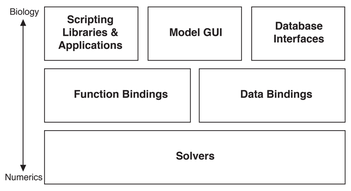
Using GENESIS 3 for single neuron modeling.

Allan D. Coop[1], Hugo Cornelis[2], Mando Rodriguez[2], James M. Bower[2].
[1]Dept. Epidemiology and Biostatistics, [2]Research Imaging Center, University of Texas Health Sciences Center, San Antonio, Texas 78229, USA.

Introduction

GENESIS (www.genesis-sim.org) has recently been reconfigured to adhere to the software design requirements specified by the CBI federated software architecture [1] and is now referred to as GENESIS 3.0 (G-3).

The CBI Federated Software Architecture



This highly modular approach to the development of simulator software leads to independent stand-alone components that integrate on a just-in-time basis. Different modules contribute functionality to the workflow of model development, model exploration and analysis, model simulation, and (ultimately) data analysis and model publication. Here we introduce the G-3 user workflow and through it present examples of how G-3 can be employed to define and examine single neuron models, run simulations, and extract results using modern and backward compatible interfaces.

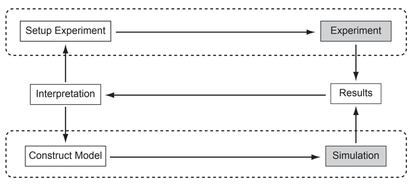
Components of GENESIS 3 (G-3)

Currently, there are nine major G-3 modules associated with either user workflow or publication functionality

1. GENESIS 3 SLI (NS-SLI): Allows G-2 scripts to be used by G-3 by supporting the major functionalities of the G-2 Script Language Interpreter.
2. Studio: Visualizes a model stored by the Model Container and allows it to be queried in the G-Shell.
3. G-3 Interactive Shell (G-Shell): A unified command line interface that enables convenient interaction with G-3 components.
4. Model Container: Used to separate biological entities and end-user concepts from the numerical solvers. By "containing" a biological model, the Model Container makes the implementation of numerical solvers independent of model representation. It provides a highly efficient solver-independent internal storage format for models. This allows user independent optimizations of the G-3 numerical core.
5. Heccer: A highly optimized compartmental solver based on G-2 *hsolve* functionality.
6. Simple Scheduler in Perl (SSP): A highly configurable scheduler that activates and synchronizes the Model Container and Heccer to work together on a single simulation in a just-in-time manner.
7. Graphic User Interface (G-Tube): Evoked from either a Unix terminal prompt or the G-Shell, a GUI is currently being developed.
8. Project Browser: A webserver that allows projects to be browsed and simulation results inspected and compared with a web browser such as Firefox.
9. GENESIS Documentation System: User and developer documentation in seven levels
Level 1: Introductory material.
Level 2: User guides and documentation.
Level 3: Automated regression testing and use cases.
Level 4: Technical guide specification.
Level 5: Algorithm documentation.
Level 6: API documentation.
Level 7: Inline source code documentation

Data Flows in Science

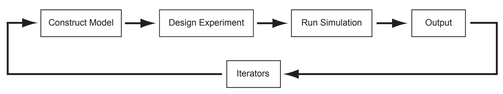
The relationship between experiment and simulation in computational neuroscience is illustrated in the following figure.



Conducting experiments and running simulations are two iterative processes connected by a feedback loop that uses interpretation of results to design new experimental setups and model constructions. From this perspective, simulations provide a framework to organize our understanding of biological systems. GENESIS supports the lower loop within the system as shown above.

GENESIS User Workflow

Over 20 years of experience in the development and use of GENESIS has led to the identification of a user workflow that can be employed to organize activities such as project development, GUI functionality, and documentation such as user tutorials. The typical workflow is composed of five basic steps, summarized in the following figure.



1. Construct Model: Create simple models directly within the G-Shell by entering commands. More complex models can be imported into the G-Shell from either the GENESIS model libraries or from other external model libraries. The model can also be explored, checked, and saved.
2. Design Experiment: Set model parameter values specific to a given simulation, the stimulus parameters for a given simulation run or "experiment", and/or the output variables to be stored for subsequent analysis.
3. Run Simulation: Configure runtime options, check, run, reset simulation, and save model state. The model state can be saved at any time step during a simulation. This allows a model to be imported into a subsequent GENESIS session. Output is flushed to raw result storage for subsequent data analysis.
4. Process Output: Check simulation output and the validity of results to determine whether the output exists in the correct locations. Output can be analyzed either within GENESIS or piped to external applications such as Matlab, xmgrace, or Mathematica.
5. Iterators: A G-3 modeling project is established by the introduction of iterators into the user workflow. The iterators achieve this by closing the loop between the output of results and model construction. Iterators include: automated construction of simulations and batch files; static parameter searching; and active parameter searching using dynamic clamp technology.

GENESIS User Interfaces

To facilitate the G-3 user workflow several different user interfaces have been developed. They include: a Script Language Interface that provides backward compatibility with previous versions of GENESIS (NS-SLI), an interactive shell that interfaces with Python and Perl (G-Shell), and two contributed G-3 components, the Studio and the Project Browser. As the various components of G-3 are independent stand-alone modules, they can also be called directly from the command line of a Unix terminal window, or the G-3 console window to be provided with the Windows version.

Overview of New GENESIS Functionality

G-3 functionality is now introduced through the steps in the user workflow. We use examples from two recent projects that employed G-3: (i) Morphological and functional comparison of cerebellar Purkinje neurons from four different species (see Box 1 for details), and (ii) Quantification of the contribution of dendritic channel activity to information processing in a cerebellar Purkinje cell model (see Box 2 for details).

Construct Model

Backward compatibility: G-3 supports backward compatibility with G-2 scripts. Model cells generated with previous versions of GENESIS (the *.g* and *.p* file formats) are accessible to the G-3 environment via the module NS-SLI. This module provides a bridge between the G-2 SLI and G-3. For example, the following command will import a given G-2 model into the Model Container inside the G-Shell, run the defined simulation, and save any generated output:

```
genesis> sli_run /usr/local/nsgenesis/tests/scripts/PurkM9_model/CURRENT9.g
```

Alternatively, the command:

```
genesis> sli_load /usr/local/nsgenesis/tests/scripts/PurkM9_model/CURRENT9.g
```

will load the G-2 model and all its dependencies. The model can then be explored, checked and saved. For example, to find the number of segments and dendritic branches loaded for the model:

```
genesis> querymachine segmenterlinearize /Purkinje
      Number of segments: 4548
      Number of segments without parents: 1
      Number of segment tips: 1474
```

The properties of a given compartment can be found with e.g.

```
genesis> show_model_parameters /Purkinje/segments/b0s03[56]
  -
    'parameter name': RA
    type: number
    value: 2.5
  -
    'parameter name': RM
    type: number
    value: 1
  -
    'parameter name': CM
    type: number
    value: 0.0164
```

The scaled value of specific parameters can be checked, e.g. the membrane capacitance:

```
genesis> show_parameter_scaled /Purkinje/segments/b0s03[56] CM
    scaled value = 6.50291e-12
```

Parameter values can also be reinitialized, e.g. the membrane potential of a given compartment:

```
genesis> set_model_parameter /Purkinje/segments/b0s03[56] Vm_init -0.0680
```

The NDF File Format: A novel feature of G-3 is its powerful new declarative file format (NDF). This replaces those aspects of the GENESIS 2 Script Language Interpreter (SLI) that support model construction and exploration.
A NDF file has four sections that are not necessarily filled, but must be present in the given order. They include a: (i) Preamble, (ii) Import, (iii) Private Models, and (iv) Public Models sections (indicated in bold in the figure below) in a file that has the following general form:

```
#!/usr/local/bin/neurospacesparse
//-*- NEUROSPACES -*-
// default location for file comments
NEUROSPACES NDF
IMPORT
    FILE <namespace> "<directorypath>/<filename.ndf>"
        . . . <other files may be imported as required>
END IMPORT
PRIVATE_MODELS
    ALIAS <namespace>::/<source label> <target label> END ALIAS
        . . . <other aliases may be defined as required>
END PRIVATE_MODELS
PUBLIC_MODELS
    CELL <morphology name>
        SEGMENT_GROUP segments
            . . . <morphological details>
        END SEGMENT_GROUP
    END CELL
END PUBLIC_MODELS
```

To enhance simulator interoperability, models specified in NDF, SWC, Python, Perl, and XML formats can be loaded into the G-Shell. They can then be operated on in a seamless and integrated manner along with any G-2 SLI models that may be present to develop a new model cell:

```
genesis> swc_load morphologies/C170897A-P3.CNG.swc
genesis> xml_load channels/hodgkin-huxley/gaba.xml
genesis> ndf_load channels/hodgkin-huxley/ampa.ndf
genesis> npl_load channels/hodgkin-huxley/na.npl
genesis> npy_load channels/hodgkin-huxley/k.npy
```

Once one or more models have been loaded they can be saved in the G-3 NDF file format along with any changes that have been made to the original model(s):

```
genesis> ndf_save /Purkinje Purkinje_1.ndf
```

Once a model is imported, it can be explored either directly from the G-Shell (as described above) or via the Studio. The Studio provides a GUI that supports direct exploration of model parameters. Importantly, for backward compatibility, as suggested above, once a G-2 model has been loaded into the G-Shell, the Studio can be employed to explore model parameters and structure. Here, e.g. are details of a rat Purkinje neuron used in the Purkinje neuron comparison study (see Box 1):

Design Experiment

Experimental design primarily consists of specifiying the inputs and outputs of a simulation. Inputs consist of how a model is activated, e.g. by current injection, voltage clamp, or synaptic activation. A 2nA current injection can be set at the soma with:

```
genesis> set_runtime_parameter /Purkinje/segments/soma INJECT 2e-9
```

and checked with:

```
genesis> show_runtime_parameters
    runtime_parameters:
      - component_name: /Purkinje/segments/soma
        field: INJECT
        value: 2e-9
        value_type: number
```

Alternatively, the Perfect Clamp utility provides a simple voltage clamp protocol to one or more specified compartment(s) of a neuronal morphology. Here, e.g. the voltage clamp circuitry object is created with a holding potential of -60mV:

```
genesis> add_inputclass perfectclamp voltage_clamp_protocol /Purkinje
    voltage_clamp_protocol command -0.060
```

Apply the voltage clamp to the Purkinje cell soma:

```
genesis> add_input voltage_clamp_protocol /Purkinje/segments/soma Vm
```

There is no limit to the number of current injection protocols or voltage clamps that can be added to a model.
Synaptic activation is easily implemented, e.g. to activate GABAergic synapses at 1 Hz and 25 Hz Poissonian stimulation on the thick dendrites and spine heads of a Purkinje cell, respectively:

```
genesis> set_runtime_parameter thickd::gaba::/Purk_GABA FREQUENCY 1
genesis> set_runtime_parameter spine::/Purk_spine/head/par FREQUENCY 25
```

The default output of a simulation is the membrane potential at the soma. However, output can be specified for any compartment, here, e.g. two dendritic compartments:

```
genesis> add_output /Purkinje/segments/b0s03[56] Vm
genesis> add_output /Purkinje/segments/b1s06[137] Vm
```

Output of all derived variables known to Heccer can be obtained with the **`add_output`** command. Specifying one or more outputs prevents the default output of the membrane potential at the soma being written to file, unless it is also specified. Simulations can also be run with the magnitude of a current injection or synaptic event times read from a data file.

Run Simulation

Prior to running a simulation, a variety of runtime options may be set, these include amongst others: the update timestep and choice of simulator and simulation algorithm. A check can be performed to ensure, e.g. that the model and the experiment to be run during a simulation are syntactically correct, variables are not out of range, and solvers are correctly initialized:

```
genesis> check /Purkinje
```

A simulation is run, e.g. 500ms with:

```
genesis> run /Purkinje 0.5
```

By default, when issuing sequential run commands, a simulation will continue from its state at the end of the previous run.
The state of a model can be saved at any timestep of a simulation with a command that saves the model state following the last update timestep of a run:

```
genesis > simulation_state_save /Purkinje Purkinje_1_run_1.nms
```

A simulation can then be (re)started from this saved state:

```
genesis > simulation_state_load Purkinje_1_run_1.nms
```

The next run command will start the simulation from this reinitialized state. Such functionality provides a convenient way to save different states of a simulation and is particularly useful for eliminating redundant startup calculations.
A model can easily be reset to its initial state along with any user defined runtime parameters with:

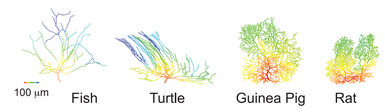
```
genesis> reset /Purkinje
```

Output

Simulation output can be found at its default location:

```
genesis> sh cat /tmp/data.out
```

Output can be explored both visually and quantitatively via the Studio. For example in the following figure (neurons from the Purkinje comparison study) the dendrites are color-coded (Red: 1ms, Blue: 20ms) according to the somatic response after unitary stimulation of a given dendritic location:



To determine whether the dendritic membrane potential is stable or oscillates in conjunction with somatic spiking, a more complex output such as the standard deviation from the average dendritic membrane potential during somatic spiking can easily be visualized with the Studio following post-processing of simulation output (result not shown).
Simulation output can also be piped to external stand-alone applications such as Matlab, xmgrace, or Mathematica for post-processing and visualization.

Iterators

Provide a crucial step in model development by matching simulation output to experimental data to tune model parameters and thus model behavior. This can be done either by brute force, as is typically the case with static parameter space searching with automatically generated batch files or dynamically by using dynamic clamp output to tune model parameters in real time.

Simple Scheduler in Perl (SSP)

A schedule is a hierarchical enumeration of 'keys' that tells SSP what to do. A SSP schedule defines: which external modules must be loaded by SSP, how these modules are linked together (if necessary), how to activate the modules, what the outputs of a simulation are, and how to finish and clean up at the end of a simulation. If some of these things are not defined in the schedule, SSP automatically assumes defaults.
Both the Purkinje cell comparison and information processing study referred to in this poster were sufficiently complex that simulations were run under SSP control. To override the builtin schedule that runs a default simulation of a cell in GENESIS, it is sufficient to load the required SSP file:

```
genesis> ssp_load
    generated__edsjb1994__excitation_12__inhibition_0.5__kc_800__cap_45.yml
```

An SSP file can be automatically generated from the runtime parameters and options, inputs, and outputs specified during a GENESIS session:

```
genesis> save_ssp
generated__edsjb1994__excitation_25__inhibition_1.0__kc_800__cap_45.yml
```

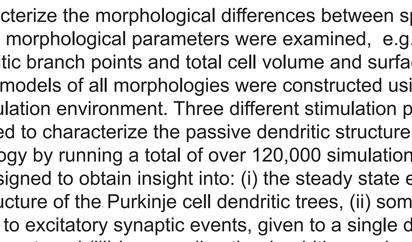
This command does not save the model (for which **`ndf_save`** should be used), but rather all the information required to successfully run the specified simulation.

The GENESIS Documentation System

As outlined above, the GENESIS documentation system provides documentation ranging from introductory background material and tutorials for users, to Doxygenized APIs and HTMLified browsable source code for developers. This documentation is available both from the GENESIS web site (http://www.genesis-sim.org/) and as context dependent help in the new GENESIS GUI (referred to as G-Tube, see below). With the exception of Level 1 and 2, documentation is automatically generated directly from regression tests and source code. This provides a guide for writing the Level 1 and 2 documentation covering specific GENESIS functionality. It employes a YAML-tagged flat file system maintained in a server repository under control of a distributed version control system. Tags define whether these documents are maintained only locally and visible only to authorized members of a given project or are exposed to the world by publication at the GENESIS web site. The system is fully automated and rebuilds itself every 2 hours, such that any new files are quickly and automatically incorporated into the documentation system.

G-Tube

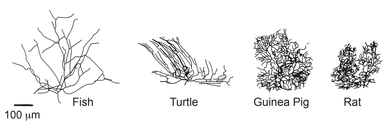
The G-3 GUI is fully configurable and starts from the user workflow. The figure below shows a popup menu revealed by rolling over Step 1 in the user workflow. The menu item Load Model has been selected. This generates a list of the NDF files that are available. The model *edsjb1994_partitioned.ndf* has been selected and loaded into the G-Shell. The G-Shell (lower window in GUI) reports that the model is successfully loaded. It can now be explored before continuing with Step 2 of the user workflow.



Conclusions

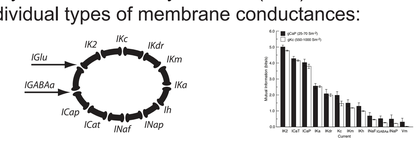
The CBI architecture exhibits several novel features. For example, the separation of stimulus protocols from a given model cell to partition a model from its numerical simulation. This allows both to be combined dynamically at run time for a given simulation and simplifies the generation of batch files for multiple simulations. Importantly, this separation allows any simulation to be run with any model, a feature that greatly facilitates iterative model development and comparison.
The isolation of a cell model from a given simulation configuration also enables convenient exploration and quantification of properties of the biological model. For example, no special functions or scripting are required to quantify properties of a model cell such as total or partial volumes or surface areas, the number of dendritic branches or branch points per dendritic tip, or the average somatopetal to dendritic tip distance. Further, with G-3, the Model Container can be queried directly by simple commands and command line options.

BOX 1: Morphological and functional comparison of cerebellar Purkinje neurons from four different species.
This study aimed to (i) determine the computational consequences of differences in the dendritic morphologies of Purkinje cells found in different species of mammals and non-mammals, and (ii) characterize the morphological and functional difference between cerebellar Purkinje neurons from different species, including: fish, turtle, guinea pig, and rat.



To characterize the morphological differences between species, common morphological parameters were examined, e.g. number of dendritic branch points and total cell volume and surface area. Passive models of all morphologies were constructed using the G-3 simulation environment. Three different stimulation protocols were used to characterize the passive dendritic structure of each morphology by running a total of over 120,000 simulations. Studies were designed to obtain insight into: (i) the steady state electrotonic structure of the Purkinje cell dendritic trees, (ii) somatic responses to excitatory synaptic events, given to a single dendritic compartment, and (iii) by recording the dendritic membrane potential resulting from somatic action potentials generated with a simulated dynamic voltage clamp at the soma. Morphologies were visually and quantitatively different between the different species. Mammalian Purkinje cells have more dendritic tips and branch points, and a greater overall surface, but the overall cell volume is lower, resulting in a greater surface to volume ratio. The distance between the soma and the dendritic tips is decreased in mammals. The steady state membrane potential of the dendrites is heavily attenuated after voltage clamp at the soma for some parts of the dendritic tree in fish and turtle Purkinje cells. For all the examined morphologies, somatic spikes resulted in an elevated non-oscillating dendritic membrane potential.

BOX 2: Quantification of the contribution of dendritic channel activity to information processing in a cerebellar Purkinje cell model.
This study used a previously published Purkinje cell model [2, 3] with updated synaptic kinetics. The model consisted of 1,600 compartments. 1,474 identical spines each composed of a neck and a head were attached to the dendritic compartments. One excitatory synaptic contact was made with each dendritic spine and 1,695 inhibitory GABAa-type synaptic contacts were distributed at random across the dendrites. Ionic channels were distributed over three zones of the model Purkinje cell, with Na and fast K channels in the soma, fast K channels in main dendrite, and Ca channels and Ca-activated K channels distributed throughout the entire dendritic tree. We calculated the mutual information between the sum of dendritically located excitatory currents (IGlu) and the summed current of individual types of membrane conductances:



As ICaP and IKc have been previously reported to exhibit the largest currents during cell discharge (De Schutter & Bower, 1994), gCaP and gKc were modulated over the ranges of 45-70 and 550-1000 Sm^{-2} (around the fitted values of 45 and 800 Sm^{-2}, respectively) to quantify their influence on information transfer.
While the firing rate was constant for different combinations of synaptic input, mutual information was very sensitive to such changes, thus disambiguating synaptic activity in dendrites. Results suggest that dendritic excitability modulated by Ca^{2+} and KCa channels is effective in regulating information transfer between excitatory synapses and membrane channels studied, and thus any possible processing of that information.

References

[1] Cornelis H, Edwards M, Coop AD, Bower JM (2008) The CBI architecture for computational simulation of realistic neurons and circuits in the GENESIS 3 software federation. BMC Neuroscience 9:P88.
[2] De Schutter E & Bower JM (1994) An active membrane model of the cerebellar Purkinje cell. I. Simulation of current clamps in slice. J Neurophysiology 71:375-400.
[3] De Schutter E & Bower JM (1994) An active membrane model of the cerebellar Purkinje cell II. Simulation of synaptic responses. J Neurophysiology 71:401-419.