# Chapter 4

# Computational neuroscience software

Neural software can be arranged in two groups:

- packages designed for modelling and simulation of neurons and neural networks

- packages supporting modelling, e.g. databases, software for data analysis and visualization, differential equation solvers

As mentioned in the previous chapter, neural modelling can be arranged according to the levels of modelling. The modelling levels range from abstract to detailed, incorporating anatomical and biophysical parameters. Since my work is focused on the construction of software for simulation of structurally realistic models of biological neural systems, I will in this chapter describe the most important packages for detailed neural modelling [53]. These packages are typically based on compartmental modelling, the standard modelling approach for learning about the detailed anatomical structure and physiological characteristics of neural systems.

The second group of software, packages supporting modelling, makes easier exchange of data developed with different simulation packages and allows its well-ordered storage. This is becoming more and more important because of the growing number of available simulation systems, many of which have different characteristics and simulation abilities. This group of neural software also contains packages for data analysis, e.g. for 3-dimensional presentations, or to convert the morphology of complex dendritic trees into input files for the simulators. These programs allow one to use the effective programs for solving of differential equations that do not provide graphical user interfaces.

## 4.1 General versus special-purpose simulators

The most important aspect of the choice of appropriate simulation system is what particular physiological data can be simulated using the system. The variety of modelling elements that are possible to study is in the focus of attention both for the developers of a program and the modellers using it.

When studying the available simulation packages and their particular characteristics, one soon recognizes that some of them have a quite general domain of application while others are rather specialized. Some Well-known packages, such as Genesis [35] and Neuron [32], were developed as an attempt to create a "general" simulation system. They were designed to be capable of solving problems at many different levels of modelling. The development of general simulation systems is important in computational neuroscience, since the range of applications arising in the modelling of real neurobiological systems can vary from sub-cellular components to complex network models.

The application of general simulation systems in neurobiological research have a number of advantages compared to programs developed and dedicated to performing of certain tasks.

First of all, these systems are powerful neural simulators which can provide good performance for simulations. Users with good programming skills and knowledge in computational neuroscience can argue that simulation code written for a particular purpose can be better optimized for that particular task. In certain cases the optimal use of computation resources is of high importance. However, development of dedicated application requires a lot of time to write and test the computer code. The time required for simulation preparation is practically always much bigger than for performing the simulation itself.

When using the general simulation packages one might also have to spend time for learning the user interface and maybe the scripting language. But in return they provide a large set of dedicated routines and available computational methods. This allows the user to construct the model from standard elements; this can substantially reduce the time for simulation preparation.

The important advantage of the general simulation packages is the extensibility of available models and elements. Modification of specially written dedicated programs for new simulation tasks often requires the user to rewrite the program code, on which the complete sequence of testing procedures follows. In contrast, if one uses a general simulation system, much less efforts are usually required to modify and test the simulation model. One can use and extend previously defined elements and structures as building blocks for the new model.

The general simulation systems are often capable of using specially developed techniques and methods. They provide a number of integration methods and their effective realizations, as for example the integration method for branching structures developed by Hines, the author of Neuron [29]. Genesis, another general simulator, provides the opportunity to perform the simulation on parallel computers [35].

The possibility to construct new simulations using elements and structures developed in previous simulations stimulates the organization of their collection in databases. Such developed databases can be considered as the tools for collecting and exchanging knowledge in the field of modelling of neural systems. The collections of simulated elements are usually functionally ordered, so that their physiological properties are easy to inspect and use in the new applications. For example, the Neuron User's Group was established to provide information about the latest updates of the program features and for communication about useful tips concerning the use of the program [46]. The Genesis users group, BABEL, maintains a database of published simulations [14]. The development of general simulations in the form of databases of neural elements can be explained by their large number of users. The communication between users results in the necessity to establish users groups.

The wide spectrum of possible modelling tasks appearing in the field of computational neuroscience has so far not allowed the realization of a simulation program or package which can be optimally applied to every problem. Usually even the general simulation systems do not optimally deal with certain spectra of problems. Before starting to learn a particular simulation system, one needs to check in detail which elements the system is capable to simulate.

Sometimes specialized programs are more effective and preferable to use for solving the problems to which they were dedicated. Because of the narrow specialization, these programs usually provide a well-developed user interface.

## 4.2 Important aspects of the program choice

In this paragraph the most important aspects that should be taken into account when choosing program will be discussed. As was mentioned previously, a simulation system could primarily be characterized by the physiological elements it is able to simulate. Therefore, at the first step of choosing a program every researcher should clearly define the problem and check whether the simulation system is capable of simulating it.

All packages which I will describe in this chapter can perform simulations for detailed study of the membrane properties and are based on the compartmental modelling approach. As was described in the previous chapter, the compartmental modelling approach is derived from the cable theory by replacing the continuous equations with a set of ordinary differential equations. Thus, it is possible to accurately simulate the morphology of a cell, since each differential equation describes the voltage dynamics of the small membrane section. At the same time, the differential equations describing the dynamics of ionic and synaptic channels are also included. Usually, simulators are able to simulate passive membrane models. However, not every simulation system is flexible enough to be able to model different kinds of ionic channels and other specific point processes.

The second important property of the simulation system is its portability. Most of the available simulation packages are Unix-based, since they require extensive computational power for solving the system of differential equations arising from compartmental modelling. For users with insufficient IT-experience, problems can arise already in the installation phase, and then again later during the usage of the programs.

The next important characteristic of a particular software is how the user interface is organized. That determines the time that will be spent on the modelling process including constructing and adjusting the particular simulation. Very often, packages have their own interpreted scripting language, in which users define the components and running parameters for their simulations. This provides a flexible interface, since the scripting programs can be specifically written for each modelling situation. The simulation can be started both from the command prompt and by reading the model description from the scripting files. The script interpreters allow control over the modelling parameters during the simulation. But these packages require programming skills and time for learning the scripting language.

"General" simulation systems often do not provide a Graphical User Interface (GUI) or otherwise have a very simple one, and as a result they cannot visually represent the simulation process. Having a wide spectrum of applications makes it difficult to construct a GUI for each model. The complexity of the model (sometimes thousands of compartments and channels in each cell) requires flexible editing and display modes for both single compartments and groups of compartments. Genesis provides the possibility to construct the user interface using a set of graphical modules. But this means that one needs time and patience to build the appropriate interface in addition to the model construction. Neuron provides a GUI only for primal research with functions for parameter setting, control of voltage and current stimuli, and graphical presentation of the results as a function of time and position.

The user extensibility can be essential for creating new modelling elements and developing new modelling tests. The extensibility depends on the following characteristics: modelling language, modular design, and open-ended code. Most systems use an object-oriented approach, which enables the user to easily add new modules to extend the system for a particular application. The modular design means that there are libraries or databases of standard simulation components, which can be chosen and used to quickly construct new simulations.

Of course, the number of available integration techniques is an important property of simulation systems. The methods for solving neural equations describing circuit spread in the cells can range from explicit methods to highly implicit methods. The explicit forward Euler method is easy to implement and takes fewer computational resources, but it can be unstable in some conditions. In contrast, implicit methods take a lot of extra work but are more accurate and more stable [49]. Most "general" simulation systems use an implicit integration method based on the one developed by Hines [29], which is the most effective for detailed cell models which contain many compartments. But this method can be applied only to problems of a special kind because of its instability. If an implicit method becomes unstable one can apply one of the explicit methods.

In the case of a model with a huge number of neurons or a complicated structure of activated channels, the available computer resources can restrict the choice of simulation program. Recently developed architectural techniques like a client-server architecture of the program or parallel computing can be a solution in such a case. The client-server architecture assumes a separation of the program into two parts: the server performing numerical integration and the client providing the user interface for preparation and control of the simulation process. The server performs the function of the equation solver and should be located on a powerful computer. The client, installed on the user's computer, provides access for controlling the simulation and displays the results. This separation gives an effective solution in the usage of computer resources, especially for detailed modelling of the morphological properties of the membrane. The client-server architecture has not been realized in any of the neural simulation packages.

The opportunity to start the system on a parallel computer (a computer with many processors running simultaneously) is provided by some simulation systems, e.g. by Genesis. The parallel computing can increase the computational speed compared with the standard single processors. Therefore, complex models can be implemented, since the computation is performed on distributed processors.

To conclude, a simulation system can be characterized by the elements possible to simulate. As important features the program extensibility, its portability, user interface and special developed techniques for effective use should be considered.

## 4.3 Compartmental modelling packages

In this section I will go through those simulation packages that are extensively used by wide groups of users and compare their characteristics based on the most important features discussed above.

### 4.3.1 Neuron

Neuron is developed with the idea to realize a general simulation system. It is designed on the basis of a script interpreter for model definition and running a simulation. In common experiments, it provides the graphical user interface with tools for specifying neural elements and analyzing simulation results. In more general cases, it is convenient to write the procedural code using the interpreter [32, 43].

**Spatial discretization: sections versus compartments** Although Neuron is based on the compartmental modelling approach, it does not use a compartment as the main building element [30]. The models are described as a number of continuous cables, which are defined in terms of "sections". The sections are the main components of the Neuron. They are connected together to form the branched cable tree. Each section is ultimately discretized in segments (compartments) of equal length, the number of which is determined by parameter *nseg* [31].

The biophysical and anatomical parameters of each section are continuous variables depending on the position along the cable and vary between segments. The cell properties, depending on the position along a cable branch (section), are defined using so-called "range" variables. They are expressions in the form $rangevar(xmin : xmax) = e1 : e2$ specified as functions of $x$ varying from $xmin >= 0$ to $xmax <= 1$. A function defined with a "range" variable can take values between $e1$ and $e2$.

Model structures and parameters defined in such a way are effectively used to find the numerical solution with the Crank-Nicholson integration method optimized for branched structures [29].

Another advantage of representing neurons in terms of cable branches rather than compartments, mentioned by Neuron's authors, is that this way of description is more natural for neuroscientists [43].

**The user interface** Neuron uses the interpreter HOC with a syntax based on the C language. The main method for model definition, specification of the sections and parameter determination is based on the use of HOC. HOC is also used for simulation control and the HOC code is saved as ASCII files. Because of the object-oriented structure of Neuron's interpreter, implementation of new data types is possible.

By default, the graphical user interface is developed for definition of the basic neural elements, parameters setting, control of voltage, current stimuli and for presenting graphs as a function of time and position. These functions can be extended for particular tasks by preparing scripting procedures to the interpreter.

**A practical example** Here I will consider an example of the construction of a simple neuron consisting of a soma connected with two dendrite cables. It demonstrates how Neuron deals with the cable "sections" mentioned above. The soma contains ionic channels with Hodgkin-Huxley dynamics, whereas each dendrite has a passive leakage channel. The following HOC code defines the model structure and specifies the morphological properties:

```
create soma, dendrite[2]
for i=0,1 {connect dendrite[I](0),soma(1)}
          forall Ra=35.4
                  soma {nseg=1
                        L=30      //length , micron
                        diam=30  //diam , micron
                        insert hh  //hh channel
                        gnabar_hh=0.5*0.120 }
for i=0,1 dendrite[i]  {nseg=5
                        L=100
                        diam(0:1)=10:3
```

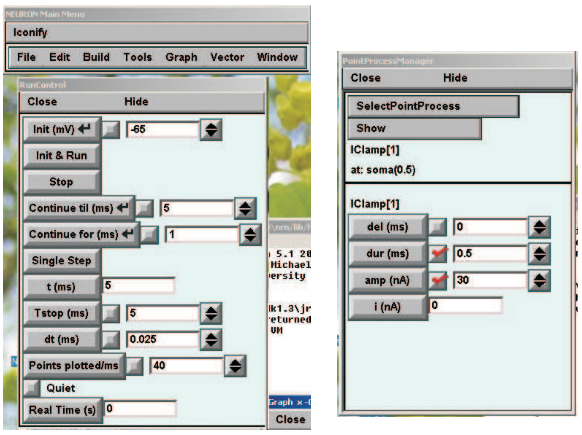


Figure 4.1: Neuron's `Run Control window` for executing and controlling the simulation and the `PointManager window` for setting up the Voltage Clamp.

```
insert pas //passive channel
e_pas=-65 //equilibrium potential
g_pas=0.001
}
```

One can see that each dendrite is composed of five "segments" (parameter *nseg*), whereas the soma is represented as one segment. The diameter of the dendrite's segments is described with a "range" variable as a function decreasing along the dendrite cable. The position is specified by the normalized parameter $0 <= x <= 1$. In our example the diameter of the dendrite's segments is changed from 10 to 3 along the dendrite.

**Graphical User Interface** The above given code can be saved in a *.hoc file and can be loaded via the graphical user interface, which can be started as the *nrngui* library. Neuron `Main Menu` provides functions to load data defined with the HOC interpreter, edit the model parameters, and for the graphical control of simulation results. The Run Control Window shown in Fig.4.1 contains menus for controlling the simulation. It provides functions for starting simulations with a defined number of steps or calling the built-in single-step integration function. It is possible to change the values of the membrane parameters during a run. An electrode inserted into the soma can be constructed from the PointManager window, which is activated from the Point Processes menu. One can also set the parameters of a stimulus, as illustrated in Fig. 4.1.

Another group of neural elements, which can be included in the model, are the so-called "standard density mechanisms", the sources of electrical and chemical signals that are distributed over the membrane of the cell, e.g. voltage-gated ionic channels.

Additionally, users can define new density mechanisms and point processes using the model description language NMODL. With NMODL, one specifies the equations for a channel or ionic process that will be translated into the appropriate C code and the compiled version can be used later in Neuron. Model description files written in NMODL are checked for the consistency of the units using a unit checker program [44].

**Cell Builder and Network Builder** Using Cell Builder and Network Builder, neural models can be constructed without the need to write any HOC code oneself. Cell Builder, shown in Fig. 4.2, provides a GUI to specify the branched structure of the single-cell model and assign biophysical and anatomical parameters for a section or a group of sections [46].

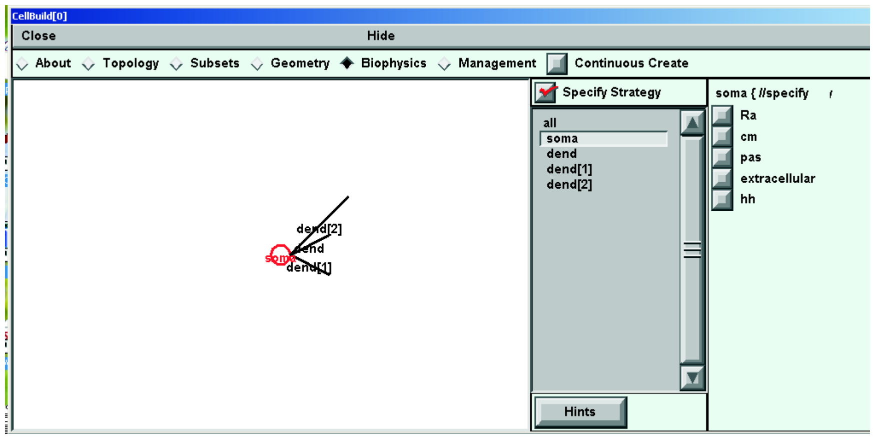
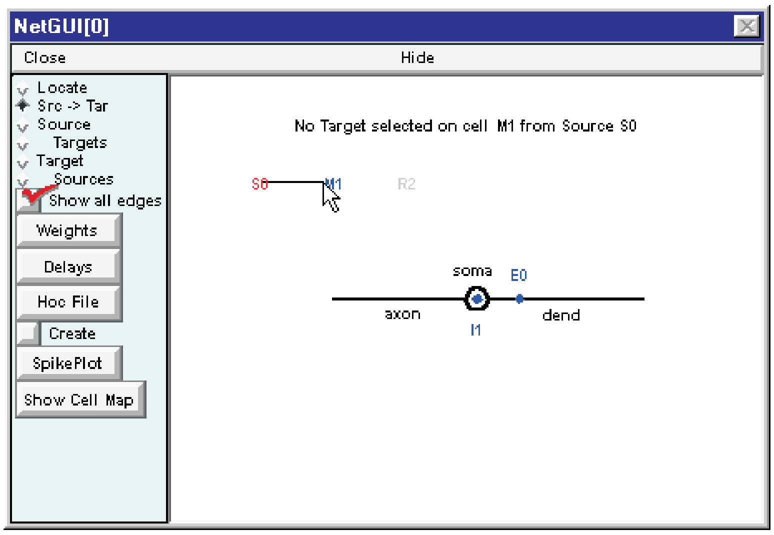


Figure 4.2: Constructing and managing models of single neurons with Cell Builder.

Network Builder was developed to construct models of real either artificial neural networks (see Fig. 4.3) incorporating some classes of artificial and pulsed neurons [46, 45].

**Integration methods** The user has a choice between two integration methods realized in Neuron: the backward Euler method and the Crank-Nicholson method. Neuron's default integration method is the first order implicit backward Euler method, which is stable. In terms of computational speed, a variant of the Crank Nicholson method, implemented by Hines, is the most effective method for models with branched cable structures with Hogkin-Huxley kinetics. The improvement is based on the staggered time step by using the backward Euler method in the first half step and then the forward Euler method in the second half step. It was shown that this method requires no extra computations cost compared to the backward Euler method step.



No Target selected on cell M1 from Source S0

Figure 4.3: Modelling networks with Network Builder.

The reason for using the backward Euler method by default is that it provides a good compromise between accuracy and stability even with large time steps, whereas the Crank-Nicholson method can produce oscillations if the time step is too large.

**Operation systems and available documentation** Neuron was initially developed in the Unix operating system and later ported to Windows and MacOs. The full source code is available with the Unix distributions. The Neuron distributions, including the source code, comprehensive documentation and tutorial material, are available free of charge from the Neuron web page.

The Neuron Users' Group is organized in the form of a mailing list for sending information about program updates and exchanging useful tips concerning the program use [46].

### 4.3.2 Genesis

Genesis is one of the most powerful and widespread general simulation systems. It has an object-oriented structure that makes it possible to exchange and reuse models and their components. The commands for model definition, simulation control and construction of the graphical interface, for each specific simulated model, are provided by an interpreter. An extensible library of neural elements was collected by the sharing of new models between members of the Genesis Users' Group [38].

**Model components** Genesis provides a library of precompiled basic components, such as compartments, various types of channels, or synaptic connections, which may be added into a model. These components can be used to construct the simulation by writing code composed of scripting language commands. The parameter values are described with the object's data fields, as it is usually done for objects in object-oriented structures.

The communication between objects is organized by calling "messages" which are predefined for the groups of modelling elements. For example, the following code constructs a soma-like compartment and sets the parameters:

```
create neutral /cell
create compartment /cell/soma
setfield /cell/soma Em {Erest}//volts
Rm {RM/area}  //Ohms
Cm {CM*area}  //Farads
Ra  {RA*length/xarea}  //Ohms
```

In the compartmental approach compartments are connected by a system of longitudinal currents from neighboring compartments. In our case, to set the connection between the soma and the dendritic compartment located to the left, one has to add "messages" containing both the axial resistance ($R_a$) and the membrane potential (previous state) for the dendrite and only the membrane potential (previous state) for the soma. Each time when the values of these fields are updated, they are sent as messages.

```
addmsg cell/dend cell/soma RAXIAL Ra previous_state
addmsg cell/soma cell/dend AXIAL previous_state
```

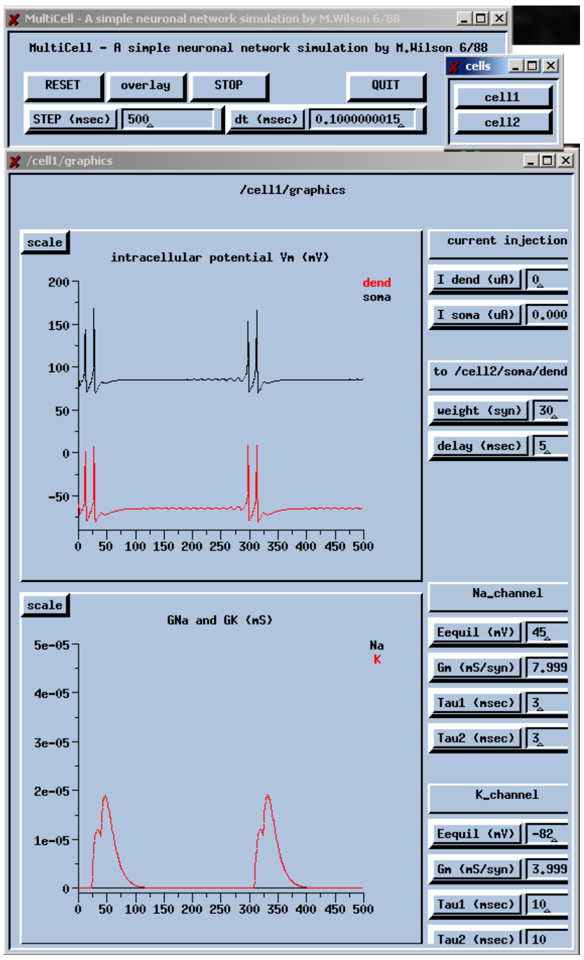
In the same way as compartments are linked with ionic channels, cells, consisting of a number of compartments, are linked together to constitute the network models.

**The scripting language interpreter and the graphical user interface** The above given commands may be called either interactively by a command prompt or by using simulation scripts files. Genesis' scripting language plays the role of the connection between the building, controlling and visualization of the simulation process.

The interpreter is based on the Scripting Language Interpreter (SLI), a "command interpreter similar to a Unix system shell" [38].

The graphical objects available with the standard Genesis distribution are linked with the model components by calling messages, for example by passing information about simulation results, which should be plotted in a graphical window, into the graphical objects.

Graphical objects constitute XODUS (the X-Windows Output and Display Utility for Simulations). The components of XODUS perform graphical functions and can be used by users to create their own GUI to control simulations and display results. For example, graphical "windows" for simulation of a two-cell network in a feedback configuration can be created as in Fig. 4.4. The simulation contains two neurons, each of which is composed of two compartments corresponding to a soma and a dendrite. The dendrite has synaptically activated channels while the soma contains ionic channels with Hodgkin-Huxley dynamics.


MultiCell - A simple neuronal network simulation by M.Wilson 6/88
MultiCell - A simple neuronal network simulation by M.Wilson 6/88
RESET
overlay
STOP
QUIT
cells
cell1
cell2
/cell1/graphics
/cell1/graphics
scale
intracellular potential Vm (mV)
dend
soma
scale
GNa and GK (mS)
Na
K
current injection
to /cell2/soma/dend
Na_channel
K_channel

STEP (msec) 500
dt (msec) 0.1000000015

I dend (uA) 0
I soma (uA) 0.000

weight (syn) 30
delay (msec) 5

Na_channel
Eequil (mV) 45
Gm (mS/syn) 7.999
Tau1 (msec) 3
Tau2 (msec) 3

K_channel
Eequil (mV) -82
Gm (mS/syn) 3.999
Tau1 (msec) 10
Tau2 (msec) 10

Figure 4.4: The GUI constructed with Genesis showing a simulation of a two-cell network in a feedback configuration.

This approach to the organizing of graphical user interfaces gives the advantage that one can build the GUI special for each simulation model and then interactively change simulation parameters, but it takes additional time to master the visualization process.

**Genesis' object libraries** The standard distribution of Genesis includes a number of common neural elements, which can be used to build a simulation.

Regardless the source code of these objects, only knowledge of the scripting language is necessary for developing a simulation.

The main "building element" of the Genesis simulation, based on the compartmental approach, is the object "compartment" which can have spherical or cylindrical shape. The Genesis object library also contains voltage- and concentration activated channels and dendro-dendritic and synaptically activated channels used for cell connections. Input to the simulation can be implemented in Genesis with so-called "device objects", among which there are different types of pulse and spike generators, voltage clamp circuitry, etc.

In addition to the standard object library, there are a number of objects that were extended by Genesis users from existing elements and also included in the Genesis distribution. The channel library contains different types of calcium, potassium and sodium channels. Single cell models are available as script files and include cerebral cortical pyramidal cells, hippocampal pyramidal cells, mitral and granule cells, etc.

Using scripting language, it is possible to create new objects from existing Genesis elements. Elements can be extended by adding new data fields or functions or by creating new messages.

However, for specialists with programming knowledge, trying to integrate self-developed elements with Genesis objects, there are several drawbacks. It is difficult to understand the logical structure of the object library. The implementation of objects and numerical integration are tied together. The interpreter commands and message callings as the important part of the Genesis system are often not separated from the objects' implementation code. Therefore, it is quite difficult to change the source code.

**Cell Reader and Neurokit simulation** Models of single multicompartmental cells can also be constructed by creating "cell parameter files" using Cell Reader. The library of prototype elements is used by Cell Reader to create model elements. A cell is constructed by copying the prototype elements into the model structure and replacing the default parameters with values from the "cell parameter files". To build a simulation with Cell Reader, one only needs to learn the form of the "cell parameter files".

The NeuroKit simulation, which can be found in the Scripts/neurokit directory, provides a graphical user interface to work with "cell parameter files", in order to build and change single cell models as well as to run simulations.

**Operating system and implementation language** The computational and graphical elements of Genesis are implemented in the C language. Genesis can be run on Unix systems having X-Windows graphical utilities.

Genesis was also adopted to perform extensive calculations on parallel computers for large simulation systems. The parallel version of Genesis is called PGenesis [47].

**Integration methods** The integration methods used in Genesis vary from explicit methods, which require more computational power, to more accurate implicit integration methods.

The default integration method is the exponential Euler method. The choice of this explicit method as the default integration method is based on its high efficiency for common differential equations arising from compartmental modelling (see Section 3.2). Other explicit methods range from forward Euler to different orders of Adams-Bashforth methods (2nd, 3rd, 4th, 5th order).

There are two implicit methods realized in Genesis: the backward Euler method and the Crank-Nicholson method. Hines' variant of the Crank-Nicholson method is the most effective for neurons having branched structure [29]. The implicit methods can be used in conjunction with a special developed object, `hsolve`, which only can be applied for a few types of elements. The object `hsolve` is responsible for calculating the solution of equations in the matrix form arising from the implicit integration techniques.

**Special techniques used in Genesis** Among the special techniques implemented in Genesis to speed up simulations one should first of all mention the `hsolve` object for effective numerical integration.

I also found that the specifying of parameters in the form of a table can be very useful. For example, one can use the `tablechannel` object which provides the dynamics of the rate parameters in a table instead of using $hh_{channel}$ which only provides standard functions for describing the "rate parameters". The tablechannel object looks up the values of the rate parameters predefined in a table. Thus, one does not need to fit parameters into the standard forms. More general types of channels can also be implemented.

The possibility to use "multiple clocks" was implemented in Genesis as another technique for speeding up and increasing the accuracy of the simulation. That means that one can perform a numerical integration and present the results graphically using different time scales for the variables.

**The Genesis distribution, documentation and tutorials** The Genesis distribution is available on the Internet. It contains the full source code including the XODUS, supported with documentation describing basic elements and simulation routines, tutorials and demonstration programs for getting started. "The Book of Genesis" [41] published by Genesis' authors can be used as a program manual. It is organized into two parts: modeling techniques used in the neurobiology and basic examples of writing Genesis simulations. The first part describes the basic approaches of computational neuroscience illustrated with Genesis simulations. The second part is written as a user guide considering basic features of Genesis and describing the process of creating Genesis simulations.

Most parts of the tutorials and the documentation were written by members of the Genesis Users' Group, BABEL, which consists of advanced users of Genesis. New simulations, and neural elements developed by members of BABEL emerging from real neural systems are included in the latest version of Genesis.

### 4.3.3 SNNAP

The SNNAP simulator can be a good choice for modellers with little or no programming skills, because of its quite well-developed graphical user interface (GUI). SNNAP provides editors embodied into the graphical interface for specifying the anatomical and biophysical properties of neurons and the structure of networks, saving the results of simulations into files and controlling the simulation results [34]. Editors consist of different graphical windows for specifying the model structure, editing the model and network parameters, setting the anatomical and biophysical parameters of neurons, plotting the simulation results and saving the information to files.

The SNAAP user does not need to learn the scripting language and simulation techniques, since all operations are performed graphically.

The graphical interface of SNNAP allows the user to reduce the time required to get started with the program, compared to interfaces based on scripting language. At the same time, SNNAP incorporates all the common elements required for simulation of real neurobiological systems. Therefore, it can be used for fast modelling of single neurons and small neural networks.

The orientation towards a fully graphical interface in SNAAP has clear advantages but it also brings some limitations. One is the limited number of possible modelling elements and possible simulation tests which are predefined in the graphical user interface. The simulator also does not provide any opportunity to extend models.

**Operating system and implementation language** SNNAP was implemented as a Java application and can run on any computer system. The SNNAP distribution can be obtained from the Internet as Java *.jar archival files consisting of the executing file, tutorial material and examples of the program demonstration.

To install the SNNAP simulator, one needs a functional version of Java on the user's computer. The installation consists only of the unzipping of the *.jar files and the allocation of them in an appropriate directory.

**SNNAP editors** When starting up the SNNAP simulator, the main menu of SNNAP, shown in Fig. 4.5, appears. It contains buttons to open the appropriate SNNAP editors which are divided into the functional groups to run a simulation and to have access to simulation parameters, cell's and connection parameters, as well as to parameters of ionic channels and inserting mechanisms.

The simulation parameters are saved in files in a hierarchical structure connected together with a *.smu simulation file. The graphical editors of SNNAP generate various functional files, for example files containing equations describing the dynamics of ionic channels or the parameters of neural connection. The hierarchical structure of the parameter files allows one to open each of them directly using an appropriate editor.

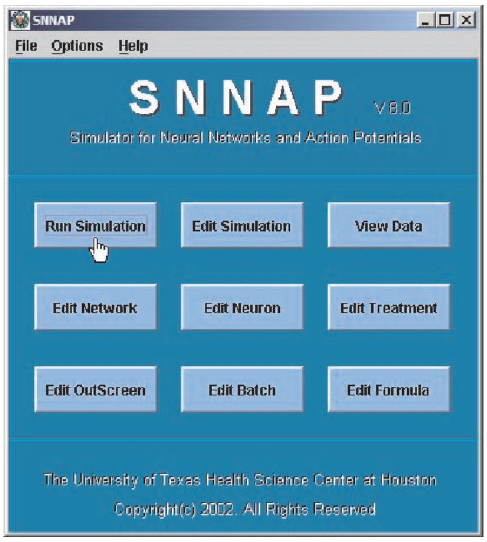

SNNAP
File Options Help
SNNAP V8.0
Simulator for Neural Networks and Action Potentials
Run Simulation
Edit Simulation
View Data
Edit Network
Edit Neuron
Edit Treatment
Edit OutScreen
Edit Batch
Edit Formula
The University of Texas Health Science Center at Houston
Copyright(c) 2002. All Rights Reserved

Figure 4.5: The main window of SNNAP.

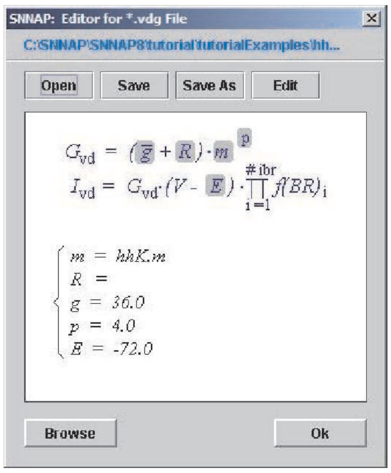
The formula editors in SNNAP are provided to edit the parameters of the mechanisms whose dynamics are described with equations. These editors show the form of equations emphasizing with different colors the parameters which may be changed (see e.g. Fig. 4.6). Thus, with the SNNAP graphical editors one can easily edit the model and the simulation parameters using a certain set of the models and their corresponding equations.

**Models simulated with SNNAP** SNNAP can simulate models constructed of neurons with Hodgkin-Huxley type voltage- and time-dependent ionic channels and integrate-and-fire neurons. The program was developed as "a simulator of single neurons and small neural networks" [34].

Let us consider an example simulation of a one-cell model with Hodgkin-Huxley ionic channels, the basic formalism structure of which is most often chosen by neuroscientists. The simulation can be found among the examples provided with the SNNAP distribution, in the directory /Examples/hhmodel. One needs to open an appropriate *.smu file, which offers access to the simulation editors. The parameters of the Hodgkin-Huxley channels (as voltage-dependent gated channels) can be viewed in the window of the *.vdg files Editor (see Fig. 4.6).

The results of the Hodgkin-Huxley neuron simulation are controlled with the simulation window shown in Fig. 4.7, where one can see an action potential initiated by the injected current with the duration of 0.1 ms. The graphs created in SNNAP can be printed and stored as a postscript or ASCII file.

SNNAP can simulate the current flow in multicompartmental neurons con-


SNNAP: Editor for *.vdg File
C:\SNNAP\SNNAP8\tutorial\tutorialExamples\hh...
Open
Save
Save As
Edit
Browse
Ok

$$G_{\mathrm{vd}} = (\bar{g} + R)\cdot m^{p}$$
$$I_{\mathrm{vd}} = G_{\mathrm{vd}}\cdot(V - E)\cdot\prod_{i=1}^{\#\mathrm{ibr}} f(BR)_{i}$$

$$\begin{cases} m = hhK.m \\ R = \\ g = 36.0 \\ p = 4.0 \\ E = -72.0 \end{cases}$$

Figure 4.6: The editor window provided by SNNAP for voltage-dependent currents.

nected into neural network models. An example of such a simulation presenting a neural network constructed from three neurons with Hodgkin-Huxley channels can be loaded from the directory /Examples/hhNetwork. The network architecture is presented in Fig. 4.8, while the network behavior is shown with the voltage graphs in Fig. 4.9.

SNNAP provides a wide range of neural elements and common experimental tests that can be used for simulations. The connections between neurons can be made by electrical, chemical and modulatory synapses. Additionally, the chemical synapses can be expressed with different kinds of plasticity, such as homo- and hetero-synaptic depression and facilitation. The kinetics of the multiple components contained in chemical synapses can be defined by the user.

Among the common experimental techniques, SNNAP can simulate for example current injections into neurons, modulation of membrane currents and voltage-clamping experiments. Examples from the directory /Example/HH _type_neuron/Biophysics_01 illustrate the voltage-clamping experiments. Fig. 4.10 displays the results of a simulation of a Hodgkin-Huxley neuron whose voltage was clamped at $V = -60\ mV$ and stepped to $V = -10\ mV$. One can see the change of the total membrane current $I_m$, the sodium current $I_{Na}$ and the potassium current $I_K$.

**Numerical integration method** SNNAP uses the forward Euler method with a fixed time step for numerical integration of the differential equations arising in the modelling. The forward Euler method, one of the explicit integration methods, is fast, but can cause problems of instability. The integration time step must be chosen small enough to avoid the problem of numerical instability of the method, leading to oscillations of the solution.

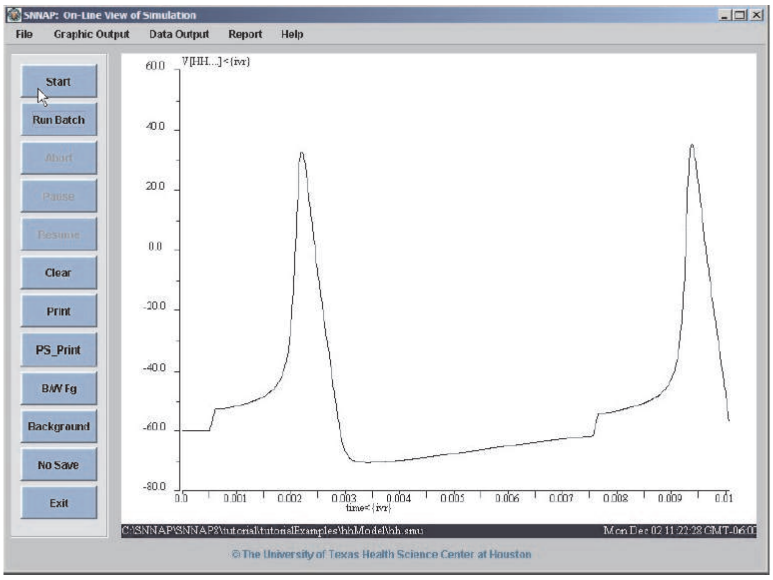

SNNAP: On-Line View of Simulation
File
Graphic Output
Data Output
Report
Help
Start
Run Batch
Abort
Pause
Resume
Clear
Print
PS_Print
B/W Fg
Background
No Save
Exit
V[HH...]<(ivr)
time<(ivr)
C:\SNNAP\SNNAP8\tutorial\tutorialExamples\hhModel\hh.smu
Mon Dec 02 11:22:28 GMT-06:00
© The University of Texas Health Science Center at Houston

Figure 4.7: The SNNAP simulation window displaying two action potentials.

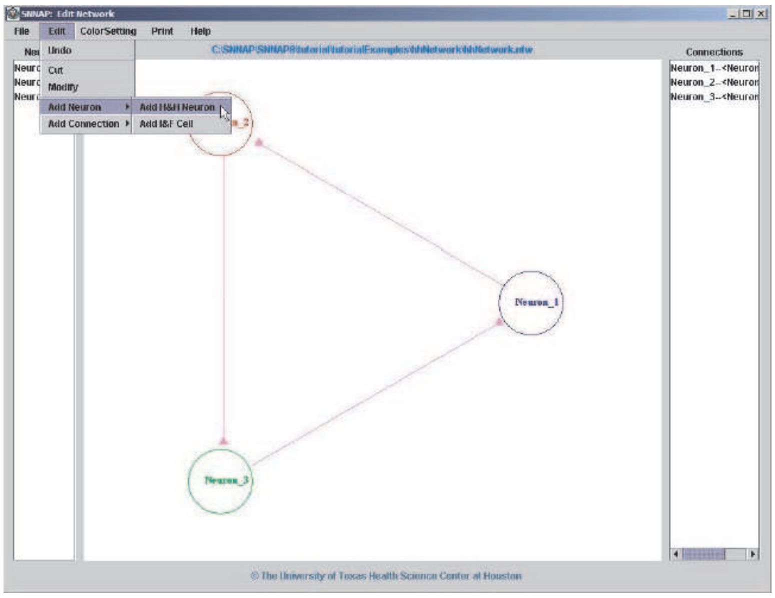


Figure 4.8: The SNNAP GUI showing the architecture of a three-neuron network.

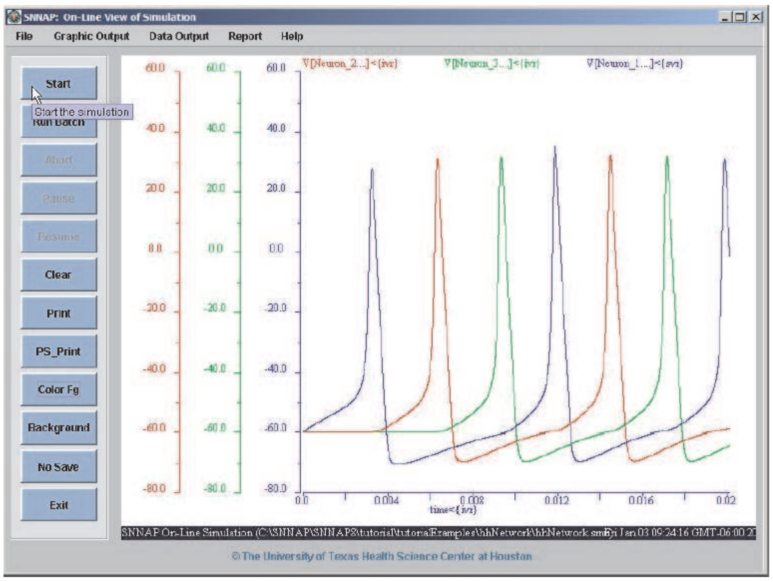


Figure 4.9: Results of a SNNAP simulation of a three-neuron network.

The SNNAP documentation and tutorials contain examples describing the problem of instability and warnings of possible errors.

### 4.3.4 Nodus

Nodus is a program for simulation of single neurons and small neural networks. All aspects of the model definition and control of the simulation results can be managed via a user-friendly graphical interface. Therefore, Nodus users do not need much computer experience.

**The Nodus distribution, operating system and programming language** Nodus was implemented in Fortran and runs on Apple Macintosh computers.

The Nodus distribution is available “for a small fee”. It consists of compiled executable files, examples and a manual description. The source code is not provided emphasizing that “the average Nodus user is not a programmer” [64].

**Integration methods** Nodus provides two numerical integration methods: an accurate Fehlberg method (fifth-order Runge-Kutta) and the forward Euler method. These are explicit integration methods, which are easier to implement but in some cases slower than implicit methods implemented in other simulation systems.

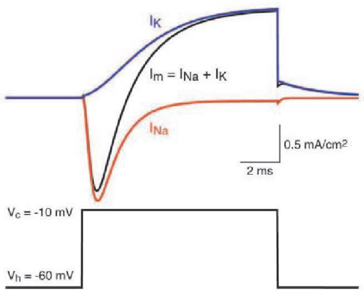
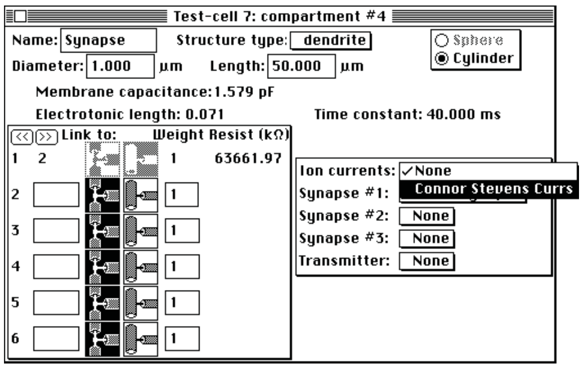


Figure 4.10: A voltage-clamping experiment with the Hodgkin-Huxley model of a squid axon implemented with SNNAP.



Test-cell 7: compartment #4

Name: Synapse Structure type: dendrite ○ Sphere [✓ Cylinder]

Diameter: 1.000 µm Length: 50.000 µm

Membrane capacitance: 1.579 pF

Electrotonic length: 0.071 Time constant: 40.000 ms

| | Link to: | Weight | Resist (kΩ) |
|---|---|---|---|
| 1 | 2 | 1 | 63661.97 |
| 2 | | 1 | |
| 3 | | 1 | |
| 4 | | 1 | |
| 5 | | 1 | |
| 6 | | 1 | |

Ion currents: ✓None

Connor Stevens Currs

Synapse #1:

Synapse #2: None

Synapse #3: None

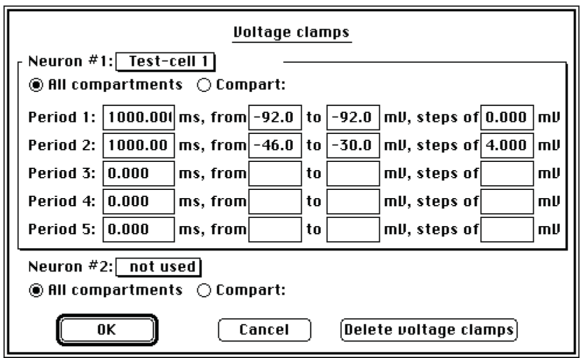
Transmitter: None

Figure 4.11: A Nodus window showing the construction of a compartment that can be passive or have activated channels.

**User interface** The models defined with Nodus are saved in different files linked together by the top simulation file. There are three types of files for storing complete model descriptions: conductance definition files, neuron definition files and (optionally) network definition files. These files are opened when loading the model. The loaded information can then be edited in the various windows.

The conductance definition files contain the parameters of the equations describing the conductance of the voltage-activated channels. The dynamics of the conductance and rate variables can be plotted as a function of the membrane voltage and can be especially useful in cases of non-standard dependencies.

The neuron definition files contain the parameters of the compartmental structure of a neuron as well as the morphological structure of all compartments. The parameters of a compartment can be edited in the compartment dialog windows, one of which is shown in Fig. 4.11. A cylindrical dendritic



Voltage clamps

Neuron #1: Test-cell 1

[✓ All compartments] Compart:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Period 1: | 1000.00( | ms, from | -92.0 | to | -92.0 | mV, steps of | 0.000 | mV |
| Period 2: | 1000.00 | ms, from | -46.0 | to | -30.0 | mV, steps of | 4.000 | mV |
| Period 3: | 0.000 | ms, from | | to | | mV, steps of | | mV |
| Period 4: | 0.000 | ms, from | | to | | mV, steps of | | mV |
| Period 5: | 0.000 | ms, from | | to | | mV, steps of | | mV |

Neuron #2: not used

[✓ All compartments] Compart:

Figure 4.12: Nodus' dialog window used to set up voltage clamps.

compartment is presented, together with the neuron morphology shown graphically. Ionic currents and synaptic connections are defined in the popup windows, which can be activated from the compartment dialog windows. In Fig. 4.11 one can see how a compartment has been made active.

The network definition files specify the parameters of the neuron connections and are always linked with the neuron definition files.

For numerical integration, data from the model definition files are transformed into the simulation database files, whose structure is optimized for mathematical calculations. These files are hidden from the user, but the simulation parameters and the specific settings for graphic and text output, can be shown in the popup windows from the simulation dialog window (see Fig. 4.11)

The compartments in Nodus can be cylindrical or spherical and are connected symmetrically (end to front) or asymmetrically (center to front).

The results of the simulation can be plotted on the following axes: conductances, currents, and voltages, as is shown in Fig. 4.13. A cell was synaptically inhibited and the results of that experiment are presented as graphs of the membrane voltage of 5 compartments (top), synaptic conductances (bottom left) and synaptic currents (bottom right).

Among standard electrophysiological experiments, Nodus can simulate different current injections (constant, pulsed, ramps, sinus, and noise) and voltage-clamping experiments. An experiment of ionic current blocking can be tested.

### 4.3.5 Surf-Hippo

Surf-Hippo is a simulation system based on compartmental modelling which allows working with model cells with a 3-dimensional geometry. Surf-Hippo is written in Lisp and runs on the Unix operating system.

**Integration method** Surf-Hippo performs numerical integration using a variant of the implicit Crank-Nicholson method developed by Hines, which is most

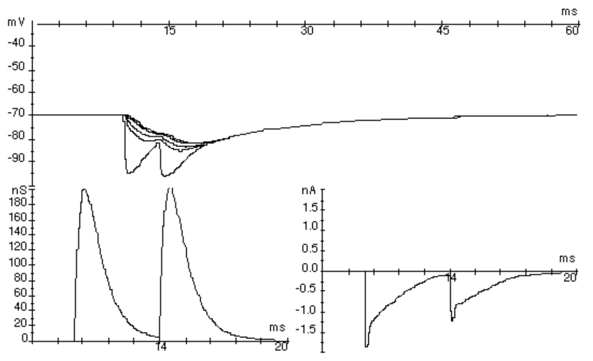


Figure 4.13: Graphic output of simulation results: membrane potential, synaptic conductances and synaptic currents (modelled with the Nodus simulator).

effective for neurons in the branched structure. The improvement made by the Surf-Hippo authors is the adaptive time step that is adjusted according to an estimate of the linear truncation error for all state variables (e.g. node voltages, channel particles). The adaptive time step allows faster numerical integrations, the results of which can be tested with fixed time step integration [4].

**Modelling elements** With Surf-Hippo one can form models with complex dendritic trees in 3-dimensional space, including ionic channels and synaptic inputs.

It is also possible to simulate ionic channels of Hodgkin-Huxley type and the synapses can be voltage- and light-dependent. Integration of such concentration modelling mechanisms as calcium-dependent processes is provided.

**Graphical user interface** The graphical user interface of Surf-Hippo is based on the libraries of the Carnegie Mellon University (CMU) Garnet GUI package and is used for model definition and presentation of the 3-dimensional structure of dendritic trees. The resulting data can be plotted and the graphical output can be exported in the postscript format.

Surf-Hippo can construct models of neurons and neural networks by loading their description from files with such anatomical formats as Neurolucida, NTS.

Knowledge of Lisp is not necessary for working with Surf-Hippo but, as it is written in the Surf-Hippo documentation, will "help in working with program considerably" [5].

**Programming language** The choice of Lisp as the programming language was motivated by its object-oriented structure and significant numerical performance similar to C and Fortran. Another advantage is the ability of Lisp to handle both symbolic and numeric representations. Thus, the description of simulated models can easily be implemented in Lisp. Moreover, Lisp provides an easily adjustable user interface with opportunities to set commands directly from the interpreter or by using scripts. It is possible to extend objects, which needs compiling new code without recompiling the whole source [5].

### 4.3.6 NeuronC

NeuronC was developed as a simulation system incorporating many characteristics of general neural simulators. The distinguishable property of NeuronC is that it was implemented as a simulation language whose syntax is based on the C programming language. NeuronC contains a subset of standard operators such as assignment and mathematical operators, conditional and loop statements, as well as variables of standard types.

The modules of the standard modelling elements are predefined in NeuronC. New components can be added by writing and compiling an appropriate C-like code.

**The origin of NeuronC** NeuronC was initially implemented for simulation of the neural networks that constitute the visual systems. Since the experimental models often are complex for such tasks, modelling at various levels of detail are required for a better understanding of the system functioning.

The origin of NeuronC determines its ability to simulate neural networks containing special elements such as two-dimensional light stimulus and receptors.

**Neural elements** NeuronC allows constructing models hierarchically using a set of "conceptual modules" in the hierarchical structure. Hierarchical structure of neural elements defines the easy extensibility of the models using standard or user predefined modules. The modules can be very simple, such as membrane channel, synapse, dendrite, or more complicated, such as synaptic interconnection pattern, neuron arrays [66].

The input files contain the information about model structure and are interpreted when building the components for simulation.

Although the detailed simulation with NeuronC is based on the compartmental modelling, the user does not need to define all compartments separately. When structure and elements of the neuron network are specified the simulator automatically generate compartmental model. Dendrite cables will be transformed into a set of longitudinally connected isopotential compartments by a cable segmentation algorithm. The predefined element “sphere” usually used for definition of the soma [59].

**Integration method** The functioning of model elements is described by a set of differential equations derived from equivalent electrical circuits of compartments. The default integration method of NeuronC is the variant of the Crank-Nicholson implicit method, which was developed by Hines. One can also

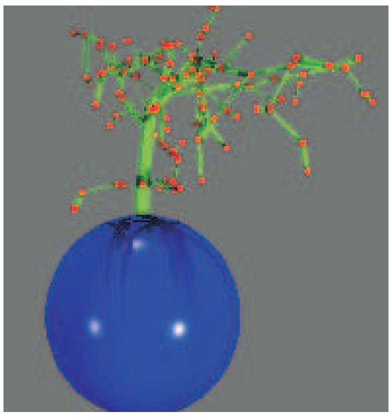


Figure 4.14: The morphology and connectivity of a 3-dimensional network constructed with NeuronC.

perform integration with the backward Euler implicit method or the forward Euler explicit method.

**Operating system** NeuronC runs on Unix. The source code is included in the distribution package.

**Graphical User Interface** Using the NeuronC commands, one can visualize the structure of 3-dimentional network, as illustrated in Fig. 4.14. The graphs of two variables such as the membrane voltage versus time can be plotted.

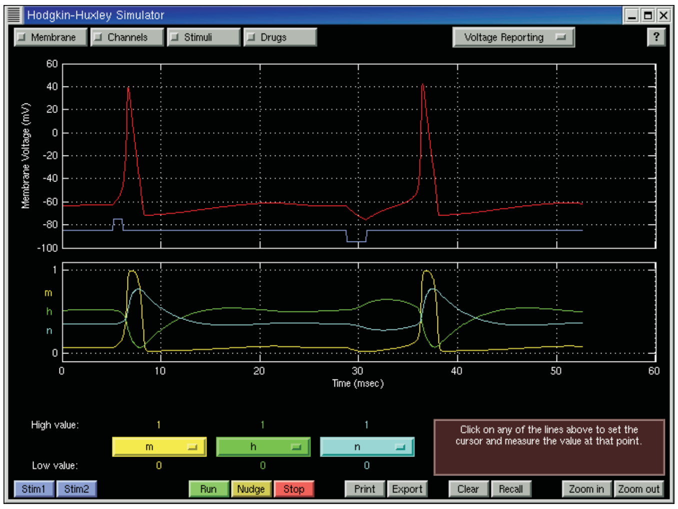
### 4.3.7 HHSim

HHSim is a neural simulator developed for educational purposes. Because of its user-friendly interface, it can also be used for quick simulation tests in neurobiological research.

Already the name of the program indicates the simulated object: a section of membrane with Hodgkin-Huxley voltage-gated channels. The section of the membrane containing ionic channels of Hodgkin-Huxley type is the subject of the test.

HHSim was written in Matlab but the current version of HHSim does not require Matlab installed on the computer, since it is available as an executable file for Windows, Unix and MacOS operating systems.

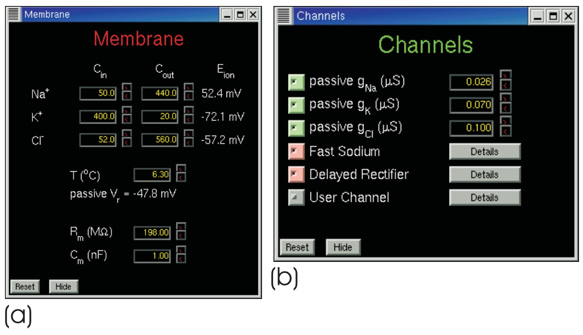
**Working with HHSim** The differential equations describing dynamics of the membrane voltage and channel conductance as well as in the gated variables are hidden from the user. By default, the user can observe the results of the simulation as plots of the membrane voltage, stimulus current and gated



Click on any of the lines above to set the cursor and measure the value at that point.

Figure 4.15: HHSim's main window to control simulation results.

parameters (see Fig. 4.15). Additionally other parameters can be plotted, such as the channel current or channel conductance. There are buttons on the top of the main window which activate the windows to edit the biophysical parameters of the membrane, the parameters of Hodgkin-Huxley channels, and the form of stimuli (see Fig. 4.16).



|  | $C_{in}$ | $C_{out}$ | $E_{ion}$ |
|---|---|---|---|
| $Na^+$ | 50.0 | 440.0 | 52.4 mV |
| $K^+$ | 400.0 | 20.0 | -72.1 mV |
| $Cl^-$ | 52.0 | 560.0 | -57.2 mV |

T (°C) 6.30

passive $V_r$ = -47.8 mV

$R_m$ (MΩ) 198.00

$C_m$ (nF) 1.00

| | | |
|---|---|---|
| ✓ | passive $g_{Na}$ (μS) | 0.026 |
| ✓ | passive $g_K$ (μS) | 0.070 |
| ✓ | passive $g_{Cl}$ (μS) | 0.100 |
| ✓ | Fast Sodium | Details |
| ✓ | Delayed Rectifier | Details |
|  | User Channel | Details |

Figure 4.16: (a) HHSim's Membrane window for adjusting the membrane parameters; (b) HHSim's Channels window providing access to the channel parameters.

The results of the simulation can be saved in postscript or ASCII format or be printed. Some additional inhibition elements are available in HHSim. Three drugs can be applied from the Drug Window: TTX, which inhibits the sodium current; TEA, which inhibits the potassium current; and pronase, which prevents sodium channel inactivation (see Fig. 4.17).

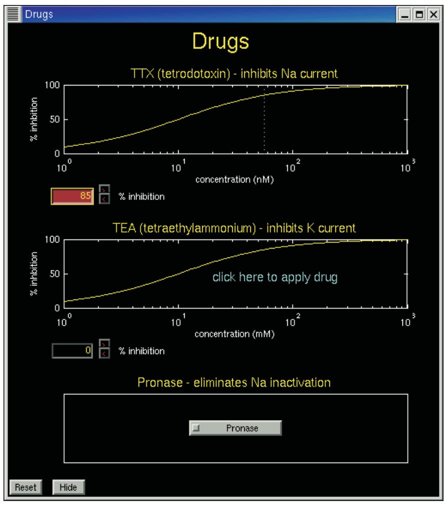

Drugs
TTX (tetrodotoxin) - inhibits Na current
% inhibition
concentration (nM)
65
TEA (tetraethylammonium) - inhibits K current
click here to apply drug
concentration (mM)
0
Pronase - eliminates Na inactivation
Pronase
Reset
Hide

Figure 4.17: The HHSim's Drug window allows application of a few drugs.

## 4.4 Summary

In this chapter we have examined seven simulation packages for biological neural. Table 4.2 summarizes the relevant information about the considered packages. The characteristics given in this table have proven useful. Table 4.1 presents the distinguishing features of the packages, playing an important role when choosing program.

| | |
|---|---|
| GENESIS | general simulation system; parameters are specified with "table lookup"; multiple time scales; extensibility; parallel computing |
| NEURON | general simulation system; numerical method developed by Hines; properties are dependent on the position in the section |
| SNNAP | graphical interface; modular organization of input files; simulation of different types of plasticity and common experimental techniques |
| Nodus | graphical interface, no programming skills are necessary; hierarchical structure of model definition files |
| NeuronC | extensibilty; simulation of experiments on vision |
| Surf-Hippo | 3-dimensional models |
| HHSim | educational software |

Table 4.1: The distinguishing features of the examined packages.

| | GENESIS | NEURON | SNNAP | Nodus | NeuronC | Surf-Hippo | HHSim |
|---|---|---|---|---|---|---|---|
| Oper. system | Unix | Unix, Win, MacOS | any | MacOS | Unix | Unix | Unix, Win, MacOS |
| Impl. language | C | C | Java | Fortran | C | Lisp | Matlab |
| User extensions | yes, for programmers | no | no | no | yes, for programmers | yes | no |
| Int. method | explicit, implicit | implicit | explicit | explicit | explicit, implicit | implicit | no inf. |
| Graphs of results | after programming | for primal research | yes | yes | yes | yes | yes |
| Model definition | interpreter | interpreter | editors | editors | interpreter | files | GUI |
| Ionic channels | yes | yes | yes | yes | yes | yes | HH |
| Synaptic channels | yes | yes | yes | yes | yes | yes | no |
| Networks | yes | yes | yes | limited | yes | yes | no |

Table 4.2: Important properties of compartmental modelling packages.