at the Woods Hole and later courses, with contributed chapters by researchers in computational neuroscience that are linked to the tutorials.

Expanding Simulator Capabilities

With the early history of simulator development now described, I will turn to the discussion of some of the issues in the construction of neural simulations that arose in the evolution of GENESIS and NEURON to their current state, their differences, and what I think this portends for the future.

It is easy to argue that the expanding base of the use of simulation systems is directly related to the expansion of their technical capabilities as well as their ease of use. During the 1990s, both GENESIS and NEURON expanded their graphical capabilities and repertoire of built-in tools specific to neural modeling, making the advantages of using a general simulator package obvious. These simulators then became the preferred method of constructing these types of models. At present, GENESIS and NEURON have very similar functionality for realistic neural modeling. However, there are some significant differences in the way that simulations are created and models are represented. An examination of the different approaches taken in their design may offer some insight into issues facing developers of the next generation of neural simulators.

Scripting and GUIs

I was attracted to the idea of using GENESIS to write tutorials because of its built-in graphical tools that I could use with the same scripting language that would be used to construct models. Matt never made much use of GUIs in his simulations, preferring to run his long network simulations in batch mode, sending the output to files for later analysis. Many modelers still follow this approach, using simulation scripts written with a text editor, and minimal graphics. However, a customizable GUI was necessary for developing tutorials. Later, I found out how important it could be for interpreting the results of parameter changes in a simulation during run time, allowing a quick exploration of a model.

At the time, most neural modelers, and even nonprogrammer users of personal computers, were at home in a command-line computer environment. Over the years, the expectations of modelers have changed, and a GUI is considered essential. However, using scripting commands to position graphical "widgets" (buttons, text fields, graphs, etc.) in a window is difficult and time-consuming. Newer graphical libraries provided for Java or Python, or the cross-platform wxWidgets library being used for G-3 provide more powerful tools than the comparatively low-level syntax used in XODUS. Nevertheless, the script code to set up a neural simulation GUI is often much longer than that needed to set up and run the simulation.

Generic GUIs such as the GENESIS Neurokit or the NEURON Cell Builder can be very useful for analyzing or tuning a single cell model, but rarely have the flexibility or unique features needed to perform and visualize the results of a research simulation, or to use as the basis of a tutorial simulation. In principle, Neurokit and the Purkinje Cell Tutorial can be modified by the user, as they are written in the XODUS extensions to the GENESIS SLI language. In practice, the scripts are far too complicated for most users to want to modify. G-3 allows for the future use of an Integrated Development Environment (IDE) such as Glade to let users create, size, and position graphical elements with a mouse, saving the layout in a standard format (Cornelis et al. 2012b).

When I began writing my first GENESIS tutorial simulation, the "Neuron" tutorial (Beeman 1994), I began to appreciate the object-oriented (OO) nature of the language that Matt had created for building simulations. I admit to being a somewhat lazy programmer who would rather hack at an existing example than to plan a program out from the beginning and start with a blank screen. I think that every computer program or simulation script that I have ever written has been a modified version of something else. Of course, I want to give a lot of thought to planning the structure of the program before I start, but I am likely to start with something that I have already written as a template. Then I fill in bits of code taken from examples or from other programs that I had written for something else. This wasn't too hard with my own FORTRAN, Pascal, or C code, but trying to merge pieces of someone else's code into my own and keep track of all the global variables and dependencies was often more work than starting over and writing it myself from the beginning.

The scripting language that Wilson developed for the GENESIS SLI had a syntax similar to C, and the ability to create simulation "elements" (or "objects" in modern terminology) from templates or "object types" (i.e., "classes"). This enabled neural models to be constructed using a "building block" approach. Simulations are constructed from modules that receive inputs, perform calculations on them, and then generate outputs. Model neurons are constructed from these basic components, such as dendritic compartments, and variable conductance ion channels. Compartments are linked to their channels and are then linked together to form multi-compartmental neurons of any desired level of complexity. Neurons may be linked together to form neural circuits. By keeping most of the variables and functions (methods or "actions") local to these elements, it was easy to pull in pieces of another model without having to understand very much about the large script in which it was embedded. This approach to building simulations has worked very well when building neural models.

Object-oriented programming concepts were well known among the artificial intelligence community at the time that GENESIS was written, but had not entered the mainstream of computer programming until the mid-1990s. The object-oriented programming language C++ was then in its infancy and would not become standardized until much later.

The parser for the scripting language devised by Matt Wilson was written by him in C. In order to make GENESIS as flexible as possible for creating different types of models, the object types were made as general as possible. For example, an "hh_channel" was not a particular Hodgkin–Huxley squid axon channel, but would model any channel that could be modeled with Hodgkin–Huxley type equations. The "tabchannel" and "tab2Dchannel" objects with tables for gate activation, introduced shortly afterwards, provided further generality. Later, De Schutter (De Schutter and Smolen 1998) added a library of GENESIS objects for modeling calcium diffusion.

By giving these objects different parameter values, creatively connecting them together in a script, and manipulating them with user-defined commands, a great amount of user-extensibility was achieved without having to do any programming outside of the scripting language. Inevitably, the time comes when a new object type or command needs to be defined and compiled into GENESIS. This requires some C programming ability of the user, although the process is simplified considerably by the detailed documentation and examples that are provided. Once the new version of GENESIS is compiled, the new functionality is available for any subsequent use of GENESIS.

NEURON is written in C, but made use of an existing scripting language and parser software by choosing HOC (Kernigan and Pike 1984). HOC has a C-like syntax and is also written in C. It was easily extended to include functions specific to modeling neurons, and over the years graphical commands and object-oriented programming concepts were added.

However, it proved difficult for users to add new channel mechanisms in HOC, so a high-level model description language, NMODL (Kohn et al. 1989) was incorporated into NEURON (Hines and Carnevale 2000). This made it much easier for users to extend the functionality of NEURON by writing definitions in NMODL, and then having them automatically compiled and linked into NEURON. This provides some advantages over the GENESIS approach, such as allowing a high-level scripting language to specify the differential equations to be solved, rather than writing lower level C modules. However, this means that most NEURON models of any complexity involve a mixture of HOC and NMODL, and require a recompilation and link step each time a simulation is run.

Parameter Search, Model Tuning, and Comparison

Upi's additions to XODUS enabled me to have pop-up help windows with scrolling text and images in my completed "Neuron" tutorial in time to use it with assigned exercises in the "Modeling and Analysis of Neural Networks" course (CS 189B) at Harvey Mudd in the spring of 1990. Jim suggested that I next write a simulation of a bursting molluscan neuron and develop a tutorial to go along with a manuscript "The Dance of the Ions," that he was writing to explain the role of the various types of ionic conductances in shaping the firing patterns in molluscan pacemaker cells.

It seemed like a simple thing to do. The principal channel types were well characterized with published voltage clamp data from the sea slugs *Tritonia* and *Aplysia californica*. GENESIS had all the features that I needed to implement a single-compartment model with six varieties of conductances and a calcium diffusion mechanism. That was my introduction to the difficulties and complexities of parameter searching and model comparison. My model was a "generic burster," loosely based on an *Aplysia* R15 neuron, with channel data taken from both *Tritonia* and *Aplysia* cells under different conditions.

I soon realized, along with many of the students that I tutored over the years in the MBL neural modeling courses and future ones in the EU Advanced Course in Computational Neuroscience and the Latin American School on Computational Neuroscience (LASCON), that parameter fitting is the most time-consuming task of single cell modeling. This requires scaling several conductance densities, shifting activation curves to account for different rest potentials, and varying time constants to account for temperature variations. Varying these many parameters in order to fit firing patterns obtained under current clamp conditions remains a difficult task today.

By the time I was happy with my model and tutorial, far better models of the *Aplysia* R15 neuron had been published (e.g., Canavier et al. 1991), but the tutorial and the one that I based on a GENESIS recreation of the Traub et al. (1991) hippocampal pyramidal cell, are still the best way I know of to get a feel for the role of the various conductances by modifying them within a GUI. The most sophisticated GENESIS single cell tutorial is the Purkinje cell tutorial, developed by Hugo Cornelis.

The experience of converting the Traub model and other published neural models to GENESIS revealed another sobering aspect of model replication and comparison. Of the many published descriptions of models that I have attempted to reimplement, I can think of very few that did not have errors or significant omissions. As discussed later, I feel that this stems from the limitations of the present system of publishing model-based research. Thus, there may be some parameter searching involved even when replicating an existing model, as well as the very important matter of making a meaningful comparison between the results from different implementations of what is ostensibly the same model.

A parameter search often involves doing the very easiest form of parallel computing: running many separate uncoupled simulations with different sets of parameters. One evening in the computer lab during the 1991 Woods Hole course, the students discovered that their simulations had suddenly slowed down to a crawl. It turned out that Erik De Shutter was doing a parameter search on the Purkinje cell model by running a background simulation on every workstation. I believe that he obligingly consented to cease, although John Uhley may have threatened actual physical violence.

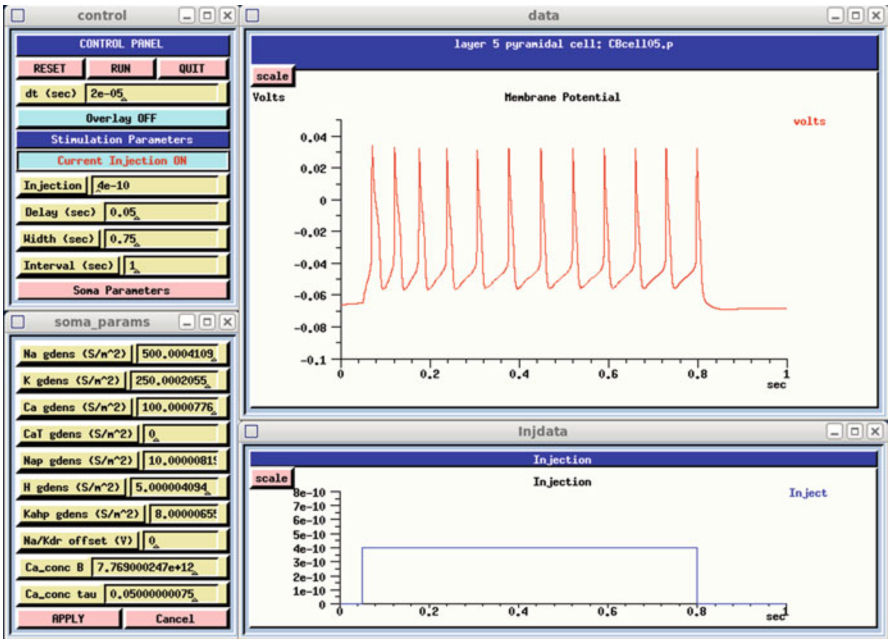
To address the problem of comparing the results of dendritic cable model simulations when run on different simulators, Bhalla et al. (1992) developed the Rallpacks set of benchmarks. These demonstrated that GENESIS and NEURON had equivalent speed and accuracy for these models. Shortly later, GENESIS gained a number of parameter search commands that were used for tuning the olfactory bulb mitral and granule cell models of Bhalla and Bower (1993). Vanier and Bower (1999) performed a detailed study of a variety of automated parameter search methods, using the GENESIS parameter search library developed by Vanier. In most cases the simulated annealing algorithm gave the best performance.

There are, however, some pitfalls in performing an automated search. In addition to the time consumed by searching unproductive regions of a large parameter space, there can be multiple regions that give equivalent local minima in the error function for the fit.

In order to know where to start a search, one needs to have an understanding of the roles that many different ionic currents play in the timing of action potentials, in order to have a sense of which parameters are most relevant. For example, knowing the role that the "H-current" plays in producing an overshoot in the membrane potential after a hyperpolarizing current injection can help define the area in parameter space to be searched.

I have found it most helpful to begin with a manual search, starting with the best available data for initial values. By varying the parameters by hand, using a custom GUI scripted with GENESIS/XODUS, and plotting the results, I can find a much better set of initial parameters for an automated search. Figure 3.2 shows such an interface for tuning a simple pyramidal cell model.

The largest problem when fitting parameters to current clamp experiments is the same as the one when comparing the results of two different simulations. Simply matching the positions of action potentials is not a sufficient condition to judge when a simulation agrees with experimental results, or those of another model, unless the model is a tonically firing one, such as the axon model used in the Rallpacks (Bhalla et al. 1992) set of benchmarks. The difficult problem of



dt (sec) 2e-05
Injection 4e-10
Delay (sec) 0.05
Width (sec) 0.75
Interval (sec) 1

Na gdens (S/m^2) 500.0004109
K gdens (S/m^2) 250.0002055
Ca gdens (S/m^2) 100.0000776
CaT gdens (S/m^2) 0
Nap gdens (S/m^2) 10.00000815
H gdens (S/m^2) 5.000004094
Kahp gdens (S/m^2) 8.0000065!
Na/Kdr offset (V) 0
Ca_conc B 7.769000247e+12
Ca_conc tau 0.05000000075

Fig. 3.2 Custom scripted GUI for adjusting channel parameters to fit response to a 0.4 nA current injection pulse to a model layer 5 pyramidal cell

reproducibility in computational neuroscience is addressed in detail in another chapter in this volume by Crook et al. (2013).

The distinction between incorrect results and those in "reasonable agreement," is particularly difficult to make in the case of cells that display spike frequency adaptation or bursting behavior. These models contain slow hyperpolarizing currents (e.g., Muscarinic or AHP) that are active near the threshold voltage for an action potential. Thus the membrane potential can hover about threshold, and these currents can have the effect of magnifying the effect of small deviations between two numerical solutions that can push the balance in one direction or the other.

Figure 3.3 shows the membrane potential for a model (Traub et al. 1994) of a burst-firing hippocampal pyramidal cell under conditions with two slightly different numerical precisions.

The upper plot, shown with a prototype Python plotting module for G-3, shows the result of a current injection over a 0.2 s interval. These small deviations eventually cause significant differences in the position of the final spike of the burst. When plotted over a 5 s period (below), the bursts have roughly the same time intervals, but drift in and out of coincidence with each other. One would call these "equivalent results," but it is difficult to quantify the differences in a meaningful way. Baldi et al. (1998) have addressed this problem by suggesting the use of Bayesian inference in the comparison of spike trains. However, making quantitative comparisons of this nature remains a largely unsolved problem.

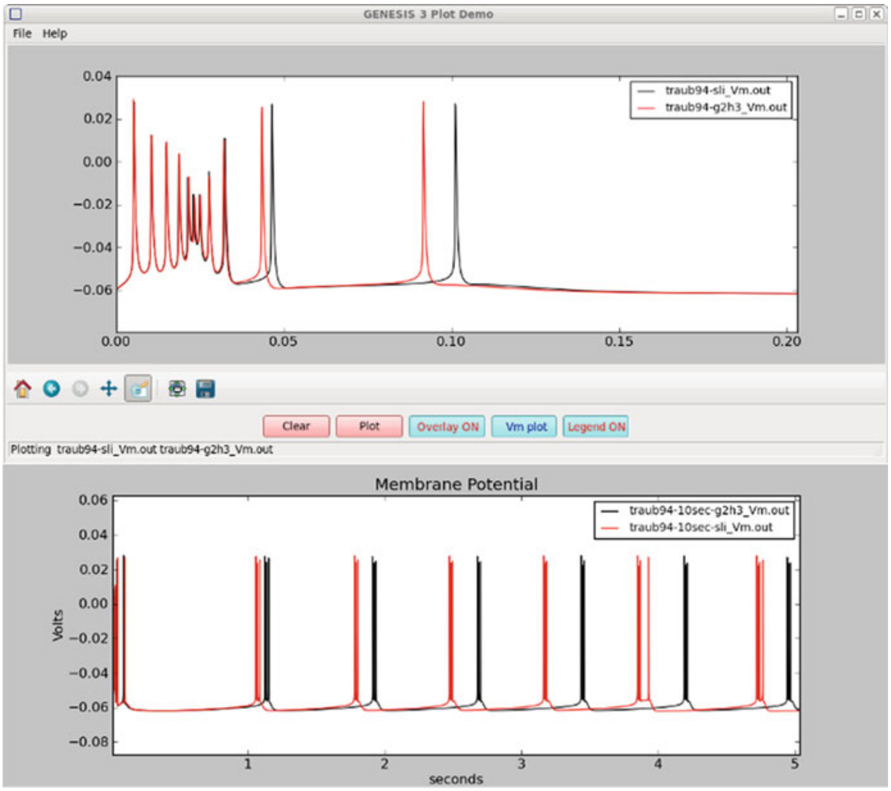


Fig. 3.3 Two simulation runs of a burst firing pyramidal cell with slightly different numerical precision. *Upper plots*: Membrane potential during the first 0.2 s. *Lower plots*: Membrane potential during 5 s of a longer run

The "Decade of the Brain" and the Human Brain Project

The US Congress established the 1990s as the "Decade of the Brain," and 16 federal agencies, including the NIH, issued program announcements in April 1993 and again in October 1995 soliciting proposals for the Human Brain Project. The research to be supported would develop informatics tools for accessing and integrating the huge amounts of data produced by neuroscience research, with an emphasis on web-based databases for data sharing (Koslow and Huerta 1997). Suddenly "neuroinformatics" became a popular word in research proposals. A great many of these proposals involved brain atlases and dealing with the huge data sets produced by neuroimaging experiments. However, there were many opportunities offered to neural modelers and simulator developers. The development of realistic neural models can benefit, not only from model sharing, but from the development of tools for managing notes and model development histories, and for linking models to experimental data and bibliographic references.

The initial goals of the SenseLab project at Yale University (Shepherd et al. 1997) were concerned with creating a comprehensive database of information about the olfactory system and tools for access. The system was to be built upon a commercial object-oriented database (OODB) called Illustra, and included an olfactory receptor database (ORDB), a database of neuron descriptions (NeuronDB), and a database of computational models of olfactory and other neurons (ModelDB).

The GENESIS group was awarded a grant in the first round to develop a "GENESIS Simulator-Based Neuronal Database" that would use an OODB with a query interface to mine the information contained within GENESIS simulations of a model neuron or network, and to link it with model descriptions, relevant data, and reference materials (Beeman et al. 1997). The initial choice of database was the commercial database UniSQL, and later prototypes were implemented with one called ObjectStore.

Two issues came up regularly at the annual Human Brain Project principal investigators meeting at the NIH in Bethesda. The question of how to best represent, store, and exchange models was a continuing theme, as well as what particular database to use. OODBs were in vogue at the time and were widely used in business software (Loomis 1995). An object-oriented representation was natural for describing neural models, and particularly for GENESIS models. However, the available OODBs were commercial products with proprietary data formats. The open-source options at that time were earlier incarnations of the relational databases PostgreSQL and MySQL. It is worth noting that none of the three commercial products mentioned above are in existence today, but PostgreSQL and MySQL are still widely used and available for a wide variety of platforms.

The other trend in the world of business software at that time was the use of SGML for representation of a large variety of data objects in a "document." A "document" could be, and often was, a description of a textual document, such a one described by a subset of SGML, HTML. The use of SGML in the publishing industry was well known. However, the use of SGML gave a powerful object-oriented description of items that could also be represented with OODBs, and it was often used as an interchange format between OODBs. An SGML document could just as well be a collection of data relating to the inventory of a business, a collection of customer contact information, or perhaps a description of a neural model.

Unfortunately, the SGML specification was needlessly complex to use or implement parsers for, and not very standard. Preliminary discussions by the World Wide Web Consortium (W3C) of a simpler standard more suitable for use with the WWW called the eXtensible Markup Language (XML) were underway at that time, but the XML 1.0 specification would not become a W3C recommendation until February 1998.

In March of 1996, Michael Arbib invited HBP participants with an interest in neural modeling to a "Workshop on Brain Models on the Web" held at the University of Southern California. His group had an HBP grant to develop a web-accessible database of neural models and to construct tools for sharing and exploring models and associated data that would be contributed by other modelers. The neural simulation system in use was NSL (Weitzenfeld 1995), a simulator for large networks of point integrate and fire neurons, and it would be advantageous to incorporate more realistic models generated with GENESIS and NEURON. These were to be stored in an Illustra OODB. Michael Hines and Matt Wilson had discussed the conversion between NEURON NMODL and GENESIS SLI scripts since 1989, and made some initial steps towards a conversion program. But, the problem was difficult because of the very different representations used in the two simulators.

The GENESIS ".p" format provides a machine-readable description of a branched compartmental cell model with active conductances. It can easily be translated to other formats. However, the channel names that appear in the file are names of elements that are created with GENESIS scripts for the SLI. Figure 3.4 is taken from a slide given at my presentation on the use of SGML for model representation, showing a fragment of an NMODL script and one of a GENESIS SLI script. Someone familiar with both simulators would recognize that both represent the same Hodgkin–Huxley model of the squid giant axon potassium channel, with one using physiological and the other SI units, but it is hard to imagine a machine translation between the two formats and their many possible variations.

The ModelDB project (Migliore et al. 2003) was one of the most successful and well known of those to come out of the HBP, primarily because of its simplicity. Rather than take the path of many others that attempted to put models into some OO format, it simply stores simulation scripts in the native simulator languages along with documentation, and is well indexed. However it suffers from the problem that the scripts are not portable, sometimes even to later versions of the simulator on which they were developed. This is because the simulators use *procedural* scripting languages that give a sequence of instructions telling the simulator how to construct a model, rather than *declarative* representations that describe the model, leaving it to the simulator to determine how it should be created in the context of the simulator implementation and its data structures.

The key to model sharing would be to translate this scripted procedural representation to a declarative object-oriented representation. I tried to argue persuasively that an SGML representation was preferable to storage in an OODB, but I don't think that anyone was convinced. My notes from the meeting say that Michael Hines was "pessimistic about the possibilities of representing a simulation outside of the structure of the simulation code." By the year 2000, when XML became widely known and open-source parsers were available, it would seem obvious that describing the objects with XML and storing XML files in a generic (and replaceable) relational database would be the best solution.

Model Sharing and Simulator Interoperability

During the final phase of the Human Brain Project, the focus of the GENESIS group turned towards creating the Modeler's Workspace (MWS). The MWS (Forss et al. 1999; Hucka et al. 2002) was a design for a graphical environment for constructing and exploring neuron and network models with simulations, experimental data, and bibliographic material. It would also allow collaborative development of models.

a NMODL example (code fragments from hh.mod)

```
COMMENT
  This is the original Hodgkin-Huxley treatment for the set of sodium,
  potassium, and leakage channels found in the squid giant axon membrane.
  ("A quantitative description of membrane current and its application
  conduction and excitation in nerve" J.Physiol. (Lond.)  117:500-544 (1952).)
ENDCOMMENT

BREAKPOINT {
        SOLVE hh METHOD runge
}

DERIVATIVE hh {
        v' = -1e3/cm * (ina + ik + il) - stim
        aux()
        rates(v)
        m' = (minf - m)/mtau
}

PROCEDURE aux() {
        gk = gkbar*n*n*n*n
        ik = gk*(v - ek)
        il = gl*(v - el)
}

PROCEDURE rates(v) {
        LOCAL  alpha, beta, sum
        .
        .
        :"n" potassium activation system
        alpha = .01*vtrap(-(v+55),10)
        beta = .125*exp(-(v+65)/80)
        sum = alpha + beta
        ntau = 1/sum
        ninf = alpha/sum
}
```

b GENESIS example (code fragments from hh_tchan.g)

```
/* FILE INFORMATION
   squid giant axon voltage-dependent channels,
   A.L.Hodgkin and A.F.Huxley, J.Physiol(Lond) 117,
   pp 500-544 (1952)

   The function setupalpha uses the form (A+B*V)/(C+exp((V+D)/F))
   for the rate variables alpha and beta to fill the channel tables
*/

// CONSTANTS
float EREST_ACT = -0.07      // resting potential (volts)
float EK        = -0.082     // potassium equil potential
float SOMA_A = len*PI*dia    // compartment area

function make K squid hh
    str chanpath = "K squid hh"
    create tabchannel {chanpath}
    setfield ^ Ek {EK} Gbar {360.0*SOMA A} Ik 0 Gk 0  \
           Xpower 4 Ypower 0 Zpower 0

    setupalpha {chanpath} X {10e3*(0.01 + EREST_ACT)} \
        -10.0e3 -1.0 {-1.0*(0.01 + EREST_ACT)} -0.01 \
        125.0 0.0 0.0 {-1.0*EREST_ACT} 80.0e-3
end
```

Fig. 3.4 Fragments of simulation scripts for a Hodgkin–Huxley potassium channel (**a**) NEURON NMODL script (**b**) GENESIS SLI script

The GENESIS HBP funding was to develop a prototype user interface for the examination and sharing of neural models with related metadata, rather than to enhance the functionality of the simulator core. Nevertheless, under continuing NSF funding, we were also looking forward to a major reorganization of GENESIS that would have the modularity and interoperability required to support the functionality of the MWS design.

As a key component, the MWS design contained an XML-based representation of cell and channel models. By the late 1990s, several other groups were also using XML-based descriptions of neuroscience-related data and models.

Daniel Gardner's group at Cornell had an HBP-funded effort to create an XML-based Common Data Model for the exchange of neurophysiology data and related metadata (Gardner et al. 2001). Although this was not intended for model description, it was very influential in our design of what eventually became a large part of NeuroML.

In late 1999, Michael Hucka, the main developer for the Modeler's Workspace Project, began working with what was then called the ERATO project at Caltech to develop SBML, the Systems Biology Markup Language (http://sbml.org). This team, consisting of Hamid Bolouri, Andrew Finney, and Herbert Sauro, was developing an infrastructure for computational modeling in systems biology. It faced many of the representation and design issues of the MWS project, and there was a great deal of overlap in the design of SBML and the MWS XML description of neural models (Hucka et al. 2003).

In June 2000, Hucka circulated the first draft of the notation used by the MWS for model descriptions (http://modelersworkspace.org/mws-rep/mws-rep.html) to the ERATO project and to Nigel Goddard's group in Edinburgh, who were working on model representations for their (now-defunct) simulator NEOSIM (Goddard et al. 2001a).

At about that time, Hugo Cornelis was finishing his Ph.D. thesis in Computer Science, while working in the laboratory of Erik De Schutter in Antwerp. Needing to develop a user-friendly declarative interface to the Hines method solver in GENESIS 2, he developed the Neurospaces model-container and the Neurospaces Description Format (NDF) for single neuron and network model representation in 1999 (Cornelis and De Schutter 2003). He described it in a meeting with the Goddard group in summer 2000 and, with Fred Howell, wrote a first draft of a proposal to use it as a representation for NEOSIM.

In a collaboration between these groups, a paper describing the initial specification for NeuroML (http://www.neuroml.org) was submitted in December 2000. After much discussion and further revision during a meeting in Edinburgh in early 2001, the representation was further clarified, incorporating the MWS cell and channel representations with the NEOSIM network-level representations (Goddard et al. 2001b).

In 2002, Fred Howell and Robert Cannon implemented the NeuroML Development Kit in Java, with classes corresponding to an extended version of the XML schema described in Goddard et al. (2001b), and tools for parsing NeuroML files. This was then used to implement a prototype MySQL database and Java-based GUI for to retrieve GENESIS ionic conductance models described with NeuroML, and convert them to procedural GENESIS SLI scripts (Beeman and Bower 2004).

The next significant application of NeuroML was the neuroConstruct project, initially begun in 2004 as a tool with a GUI for creating network models with

NEURON. NeuroConstruct (Gleeson et al. 2007) is implemented in Java, and the latest version (http://neuroconstruct.org/) uses the current NeuroML specification (Gleeson et al. 2010), which incorporates the MorphML (Crook et al. 2007) schema for cell morphology description. Rather than being a simulator, it provides an environment for creating large networks of biologically realistic neurons with complex connectivity patterns, using NEURON-, GENESIS-, MOOSE-, PSICS-, and PyNN-based simulators to perform the actual simulations. This is accomplished by generating simulation scripts for these simulators in their native scripting languages.

The International Neuroinformatics Coordinating Facility (INCF) has formed a program to develop another standardized description language for spiking neuronal network models, the Network Interchange for Neuroscience Modeling Language (NineML). NineML (Gorchetchnikov 2010; Raikov 2010; http://nineml.org) incorporates features of NeuroML and SBML and is based on a layered approach. An abstraction layer allows a full mathematical description of the models, including events and state transitions, while the user layer contains parameter values for specific models. There are frequent discussions between the NineML and NeuroML groups. It is likely that there will be some convergence between the two standards.

In the fall of 2005, Cornelis joined the Bower lab in San Antonio as a postdoctoral student, and began the integration of Neurospaces into G-3 as its internal data representation format and model container. Hucka is now the Team Leader and Chair of the SBML editors for the SBML project, working on all aspects of SBML development.

Choice of Programming Languages

In retrospect, the choice of C as the programming language for GENESIS and NEURON during the mid-1980s may seem like an obvious decision. But, there were other alternatives that could have been chosen, and C was a fortunate choice. At this time most scientific computations were performed in FORTRAN, running on mainframe computers without graphics. Graphical displays were available on workstations made by a variety of manufacturers including DEC, Xerox, Evans and Sutherland, Apollo, and Xerox, as well as Sun Microsystems and Silicon Graphics. These typically had their own specialized software libraries and software to take advantage of their hardware. C was the standard language for Unix-based systems, and the X Window System was just beginning to emerge as a standard hardware-independent protocol for the display of graphics.

There were a number of dialects of C, and much early software, including GENESIS, used the original informal specification by Kernighan and Ritchie (1978) that is often called "K&R C." The first standard for what became known as "ANSI C" was not adopted by the American National Standards Institute until 1989, and adopted by the International Organization for Standardization in 1990. During the 1990s, a great deal of time was spent updating and slowly "ANSIfying" the GENESIS base code in order to guarantee that it would compile under the various workstation operating system implementations of Unix, such as SunOS, Solaris, Irix, Ulttrix, and HPUX.

However, the US government was pushing strongly for the use of the language Ada. Although Pascal was originally proposed as a teaching language, it was becoming popular, and there were inexpensive Pascal compilers and interpreters widely available for personal computers. Many computer scientists favored Algol or Modula-2, or "fifth generation" languages such as Prolog or Lisp. In fact, a notable single neuron simulator, Surf-Hippo was written in Lisp (Borg-Graham 2000).

Many IBM mainframe computers ran a proprietary language PL/1/, and DEC minicomputers such as the PDP8 used FOCAL. Although object-oriented C++ had been under development since 1983 as an extension of C, standardization was slow to come and the C++ programming language standard was not ratified until 1998.

During the development of a neural simulator, it is obviously of great importance to "pick a winner" among emerging software standards for programming languages and graphical packages. The approach taken by GENESIS was a conservative one, using standard C and X libraries, at the cost of having to write much of the "middle-level" software such as the SLI and XODUS to connect low-level function calls to high-level commands in a scripting language. NEURON has tended to use available software packages (HOC, NMODL, InterViews, etc.) as an easy way to gain functionality, whereas GENESIS has tended to develop its own built-in libraries or modules. Thus, the SLI, XODUS, and the original mailing list management software were written "in-house," based on widely accepted standard low-level Unix and X libraries. Each of these approaches has both advantages and disadvantages.

Open Source vs. Proprietary Software

Many of the HBP-funded database projects elected to use proven, professionally developed commercial software, rather than less stable free open-source software. However, most of these companies and their software with proprietary data formats no longer exist. Today it is much easier to make an argument in favor of using open-source software to avoid dependence on a closed platform that may not exist in the future. However, it is certainly an advantage to have someone else do the hard work, and it is a safe bet that tools such as MATLAB and their data formats will be around for a long time.

However, the choice of an open-source package also presents a dilemma. Is it best to incorporate the needed code into the simulator, or rely on loadable libraries that are maintained elsewhere and distributed from an Internet-accessible repository? Obviously the latter is the easiest route, if only one can guarantee that the package will not become like so many orphaned software projects on Sourceforge.net. If the package is not large, at least the source code will be available to be incorporated and maintained by the simulator developers.

The GENESIS experience with the NetCDF package from Unidata is an example of a decision that wasn't entirely optimal. GENESIS has a very fast and efficient binary file format (FMT1) for outputting the state (virtually any variable of interest) of the neurons in a large network at specified time intervals. However, it is platform-dependent and cannot be reliably used for exchanging files between different computers. The NetCDF package provided an easy way to give GENESIS a platform-independent output format with many other features. However, it is a very large package with a great deal that is not relevant to GENESIS. The NetCDF license allowed the incorporation of parts of the code into GENESIS with proper attribution, so it was decided to make a GENESIS "netcdflib" from parts of NetCDF. During the early years of patching GENESIS in order to compile on the many variants of the Unix operating system, maintaining netcdfllib became a tiresome task. Although the compilation of netcdflib is optional in GENESIS 2, it now runs (more slowly than the default FMT1) on all the major Unix variants. NetCDF is currently still maintained by Unidata, and updated packages are available. In retrospect, life would be simpler if netcdflib were a library maintained by Unidata, or if we had written our own software with a NetCDF-compatible format, or picked another widely used and supported format.

NEURON relied heavily on open-source software packages that might now be considered orphans, but with few ill effects. HOC and MOD (from which NMODL was developed) are used nowhere else but in NEURON. The Unix InterViews package upon which the NEURON GUI is based had its last major release in 1993. As long as these are part of NEURON and can be maintained with the rest of the code, it does not matter if they are otherwise unsupported.

In 2011, there are very powerful and complex graphical libraries available that are popular and being extensively developed. Today, no one would think of writing a programmable GUI such as XODUS from scratch, using basic X Window System function calls. For example, G-3 makes use of the platform-independent libraries for wxWidgets and the Python tool Matplotlib. Compiled binary libraries for nearly any operating system are maintained and may be updated from Internet repositories. Using these is a great advantage, but there is always the gamble of betting on a loser that will fall from favor and no longer be maintained.

Some Identified Problems with Past Simulators

Scripting a simulation with SLI or HOC is difficult because they each have an idiosyncratic syntax that must be learned. They lack the generality and the data structures available in more general purpose programming languages, and have no supported external libraries that can be used to easily gain additional functionality. Creating a custom GUI for a simulation or a tutorial with SLI or HOC is even harder, because of the lack of built-in tools for designing GUIs. Modelers also require that a simulator provide a large variety of tools for visualization, analysis, or model construction, in addition to fast and accurate simulation of the model.

Increasingly, neural simulations need to cover many scales of modeling, ranging from the subcellular level of biochemical kinetics and diffusion modeling, through single cell, network, and system modeling. Specialized simulators exist that are an excellent solution at a particular level. For example, MOOSE provides very good capabilities for biochemical kinetics modeling, based on the GENESIS 2 kinetics library component. MCell (Stiles and Bartol 2001) uses Monte Carlo methods to model diffusion and stochastic activation of synapses. Due to the lack of interoperability between simulators, it has generally required extending the simulator in order to include similar capabilities.

As mentioned in the section on parameter search, model tuning, and comparison, publication of the results of a simulation in a way that allows the results to be reproduced or compared with the results from other models is a nearly impossible task. This is true even if the source code for the simulation is made available. This is because there is no way to track either the micro-evolution of a model during a single published research project or the macro-evolution of a model across longer time periods and multiple publications by multiple researchers. There is no guarantee that the model parameters that were used to generate a particular figure are the same as those described in the paper or used in the provided simulation scripts. In the traditional scientific process the actual records of day-to-day activity are kept in research notebooks or log books. In the case of computational modeling, a model may be incrementally modified with an incomplete record kept of all changes. Most neural simulators provide facilities for keeping "Notes" files, but not a comprehensive system for tracking model evolution using a verifiable digital description of the model and the exact conditions for the simulated experiments.

The monolithic architecture of simulators that were developed during the last century makes it difficult to add "plug-in" software components to the simulator without making major changes to the core simulator code. For example, GENESIS 2 has various components such as the cell reader, the hsolve compartmental solver, the SLI parser, and XODUS. However these were not implemented in a modular fashion and cannot be used as stand-alone software components.

By the end of the twentieth century, the limitations of the 1980s software that was the basis for GENESIS, NEURON, and other simulators were becoming apparent. Although emerging standards for declarative model descriptions showed promise for simulator-independent model sharing, the fundamental barrier to simulator interoperability and collaborative model development was the monolithic architecture of the simulators. The only way to communicate with GENESIS or NEURON was via an exchange of files. Not only does this limit the ability of two simulations to closely interact in real time, but it forces an "all-or-nothing" approach to the use of tools to control the simulation, provide stimuli, or analyze and display the results. Although GENESIS has many built-in tools for spike train analysis of the output of single cell models, this can easily be done with external tools such as Matlab or custom tools written in Python, if spike times are sent to a file for post-run analysis. Modelers have long used tools such as Matlab or various plotting programs to analyze the results of simulations. New tools are being developed specifically to aid neural simulation. However, they presently communicate with simulators indirectly by loading simulation scripts and communicating via files.

The problem comes if one wants to connect them more directly to a simulation during the run, without resorting to communication via data files. When calculating extracellular potentials or simulated EEG and MEG recordings from a large network model, it is necessary to sum channel currents from many compartments of very many cells. Unless the analysis application and the simulator have been designed to communicate with each other, the only other alternative is to generate enormous data files that give meaningful results only when the simulation is over, and no feedback while it is running. Although long "production runs" may be performed non-interactively, it is useful during the exploratory phase, and essential for educational tutorials, to perform these operations while the simulation is running. Thus, GENESIS 2 and NEURON each have a great deal of code that is devoted to built-in tools that are not concerned with specifying and simulating a model. Ideally, there should be no need to have a morphology file converter or spike train analyzer encapsulated within a simulator. It would be much better if these tools were implemented as external simulator-independent plug-in modules.

The Twenty-First Century: Next Generation Neural Simulators

By the beginning of the twenty-first century, the basic simulation capabilities of GENESIS and NEURON had reached maturity, although there were continuing improvements, leading up to the latest releases of GENESIS 2.3 (May 2006) and NEURON 7.1 (October 2009). Recent development efforts have shifted to providing other capabilities for the analysis or construction of models. For NEURON, the focus has been on better integrated graphical tools, such as Cell Builder, Channel Builder, Kinetic Scheme Builder, and built-in tools, for parameter fitting and importing cell morphology files, as well as model import functions for NeuroML and other standard declarative formats. As described below, the approach taken by G-3 has been somewhat different.

I believe that the twentieth century simulators described above and their architecture have reached the end of their life cycles, and it is time for a new generation of modular, interoperable realistic neural simulators. These should be built upon modern software designs, with care to pick standards, formats, and externally developed packages that will be in existence 20 years from now.

It may be useful to briefly describe here some simulators that have been developed or extended during the present century. Brette et al. (2007) reviewed eight currently available simulators that are capable of modeling networks of spiking neurons. In addition to GENESIS and NEURON, this capability is present to one degree or another in NEST (Eppler et al. 2008), NCS (Drewes et al. 2009), CSIM (http://www.lsm.tugraz.at/csim/), SPLIT (Hammarlund and Ekeberg 1998), Mvaspike (Rochel and Martinez 2003), and XPPAUT (Ermentrout 2006).

NEST specializes in very large networks of neurons having one or a small number of compartments. It uses parallelism and has its own interpreted scripting language with no GUI. NCS, the NeoCortical Simulator, has an inherently parallel implementation and is used for large networks of multi-compartmental integrate and fire neurons.

CSIM and the Python version PCSIM (Pecevski et al. 2009) model large networks of point neurons that are typically integrate and fire, but may also include Hodgkin–Huxley channels. Rather than having its own GUI, it is controlled by Matlab, and more recently with Python.

SPLIT is designed for massively parallel simulations of networks of multi-compartmental neurons with Hodgkin–Huxley dynamics. It was recently used in a neocortical simulation with eight million neurons and four billion synapses performed on the Blue Gene/3 supercomputer (Djurfeldt et al. 2005). The user specifies the model in a C++ program, rather than using an interpreted simulation language. The program is linked to the SPLIT library, compiled, and run. It has a minimal GUI and no analysis tools.

Mvaspike is based on an event-based modeling and simulation strategy, mainly using pulse-coupled integrate-and-fire point neurons.

XPPAUT is in a class by itself. It is not so much a neural simulator, but an analysis tool for understanding the equations that are used in simulations of cells and small networks. The equations used are completely specifiable by the user and can be analyzed with bifurcation diagrams and similar phase plane representations.

Another new simulator, Brian (Goodman and Brette 2008) models networks of integrate-and-fire or Hodgkin–Huxley single or few compartment neurons. It is written entirely in Python, making it highly portable, easy to learn and use, and suitable for rapid prototyping of models. However, this prevents it from being as fast as other simulators that make use of compiled C or C++ libraries to perform most of the numerical calculations in a simulation.

PyNN (Davison et al. 2009) and MUSIC (Djurfeldt et al. 2010) are not simulators, but provide interfaces for existing simulators. PyNN, discussed below, provides a common Python-based scripting interface for many simulators. MUSIC is a C++ library implementing an API which allows large-scale neuronal network simulators to exchange data during run time. NeuroConstruct (Gleeson et al. 2007), described previously, is an environment for creating simulations that run under several simulators, using NeuroML as a declarative model description language.

The GPU-SNN simulator (Richert et al. 2011) models large networks of spiking Izhikevich model neurons, having spike-timing-dependent plasticity and short-term plasticity. It can run on an "off-the-shelf" Graphical Processing Unit (GPU) such as the Nvidia GTX-280 at speeds of up to 26 times faster than a CPU version for a simulation of 100,000 neurons with 50 million synaptic connections. It is written in C and C++ and has a Python-based user interface similar to PyNN.

MOOSE (http://moose.ncbs.res.in) is the Multiscale Object-Oriented Simulation Environment for large, detailed simulations including computational neuroscience and systems biology. It was developed by Upi Bhalla as a reimplementation of GENESIS 2 in a cleaner, more modular manner. Although it retains the GENESIS

2 use of objects that pass messages between them, it does so in a more efficient manner. It is completely rewritten in C++ and has a different architecture, with no old GENESIS 2 code except for the SLI parser definition. The SLI parser allows it to maintain a high degree of backwards compatibility with GENESIS 2 scripts, and the new Python interface (Ray and Bhalla 2008), allows scripting of simulations in Python.

The development of G-3 has taken a different path from that of the simulators described above. Rather than adding new features and capabilities to GENESIS, the functionality of its monolithic architecture has been completely reimplemented as a collection of independent software components. These, and other independently developed components may be used individually or in combination with others to perform the functions desired for running a particular simulation.

The modular CBI architecture used by G-3 (Cornelis et al. 2012a) is based on plug-ins and has multiple interfaces. This modularization provides a number of advantages for simulator development and for interoperability with other simulators across scales ranging from subcellular to systems level.

The clean separation of modules allows developers and users to choose to contribute to only a single component, instead of being exposed to the complexity of the entire simulator. Decomposition of an application into multiple software components not only allows reuse and extension of individual modules, facilitating both simulator and model development, but individual components can be independently updated, enhanced, or replaced when needed. The use of multiple parsers for scripting simulations allows G-3 users to maintain backwards compatibility with GENESIS 2, while making use of scripts written in Python or other new scripting languages. Modules can be run separately on different machines. For example, the GUI and modeling environment might be run locally, while the simulation is run remotely on more powerful, possibly parallel, machines.

Some of the more relevant G-3 components for creating and running a simulation are:

- The Neurospaces Model Container (NMC) contains the biological model description and separates it from the details of the implementation.
- Multiple solvers perform numerical calculations and allow highly efficient solvers to be implemented for particular model objects.
- The Experiment component provides experimental protocols for applying stimuli, or for recording and analyzing the model behavior.
- A scheduler (SSP in Perl or SSPy in Python) binds the contents of the NMC with needed solvers and experimental protocols, and runs the simulation.
- The G-shell (or the new Python shell) provides a console for issuing interactive commands.
- G-Tube provides a GUI for running G-3 simulations.
- Studio allows the visualization of models in the Model Container.
- NS-SLI provides backwards compatibility with the GENESIS 2 SLI.
- The Exchange component provides model exchange using common standards such as NeuroML and NineML.

- The G-3 Documentation System not only provides user and developer documentation on all aspects of G-3, but is the basis for the model publication system.

Heccer is the default fast implicit numerical solver for compartmental models. It transparently incorporates the hsolve object of GENESIS 2. The Discrete Event System (DES) component is a separate solver used for delivering spike events in network simulations. The Chemesis-3 solver is a numerical solver optimized for the solution of reaction–diffusion equations (Blackwell 2000). The use of these separate numerical solvers for different types of models allows improved optimization over that obtained by the generic solver used by GENESIS 2. It also allows the use of multiple simulation engines to perform the numerical calculations of the simulation. In principle, the solvers of simulators such as NEURON and MOOSE could be used along with the G-3 solvers to perform parts of a simulation. Keeping the declarative model description separate from the simulator-dependent solver facilitates the exchange and reuse of models. The use of separate parsers for simulator commands allows simulations to be constructed with the G-shell, NS-SLI for GENESIS 2 scripts, or SSPy for scripts written in Python.

From the standpoint of a modeler constructing a simulation, G-3 preserves the GENESIS 2 paradigm of creating simulation objects that exchange messages during a simulation. However, unlike the approach taken with MOOSE, G-3 does not internally use objects with messages. This allows highly efficient numerical methods to be used without resort to “hacks” such as the GENESIS 2 hsolve object.

The G-3 model-based publication system (Cornelis et al. 2010) addresses the limitations of current paper and digital publications, by providing model comparison tools, model lineage inspection tools, and model verification tools. This is intended to lay the ground work for making models, rather than, as at present, the written description of models, the basis for scientific publication in neuroscience. The Publication System is designed to be platform independent as it adheres to the CBI federated software architecture (Cornelis et al. 2012a).

Choice of Python as a Scripting Language

The term *scripting language* is often used for a programming language that is used for control over applications or as a “glue” that links compiled libraries that are written in other languages designed for efficient numerical computations. Scripting languages are usually interpreted, for run-time interaction with the user, and are designed to be easily writeable and modifiable by the user. This is in contrast to the programming language that is used for the implementation of the core simulator functionality. In the past, simulators have used simulator-specific scripting languages, such as SLI or HOC.

As early as 1999, Michael Vanier was working on PyGenesis to provide a better object-oriented declarative scripting language than the SLI syntax used with GENESIS 2. However, the monolithic architecture of GENESIS 2 prevented this from being an easily interfaced plug-in module, and it was never officially released. Further development of Python interfaces to GENESIS was postponed until recently, when it became possible to use them as plug-in components of G-3 (Cornelis et al. 2012b).

Python has recently become very popular as a scripting language for neurosimulators because it has a far simpler syntax than other languages such as Perl, with very flexible object-oriented capabilities and powerful built-in data structures. It also has a wide variety of well-supported open-source libraries for scientific computing and graphical display. For example, NumPy (http://numpy.scipy.org) provides arrays and fast matrix manipulation tools, and matplotlib (http://matplotlib.sourceforge.net) can duplicate most of the functionality of Matlab. These modules are widely used for neuroscience data acquisition and analysis (Spacek et al. 2009). Although it may be considered a procedural language, the ability to create purely declarative representations of models using Python objects makes it a good choice as a scripting language for OO model descriptions.

The PyNN project (Davison et al. 2009) attempts to provide a common Python-based scripting interface for nearly any neural simulator, allowing a mixture of Python and native simulator code. Many of the neural simulators described in the previous section have developed Python interfaces that aim for some degree of compatibility with PyNN, including G-3 (Cornelis et al. 2012b), NEURON (Hines et al. 2009), MOOSE (Ray and Bhalla 2008), and NEST (Eppler et al. 2008).

Conclusion: What Have We Learned?

In the preceding narrative history of neural simulator development, certain issues arose repeatedly. A summary of these may provide some guidance for future simulator development.

The advantages of using a well-supported simulator rather than dedicated simulation-specific code have now been widely recognized. However, the choice of a particular simulator usually means a commitment to spending time learning the details of using that simulator. That tends to lock the user into that choice and discourage the use of another tool more appropriate for the task. Modern simulator development has attempted to avoid this problem with efforts towards simulator interoperability and model sharing. New simulator architectures allow the use of standard, well-supported external modules, or specialized tools for neural modeling, that are implemented independently from the means of running the model simulation. This allows not only sharing of models, but sharing of research tools.

The OO paradigm of constructing models from basic reusable "objects" has now become nearly standard in current neural simulator interfaces, whether through a scripting language or a GUI. This trend has been encouraged by the development of standard declarative representations for models such as NeuroML and NineML, which are based on an inherently OO structure. However, there is more to be done in the way of standardization, and of representation of models that lie beyond the current capabilities of GENESIS 2 and NEURON.

Parameter search is a necessary part of modeling. Some simulators have implemented parameter search algorithms within the simulator. A more modular simulator architecture and the use of standard scripting languages can allow the use of more general purpose external simulator-independent search tools. Parameter searching is a very appropriate and simple use of parallelism, and simulator architectures should be designed for easy and transparent division of a simulation or many simulation runs over multiple computers or processors. It is also important to have the ability to easily add numerical solvers for new hardware devices such as powerful GPUs that were developed for display rendering and are now being used for high performance scientific computing.

As extensively discussed in the chapter by Crook et al. (2013) and earlier in this one, reproducibility of simulation results is hampered not only by insufficient, ambiguous, or inaccurate descriptions of the model in the original publication, but by sensitivity to implementation details, and dependencies on the computing environment. As with other identified problems with past simulators discussed earlier, the key to these problems seems to be a combination of standard declarative model descriptions, and modular simulator architectures that permit the use of external tools to perform the ancillary tasks of model tracking, publication, comparison, and parameter fitting.

The choice of a programming or scripting language and whether to use code developed in-house, open-source code, or proprietary commercial software has become simpler with the increased availability of well-supported open-source software packages. Their use has been made easier by attempts to increase the modularity of new simulator architectures and through the use of standard scripting languages such as Python. I have seen languages come and go in popularity. Perl is a powerful scripting language with very good string handling capabilities. It has long been a favorite scripting language of software developers and system administrators, but is difficult for the novice or occasional "script hacker." Not long ago, Java was everyone's favorite bet for a programming or scripting language with great promise to run on all platforms. Now Python is the favorite and is being challenged by other alternatives such as Ruby or C#, which also have OO capabilities, and can be used as a "glue" to provide access to compiled libraries within a script. C# (by Microsoft) and other new languages such as Go (by Google) are also "Internet aware," with built-in security and run-time execution distribution support, favoring modular architectures to prevent vendor lock-in. For these reasons, it is important to design a simulator so that its operations can easily be bound to a user's choice of scripting language. For example, the use of separate modules for a declarative model description, numerical solver, and command parser can reduce the dependency on a particular scripting language, as well as facilitate the exchange of models.

Fortunately for the future of computational neuroscience, the "lessons" mentioned above, and throughout this chapter, appear to be taken seriously by today's simulator developers. Annual workshops held at the Computational Neuroscience conference (http://cnsorg.org), by the NeuroML developers (http://neuroml.org), and the INCF (http://incf.org) bring together participants from all the major neural simulator and database projects. Standards and designs are vigorously debated, and progress continues.

Acknowledgment The author acknowledges support from the National Institutes of Health under grant R01 NS049288-06S1.

References

Alben R, Kirkpatrick S, Beeman D (1977) Spin waves in random ferromagnets. Phys Rev B15:346

Baldi P, Vanier MC, Bower JM (1998) On the use of Bayesian methods for evaluating compartmental neural models. J Comput Neurosci 5:285–314

Beeman D (1994) Simulation-based tutorials for education in computational neuroscience. In: Eeckman FH (ed) Computation in neurons and neural systems. Kluwer Academic, Norwell, MA, pp 65–70

Beeman D, Boswell J (1977) Computer graphics and electromagnetic fields. Am J Phys 45:213

Beeman D, Bower JM (2004) Simulator-independent representation of ionic conductance models with ChannelDB. Neurocomputing 58–60:1085–1090

Beeman D, Bower JM, De Schutter E, Efthimiadis EN, Goddard N, Leigh J (1997) The GENESIS simulator-based neuronal database (chap 4). In: Koslow SH, Huerta MF (eds) Neuroinformatics: an overview of the human brain project. Lawrence Erlbaum Associates, Mahwah, NJ, pp 57–80

Bhalla US (1998) Advanced XODUS techniques (chap 22). In: Bower JM, Beeman D (eds) The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System, 2nd edn. Springer, New York, pp 381–405

Bhalla US (2000) Modeling networks of signaling pathways (chap 2). In: De Schutter E (ed) Computational neuroscience: realistic modeling for experimentalists. CRC Press, Boca Raton, FL, pp 25–48

Bhalla US (2003) Managing models of signalling networks. Neurocomputing 52–54:215–220

Bhalla US, Bower JM (1993) Exploring parameter space in detailed single neuron models: simulations of the mitral and granule cells of the olfactory bulb. J Neurophysiol 69:1948–1965

Bhalla US, Iyengar R (1999) Emergent properties of networks of biological signaling pathways. Science 283:381–387

Bhalla US, Bilitch DH, Bower JM (1992) Rallpacks: a set of benchmarks for neuronal simulators. Trends Neurosci 15:453–458

Blackwell KT (2000) Evidence for a distinct light-induced calcium-dependent potassium current in Hermissenda crassicornis. J Comput Neurosci 9:149–170

Borg-Graham LJ (2000) Additional efficient computation of branched nerve equations: adaptive time step and ideal voltage clamp. J Comput Neurosci 8:209–226

Bower JM (1991) Relations between the dynamical properties of single cells and their networks in piriform (olfactory) cortex. In: McKenna T, Davis J, Zornetzer S (eds) Single neuron computation. Academic, San Diego, pp 437–462

Bower JM (1992) Modeling the nervous system. Trends Neurosci 15:411–412

Bower JM (2005) Looking for Newton: realistic modeling in modern biology. Brains Minds Media 1:bmm217 (urn:nbn:de:0009-3-2177)

Bower JM, Beeman D (1998) The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System, 2nd edn. Springer, New York, http://www.genesis-sim.org/GENESIS/bog/bog.html

Brette R, Rudolph M, Carnevale T, Hines M, Beeman D, Bower JM, Diesmann M, Morrison A, Goodman PH, Harris FC, Zirpe M, Natschläger T, Pecevski D, Ermentrout B, Djurfeldt M, Lansner A, Rochel O, Vieville T, Muller E, Davison AP, El Boustani S, Destexhe A (2007) Simulation of networks of spiking neurons: a review of tools and strategies. J Comput Neurosci 23:349–398. doi:10.1007/s10827-007-0038-6

Canavier CC, Clark JW, Byrne JH (1991) Simulation of the bursting activity of neuron R15 in *Aplysia*: role of ionic currents, calcium balance, and modulatory transmitters. J Neurophysiol 66:2107–2124

Carenvale NT, Woolfe TB, Shepherd GM (1990) Neuron simulations with SABER. J Neurosci Methods 33:135–148

Cole K (1968) Membranes, ions, and impulses: a chapter of classical biophysics. University of California Press, Berkeley

Connor JA, Stevens CF (1971) Prediction of repetitive firing behavior from voltage clamp data on an isolated neurone soma. J Physiol 213:31–53

Cornelis H, De Schutter E (2003) Neurospaces: separating modeling and simulation. Neurocomputing 52–54:227–231. doi:10.1016/S0925-2312(02)00750-6

Cornelis H, Coop AD, Bower JM (2010) Development of model-based publication for scientific communication. BMC Neurosci 11(suppl 1):P69. doi:10.1186/1471-2202-11-S1-P69

Cornelis H, Coop AD, Bower JM (2012a) A federated design for a neurobiological simulation engine: the CBI federated software architecture. PLoS One 7:e28956. doi:10.1371/journal.pone.0028956

Cornelis H, Rodriguez AL, Coop AD, Bower JM (2012b) Python as a federation tool for GENESIS 3.0. PLoS One 2:e29018

Crook S, Gleeson P, Howell F, Svitak J, Silver R (2007) MorphML: Level 1 of the NeuroML standards for neuronal morphology data and model specification. Neuroinformatics 5:96–104. doi:10.1007/s12021-007-0003-6

Crook S, Davison AP, Plesser HE (2013) Learning from the past: approaches for reproducibility in computational neuroscience. In: Bower JM (ed) 20 Years of computational neuroscience. Springer, New York

Davison AP, Brüderle D, Eppler JM, Kremkow J, Muller E, Pecevski D, Perrinet L, Yger P (2009) PyNN: a common interface for neuronal network simulators. Front Neuroinform 2:11. doi:10.3389/neuro.11.011.2008

De Schutter E, Smolen P (1998) Calcium dynamics in large neuronal models. In: Koch C, Segev I (eds) Methods in neuronal modeling: from ions to networks, 2nd edn. MIT Press, Boston, pp 211–250

Djurfeldt M, Johansson C, Ekeberg Ö, Rehn M, Lundqvist M, Lansner A (2005) Massively parallel simulation of brain-scale neuronal network models. Tech. Rep. QC 20100709. KTH, School of Computer Science and Communication (CSC), oai:DiVA.org:kth-10606

Djurfeldt M, Hjorth J, Eppler J, Dudani N, Helias M, Potjans T, Bhalla U, Diesmann M, Hellgren Kotaleski J, Ekeberg Ö (2010) Run-time interoperability between neuronal network simulators based on the MUSIC framework. Neuroinformatics 8:43–60. doi:10.1007/s12021-010-9064-z

Dodge FA, Cooley JW (1973) Action potential of the motor neuron. IBM J Res Dev 17:219–229

Drewes RP, Zou Q, Goodman PH (2009) Brainlab: a Python toolkit to aid in the design, simulation, and analysis of spiking neural networks with the neocortical simulator. Front Neuroinform 3:16. doi:10.3389/neuro.11.016.2009

Eppler JM, Helias M, Muller E, Diesmann M, Gewaltig MO (2008) PyNEST: a convenient interface to the NEST simulator. Front Neuroinform 2:12. doi:10.3389/neuro.11.012.2008

Ermentrout B (2006) XPPAUT. Scholarpedia 1(10):1399. doi:10.4249/scholarpedia.1399

Forss J, Beeman D, Bower JM, Eichler West RM (1999) The modeler's workspace: a distributed digital library for neuroscience. Future Gener Comp Syst 16:111–121

Gardner D, Knuth KH, Abato M, Erde SM, White T, DeBellis R, Gardner E (2001) Common data model for neuroscience data and data model interchange. J Am Med Inform Assoc 8:17–33

Getting PA (1989) Reconstruction of small neural networks (chap 6). In: Koch C, Segev I (eds) Methods in neuronal modeling. MIT Press, Cambridge, MA, pp 171–194

Gleeson P, Steuber V, Silver RA (2007) neuroconstruct: a tool for modeling networks of neurons in 3d space. Neuron 54:219–235

Gleeson P, Crook S, Cannon RC, Hines ML, Billings GO, Farinella M, Morse TM, Davison AP, Ray S, Bhalla US, Barnes SR, Dimitrova YD, Silver RA (2010) NeuroML: a language for describing data driven models of neurons and networks. PLoS Comput Biol 6(6):e1000–e1815. doi:10.1371/journal.pcbi.1000815

Goddard NH, Lynne KJ, Mintz T (1987) Rochester connectionist simulator. Tech. Rep. ADA191483. Department of Computer Science, University of Rochester

Goddard NH, Hood G, Howell FW, Hines ML, De Schutter E (2001a) NEOSIM: portable large-scale plug and play modelling. Neurocomputing 38–40:1657–1661. doi:10.1016/S0925-2312(01)00528-8

Goddard NH, Hucha M, Howell F, Cornelis H, Shankar K, Beeman D (2001b) Towards NeuroML: model description methods for collaborative modelling in neuroscience. Philos Trans R Soc Lond B Biol Sci 356:1209–1228. doi:10.1098/rstb.2001.0910

Goodman DFM, Brette R (2008) Brian: a simulator for spiking neural networks in Python. Front Neuroinform 2:5. doi:10.3389/neuro.11.005.2008

Gorchetchnikov A, The INCF Multiscale Modeling Taskforce (2010) Nineml: a description language for spiking neuron network modeling: the user layer. BMC Neurosci 11(suppl 1):P71. doi:10.1186/1471-2202-11-S1-P71

Gray CM, Konig P, Engel AK, Singer W (1989) Oscillatory responses in cat visual-cortex exhibit inter-columnar synchronization which reflects global stimulus properties. Nature 338:334–337

Hammarlund P, Ekeberg Ö (1998) Large neural network simulations on multiple hardware platforms. J Comput Neurosci 5:443–459. doi:10.1023/A:1008893429695

Hartree DR (1932) A practical method for the numerical solution of differential equations. Mem Manchester Lit Phil Soc 77:91–107

Hines M (1984) Efficient computation of branched nerve equations. Int J Biomed Comput 15:69–79

Hines M (1989) A program for the simulation of nerve equations with branching geometries. Int J Biomed Comput 24:55–68

Hines ML, Carnevale NT (2000) Expanding NEURON's repertoire of mechanisms with NMODL. Neural Comput 12:995–1007

Hines M, Davison AP, Muller E (2009) NEURON and Python. Front Neuroinform 3:1. doi:10.3389/neuro.11.001.2009

Hodgkin A, Huxley A (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol (London) 117:500–544

Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci USA 79:2554

Hucka M, Shankar K, Beeman D, Bower JM (2002) The Modeler's workspace: making model-based studies of the nervous system more accessible (chap 5). In: Ascoli G (ed) Computational neuroanatomy: principles and methods. Humana Press, Totowa, NJ, pp 83–115

Hucka M, Finney A, Sauro H, Bolouri H, Doyle J, Kitano H, Arkin A (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19:524–531

Kernigan BW, Pike R (1984) The Unix programming environment. Prentice-Hall, Englewood Cliffs, NJ

Kernighan B, Ritchie D (1978) The C programming language. Prentice-Hall, Englewood Cliffs, NJ

Kohn MC, Hines ML, Kootsey JM, Feezor MD (1989) A block organized model builder. Math Comp Mod 19:75–97

Koslow SH, Huerta MF (eds) (1997) Neuroinformatics: an overview of the human brain project. Vol: Progress in neuroinformatics research series. Lawrence Erlbaum Associates, Mahwah, NJ

Loomis ME (1995) Object databases—the essentials. Addison-Wesley, Reading, MA

Maley N, Beeman D, Lannin JS (1988) Dynamics of tetrahedral networks: amorphous Si and Ge. Phys Rev B38:10,611

Mcullough WS, Pitts WH (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5:115–133

Migliore M, Morse TM, Davison AP, Marenco L, Shepherd GM, Hines ML (2003) ModelDB: making models publicly accessible to support computational neuroscience. Neuroinformatics 1:135–139. doi:10.1385/NI:1:1:135

Nelson M, Rinzel J (1998) The Hodgkin-Huxley model (chap 4). In: Bower JM, Beeman D (eds) The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System, 2nd edn. Springer, New York, pp 29–49

Nelson M, Furmanski W, Bower JM (1989) Simulating neurons and neuronal networks on parallel computers (chap 12). In: Koch C, Segev I (eds) Methods in neuronal modeling. MIT Press, Cambridge, MA, pp 397–438

Pecevski D, Natschläger T, Schuch K (2009) PCSIM: a parallel simulation environment for neural circuits fully integrated with Python. Front Neuroinform 3:11. doi:10.3389/neuro.11.011.200

Pellionisz A, Llinás R, Perkel DH (1977) A computer model of the cerebellar cortex of the frog. Neuroscience 2:19–35

Perkel DH, Watt JH (1981) A manual for MANUEL. Stanford University Press, Stanford CA

Raikov I, INCF Multiscale Modeling Taskforce (2010) NineML: a description language for spiking neuron network modeling: the abstraction layer. BMC Neurosci 11(suppl 1):P66. doi:10.1186/1471-2202-11-S1-P66

Rall W (1959) Branching dendritic trees and motoneuron membrane resistivity. Exp Neurol 1:491–527

Rall W (1962a) Electrophysiology of a dendritic neuron model. Biophys J 2:145–167

Rall W (1962b) Theory of physiological properties of dendrites. Ann N Y Acad Sci 96:1071–1092

Rall W (1964) Theoretical significance of dendritic tress for neuronal input–output relations. In: Reiss RF (ed) Neural theory and modeling. Stanford University Press, Stanford CA, pp 73–97

Rall W (1967) Distinguishing theoretical synaptic potentials computed for different soma-dendritic distributions of synaptic input. J Neurophysiol 30:1138–1168

Rall W, Agmon-Smir H (1998) Cable theory for dendritic neurons (chap 2). In: Koch C, Segev I (eds) Methods in neuronal modeling: from ions to networks, 2nd edn. MIT Press, Boston, pp 27–92

Rall W, Shepherd GM (1968) Theoretical reconstruction of field potentials and dendrodendritic synaptic interaction in olfactory bulb. J Neurophysiol 31:884–915

Ray S, Bhalla US (2008) PyMOOSE: interoperable scripting in Python for MOOSE. Front Neuroinform 2:6. doi:10.3389/neuro.11.006.2008

Richert M, Nageswaran JM, Dutt N, Krichmar JL (2011) An efficient simulation environment for modeling large-scale cortical processing. Front Neuroinform 5:19

Rinzel J (1990) Electrical excitability of cells, theory and experiment: review of the Hodgkin-Huxley foundation and an update. Bull Math Biol 52:5–23

Rochel O, Martinez D (2003) An event-driven framework for the simulation of networks of spiking neurons. In: ESANN-2003, Bruges, Belgium, pp 295–300

Santamaria F, Tripp PG, Bower JM (2007) Feedforward inhibition controls the spread of granule cell: induced Purkinje cell activity in the cerebellar cortex. J Neurophysiol 97:248–263. doi:10.1152/jn.01098.2005, http://jn.physiology.org/content/97/1/248.full.pdf+html

Sasaki K, Bower JM, Llinás R (1989) Purkinje cell recording in rodent cerebellar cortex. Eur J Neurosci 1:572–586

Segev I, Fleshman JW, Miller JP, Bunow B (1985) Modeling the electrical behaviour of anatomically complex neurons using a network analysis program: passive membrane. Biol Cybern 53:27–40

Shepherd GM, Brayton RK (1979) Computer simulation of a dendro-dendritic synapse circuit for self- and lateral-inhibition in the olfactory bulb. Brain Res 175:377–382

Shepherd GH, Healy MD, Singer MS, Peterson BE, Mirsky JS, Wright L, Smith JE, Nadkarni P, Miller PL (1997) SenseLab: a project in multidisciplinary, multilevel sensory integration (chap 3). In: Koslow SH, Huerta MF (eds) Neuroinformatics: an overview of the human brain project. Lawrence Erlbaum, Mahwah, NJ, pp 21–56

Spacek MA, Blanche T, Swindale N (2009) Python for large-scale electrophysiology. Front Neuroinform 2:1. doi:10.3389/neuro.11.009.2008

Stiles JR, Bartol TM (2001) Monte Carlo methods for simulating realistic synaptic microphysiology using MCell. In: Schutter ED (ed) Computational neuroscience: realistic modeling for experimentalists. CRC Press, Boca Raton, pp 87–127

Thorpe MF, Beeman D (1976) Thermodynamics of an Ising model with random exchange interactions. Phys Rev B14:188

Traub R (1977) Motor neurons of different geometry and the size principle. Biol Cybern 25:163–176

Traub RD (1982) Simulation of intrinsic bursting in CA3 hippocampal neurons. Neuroscience 7:1233–1242

Traub RD, Llinás R (1979) Hippocampal pyramidal cells: significance of dendritic ionic conductances for neuronal function and epileptogenesis. J Neurophysiol 42:476–496

Traub RD, Wong RKS, Miles R, Michelson H (1991) A model of a CA3 hippocampal neuron incorporating voltage-clamp data on intrinsic conductances. J Neurophysiol 66:635–650

Traub RD, Jeffereys JGR, Miles R, Whittington MA, Tóth K (1994) A branching dendritic model of a rodent CA3 pyramidal neurone. J Physiol (London) 481:79–95

Vanier MC, Bower JM (1999) A comparative survey of automated parameter-search methods for compartmental neural models. J Comput Neurosci 7:149–171

Weitzenfeld A (1995) NSL—neural simulation language. In: Arbib MA (ed) The handbook of brain theory and neural networks, 1st edn. Bradford Books/MIT Press, Cambridge, pp 654–658

Wilson MA, Bower JM (1989) The simulation of large scale neural networks (chap 9). In: Koch C, Segev I (eds) Methods in neuronal modeling. MIT Press, Cambridge, MA, pp 291–333

Wilson M, Bower JM (1991) A computer simulation of oscillatory behavior in primary visual cortex. Neural Comput 3:498–509

Wilson M, Bower JM (1992) Cortical oscillations and temporal interactions in a computer simulation of piriform cortex. J Neurophysiol 67:981–995

Wilson MA, Bhalla US, Uhley JD, Bower JM (1989) GENESIS: a system for simulating neural networks. In: Touretzky D (ed) Advances in neural information processing systems. Morgan Kauffman, San Mateo, CA, pp 485–492