A Simulation and Optimization Environment for Models in Computational Neurobiology[*]

Blaž Zupan[a] and John A. Halter[b]

[a]AI Laboratory, J. Stefan Institute, Jamova 39, Ljubljana, Slovenia

[b]Div. Restorative Neurology and Human Neurobiology, Baylor College of Medicine, One Baylor Plaza, Houston, TX 77030, U.S.A.

Computational neurobiology is an emerging field which employs models to simulate the behavior of real neural systems. We present an environment that can be used to guide simulations using existing neural models. This environment is based on Perl, a powerful and standard script language. It also contains a standard library of routines for common simulation tasks. Included in the environment is an optimization package. Both simulation and optimization routines incorporate a novel approach to simulation data storage and retrieval.

1. Introduction

Contemporary simulation packages for modeling neural systems vary in complexity, flexibility, and performance. Such packages are used to model systems from a single nerve fiber [1] to a complex multicellular neural system [2,3]. To utilize these models, one usually has to provide extensive number of model parameters, do a series (several hundred or thousands, cf. [4–6]) of simulations and interpret the results. To assist in this process, we present a simulation environment that: (1) is used to guide the simulations using the existing neural models, (2) is based on a powerful and standard script language, (3) incorporates a novel approach to simulation data storage and retrieval, and (4) includes a flexible optimization package.

2. Method

A case for which we have originally developed our environment is a realistic distributed-parameter model of the mammalian myelinated nerve fiber [1,7] (Fig.1). A simulation of this model requires a set of input parameters which specify the electrophysiological and anatomical data (e.g. conductivities and resistances, geometries of the segments, stimulus specification). There are circa 30 parameters, and most of them have to be specified for

[*]Support for this effort was generously provided by the W.M. Keck Center for Computational Biology, the Whitaker Foundation, and Developmental Support from Information Technology Program at Baylor College of Medicine. B. Zupan is also supported by the Ministry of Science and Technology of the Republic of Slovenia.

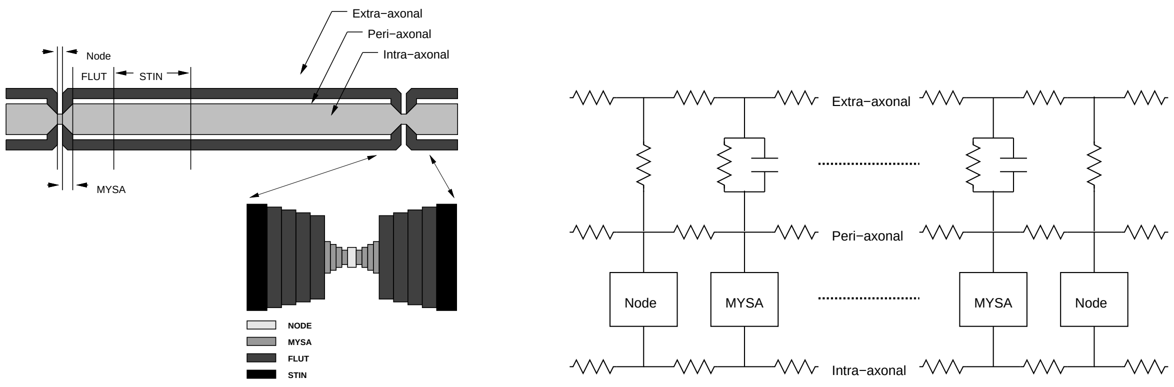

Figure 1. Structure of the myelinated nerve fiber model. NODE: Node of Ranvier, MYSA: MYelin Sheath Attachment region, FLUT: FLUTed region (non-cylindrical), STIN: STerotyped Internodal Region (left) and its multi-axial electrical equivalent circuit (right).

each of the segments of the neuron. Each simulation run can produce different data showing spatial and temporal dependencies of potentials and currents. These are then used to extract clinical measures such as the conduction velocity of propagation of action potentials. When the model is used for exploration of parameter space, several simulations are required. These are usually driven by the results of the previous simulation runs, each of which can take several minutes or more of CPU time.

2.1. Simulation

The simulation environment is based on the standard script language Perl [8]. The user provides a script which drives a simulation such as a loop that reads the parameter specification file, modifies some of the parameters, executes the simulation, and extracts the results (cf. Fig.2).

A typical simulation script file consists of a series of calls to simulation library routines and other user defined statements contained within Perl control statements. Simulation library routines are divided into low-level and high-level routines. Low-level routines perform the tasks such as changes of electrophysiological parameters, changes of the shape of the neuron (e.g. shrinking the neuron's diameter), invocation of model simulation, determination of conduction velocity from simulation output data, etc.

Simulation library high-level routines use the low-level routines and provide a compact interface to complex tasks. The user sets a type of the task to be performed and then calls a specific high-level routine which then builds a corresponding model(s), executes one or a series of simulations, analyzes the simulation results, and returns the outcome of the analysis.

For example, to determine the rheobase (minimal stimulus where neuron "fires") one would have to iteratively set the stimulus, execute the simulation, and test for the firing condition using the axon potential output data. Instead, setting the simulation method to `rheobase`, a call to `&sim_exec` library routine does all the work. Other high-level library routines are used for the tasks such as conduction velocity and action potential amplitude measurements and determination of depolarization required to reach the firing threshold. Each such task might require a single simulation of a model (as in the case of conduction velocity measurements) or a series of simulations (e.g. determination of the rheobase).

Through the use of library routines the user is assured that the course of simulation will be recorded to the log file. This stores all the routines calls that where used to build or change the model, names of the reference models used, and types, times and a reference codes of the invocations of the model simulator. This log file can be used for data retrieval purposes (see below) and for performance analysis.

```
require "seng.pl";
$in_data_name = "neuron.data";
&init("multiple");
print RES "sim\tscale\tcv\n";
do {
  &sim_exec;
  $cv = &get_cv;
  printf(RES "%d:\t%lf\%\t%lf\n",
   $snum-1,0.9**($snum-1)*100,$cv);
  &scale_fib_diam(0.9);
} until $cv < 70;
&sim_close;
```

```
sim scale  cv
0:  100.0  71.379
1:   90.0  71.186
2:   81.0  70.943
3:   72.9  70.428
4:   65.6  69.986
```

```
SIMN 0
TIME Wed Mar 8 14:33:07 1995
SRUN triax -iv tmp.data
COMM extracting cv = 71.378
OPER &scale_fib_diam(0.900);
...
SIMN 4
TIME Wed Mar 8 14:41:37 1995
SRUN triax -iv tmp.data
COMM extracting cv = 69.985
OPER &scale_fib_diam(0.900);
```

Figure 2. An example of simulation script (left), an output file produced with the script (middle), and a corresponding log file (right). The script scales the nerve fiber radious of the model defined in `neuron.data` and determines the conduction velocity. The iterations stop when conduction velocity falls below 70m/s.

2.2. Data Storage and Retrieval

Each input parameter file usually stores several hundred kilobytes of data. For information retrieval reasons, rather than storing the parameter file for each simulation a series of simulations are based on a single “reference” data file. A log file is used to record every change to the reference file. These changes are recorded by a library routine invocation that resulted in a data change. After the completion of simulation script, one can extract the input data for each simulation by scanning the log file and modifying the reference data file accordingly.

Substantial memory space (several megabytes) can be occupied by the output data from each simulation. As the process of using the model can be time consuming because of the number of simulations required versus the duration of a single simulation, the output data is not stored and can be retrieved by re-executing the simulation. This substantially conserves secondary memory space with minimal time cost.

2.3. Optimization

Commonly, models of nerve fiber are used to answer questions like "What should the fiber diameter be for a nerve to conduct impulses with a velocity of 72 meters-per-second?" or "What is the optimal internodal distance?". These types of questions can be answered with optimization.

As a part of environment presented in this paper, we have developed an optimization package. The user provides a set-up file specifying the criteria function and optimization parameters (precision, method, files to be used) and a script file that computes the variables needed to evaluate the criteria function (see Fig.3). This script is basically a simulation script augmented with optimization specific routine calls. The trace of simulation performed during optimization is again recorded in the log file, which also stores the simulation results.

In general, a criteria function may be an expression over any of the variables defined within the optimization script. It usually addresses the parameters that are obtained from the output files of simulations, such as conduction velocity, rheobase, etc. Recently we have upgraded the package to include also a set of routines that evaluate the dynamic behavior of the model. For example, given a reference action potential data of some simulation or even real experiment, one can optimize over the range of the parameters so as to get the model that exhibits the most similar behavior in terms of action potential. Due to the generality of our approach, one of course does not need to limit herself only on a single reference file. Namely, the measure of similarity derived using this file might be just one of the factors in the criteria function.

```
DATA   elev.data

OPTIMIZE ON
$scf FROM 0 TO 100=70

OUTPUT $cv

CRITERIA ($cv-70)**2
```

```
require "../../optlib.pl";
&set_opt_val();
&sim_init_method("cv_vax");
&scale_fib_diam($scf/100);
&sim_exec;
$cv = &get_cv;
&report_results;
&sim_close;
```

```
# $scf      $cv       ($cv-70)**2

1 7.000e+01 7.051e+01 2.695e-01
2 7.000e+01 7.051e+01 2.690e-01
3 6.972e+01 7.065e+01 4.236e-01
4 7.044e+01 7.062e+01 3.925e-01
5 7.017e+01 7.043e+01 1.919e-01
...
```

Figure 3. An example of a simple optimization setup (left), corresponding optimization script (middle), and an excerpt of autoproduced result file (right). The question posed to the optimizer was to find the nerve fiber diameter (scaled from the reference model with factor `$scf` that exhibits the conduction velocity of 70m/s. Optimizer determined that such fiber should have diameter of 70.03% of that of the reference model. Optimizer executed 23 simulations and used 56 approximations.

Optimization is based on the conjugate gradient method [9]. In contrast with similar packages (cf. [4] where the conjugate gradient method is combined with B-splines fitting to derive the local minima), ours utilizes the results of previous simulation runs. These are used for approximation of a criteria function every time the one has to be evaluated with a new set of parameters. If the approximation is estimated to be "sufficiently confident", the approximation is used instead of simulation. This technique can substantially reduce the time spent for optimization and minimize the computational requirements.

Optimization uses different approximation techniques, like nearest neighbors method, regression trees, estimation with back-propagation artificial neural network, and local regression models. An optimization set-up file is used to specify which approximation technique or their combination to use. Nearest neighbors method might be the simplest and most efficient when a parameter space is dense with data points for which the criteria function might be computed. Other techniques are more computationally expensive and, with unappropriate setup, might even use more time for approximation than a simulation would require. From other methods, our experiments showed a substantial reduction of overall optimization time when using a combination of nearest neighbors method with local regression. First method is used to select the data points, while the second builds a local regression using the techniques described in [10]. Starting with empty set of known experimental results and adding to this set a result of each simulation performed, we have observed that such techniques might save up to 50% of optimization time compared to the optimization that does not use past experimentation results.

3. Conclusion

When we started to design the simulation environment presented here, the goal, apart from functionality and flexibility, was to provide a tool to minimize the time researcher spends interacting with the simulator as well as the required computer resources. Both were observed to be substantial when we re-executed the experiments reported in [5] and in current experimentation with the influence of structural changes on nerve fiber behavior.

Though the environment was initially used with a specific neurocomputational model, it is general enough to be fitted for other models. We are currently using this new environment for management of a more complex type of nerve fiber model. We are also extending the environment to explore the higher grain concurrency, parallelizing the optimization method and introducing parallel constructs in the simulation scripts.

REFERENCES

1. J. A. Halter and J. W. Clark. A distributed-parameter model of the myelinated nerve fiber. *J. Theo. Biol.*, 148:345–382, 1991.
2. M. A. Wilson, U. S. Bhalla, J. D. Uhley, and J. M. Bower. GENESIS: A system for simulating neural networks. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, pages 485–492. Morgan Kaufmann, San Mateo, CA, 1989.
3. M. Hines. NEURON - a program for simulation of nerve equations. In F. Eeckman, editor, *Neural Systems: Analysis and Modeling*, pages 127–136. Kluwer Academic Publishers, 1993.
4. U. S. Bhalla and J. M. Bower. Exploring parameter space in detailed single neuron models: simulations of the mitral and granule cells of the olfactory bulb. *J. Neurophysiology*, 69(6):1948–1965, June 1993.
5. J. A. Halter, J. S. Carp, and J. W. Woplaw. Operantly conditioned motoneuron plasticity: possible role of sodium channels. *J. Neurophysiology*, 73(2):867–871, 1995.

6. J. A. Halter and J. W. Clark. The influence of nodal constriction on conduction velocity in myelinated nerve fibers. *Neuro Report*, 4:89–92, 1993.
7. J. A. Halter, D. Micci Barreca, and B. Zupan. Implementation of myelinated nerve fiber model on a MIMD computer. In *Proc. of the IFAC Symposium*, pages 511–512, Galveston, TX, 1994.
8. L. Wall and R. L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., Sebastopol, CA, 1992.
9. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1994.
10. W. S. Cleveland and E. Grosse. Computational methods for local regression. *Statistics and Computing*, 1:47–62, 1991.