A Java Interactive Pedagogical Tool for Experimenting with Neural Circuits

Ethan McBride *egmcbride@ucsd.edu*
Kyle Fischer *kbfische@ucsd.edu*
Stephen Johnston *stjohnst@ucsd.edu*

Neurosciences Graduate Program, University of California, San Diego

Abstract

This project aims to develop an online user interface to implement simple user designed neural circuits to serve as demonstration and education material for high school and college level students and educators. The user interacts with a graphical user interface (GUI) which on the front end allows the user to assemble, probe, and alter simple neural circuits; while on the back end the GUI implements the FitzHugh-Nagumo neuron model in Java as a system of alterable point neurons with adjustable connection matrices and a low-order differential equation solver to allow for real-time computation and visualization of action potential generation and propagation. This interface can serve as a demonstration of the complex nonlinear interactions that drive neurons, while increasing understanding of how the brain works on a cellular and computational level in a way accessible to the general public. The final aim is the publication of the GUI online, where users can interact with it, and increasing the complexity of the model and interface to demonstrate more complex behaviors and greater user definability.

1 Introduction

In the last few decades, the field of computational neuroscience has expanded exponentially, due in part to the emergence of new technologies that enable researchers to gain new insight into various aspects of how the brain works. Mathematical models of neuronal activity have existed for a century, beginning with Lapicque's publication of a simple integrate and fire model in 1907, and continuing with the more biologically accurate Hodgkin and Huxley model published in 1952. However, it was not until the advent of modern computers and the rapid, continuous increase in their computational power that the activity of single neurons and even networks of neurons could

be simulated at a large scale. These advances in simulation technologies have provided countless valuable discoveries. One of the consequences is that it, and neuroscience in general, is becoming an increasingly popular field. While many research tools have been developed that enable investigators to simulate everything from highly realistic neurons to entire brains, these techniques require a large amount of technical skill and as a result have a steep learning curve, especially for somebody new to neuroscience.

In order to provide beginning neuroscientists and laypeople with a tool they could use to better understand basic principles of neuroscience, we decided to build a simple, easy-to-use, and still relatively accurate neural circuit simulator. Our specific goals for the project were that the simulator could run on any modern personal computer while allowing the user to dynamically alter the properties of individual neurons, the configuration and size of the network, and that this would be published online and accessible to anyone. We decided to use the Processing language to build our simulator because it is designed for coding visual projects, and the Fitzhugh-Nagumo neuron model because of its relative simplicity, and thus lower computational requirements.

2 Methods

2.1 The Processing programming language

The object-oriented programming language Processing was employed for the construction of the GUI. Processing is a open source programming language and environment modified from java which makes it easy to create live animations and implement in html5 (available at: processing.org). This allows not only the authors to create a readily accessible front-end GUI but also allows users, even with relatively little programming experience, open access to alter back-end coding. Neuron spiking is visualized in the GUI by a change in neuron color saturation with the capability to add "electrodes" to measure the actual voltage traces of individual neurons, the ability to increase neuron firing by altering current injected into an individual neuron, switch the synapses of a given neuron between excitatory and inhibitory (visualized as green for excitatory and red for inhibitory), and move and connect neurons and synapses using simple keyboard and mouse commands. Individual "neurons" were implemented as objects to allow for easy addition, adjustment of individual neuron tuning parameters, and neuron-neuron interactions. processing.js (available at: processingjs.org) was used to convert the GUI processing file (.pde) to JavaScript in browser and then implemented as an HTML5 canvas. The GUI can be found at www.neurodraw.com, after December 17th, 2012.

2.2 Biological neuron models

To this end, several common biological neuron models were implemented: the Hodgkin-Huxley model (HH), the Fitzhugh-Nagumo model (FN), and the Hindmarsh-Rose model (HR). The classic HH model, as the first biologically derived mathematical neuron model, allows for a great deal of complex behavior and nonlinear dynamics by accounting for various ion channel conductance with corresponding voltage-dependent state variables. The

system of differential equations is as follows:

$$C_m \frac{dV}{dt} = -I_{Na} - I_K - I_L + I_{ext}$$
$$I_{Na} = g_{Na} m^3 h\,(V - E_{Na})$$
$$I_K = g_K n^4\,(V - E_K)$$
$$I_L = g_L\,(V - E_L)$$

where C_m is the membrane capacitance. $I_{Na}$is the Sodium current with capacitance g_{Na}, gating variables m and h, and equilibrium potential E_{Na}. The same hold for I_K, the Potassium current, and I_L.the leak current (Table S1). I_{ext} is injected current. The gating variables, for x = {m, n, h}, are then defined as:

$$\frac{dx}{dt} = \alpha_x(V)(1 - x) - \beta_x(V)x$$
$$\alpha_m(V) = 0.1\frac{V + 45}{1 - e^{-\frac{V+45}{10}}}$$
$$\beta_m(V) = 4.0\, e^{-\frac{V+70}{18}}$$
$$\alpha_h(V) = 0.07 e^{-\frac{V+70}{20}}$$
$$\beta_h(V) = \frac{1}{1 - e^{-\frac{V+40}{10}}}$$
$$\alpha_n(V) = 0.01\frac{V + 60}{1 - e^{-\frac{V+60}{10}}}$$
$$\beta_n(V) = 0.125\, e^{-\frac{V+70}{80}}$$

The FN model retains spiking properties and nonlinear dynamics while simplifying the system to two state variables:

$$\frac{dV}{dt} = V\,(\alpha - V)(V - 1)w + I_{ext} \quad \textit{with gating variable}: \frac{dw}{dt} = \epsilon(V - \gamma w)$$

Constant values can be found in Table S2.

Finally, the Hindmarsh-Rose model allows for increased complexity with the addition of third "bursting" variable: z(t), which increases the qualitative description of the neuron model by accounting for additional empirically observed spiking patterns (i.e. bursting):

$$\frac{dV}{dt} = y + aV^2 + V^3 - z + I_{ext}$$
$$\frac{dy}{dt} = 1 - bV^2 - y$$
$$\frac{dz}{dt} = r[s(V - V_r) - z]$$

Constant values can be found in Table S3..

In the present iteration only the FN model is available to users due to its

reliance on fewer dependent variables, increased computation speed, and back-end accessibility without a major loss in the demonstrative capacity of the program. However, it is our hope that future versions will include ready access to multiple models for comparison and increased dynamic complexity.

Each of these models had the addition of synaptic currents, where they were formed by contact between a synapse and a neuron in the GUI, modeled by:

$$I_{syn} = g_{syn} r \left(V_{post} - E_{syn}\right)$$

$$\frac{dr}{dt} = \alpha_r [T](1 - r) - \beta_r r \; where \; [T] = \; [\mathrm{T}]_{\max}/(1 \; + \; \mathrm{e}^{-\frac{\mathrm{V_{pre}} - \mathrm{V_p}}{\mathrm{K_p}}})$$

2.3 Integration methods

Several integrative methods were explored to varying success. First, the first-order Euler method proved simple to implement, however the system dynamics proved to fast to yield accurate results. Second, the classical fourth order Runge-Kutta method (e.g. RK4) gave highly accurate results with an increased cost of computation. Third, the Runge-Kutta-Fehlberg method (e.g. RKF45), with its capacity to estimate the error between fourth and fifth order solutions and then automatically determine integrative step-size, provided an increased capacity to reduce error below a defined threshold; however, for the purpose of these simulations the increased accuracy was negligible. Therefore, the RK4 method was employed to provide highly accurate results while minimizing computational demand and thereby provide greater usability, in particular when a large number of neurons are simultaneously implemented.

3 Results

3.1 Processing was an effective language

The use of Processing as the basis of our program provided a sleek and simple interface. Being a stripped down scripting language based on Java and designed for artists, the language was easy to learn and our group was quick to create both programs for testing neuronal models as well as visually appealing user interfaces. While perhaps not up to the caliber of professional game designers, our final interface is easy to use and manipulate: we have been able to implement nearly every design we have imagined. Furthermore, the new Processing JavaScript library in conjunction with new HTML5 standards allow for a simple online publication of the script. Importantly, this allows users to access the interface from any browser without having to download it as an application and keeping all computation client-side. The simplicity of this language, however, does have its drawbacks. The script is built around an infinite draw() loop which has a frame rate which is relatively computationally intensive to dynamically regulate and made the application of variable-step differential equation solvers difficult to apply. As we further develop the environment and receive feedback we may look into moving the script to Adobe Flash, in which many games are designed, or Java, which

would allow for more control at the loss of simplicity the Processing package provides.

3.2 The Fitzhugh-Nagumo model was ideal for the project

The Fitzhugh-Nagumo model was targeted due to its simplicity. With two state variables, the model significantly decreased the computational load when compared to the 4 state variables of the Hodgkin-Huxley model. We were also unable to produce a functioning Hodgkin-Huxley model at all but the smallest of steps, which made its application slow and cumbersome. Further development of efficient variable-step solvers may remedy this problem.

We also had success with the Hindmarsh-Rose model. Due to its similarity with Fitzhugh-Nagumo model, this model was similarly attractive for its light computational load and flexibility. This model is also interesting due to its Z state, which allows for more chaotic neural activity. While we decided to stick with the Fitzhugh-Nagumo model for simplicity, future editions of the interface may utilize the Hindmarsh-Rose model and allow for modification of cell "burstiness" through its Z state variable.

3.2 The RK4 integration method was the most useful

The Runge-Kutta 4 (RK4) method proved to be the most useful numeric differential equation solver we attempted to implement. The success of any of our solvers was highly based on the chosen step size.

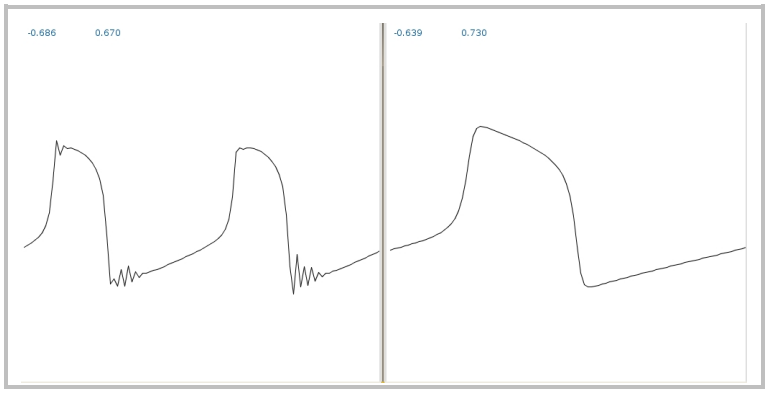

Figure 1: Two waveforms of Fitzhugh-Nagumo model neurons with 1 (left) and 0.5 (right) stepsizes. While a step size of 1 creates more noise, this is rarely perceivable and more accurately shows the spiking shape of an action potential than the right trace which, despite being much smoother, is sluggish and impractical for our application.

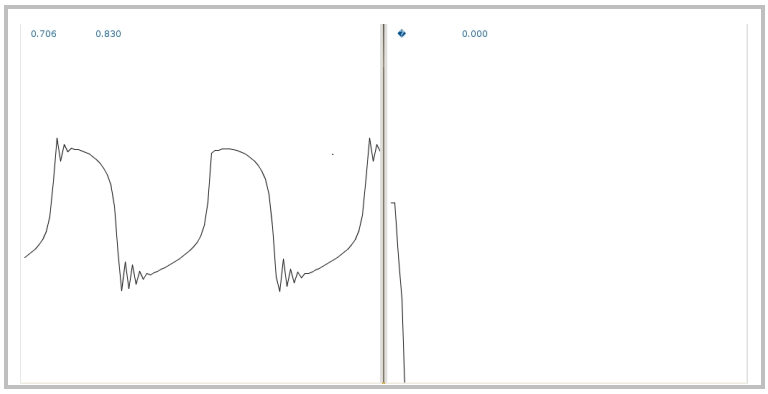

Figure 2: These two waveforms of Fitzhugh-Nagumo model neurons illustrate the importance of proper step size choice: the left is at a step size of 1, while the right is at 1.1. The increase has catastrophic results, leading the state variable for voltage to jump to negative infinity.

While the Euler method was sufficient in producing the most basic sinusoidal shape of action potentials in the FN model, severe high frequency oscillations were produced at points of quick voltage change, such as the peaks and troughs of action potentials, in even the best circumstances.

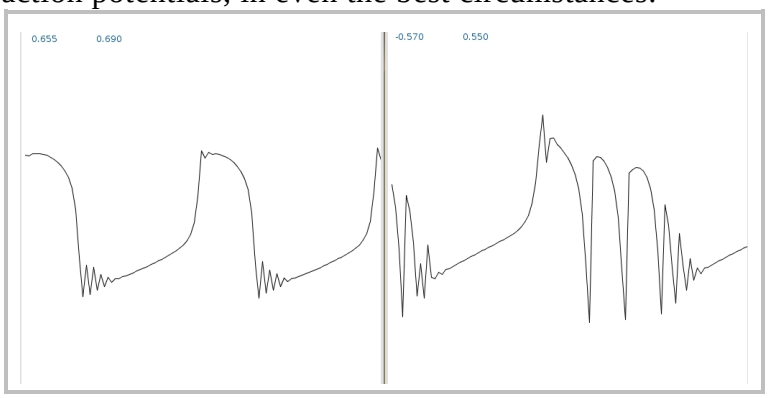

Figure 3: Two waveforms of Fitzhugh-Nagumo model neurons using the RK4 (left) and Eulers (right) solving methods. The Euler's method, though functional is much noiser and more volatile than the RK4 method.

At the double the computational cost (4 calculations verses 2), the RK4 proved to be much more reliable, even if it was lacking some of the "peakiness" of action potentials.

Due to our success with the RK4 model, our attempts to apply a variable-step solver were limited. We attempted to apply the Runge-Kutta-Fehlberg solver, often referred to as RK45. While this function was able to give a more faithful representation of the FN model, we chose not to use it for two reasons: (1) the computed optimal step-size was often incredibly small, and due to the static frame rate, made the model progress at variable speeds and (2) the computation required 12 computations to produce the suggested step size and the final increment, 3 times the number calculations of the basic RK4. While

we came up with ways of circumventing the first issue, by dynamically varying the frame rate or pausing the draw() function or optimizing the threshold step-size variable, the added computational weight of the process and functionality of the basic RK4 made the use of the RK45 unwieldy and unnecessary. Further research into the application of a technique called “step doubling”, which is less computationally costly, may yield more useful results.

3.3 “Final” product

We were largely successful in implementing functional neural models in a simple user interface. The final product contains two neuron types, excitatory and inhibitory, that are in green and red, respectively. The interface also includes two “recording electrodes” which, when placed over the desired cell, show the waveform at the bottom of the application. All cells, electrodes, and synapses can be moved with the mouse. Synaptic current is only transmitted when the synapses are moved over a neighboring cell and new axons can be made by hitting the “+” key while hold the mouse over the target cell or destroyed by hitting the “-” while holding the mouse over the target axonal buton. New neurons can be produced by hitting the “n” key and the neural type (inhibitory/excitatory) can be changed by holding the mouse over the target cell and hitting the “i” or “e” keys. The synaptic current for each synapse is currently hard coded, but future editions may allow for direct user control.

While the design process has been smooth, a heated debate has grown amongst the group members over the best website name on which to host the program. The debate has delayed the online publication of the program, but the final consensus appears to be on the name www.NeuroDraw.com, where the program will be published on Dec. 17th, 2012.

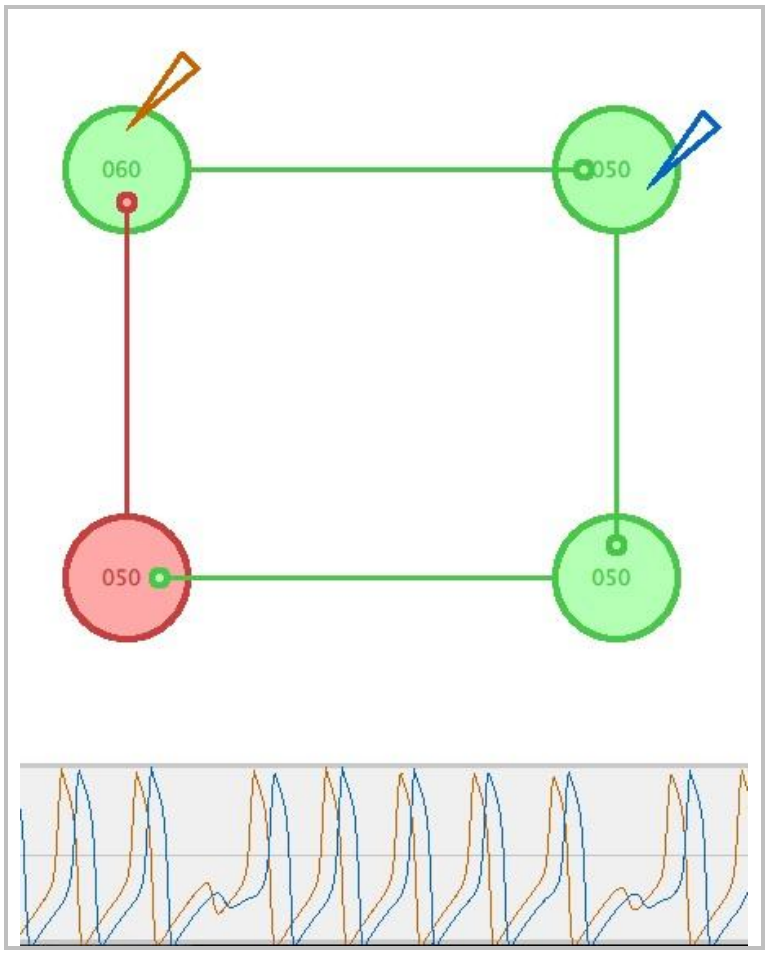

Figure 4: Screen shot of a four neuron network with electrode traces from NeuroDraw.

4 Discussion

4.1 Project summary

Our project in its current state fulfills our initial goals: It is a user-friendly interface that will run on any modern computer, and it allows dynamic manipulation of simple neural circuits with a real-time readout of neuronal activity.

4.2 What needs improvement

While the Processing language was convenient for this project, it has some disadvantages, most notably the fixed frame rate, which makes implementation of variable-step differential equation solvers very difficult. We may look into moving the script to Adobe Flash, in which many games are designed, or Java, which would allow for more control, but with a loss of simplicity the Processing package provides.

Another area we would like to improve is the interface. At present, most of the commands to the program are issued by pressing different keys, but as more variables are modifiable, we need to assign and remember more keys. One thing we could do to improve this area would be to implement a toolbar into the program where instead of pressing keys to change variables, the user could simply press clearly labeled buttons within the interface. On a related note, we would like to implement different "modes" of complexity into the program so that users without much knowledge of the nervous system wouldn't be overwhelmed by the multitude of variables, and so more advanced users could still change any variables they wanted. An easy way to do this would be to add a button to the toolbar where you can toggle between "Beginner Mode," where only the injected current to each neuron could be changed, and "Advanced Mode," where many other variables, such as synaptic strength, could be altered. Additionally, a similar button which toggles the use of different neuronal models for the simulation could be interesting for people wanting to compare the advantages and disadvantages of the different models.

4.3 Future directions

There are several future directions that we could pursue with this project. Our primary goal is to promote this GUI as an educational tool. Along these lines, we have implemented it on a website and plan to ask various neuroscientists and non-neuroscientists to try it out and give us feedback. This will enable us to continue to improve the GUI in ways we may not have thought of ourselves. We also would like to add a sensory neuron class, which could respond to the proximity of the mouse to the sensor, for example. If we also were to add a motor neuron class and a simple muscle, we could build a whole circuit with a sensory input and a motor output. This would allow users to play around with realistic circuits, such as the stretch reflex circuit found in the spinal cord, and learn basic principles of how sensory input can lead to motor output.

One could imagine other potential uses for this project besides straight education, including using it as a first-pass test for hypotheses about simple neural circuits. Because of its simplicity it may not be able to test complex hypotheses, but it could be used as more of a brainstorming tool, or even a hypothesis generator. Another idea is that it could be built into a game format, i.e. building circuits to accomplish some motor output task. If implemented well, this game format could potentially be used with crowdsourcing to find ways that small neural networks accomplish simple tasks. However, bringing these ideas to fruition will require a lot more time invested in the project.

4.4 Summary/Conclusions

In summary, we are very satisfied with the current state of our project, but we would like to continue improving it and adding features. We believe that this GUI could be a very useful educational tool for anyone wanting to explore the basics of how neurons interact with each other within simple circuits.

289 **Supplemental**

Table S1: HH model																				
	I_{Na}		I_K		I_{Na}		$I_{inhibitory}$							$I_{excitatory}$						
C_m	g	E	g	E	g	E	g	E	α_r	β_r	$[T]_m$	V_p	K_p	g	E	α_r	β_r	$[T]_{max}$	V_p	K_p
1.0	120	45	36	-82	0.3	-59.4	1.0	-80	5.0	.18	1.5	7.0	5.0	0.4	-38	2.4	.56	1.0	5	7

290

Table S2: FN model																
			$I_{inhibitory}$							$I_{excitatory}$						
α	ϵ	γ	g	E	α_r	β_r	$[T]_{max}$	V_p	K_p	g	E	α_r	β_r	$[T]_{max}$	V_p	K_p
0.15	0.009	0.8	0.2	-0.8	0.1	0.01	1.0	0.5	0.1	0.2	0.8	0.1	0.01	1.0	0.5	0.1

291

Table S3: HR model																		
					$I_{inhibitory}$							$I_{excitatory}$						
a	b	r	s	V_r	g	E	α_r	β_r	$[T]_{max}$	V_p	K_p	g	E	α_r	β_r	$[T]_{max}$	V_p	K_p
3.0	5.0	10^{-3}	4.0	-1.4	0.2	-0.8	0.1	0.01	1.0	0.5	0.1	0.2	0.8	0.1	0.01	1.0	0.5	0.1

292