Computational Neuroscience Summer Program: Introductory Course

June 1 – 4, 2010

Instructors: Dr. Joshua Jacobs (`jojacobs@psych.upenn.edu`)

Jeremy Manning (`manning3@mail.med.upenn.edu`)

Suggested texts: *Theoretical Neuroscience*, Dayan and Abbott

Principles of Neural Science, Kandel, Schwartz, and Jessell

Matlab for Neuroscientists, Wallisch *et al.*

Course overview: This intensive introductory course is intended to familiarize students with basic techniques in computational modeling and analysis of neural data using Matlab. Students may (and are encouraged to) work together on assignments, but each student will be expected to hand in their own work. Assignments will be reviewed, but no formal grades will be assigned.

Course Outline:

Ethics training	June 1 (AM)
Introduction to programming in Matlab	June 1 (PM)
Introduction to computational modeling	June 2 (AM)
Integrate-and-fire neuron model	June 2 (PM)
Hodgkin-Huxley neuron model	June 3 (AM)
Extensions of the Hodgkin-Huxley model	June 3 (PM)
Introduction to data analysis	June 4 (AM)
Free lab time	June 4 (PM)

Note: The above course outline is approximate and is subject to change pending students' needs and interests. Because of the brief duration of this course, we are only able to provide a small "taste" of the diverse and evolving field of computational neuroscience. Students seeking more in-depth coverage of computational neuroscience, including the topics discussed in this course, are encouraged to read the suggested texts.

MATLAB Cheat Sheet

Basic Commands

`%`	Indicates rest of line is commented out.
`;`	If used at end of command it suppresses output. If used within matrix definitions it indicates the end of a row.
`save filename`	Saves all variables currently in workspace to file `filename.mat`.
`save filename x y z`	Saves x, y, and z to file `filename.mat`.
`save -append filename x`	Appends file `filename.mat` by adding x.
`load filename`	Loads variables from file `filename.mat` to workspace.
`!`	Indicates that following command is meant for the operating system.
`...`	Indicates that command continues on next line.
`help function/command`	Displays information about the function/command.
`clear`	Deletes all variables from current workspace.
`clear all`	Basically same as clear.
`clear x y`	Deletes x and y from current workspace.
`home`	Moves cursor to top of command window.
`clc`	Homes cursor and clears command window.
`close`	Closes current figure window.
`close all`	Closes all open figure windows.
`close(H)`	Closes figure with handle H.
`global x y`	Defines x and y as having global scope.
`keyboard`	When placed in an M-file, stops execution of the file and gives control to the user's keyboard. Type `return` to return control to the M-file or `dbquit` to terminate program.
`A=xlsread('data',...` `'sheet1','a3:b7')`	Sets A to be a 5-by-2 matrix of the data contained in cells A3 through B7 of sheet `sheet1` of excel file `data.xls`
`Succes=xlswrite(...` `'results',A,'sheet1','c7')`	Writes contents of A to sheet `sheet1` of excel file `results.xls` starting at cell C7. If successful `success`= 1.
`path`	Display the current search path for `.m` files
`addpath c:\my_functions`	Adds directory `c:\my_functions` to top of current search path.
`rmpath c:\my_functions`	Removes directory `c:\my_functions` from current search path.
`disp('random statement')`	Prints `random statement` in the command window.
`disp(x)`	Prints only the value of x on command window.
`disp(['x=',num2str(x,5)])`	Displays `x=` and first 5 digits of x on command window. Only works when x is scalar or row vector.
`fprintf(...` `'The %g is %4.2f.\n', x,sqrt(x))`	Displays `The 3 is 1.73.` on command window.
`format short`	Displays numeric values in floating point format with 4 digits after the decimal point.
`format long`	Displays numeric values in floating point format with 15 digits after the decimal point.

Plotting Commands

`figure(H)`	Makes H the current figure. If H does not exist is creates H.

	Note that H must be a positive integer.
`plot(x,y)`	Cartesian plot of x versus y.
`plot(y)`	Plots columns of y versus their index.
`plot(x,y,‘s’)`	Plots x versus y according to rules outlined by s.
`semilogx(x,y)`	Plots $\log(x)$ versus y.
`semilogy(x,y)`	Plots x versus $\log(y)$.
`loglog(x,y)`	Plots $\log(x)$ versus $\log(y)$.
`grid`	Adds grid to current figure.
`title(‘text’)`	Adds title `text` to current figure.
`xlabel(‘text’)`	Adds x-axis label `text` to current figure.
`ylabel(‘text’)`	Adds y-axis label `text` to current figure.
`hold on`	Holds current figure as is so subsequent plotting commands add to existing graph.
`hold off`	Restores hold to default where plots are overwritten by new plots.

Creating Matrices/Special Matrices

`A=[1 2;3 4]`	Defines A as a 2-by-2 matrix where the first row contains the numbers 1, 2 and the second row contains the number 3, 4.
`B=[1:1:10]`	Defines B as a vector of length 10 that contains the numbers 1 through 10.
`A=zeros(n)`	Defines A as an n-by-n matrix of zeros.
`A=zeros(m,n)`	Defines A as an m-by-n matrix of zeros.
`A=ones(n)`	Defines A as an n-by-n matrix of ones.
`A=ones(n,m)`	Defines A as an m-by-n matrix of ones.
`A=eye(n)`	Defines A as an n-by-n identity matrix.
`A=repmat(x,m,n)`	Defines A as an m-by-n matrix in which each element is x.
`linspace(x1, x2, n)`	Generates n points between $x1$ and $x2$.

Matrix Operations

`A*B`	Matrix multiplication. Number of columns of A must equal number of rows of B.
`A^n`	A must be a square matrix. If n is an integer and $n > 1$ than `A^n` is A multiplied with itself n times. Otherwise, `A^n` is the solution to $A^n v_i = l_i v_i$ where l_i is an eigenvalue of A and v_i is the corresponding eigenvector.
`A/B`	This is equivalent to `A*inv(B)` but computed more efficiently.
`A\B`	This is equivalent to `inv(A)*B` but computed more efficiently.
`A.*B,A./B,` `A.\B,A.^n`	Element-by-element operations.
`A’`	Returns the transpose of A.
`inv(A)`	Returns the inverse of A.
`length(A)`	Returns the larger of the number of rows and columns of A.
`size(A)`	Returns of vector that contains the dimensions of A.
`size(A,1)`	Returns the number of rows in A.
`reshape(A,m,n)`	Reshapes A into an m-by-n matrix.

`kron(A,B)`	Computes the Kronecker tensor product of A with B.
`A = [A X]`	Concatenates the m-by-n matrix A by adding the m-by-k matrix X as additional columns.
`A = [A; Y]`	Concatenates the m-by-n matrix A by adding the k-by-n vector Y as additional rows.

Data Analysis Commands

`rand(m,n)`	Generates an m-by-n matrix of uniformly distributed random numbers.
`randn(m,n)`	Generates an m-by-n matrix of normally distributed random numbers.
`max(x)`	If x is a vector it returns the largest element of x. If x is a matrix it returns a row vector of the largest element in each column of x.
`min(x)`	Same as `max` but returns the smallest element of x.
`mean(x)`	If x is a vector it returns the mean of the elements of x. If x is a matrix it returns a row vector of the means for each column of x.
`sum(x)`	If x is a vector it returns the sum of the elements of x. If x is a matrix it returns a row vector of the sums for each column of x.
`prod(x)`	Same as `sum` but returns the product of the elements of x.
`std(x)`	If x is a vector it returns the standard deviation of the elements of x. If x is a matrix it returns a row vector of the standard deviations for each column of x.
`var(x)`	Same as `std` but returns the variance of the elements of x.

Conditionals and Loops

```
for i=1:10
  procedure
end
```

Iterates over `procedure` incrementing i from 1 to 10 by 1.

```
while(criteria)
  procedure
end
```

Iterates over `procedure` as long as `criteria` is true.

```
if(criteria 1)
  procedure 1
elseif(criteria 2)
  procedure 2
else
  procedure 3
end
```

If `criteria 1` is true do `procedure 1`, else if `criteria 2` is true do `procedure 2`, else do `procedure 3`.

Problem Set 1 -- Introduction to programming in Matlab

Computational Neuroscience Summer Program

June, 2010

Log into the computer with your PennKey and start Matlab. Please put your answers to this assignment in a single Microsoft Word document. This document should include your raw Matlab code, the commands you type to run the code, and its output text and plots.

Some of these questions require functions that you may not have seen before. If you have questions, please ask the instructors and feel free to consult Matlab's great built-in help functions and online tutorials at mathworks.com.

1. Concatenation

Write a script that creates the variables a = [1 2 3], b = 4, c = 5, d = [6; 9], e = [7 8]. Use only these variables to create the matrix m = [1 2 3; 4 5 6; 7 8 9] in a single line of code through horizontal & vertical concatenation. (Hint: use nested [ ] operations.) Now change the script into a function. What is the effective difference between calling the script and the function?

2. Number Classification

Write a function that will indicate whether an input number is negative, positive, or zero, as well as even or odd. The function should print these results to the command window and not return anything.

3. For loop

Write a function that creates a Fibonacci sequence (a series of numbers where each element is the sum of the previous two elements; you'll have to start it with [1 1]). It should have an input parameter that indicates the desired length of the output sequence.

4. Plotting

Download this file and load it into matlab:

http://memory.psych.upenn.edu/~josh/Q4.mat

Plot the variable 'Z' from this file using the standard plot command.

Next, create a function *plot_labeled_peak* that accepts one input vector. It should plot the vector in black. In addition, the largest data point in this vector should be labeled with a red circle, and the smallest data point should be labeled with a blue 'x'. Show the result of this function when plotting 'Z'.

5. Challenge questions: Performance analysis

A.

Here we are interested in calculating the lengths of many three-element vectors according to the pythagorean theorem (i.e., sqrt($a^2 + b^2 + c^2$)). Write a function that takes a three-column matrix, and returns the length of the vector in each row using a loop. Use *randn* to create random three-column matrices with 10, 100, 1000, & 10000 rows, and measure how long it takes your function to compute each one. Next, create a similar function that does the same calculations in a vectorized fashion. Compare the performance of the two by plotting the execution times of each of the two functions versus the number of rows in the matrix.

B.

In Matlab, preallocating a vector in a single command (by using the ‘zeros’ or ‘ones’ functions, for example) can make things operate much more quickly than repeatedly appending individual elements to the end of a matrix. Write a version of the Fibonacci function from Question 3 that preallocates the output. How fast does it calculate a 20,000 element Fibonacci sequence? How does this speed compare to the previous version of this function? Plot the Fibonacci execution times of various sequence lengths, for both versions of this procedure.

C.

A common data analysis tool is to *normalize* a dataset so that it has a mean of zero and a standard deviation of 1. Here is a function that takes in a large matrix and returns a normalized dataset where each row has a mean of zero and a standard deviation of 1:

```
function x=normData_loop(x)
for row=1:size(x,1)
    m=mean(x(row,:));
    s=std(x(row,:));
    x(row,:)=(x(row,:)-m)/s;
end
```

Can you write a vectorized version of this function? (Hint: use `repmat`.) How does the execution time of your function compare with `normData_loop` on a 1000 by 1000 matrix (e.g., `rand(1000,1000)`)?

Lecture 2 – Introduction to computational modeling

Computational Neuroscience Summer Program

June, 2010

Motivation. This lecture is intended to give students a general intuition for basic mathematical language used to describe and model neurons. These principles will serve as the foundation for future lectures.

Basic organization of the brain. The brain is typically divided into 4 *lobes*. The *temporal lobe* contains neural machinery for processing speech and sounds, spatial information, and for encoding episodic (autobiographical) memories. The *parietal lobe* is involved with sensory perception, sensory integration, and memory. The *frontal lobe* is associated with personality, reasoning, planning, problem solving, working memory, and movement. The *occipital lobe* is primiarily involved with vision and visual processing. The *central sulcus* separates the primary motor cortex (frontal lobe) from the primary somatosensory cortex (parietal lobe). The *medial longitudinal fissure'* separates the right and left hemispheres of the brain. The *spinal cord* sends signals from the primary motor cortex to the body's skeletal muscles.

Neuron anatomy. Neurons are electrically excitable cells in the brain. Neurons "listen" to other cells via branch-like *dendrites*. Signals from the dendrites travel down to the cell body, or *soma*, which contains the nucleus of the cell. Neurons communicate with other cells by sending electrical impulses down their *axons*, which most often synapse onto the dendrites of other neurons.

Action potentials. Neurons communicate with each other by changes in their membrane voltage (we'll get to what this means in a bit). Small changes in membrane voltage are picked up by neurons up to approximately 1 mm away. Thus, for nervous systems on the scale of 1 mm, such as the fruit fly nervous system, no other special mode of communication is needed. However, in larger nervous systems (e.g. ours), neurons fire action potentials – sudden changes in voltage. These form the basic mode of neural communication in the brain. Over the next few lectures we'll be trying to understand how action potentials come about by modeling neurons in increasing levels of detail.

The neuron as a fluid-filled ball. Each μm^3 of cytoplasm contains on the order of 10^{10} water molecules, 10^8 ions (e.g. sodium, potassium, calcium, chloride), 10^7 small molecules (e.g. amino acids, nucleic acids), and 10^5 proteins. Relative to the extracellular space, the inside of the cell is negatively charged (the difference is carried by about 1 out of every 100,000 ions). This results in a voltage (V) across the membrane of approximately -70 mV.

The neuron as a capacitor. Excess negative charges in the cell oppose each other and line up around inside of membrane. This attracts an equal number of extracellular positive ions, which line up outside the cell. In this way, the membrane builds up charge – it's acting as a capacitor! The amount of charge (Q) stored by the membrane is given by the following equation:

$$C_m V = Q$$

English description: the amount of charge stored by the membrane is equal to the ability of the membrane to store charge (i.e., its capacitance) multiplied by the voltage difference across the membrane. The total membrane capacitance (C_m) is proportional to the surface area of the cell (A):

$$C_m = c_m A$$

Specific capacitance (c_m) depends on conductance and thickness of membrane, which is about the same for all neurons – about 10 nF/mm^2. Neurons typically have a surface area of 0.01 – 0.1 mm^2, so C_m ranges from around 0.1 – 1 nF. We can now compute the number of charges stored by a given neuron (we'll assume 1 nF total capacitance and -70 mV membrane potential):

$$1\text{nF} \times -70\text{mV} = 10^{-9}\text{F} \times 70 \times 10^{-3}\text{V} = 70 \times 10^{-12}\text{C} = 10^{9}\text{charges}.$$

Note: A Columb is 1 Farrad × volt.

Changes in current. Membrane current is a measure of the number of charges per second that travel across the membrane. Current is measured in amps – 1 amp is 1 Columb per second:

$$I = \frac{dQ}{dt}$$

In order to compute the membrane current, we can take the time derivative of the equation for determining how much charge the membrane stores:

$$C_m \frac{dV}{dt} = \frac{dQ}{dt} = I$$

Example: suppose $C_m = 1$ nF. Then injecting $I = 1$ nA of current causes the membrane voltage to rise by 1 volt per second (i.e., 1 mV per millisecond).

Membrane current. There are two components of current (I). The first is membrane current. The membrane contains ion channels – these let specific neurons through. They can open and close.

One type of channel is the sodium channel. The inside of the cell contains fewer sodium ions than outside the cell. When the sodium channels open, sodium (positively charged) flows into the cell and causes the membrane voltage to increase. Diffusion of sodium and other ions (e.g. potassium) is called the membrane current, I_m.

Driving force and the equilibrium potential. In addition to sodium being driven to flow down its concentration gradient, one can make it more or less difficult for sodium to enter the cell by changing the membrane voltage. Because sodium is positively charged, decreasing, or *hyperpolarizing*, the membrane voltage (inside relative to outside) will make sodium ions more likely to flow into the cell. Conversely, increasing, or *depolarizing*, the membrane voltage will make sodium ions less likely to flow into the cell. The membrane potential at which net flow of an ion stops is called the equilibrium potential, E. When $V > E$, positive ions flow out of the cell. When $V < E$, positive ions flow into the cell. This means that V is driven towards E. Thus, we sometimes refer to the quantity $(V - E)$ as the *driving force* across the cell membrane. When the driving force is negative, positive ions are driven out of the cell. When the driving force is positive, positive ions are pulled into the cell.

External current. The second component of current is current that is injected into the neuron from external sources (e.g. if we stick an electrode into the neuron and pump in current).

The change in membrane voltage V due to some change in current I follows Ohm's Law:

$$V = IR,$$

where R is the membrane resistance, described next.

Membrane resistance. Ion channels are like little holes in the membrane. They let ions pass through them – i.e., they conduct ions. A given unit area of membrane has some number of open channels, and we can measure the ease with which ions pass through those channels – the specific conductance, g_m. The total conducatance is proportional to the neuron's area:

$$G_m = g_m A$$

By convention, we tend to talk about the inverse of conductance, which is called resistance. Whereas conductance is proportional to the surface area of the neuron, resistance is proportional to the inverse of the surface area of the neuron:

$$R_m = \frac{r_m}{A}$$

Note that the membrane resistance often changes as a function of voltage, which makes things interesting – we'll get to this later.

The Neuron Equation. Previously we had:

$$C_m \frac{dV}{dt} = I,$$

which we can update to reflect that I is comprised of both membrane and external currents:

$$C_m \frac{dV}{dt} = I_e + I_m.$$

I_m depends on the driving force $V - E$ and also difficulty with which ions flow through the membrane – i.e., the membrane resistance, R_m. In particular

$$I_m = \frac{1}{R_m}(E - V)$$

Note that the order of the E and V terms in the driving force have been swapped. This is because the internal and external currents need to go in opposite directions. We can multiply both sides of the equation by R_m for convenience:

$$C_m R_m \frac{dV}{dt} = R_m I_e + E - V$$

Since $C_m = c_m A$ and $R_m = \frac{r_m}{A}$, the A's cancel, and we get $c_m r_m$, which is independent of the cell's surface area. Since $c_m r_m$ determines the rate at which the cell's membrane potential changes, it is given a special variable, τ_m, or the membrane time constant. ***The equation for all neuron models we'll see in this mini-course is***:

$$\tau_m \frac{dV}{dt} = R_m I_e + E - V$$

V_∞. From the above equation, we see that the change in membrane voltage is some fraction (proportional to τ_m) of the difference between $R_m I_e + E$ and the current membrane voltage, V. By this equation the membrane voltage approaches $R_m I_e + E$ over time. For convenience we can define

$$V_\infty = E + R_m I_e,$$

where V_∞ is the membrane voltage that will be reached given an external current and membrane resistance, and an infinite amount of time.

Resting potential. If you shut off the external current (i.e., set $I_e = 0$), then $V_\infty = E$. For this reason, we call E the resting potential of the cell – the potential the cell approaches if we remove external forces.

Computing the voltage at a particular time, t. We need to solve the following differential equation for V:

$$\tau_m \frac{dV}{dt} = R_m I_e + E - V$$

We know that, given enough time, V tends towards V_∞, so we can say that at time t:

$$V(t) = V_\infty + f(t)$$

Now we need to find $f(t)$ (we'll abbreviate $f(t)$ as f for convenience). We use the equation above, substituting in $V_\infty + f$ for V:

$$\tau_m \frac{df}{dt} = R_m I_e + E - V_\infty - f$$

Since $V_\infty = R_m I_e + E$, those terms cancel and we have:

$$\tau_m \frac{df}{dt} = -f$$

so

$$\tau_m df = -f dt$$

$$\tau_m \frac{df}{f} = -dt$$

Now that we have the f's and t's on different sides of the equation, we can take the integral from 0 to t of both sides:

$$\int_{f(0)}^{f(t)} \tau_m \frac{df}{f} = \int_0^t -dt$$

Solve the right hand side yields:

$$\int_{f(0)}^{f(t)} \tau_m \frac{df}{f} = -t$$

We can solve the left hand side using the rule $\int \frac{1}{x}dx = ln(x)$:

$$\tau_m[ln(f(t)) - ln(f(0))] = -t$$

$$\tau_m ln\frac{f(t)}{f(0)} = -t$$

Now we can solve for $f(t)$:

$$ln\frac{f(t)}{f(0)} = \frac{-t}{\tau_m}$$

$$\frac{f(t)}{f(0)} = e^{\frac{-t}{\tau_m}}$$

$$f(t) = f(0)e^{\frac{-t}{\tau_m}}$$

Previously, we said that the voltage changes as a function of the distance between the voltage at the present time and V_∞. So $f(0) = V(0) - V_\infty$, and $f(t) = f(0)e^{\frac{-t}{\tau_m}}$.

Plugging $f(t)$ back into the original equation gives:

$$V(t) = V_\infty + f(t) = V_\infty + (V(0) - V_\infty)e^{\frac{-t}{\tau_m}}$$

This gives us a way to compute how long it will take to charge up the neuron to an arbitrary voltage $V(t)$, by solving for t.

Sanity checks. For $t = 0$, $e^{\frac{-t}{\tau_m}} = 1$, so we get:

$$V(0) = V_\infty + V(0) - V_\infty = V(0)$$

When t is very large, $e^{\frac{-t}{\tau_m}}$ approaches 0, so $V(t)$ approaches V_∞.

Problem Set 2 – Introduction to computational modeling

Computational Neuroscience Summer Program

June, 2010

In this problem set you will be exploring the relation between several fundamental neuronal properties. For all problems, you should assume a specific membrane capacitance of $c_m = 10$ nF/mm^2, a specific membrane resistance of $r_m = 1$ MΩ·mm^2, and a resting membrane potential of $E = -70$ mV. You should perform all calculations for a variety of cell surface areas (use realistic values – between 0.01 and 0.1 mm^2). Write up your results in a text editor of your choosing. Include any relevant figures. Each question can be answered in 1-2 sentences. Include a printout of your Matlab code as well as any calculations that aren't in the code. You may work individually or in groups, but each student should hand in their own report.

Equations

$$C_m = A \cdot c_m \qquad R_m = \frac{r_m}{A}$$
$$\tau_m = c_m \cdot r_m \qquad V_\infty = E + R_m I_{ext}$$
$$V(t) = V_\infty + (V(0) - V_\infty)e^{\frac{-t}{\tau_m}}$$

Problems

1. Plot (seperately) total membrane capacitance (C_m) and total membrane resistance (R_m) as a function of cell surface area (A). Your graph should include appropriate units. Briefly describe the relation between C_m, R_m, and A.

2. Compute the membrane time constant τ_m. What does τ_m mean in terms of the neuron?

3. How much external electrode current would be required to hold the neuron at the membrane potentials below?

$$V_\infty \in \{-80, -75, -70, -65, -60, -55, -50\} \text{ mV}$$

For each cell surface area (see above) plot the I_{ext} required to hold the neuron at each of the membrane potentials listed. Use a different color for each surface area, and include a legend in your plot. In your own words, describe the relation between cell surface area and I_{ext}.

4. Challenge problem. Assume an external current of $I_{ext} = 8$ nA. How much time would it take to reach the membrane potentials below?

$$V_m \in \{-70, -65, -60, -55, -50\} \text{ mV}$$

Repeat your calculations for several cell surface areas (see above). In your own words, describe the relation between cell surface area and time to reach target voltage.

Lecture 3 – Integrate-and-fire neuron model

Computational Neuroscience Summer Program

June, 2010

Motivation. This lecture builds on the simple model neuron that we developed in the last lacture by adding in action potentials (APs). Rather than modeling the biophysical basis of the AP, in this model we manually cause the neuron to spike when its membrane voltage reaches a threshold value.

Recap. Our working model is that neurons are like capacitors connected to resistors – the cell membrane stores charge, which can leak out through ion channels. As developed in the last lecture, the equation we'll be using for all model neurons is:

$$\tau_m \frac{dV}{dt} = E - V + R_m I_e$$

Also from the previous lecture, we can solve for $V(t)$ as follows:

$$V(t) = V_\infty + (V(0) - V_\infty)e^{\frac{-t}{\tau_m}}$$

Running a simulation – the "integrate" part of the model. Running a simulation entails computing changes in the cell's membrane voltage for each iteration of the simulation (dt ms). We can use the same equation as above, but replace $V(t)$ with $V(t+dt)$ (i.e. the voltage in the next time step of the simulation) and $V(0)$ (the starting voltage) with $V(t)$:

$$V(t+dt) = \text{(where we are going)} + \text{(distance)}e^{\frac{-t}{\tau_m}} = V_\infty + (V(t) - V_\infty)e^{\frac{-t}{\tau_m}}$$

In practice, these computations work best for small values of dt (in most cases we'll use $dt \leq 0.1$ ms).

Running a simulation – the "fire" part of the model. Now the integrate-and-fire model is almost entirely in place. The only thing we need to add is the rule that when $V(t + dt) \geq V_{thresh}$, simulate an action potential by setting $V(t) = V_{peak}$ and $V(t + dt) = V_{reset}$. V_{thresh} is generally somewhere around -55 mV, V_{peak} is around 40 mV, and V_{reset} is around -80 mV. In the next lectures we'll discuss the biophysical basis of why the neuron depolarizes (increases its membrane voltage) suddenly during the start of the action potential and why the membrane voltage becomes hyperpolarized (decreased) after the action potential is fired.

Computing firing rate. This is straightforward. We can simply count up the number of spikes that were fired during the simulation (i.e. times when $V \geq V_{thresh}$ and divide by the length of time we were simulating.

Analytic solution for firing rate. While the full integrate-and-fire simulation is often useful (and is necessary if you want to model things like spike timing), it turns out that there is an analytic method for computing the expected firing rate of the model, given a constant external current I_e. We start with the equation for finding the membrane voltage at time t:

$$V(t) = V_\infty + (V(0) - V_\infty)e^{\frac{-t}{\tau_m}}$$

We can then solve for t as follows:

$$V(t) - V_\infty = (V(0) - V_\infty)e^{\frac{-t}{\tau_m}}$$

$$\frac{V(t) - V_\infty}{(V(0) - V_\infty)} = e^{\frac{-t}{\tau_m}}$$

$$ln(\frac{V(t) - V_\infty}{V(0) - V_\infty}) = \frac{-t}{\tau_m}$$

$$\tau_m ln(\frac{V(t) - V_\infty}{V(0) - V_\infty}) = -t$$

$$t = -\tau_m ln(\frac{V(t) - V_\infty}{V(0) - V_\infty})$$

Now let's suppose our model neuron has just fired a spike in the previous timestep of our simulation. We start by setting $V(0) = V_{reset}$. We next need to know how long it is until the neuron next fires a spike (the inter-spike interval, t_{isi} – or, in other words, the time t at which $V(t) = V_{thresh}$ after starting at $V(0) = V_{reset}$. Plugging in the appropriate values, we can compute t_{isi} as follows:

$$t_{isi} = -\tau_m ln(\frac{V_{thresh} - V_\infty}{V_{reset} - V_\infty})$$

Recall that $V_\infty = E + R_m I_e$. Thus

$$t_{isi} = -\tau_m ln(\frac{V_{thresh} - (E + R_m I_e)}{V_{reset} - (E + R_m I_e)}) = -\tau_m ln(\frac{V_{thresh} - E - R_m I_e}{V_{reset} - E - R_m I_e})$$

The firing rate (r_{isi}) is the inverse of the inter-spike interval:

$$r_{isi} = (-\tau_m ln(\frac{V_{thresh} - E - R_m I_e}{V_{reset} - E - R_m I_e}))^{-1}$$

Putting it all together. To run the simulation, start with the basic model neuron equation:

$$\tau_m \frac{dV}{dt} = E - V + R_m I_e$$

Now solve for dV:

$$\frac{dV}{dt} = \frac{E - V + R_m I_e}{\tau_m}$$

$$dV = (\frac{E - V + R_m I_e}{\tau_m})dt$$

Start the simulation by setting $V(0) = E$. With each timestep set $V(t + dt) = V(t) + dV$. You'll need to re-compute dV for each time-step given $V(t)$ and $I_e(t)$ for the appropriate time t. Remember to include the rule for firing a spike (and resetting) when $V > V_{thresh}$ – otherwise the neuron won't fire spikes.

General MATLAB stuff. Set up your environment:

```
E = -70;         %mV
c_m = 10;        %nF / mm^2
r_m = 1;         %M ohm * mm^2
A = 0.025;       %mm^2
V_reset = -80;   %mV
V_thresh = -55;  %mV
V_peak = 40;     %mV

dt = 0.1;        %ms
t = 1:dt:1000;   %ms
```

Now loop:

```
V(1) = E;
for i = 2:length(t)
  if V(i-1) > V_thresh
    {fire a spike}
  else
    {compute dV}
    V(i) = V(i-1) + dV;
end
```

For the problem set, you should write a function that runs the integrate-and-fire model for a given set of parameters. Your function should at return the firing rate (computed numerically, not using the r_{isi} equation). You also might want to have it return the vector of V over time, depending on how you set up your code.

Problem Set 3 – Integrate-and-fire neuron model

Computational Neuroscience Summer Program

June, 2010

In this problem set you will be building a simple integrate-and-fire neuron. You should assume a specific membrane capacitance of $c_m = 10$ nF/mm^2, a specific membrane resistance of $r_m = 1$ MΩ·mm^2, a resting membrane potential of $E = -70$ mV, a reset potential of $V_{reset} = -80$ mV, an action potential threshold of $V_{threshold} = -55$ mV, and a cell surface area of $A = 0.025$ mm^2. Write up your results in a text editor of your choosing. Include any relevant figures, your Matlab code, and any other calculations related to the problem set. You may work individually or in groups, but each student should hand in their own report.

Equations

$$C_m = A \cdot c_m \qquad R_m = \frac{r_m}{A}$$
$$\tau_m = c_m \cdot r_m \qquad V_\infty = E + R_m I_{ext}$$
$$V(t) = V_\infty + (V(0) - V_\infty)e^{\frac{-t}{\tau_m}} \qquad r_{isi} = (\tau_m \ln(\frac{R_m I_{ext} + E - V_{reset}}{R_m I_{ext} + E - V_{threshold}}))^{-1}$$
$$\tau_m \frac{dV}{dt} = E - V(t-1) + R_m I_{ext}$$

Problems

1. Model an integrate-and-fire neuron using the equations above and the following rule: when the neuron's membrane voltage exceeds $V_{threshold}$, set the voltage in that timestep to $V_{peak} = 40$ mV, and in the next timestep set the voltage to V_{reset}. Set $dt = 0.1$ ms. Apply a square pulse of 0.5 nA from $t = 250$ ms until $t = 750$ ms in your simulation. Use Matlab's subplot command to plot the membrane voltage over time in the top panel and I_{ext} in the bottom panel (use the same time scale for the horizontal axis of both plots). No text is required for this question; just include a plot.

2. Compute the average firing rate (spikes per second) of the integrate-and-fire neuron for the pulse interval you used in quesion 1 (500 ms). Now plot simulated firing rate vs r_{isi} for several values of I_{ext} (use I_{ext} between 0 and 1 nA). How does the firing rate of the modeled neuron compare to the estimated firing rate given by r_{isi}? Note: the equation for r_{isi} only holds if $V_\infty > V_{thresh}$; otherwise, $r_{isi} = 0$.

3. Starting from 0 nA, gradually increase the amount of external current injected into the integrate-and-fire neuron in steps of 0.01 nA. Keep the pulse duration constant at 500 ms. What is the smallest amount of current you can inject which will still result in an action potential? Is there a maximum firing rate this neuron can achieve? Why or why not?

4. Challenge problem. Compute firing rate as a function of pulse duration, $I_{duration}$, using 20 durations between 10 and 500 ms. Repeat this for several different values of I_{ext} (try using 10 log-spaced values between 0.1 and 5 nA). Explain what you see. In particular, are the firing rate curves smooth or jagged? Why?

5. Vary the resting potential, specific capacitance, specific resistance, and surface area variables. How do increases or decreases in these values affect firing rate of the integrate-and-fire neuron? Explain (try to stay at or under 1-2 sentences per variable). Include plots for each of these variables.

Lecture 4 – Hodgkin-Huxley neuron model

Computational Neuroscience Summer Program

June, 2010

Motivation. The integrate-and-fire model allows us to model spiking rates, but ignores the biophysical underpinnings of the action potential. The Hodgkin-Huxley model, proposed by Alan Hodgkin and Andrew Huxley in 1952 explains how the action potential arises from ionic currents. They won the Nobel Prize for this model in 1963, and their model is still taught today in neuroscience classes around the world.

Recap. The model neuron equation we've been using is:

$$\tau_m \frac{dV}{dt} = E - V + R_m I_e$$

In this equation, all of the membrane biophysics are (implicitly) lumped into the $E - V$ term. The $R_m I_e$ term represents how external currents affect the cell. The basic idea of the Hodgkin-Huxley model is that we'll expand on the membrane biophysics portion of the equation by modeling ionic currents flowing through ion channels. We'll consider two ions: sodium and potassium. Before we fully explain the Hodgkin-Huxley model we'll review over some basic principles from physics and chemistry to gain a deeper understanding of the forces acting on ions inside and surrounding the cell.

Diffusion. Imagine adding a drop of food coloring to a beaker of water. The food coloring diffuses ("spreads out") throughout the water uniformly. This is because of principle 1: *molecules flow down their concentration gradient.*

Selective permeability. Let's add a barrier to our imaginary beaker. If we drop some blue food coloring in the left side, it will spread throughout that side but won't spread to the right side. Now suppose the barrier is permeable to red, but not to blue. If we drop some red food coloring in either side, it will spread throughout the entire beaker, even though blue coloring is confined to the side it was dropped into. Ion channels allow the cell's membrane to become selectively permeable to a single type of ion.

Charges. Now let's suppose blue represents negatively charged ions and red represents positive charged ions. Since opposite charges attract, we need to consider this force acting on the ions in addition to simple diffusion. In particular, rather than spreading out evenly throughout the beaker, some of the red ions will be attracted over to the blue side. This is because of principle 2: *molecules flow down their charge gradient.*

Energy required to transport ions across the membrane. Membrane potentials are small, so ions are transported across the membrane by thermal fluccuations. The thermal energy of an ion is $k_B T$, where k_B is Boltzman's constant and T is the temperature in degrees Kelvin. Biologists and chemists typically like to think about moles of ions rather than single ions. A mole of ions has Avagadro's number times as much energy as a single ion, or RT, where R is the universal gas constant (8.31 Joules/mole °K = 1.99 cal/mole °K. At normal temperatures, $RT \approx 2,500$ joules/mole, or 0.6 kCal/mole.

We can compute the energy gained or lost when a mole of ions crosses the membrane with a potential difference V_T across it. This energy is equal to FV_T, where F is Faraday's constant (F = 96,480 Columbs/mole), or Avagadro's number times the charge of a single proton, q. Setting $FV_T = RT$ gives:

$$V_T = \frac{RT}{F} = k_B T q$$

Equilibrium potential. The *Equilibrium potential* is the membrane voltage at which the net flow of a particular ion into or out of the cell is zero (i.e., the concentration gradient perfectly offsets the charge gradient). If an ion has an electrical charge zq, it must have a thermal energy of at least $-zqV$ to cross the membrane (this quantity is positive for $z > 0$ and $V < 0$). The probability that an ion has a thermal energy greater than or equal to $-zqV$ when the temperature (degrees Kelvin) is T is $e^{\frac{zqV}{k_B T}}$, which is determined by integrating the Boltzmann distribution for energies $\geq -zqV$. In molar units, this can be written as:

$$e^{\frac{zFV}{RT}}$$

Then the concentration gradient offsets the charge gradient when

$$[\text{outside}] = [\text{inside}] e^{\frac{zFE}{RT}}$$

Where E is the "equilibrium potential" – the membrane potential at which the gradients cancel. We can solve for E:

$$\frac{[\text{outside}]}{[\text{inside}]} = e^{\frac{zFE}{RT}}$$

$$ln(\frac{[\text{outside}]}{[\text{inside}]}) = \frac{zFE}{RT}$$

$$E = \frac{RT}{zF} ln(\frac{[\text{outside}]}{[\text{inside}]})$$

Plugging in the appropriate sodium and potassium concentrations, we find that $E_K \approx$ -70 to -90 mV and $E_{Na} \approx 50$ mV.

Reversal potential. When $V < E$, positive charges flow into the cell, causing the cell to depolarize (become more positive). When $V > E$, positive charges flow out of the cell, causing it to hyperpolarize (become less positive). Because E is the membrane potential at which the direction of net current flow reverses, E is also often called the *reversal potential*.

Resting potential. When no current is being pumped into the neuron, the equilibrium potentials of the different ions contained in the neuron and extracellular fluid all fight to bring the neuron's membrane voltage to their respective equilibrium potentials. The "strength" with which each ion pulls the membrane potential towards its equilibrium potential is proportional to the permeability ("conductance") of the the membrane to that ion, g_i – this depends on the number of open ion channels for that ion. The resting membrane potential is equal to:

$$V_{rest} = \frac{\sum_i (g_i E_i)}{\sum_i g_i}$$

This is called the Goldman-Hodgkin-Katz equation.

Intuition underlying the shape of the action potential. The key biophysical property of neurons that gives rise to the characteristic shape of the action potential is that the membrane's permeability to different ions depends on the membrane voltage. During the rising phase of the action potential, sodium channels open quickly and potassium channels open slowly. Since the sodium conductance outweighs the potassium conductance, we see from the Goldman-Hodgkin-Katz equation that the membrane voltage will head towards E_{Na}. At the peak of the action potential, sodium channels become blocked, and the membrane becomes almost exclusively permeable to potassium – so during the falling phase, the membrane plummets towards E_K. Then some resetting happens, and the membrane potential returns to V_{rest}. Now let's get into the details and the equations...

Voltage-gated potassium channels. The voltage-gated potassium channel is comprised of four identical (independent) subunits. For the channel to conduct potassium ions, all four subunits have to be in their open configuration. The probability that a given potassium channel is open, P_K is equal to the probability that a given subunit is open, n, raised to the fourth power:

$$P_K = n^4$$

Note that if n is the probability that a given subunit is open, then $1 - n$ is the probability that the subunit is closed.

The potassium channel subunits contain voltage sensors which make it more likely that the subunits will be open (activated) when the membrane is depolarized and less likely that the subunits will be open (deactivated) when the membrane is hyperpolarized. Thus, we need to use (and update) the membrane voltage at each time t in our calculations.

Let's define an opening rate, $\alpha_n(V)$ and a closing rate, $\beta_n(V)$ for the Potassium channel subunits. The probability that a subunit gate opens over a short interval of time is equal to the probability of finding the gate closed $(1 - n)$ multiplied by the opening rate, $\alpha_n(V)$. Likewise, the probability that a subunit gate closes over a short interval of time is equal to the probability of finding the gate open (n) multiplied by the closing rate, $\beta_n(V)$. The the rate at which the open probability for a subunit gate changes is given by the difference between these two terms:

$$\frac{dn}{dt} = \alpha_n(V)(1-n) - \beta_n(V)n$$

Another useful form of this equation is to divide through by the term $\alpha_n(V) + \beta_n(V)$:

$$\tau_n(V)\frac{dn}{dt} = n_\infty(V) - n, \text{ where}$$

$$\tau_n = \frac{1}{\alpha_n(V) + \beta_n(V)} \text{ and}$$

$$n_\infty(V) = \frac{\alpha_n(V)}{\alpha_n(V) + \beta_n(V)}$$

This equation indicates that for a given voltage, V, n approaches the limiting value $n_\infty(V)$ exponentially with time constant τ_n. Hodgkin-Huxley found that the following rate functions fit their data:

$$\alpha_n(V) = \frac{0.1(V+55)}{1 - e^{-0.1(V+55)}}$$

This function first increases gradually, and then increases approximately linearly.

$$\beta_n(V) = 0.125e^{-0.0125(V+65)}$$

This function decays exponentially towards zero.

Since the membrane contains a very large number of potassium channels, the fraction of channels open at any given time is equal to $P_K = n^4$. The membrane's ability to conduct potassum (g_K) is equal to P_K times the membrane's maximal potassium conductance, $\bar{g}_K$.

Voltage-gated sodium channels. Voltage-gated sodium channels consist of three main subunits, each of which are open with probability m. As with potassium channel subunits, sodium subunit opening and closing rates are given by α_m and β_m, respectively. These subunits open (activate) quickly when the membrane is depolarized and close (inactivate) when the membrane is hyperpolarized.

In addition to the three main subunits, the sodium channel contains a fourth subunit which has a negative charge, and is open with probability h. When the cell is depolarized, the subunit gets attracted to the inside of the cell and blocks ("inactivates") the channel. In order to "unblock" the channel, the cell needs to be sufficiently hyperpolarized (past its normal resting potential). The unblocking is called "de-inactivation." The equations for dm and dh are identical to the equations for dn, but with n replacing with m or h. The equations for α_m and β_m are:

$$\alpha_m(V) = \frac{0.1(V+40)}{1-e^{-.1(V+40)}}$$

$$\beta_m(V) = 4e^{-0.0556(V+65)}$$

(these are of the same form as the equations for α_n and β_n. The equations for α_h and β_h are:

$$\alpha_h(V) = 0.07e^{-.05(V+65)}$$

$$\beta_n(V) = \frac{1}{1+e^{-.1(V+35)}}$$

which are of the opposite form as for the n and m equations, since the h subunit inactivates with depolarization and de-inactivates with hyperpolarization.

Since the membrane contains a very large number of sodium channels, the fraction of channels open at any given time is equal to $P_{Na} = m^3h$. The membrane's ability to conduct sodium (g_{Na}) is equal to P_{Na} times the membrane's maximal Potassium conductance, $\bar{g}_{Na}$.

Leak channels. The last type of channels in the Hodgkin-Huxley model is the leak channel. Leak channels are always open, regardless of membrane voltage. The total leak conductance is represented by $\bar{g}_L$.

Derivation of full model. For the integrate-and-fire model we used:

$$\tau_m \frac{dV}{dt} = E - V + R_m I_e$$

We can re-write as:

$$c_m r_m \frac{dV}{dt} = (E - V) + \frac{r_m}{A} I_e$$

Now we divide both sides by r_m:

$$c_m \frac{dV}{dt} = \frac{1}{r_m}(E - V) + \frac{I_e}{A}$$

The full model is of the form:

$$c_m \frac{dV}{dt} = -i_m + \frac{I_e}{A},$$

where

$$i_m = \bar{g}_L(V - E_L) + g_K(V - E_K) + g_{Na}(V - E_{Na})$$

This is simply the written-out form of the Goldman-Hodgkin-Katz equation. As in the integrate-and-fire model, we start by solving for dV and setting $V(t + dt) = V(t) + dV$ in each time step of the simulation.

Problem Set 4 – Hodgkin-Huxley neuron model

Computational Neuroscience Summer Program

June, 2010

In this problem set you will be building a Hodgkin-Huxley model neuron. Write up your results in a text editor of your choosing. Include any relevant figures. Include a printout of your Matlab code as well as any calculations that aren't in the code. You may work individually or in groups, but each student should hand in their own report.

Equations

$\tau_x(V)\frac{dx}{dt} = x_\infty(V) - x$	$\tau_x(V) = \frac{1}{\alpha_x(V)+\beta_x(V)}$	$x_\infty(V) = \frac{\alpha_x(V)}{\alpha_x(V)+\beta_x(V)}$
$\alpha_n(V) = \frac{.01(V+55)}{1-exp(-.1(V+55))}$	$\alpha_m(V) = \frac{0.1(V+40)}{1-exp(-.1(V+40))}$	$\alpha_h(V) = 0.07 * exp(-.05(V+65))$
$\beta_n(V) = 0.125 * exp(-0.0125(V+65))$	$\beta_m(V) = 4 * exp(-.0556(V+65))$	$\beta_h(V) = \frac{1}{1+exp(-.1(V+35))}$
$P_K = n^4$	$P_{Na} = m^3 h$	
$g_K = \bar{g}_K P_K$	$g_{Na} = \bar{g}_{Na} P_{Na}$	

Problems

1. Build a Hodgkin-Huxley model using the following equation, in addition to those listed above:

$$V(t+dt) = V(t) + \frac{\left[-i_m + \frac{I_{ext}}{A}\right]dt}{c_m},$$

where

$$i_m = \bar{g}_L(V(t) - E_L) + g_K(V(t) - E_K) + g_{Na}(V(t) - E_{Na}),$$

and where $\bar{g}_L = 0.003$ mS/mm^2, $\bar{g}_K = 0.36$ mS/mm^2, $\bar{g}_{Na} = 1.2$ mS/mm^2, $E_L = -54.387$ mV, $E_K = -77$ mV, $E_{Na} = 50$ mV, and $c_m = 0.1$ nF/mm^2. Set $V(0) = E = -65$ mV. You should simulate a 15 ms segment (use $dt \leq 0.01$ ms). Pulse the neuron with a 5 nA/mm^2 pulse between 5 and 8 ms in your simulation; this should result in a single action potential at between 5 and 10 ms. Plot the V, n, m, and h as a function of time. Congratulations — you've just replicated a Nobel prize-worthy result!

2. Explain what's happening in problem 1. In particular: what do V, n, m, and h mean in terms of the simulated neuron? Explain the relative time course of each of these variables. How do n, m, and h interact to produce V?

3. Tetrodotoxin (TTX), a pufferfish-derived toxin, selectively blocks voltage-sensitive sodium channels, effectively setting $P_{Na} = 0$. Simulate the addition of TTX at $t = 0$ ms, and re-plot V, n, m, and h. Does the neuron still fire an action potential? Why or why not?

4. Tetraethylammonium (TEA) selectively blocks voltage-sensitive potassum channels, effectively setting $P_K = 0$. Simulate the addition of TEA at $t = 0$ ms, and re-plot V, n, m, and h. Does the neuron still fire an action potential? Why or why not? (Remember to remove TTX first!)

5. Challenge problem. How might you compute the firing rate of your Hodgkin-Huxley neuron? Simulate a 1000 ms run with a pulse from 250 to 750 ms, and measure how many spikes are fired. Repeat this procedure for 10 pulse strengths between 0.01 and 10 nA. Plot firing rate as a function of pulse strength.

Lecture 5 – Extensions of the Hodgkin-Huxley model

Computational Neuroscience Summer Program

June, 2010

Motivation. The Hodgkin-Huxley model gave us a way to explicitly model ionic currents using the expected conductances of each ion and the driving forces on those ions. We will now go one step further by explicitly simulating the stochastic actions of individual subunits of the potassium and sodium ion channels.

Potassium channel review. The potassium channel is comprised of four identical subunits. In the Hodgkin-Huxley model, the probability that a given channel is open is equal to $p_K = n^4$, where n is the probability of a single subunit being open. Recall that n increases when the cell is depolarized and decreases when the cell is hyperpolarized. The opening rate of each subunit is α_n and the closing rate is β_n:

$$\alpha_n(V) = \frac{0.01(V+55)}{1-e^{-0.1(V+55)}}$$

$$\beta_n(V) = 0.125e^{-0.0125(V+65)}$$

Stochastic model. In the stochastic model, we represent each potassium channel using a state diagram:

$$\begin{tikzcd}
\begin{matrix}1\\ \text{closed}\end{matrix} \arrow[r, shift left, harpoon, "4\alpha_n"] & \begin{matrix}2\\ \text{closed}\end{matrix} \arrow[l, shift left, harpoon, "\beta_n"] \arrow[r, shift left, harpoon, "3\alpha_n"] & \begin{matrix}3\\ \text{closed}\end{matrix} \arrow[l, shift left, harpoon, "2\beta_n"] \arrow[r, shift left, harpoon, "2\alpha_n"] & \begin{matrix}4\\ \text{closed}\end{matrix} \arrow[l, shift left, harpoon, "3\beta_n"] \arrow[r, shift left, harpoon, "\alpha_n"] & \begin{matrix}5\\ \text{open}\end{matrix} \arrow[l, shift left, harpoon, "4\beta_n"]
\end{tikzcd}$$

where the labels represent the probability of transitioning between each state in the indicated direction. We simulate a state transition from state X to Y during a time interval dt if a random number, chosen with each new timestep of the simulation, is less than the probability of transitioning between X and Y. The channel is considered to be open at time t if the ion channel is in the 5^{th} state at time t. In simulations, we'll need to keep track of how many of the ion channels are open vs. closed during each time step. As the number of channels (N) increases, this model becomes arbitrarily similar to the Hodgkin-Huxley version (you'll be verifying this in the problem set). Note that you can compute P_K for the stochastic model at time t by simply computing the fraction of channels that are in state 5 at time t.

Sodium channel review. The sodium channel is comprised of three identical subunits and an inactivation gate. The three identical subunits each open with probability m (where m increases as the cell is depolarized and decreases as the cell is hyperpolarized). The opening rates for the three subunits are α_m, and the closing rates are β_m. The inactivation gate is open with probability h, where h decreases as the cell is depolarized and increases when the cell is hyperpolarized. The opening and closing rates are α_h and β_h, respectively.

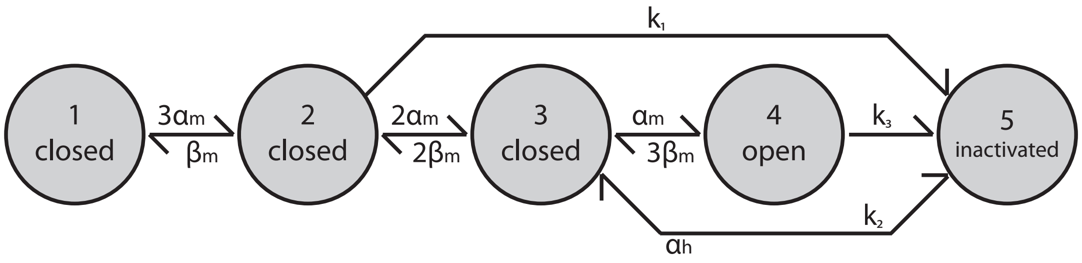
$$\alpha_m(V) = \frac{0.1(V+40)}{1-e^{-.1(V+40)}}$$

$$\beta_m(V) = 4e^{-0.0556(V+65)}$$

$$\alpha_h(V) = 0.07e^{-.05(V+65)}$$

$$\beta_n(V) = \frac{1}{1 + e^{-.1(V+35)}}$$

Stochastic sodium channel model. In the Hodgkin-Huxley model, the three subunits and the inactivation gate are assumed to be independent (that's why the probabilities are multiplied into m^3h). However, this is not quite true. A more accurate description is something like the following:



In particular, the ball mechanism of the inactivation gate is located inside the cell membrane, and cannot be directly affected by potential across the membrane. The inactivation gate only comes into play when at least one of the subunits is open (i.e., when the channel occupies states 2, 3, or 4 in the diagram). In addition, according to this model, if the neuron is in the inactivate state (state 5), it can only transition to state 3.

Whereas the transitions of the three subunits between states 1, 2, 3, and 4 in the stochastic model are identical to in the Hodgkin-Huxley model, the behavior of the inactivation gate is much different – in particular, the inactivation gate in the stochastic model depends on the states of the three subunits.

As in the stochastic potassium channel model, you can compute the P_{Na} for the stochastic sodium channel at time t by computing the fraction of sodium channels which occupy state 4 at that time.

Replacing the Hodgkin-Huxley channels with stochastic channels. In the Hodgkin-Huxley model, we computed $P_K = n^4$ and $P_{Na} = m^3h$ with each time step, updating $n, m,$ and h as we stepped through the model. In the stochastic model, we compute P_K and P_{Na} directly, so we no longer need to compute n, m, or h. Other than the difference in computing P_K and P_{Na}, the stochastic model is identical to the Hodgkin-Huxley model.

Some implementation suggestions. There are a number of possible ways to implement the stochastic channel models, with some methods being more efficient than others. Particularly when the number of channels is large, it becomes very important to code the simulation efficiently (read: vectorize!) if the simulation is to finish running in a reasonable amount of time. To start you off, we'll go through a simple two-state example. The states are x (closed) and y (open). Let's suppose that the probability of transitioning from state x to state y is p_{xy} and the probability of transisitioning from y to x is p_{yx}. To simulate $N = 1000$ of these simple two-state channels, we could write something like the following:

```
dt = 0.1;
t = 0:dt:1000;
states = ones(1,N);
n_open = zeros(size(t));

pOpen = [p_xy 0];
```

```
pClose = [0 p_yx];
for i = 1:length(t)
  open_chooser = rand(size(states)) < (dt*pOpen(states));
  states(open_chooser) = states(open_chooser) + 1;
  states(states > 2) = 2;

  close_chooser = rand(size(states)) < (dt*pClose(states));
  states(close_chooser) = states(close_chooser) - 1;
  states(states < 1) = 1;

  n_open(i) = sum(states == 2);
end
```

The potasium channel simulation is identical to the above code, but with the `pOpen` and `pClose` variables modified as in the state diagram. To implement the stochastic sodium channel model, you need to seperately compute the probabilities of opening subunits and opening the inactivation gate.

Problem Set 5 – Extensions of the Hodgkin-Huxley model

Computational Neuroscience Summer Program

June, 2010

In this problem set you will be extending the standard Hodgkin-Huxley model you constructed in Problem Set 4. In this extended model we will be simulating the actions of individual voltage-dependent Sodium and Potassium channels. Feel free to re-use any code from Problem Set 4 that you think would be useful. Write up your results in a text editor of your choosing. Include any relevant figures. Include a printout of your Matlab code as well as any calculations that aren't in the code. You may work individually or in groups, but each student should hand in their own report.

Equations

$\tau_x(V)\frac{dx}{dt} = x_\infty(V) - x$	$\tau_x(V) = \frac{1}{\alpha_x(V)+\beta_x(V)}$	$x_\infty(V) = \frac{\alpha_x(V)}{\alpha_x(V)+\beta_x(V)}$
$\alpha_n(V) = \frac{.01(V+55)}{1-exp(-.1(V+55))}$	$\alpha_m(V) = \frac{0.1(V+40)}{1-exp(-.1(V+40))}$	$\alpha_h(V) = 0.07 * exp(-.05(V+65))$
$\beta_n(V) = 0.125 * exp(-.0125(V+65))$	$\beta_m(V) = 4 * exp(-.0556(V+65))$	$\beta_h(V) = \frac{1}{1+exp(-.1(V+35))}$
$P_K = n^4$	$P_{Na} = m^3 h$	
$g_K = \bar{g}_K P_K$	$g_{Na} = \bar{g}_{Na} P_{Na}$	

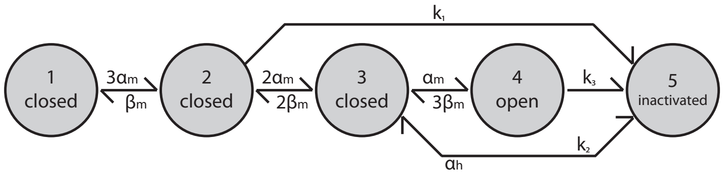
Problems

1. Construct and simulate the stochastic K$^+$ channel model as shown in the state diagram below. The diagram shows transitions between different states of a K$^+$ channel. The symbols α_n and β_n represent the opening and closing rates, respectively, of individual K$^+$ channels (these are the same variables you used in Problem Set 4). Set the rate constants equal to their value at 10 mV (i.e., use $\alpha_n(10\text{mV}) = 0.65/\text{ms}$ and $\beta_n(10\text{mV}) = 0.05/\text{ms}$). Transitions are made between states with each iteration of your program if a random number chosen uniformly between 0 and 1 is less than the corresponding rate for that transition time. For example, if a channel is in state 1, that channel will transition to state 2 during the current iteration of your program if the chosen random number is less than $4\alpha_n dt$. As the channel(s) make transitions between states, keep track of whether state 5 is occupied. If so, assume that each channel conducts 1 pA (10^{-12} A) of current; otherwise no current flows through the channel. Plot currents generated by this model for N =1, 10, and 100 channels over a 20 ms period (use dt = 0.01 ms).

$$\begin{tikzcd}
\text{1 closed} \arrow[r, shift left, "4\alpha_n"] & \text{2 closed} \arrow[l, shift left, "\beta_n"] \arrow[r, shift left, "3\alpha_n"] & \text{3 closed} \arrow[l, shift left, "2\beta_n"] \arrow[r, shift left, "2\alpha_n"] & \text{4 closed} \arrow[l, shift left, "3\beta_n"] \arrow[r, shift left, "\alpha_n"] & \text{5 open} \arrow[l, shift left, "4\beta_n"]
\end{tikzcd}$$

2. The Hodgkin-Huxley description of the K$^+$ channel predicts the current flowing in these simulations would be Nn^4 pA, where N is the number of K$^+$ channels and n is the K$^+$ activation variable (same as in Problem Set 4). On a single plot, show the amount of current predicted by the Hodgkin-Huxley prediction and the amount of current predicted with the stochastic model. (Use N = 1000.)

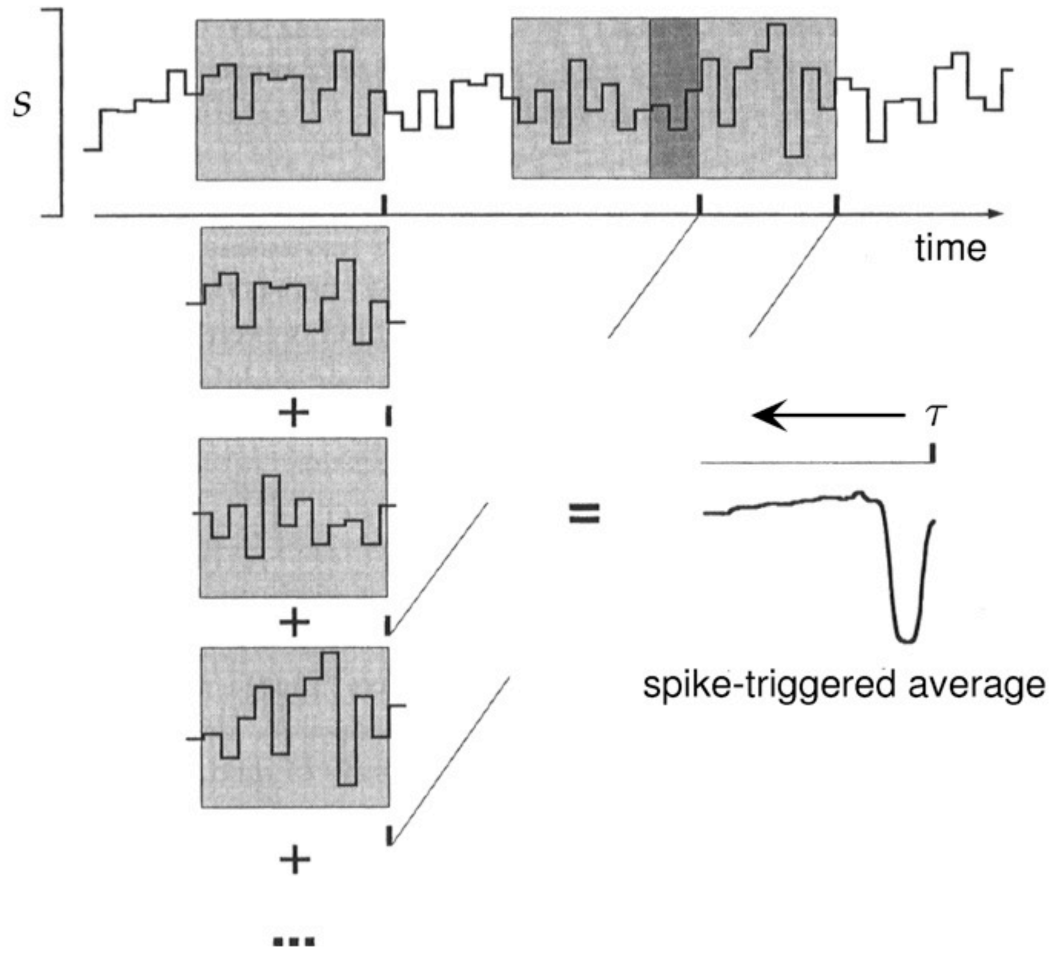
3. Construct an stochastic Na$^+$ channel model analogous to the K$^+$ model, using the state diagram below. The symbols α_m and β_m represent opening and closing rates of the three main subunits of the sodium channel. Activation and deinactivation rates of the sodium channel gate are represented by k_1, k_2, and k_3. Set the rate constants equal to their value at 10 mV (i.e., use α_m(10 mV) =5.034/ms, α_h(10 mV) =0.0016/ms, and β_m(10 mV) =0.0618/ms). Use k_1 = 0.24/ms, k_2 = 0.4/ms, and k_3 = 1.5/ms. As the channel(s) make transitions between states, keep track of whether state 4 is occupied. If so, assume that each channel conducts -1 pA (10^-12 A) of current; otherwise no current flows through the channel. Plot currents generated by this model for M =1, 10, and 100 channels over a 20 ms period.

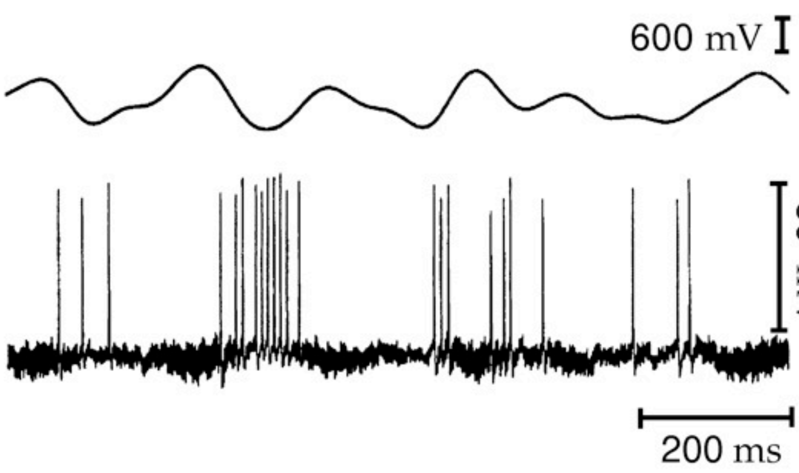


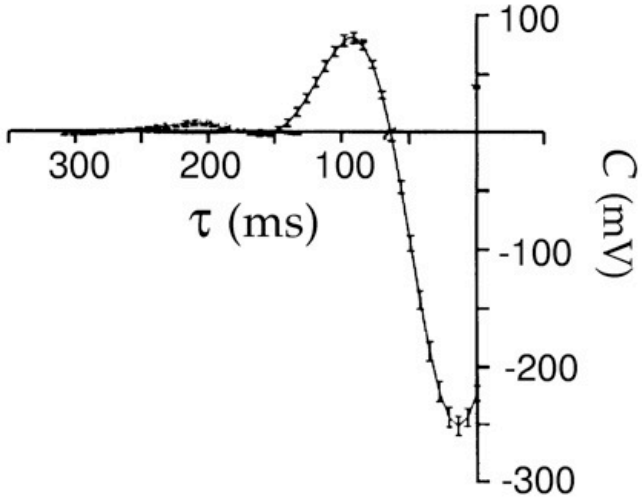
4. The Hodgkin-Huxley description of the Na$^+$ channel predicts the current flowing in these simulations would be Mm^3h pA, where M is the number of Sodium channels. On a single plot, show the amount of current predicted by the Hodgkin-Huxley prediction and the amount of current predicted with the stochastic model. (Use M = 1000.)

5. Challenge problem. Modify the Hodgkin-Huxley model from Problem Set 5 to explicitly simulate Sodium and Potassium channels using the stochastic models above (to start, use $N = M = 1000$). You will need to update the transition probabilities $\alpha_n, \alpha_m, \alpha_h, \beta_n$, and β_m with each time step, since those transition probabilities are voltage-dependent. k_1, k_2, and k_3 are constants. Hint: P_X is equal to the total proportion of X channels currently open. Simulate a 20 ms interval. Apply an external current of I_{ext} = 5 nA/mm^2 from 5 to 8 ms. Plot V, P_K, and P_{Na} as a function of time.

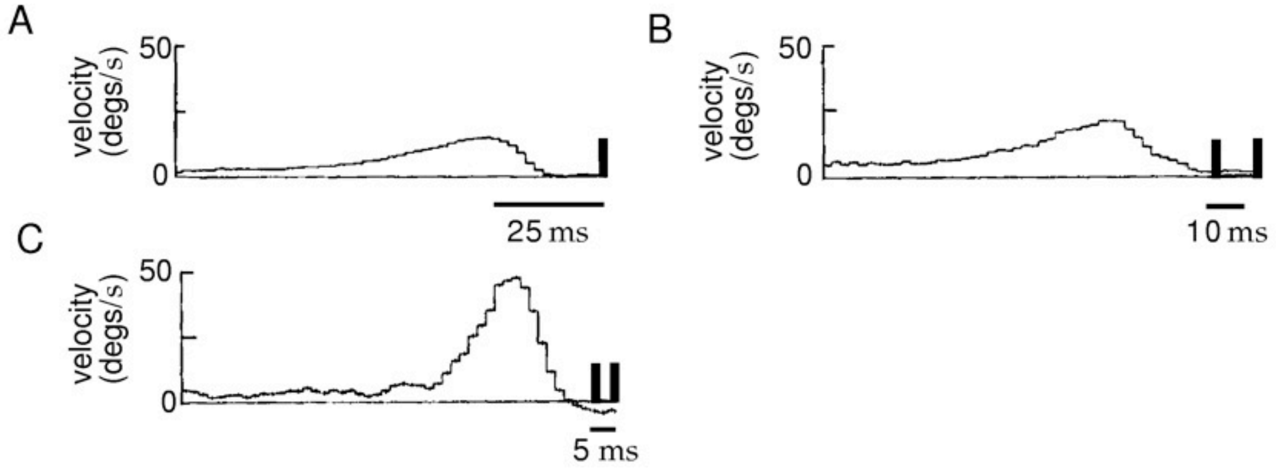
Computing a spike-triggered average




600 mV
50 mV
200 ms


100
-100
-200
-300
C (mV)
300
200
100
τ (ms)

Two-spike-triggered average



- You might expect that the two-spike average would be a linear summation of the single-spike averages.
- However, at some neurons multiple spikes are caused by especially large input signals.

Problem Set 6 -- Introduction to data analysis

Computational Neuroscience Summer Program

June, 2010

These questions use two Matlab files:

`http://www.neurotheory.columbia.edu/~larry/book/exercises/c1/data/c1p8.mat`

`http://www.neurotheory.columbia.edu/~larry/book/exercises/c1/data/c2p3.mat`

These questions were adapted from Dayan & Abbott, who in turn took inspiration from Sebastian Seung.

1. Reverse correlation

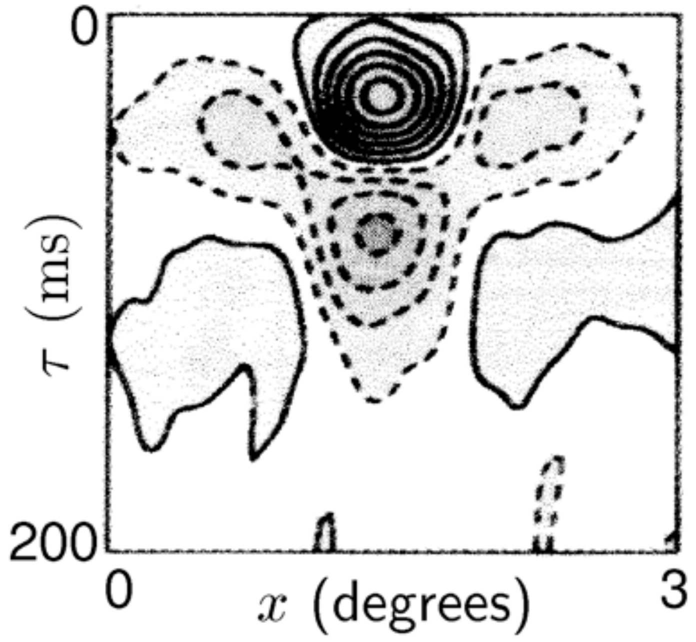
Load `c1p8.mat`. These data were collected for 20 minutes at a sampling rate of 500 Hz. In the file, rho is a vector that gives the sequence of spiking events or nonevents at the sampled times (every 2 ms). When an element of rho is one, this indicates the presence of a spike at the corresponding time, whereas a zero value indicates no spike. The variable stim gives the sequence of stimulus values at the sampled times. Calculate and plot the spike-triggered average from these data over the range from 0 to 300 ms (i.e, 150 time steps). How do you explain the shape of this curve for values of $t > 0$?

2. Challenge problem: Two-spike reverse correlation

Again using `c1p8.mat`, calculate and plot stimulus averages triggered on events consisting of a pair of spikes (which need not necessarily be adjacent) separated by a given interval. Plot these two-spike-triggered average stimuli for various separation intervals ranging from 2 to 1 0 0 ms. (Hint: you can use convolution for pattern matching: e.g., `find(conv(rho,[1 0 1])==2)` will contain the indices of all the events with two spikes separated by 4 ms.) Plot, as a function of the separation between the two spikes, the sum of the magnitudes of the differences between the two-spike-triggered average and the sum of two single-spike-triggered averages (obtained in exercise 1) separated by the same time interval. At what temporal separation does this difference become negligibly small?

3. Two-dimensional reverse correlation.

Load `c2p3.mat`. This file contains the responses of a cat LGN cell to two-dimensional visual images (these data are described in Kara et al., 2000). In the file, `counts` is a vector containing the number of spikes in each 15.6-ms bin, and `stim` contains the 32767, 16 x 16 images that were presented at the corresponding times. Specifically, `stim(x,y,t)` is the stimulus presented at the coordinate `(x,y)` at time-step `t`. Note that stim is an `int8` array that must be converted into a `double` using the matlab command `stim=double(stim)` in order to be manipulated within Matlab. Calculate the spike-triggered average images for each of the 12 time steps before each spike and show them all (using `imagesc` and `subplot`). Note that in this example, the time bins can contain more than one spike, so the spike-triggered average must be computed by weighting each stimulus by the number of spikes in the corresponding time bin, rather than weighting it by either 1 or 0 depending on whether a spike is present or not. (Tip: Make the plots look even cleaner by using the same color scale limits for all plots (search '`clim`').) In the averaged images, you should see a central receptive field that reverses sign over time. Also, by summing up the images across one spatial dimension, produce a figure like the one below that plots the response as a function of time (τ) and one spatial dimension (x).


τ (ms)
x (degrees)