The Hodgkin-Huxley Model

The Hodgkin-Huxley model is the most famous biophysically based neuron model. It was originally used to describe the dynamics of the action potential in the squid giant axon but the description of the conductances in terms of activation and inactivation variables is now used ubiquitously to model neurons in a wide range of species and brain areas. It is a "conductance-based neuron," meaning that it models the voltage- and time-dependent conductances of real ion channels (contrast this with the integrate-and-fire neuron model which does not model real ionic conductances).

I. Mathematical description of the Hodgkin-Huxley neuron

Today, we will build a Hodgkin-Huxley model neuron by numerically integrating the model equations for V, m, h, and n given below:

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

where A is the area of the cell and the membrane currents i_m are given by

$$i_m = \bar{g}_L(V - E_L) + \bar{g}_K n^4 (V - E_K) + \bar{g}_{Na} m^3 h (V - E_{Na})$$

The activation variables m and n, as well as the inactivation variable h, are all described by equations of the form (note that rather than referring explicitly to which of these are "activation" versus "inactivation" variables, we sometimes just refer to them all less specifically as "gating" variables because they control the gateway to letting ions flow through the channels):

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

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

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

(and similarly for m and h), with

$$\alpha_n(V) = \frac{.01(V+55)}{1-\exp(-.1(V+55))}, \quad \beta_n(V) = 0.125\exp(-0.0125(V+65)),$$

$$\alpha_m(V) = \frac{.1(V+40)}{1-\exp(-.1(V+40))}, \quad \beta_m(V) = 4\exp(-0.0556(V+65)),$$

$$\alpha_h(V) = .07\exp(-.05(V+65)), \quad \beta_h(V) = \frac{1}{1+\exp(-.1(V+35))}$$

In the above equations, time is in ms and V in mV.

In the model of the giant squid axon, the corresponding parameters are c_m=10 nF/mm^2, $\bar{g}_L$=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, and E_{Na}=50 mV.

II. Today's modeling goal

The goal of this lab is to generate action potentials in the Hodgkin-Huxley neuron in response to a constant current injection I_e, and plot the cell's voltage V and the values of the activation and inactivation variables m, h, and n during the action potential. We will then compute the firing rate of the model neuron.

To do this, we will build a Hodgkin-Huxley model as described in section I above. For initial values we will take V = -65 mV, m = 0.0529, h = 0.5961, and n = 0.3177 (these equal the steady-state values at this voltage: $m=m_\infty(-65)$, $h= h_\infty(-65)$, $n=n_\infty(-65)$). We will use an external current of I_e/A = 200 nA/mm^2 and we will integrate the equations using a time step dt=0.1 ms.

III. Step 1: Response to a constant current injection

As usual, let's open a new m-file, name it something memorable like "HHStep1.m", put a comment at the top describing what is being done in this assignment, and clear all variables that might be leftover from previous runs or lines typed into the command window:

```
% Lab 3: Build a Hodgkin-Huxley model neuron; stimulate to produce AP's; plot V and
% activation and inactivation variables m, h, and n; and calculate firing rate

clear all;  %clear all variables
close all; %close any open matlab windows
```

Next let's define the model parameters. We immediately have to be *very careful(!!!)* to make sure that the units that we use are all consistent, just as in your Flicker Fusion homework question you had to convert the frequency f of the cosine from Hz (=1/sec) to (1/ms). As given, the numbers in fact *do not produce consistent units*: I_e/A is given in nA/mm^2; c_mdV/dt is given in [nF/mm^2]*[mV/ms] = nA/mm^2 (so this is consistent so far…), *but* $\bar{g}$ V is given in units of [mS/mm^2][mV] = μA/mm^2. To get this term in nA/mm^2, we should instead use $\bar{g}$ values given in μS/mm^2 (exercise: check that this is correct). To do this, we need to multiply the given values by 1000 because there are 1000 μS per mS. We can do this in scientific notation by appending "e3" (which means 'times 10^3' in the computer version of scientific notation) to the end of the values given per mS. Doing this, and listing the other parameters for our simulation, gives:

```
%DEFINE PARAMETERS
dt = 0.1; %time step [ms]
t_end = 70; %total time of run [ms]
t_StimStart = 10;  %time to start injecting current [ms]
t_StimEnd = 60; %time to end injecting current [ms]
c = 10;  %capacitance per unit area [nF/mm^2]
gmax_L = 0.003e3; %leak maximal conductance per unit area [uS/mm^2]
E_L = -54.387; %leak conductance reversal potential [mV]
gmax_K = 0.36e3; %hodgkin-huxley maximal K conductance per unit area [uS/mm^2]
E_K = -77; %hodgkin-huxley K conductance reversal potential [mV]
gmax_Na = 1.2e3; %hodgkin-huxley maximal Na conductance per unit area [uS/mm^2]
E_Na = 50; %hodgkin-huxley Na conductance reversal potential [mV]
```

Sometimes it is useful to work backwards from the end of a program when writing computer code. Let's next write the plotting code that will appear at the end of the program. This will help us to figure out what variables need to be set up in vectors that will contain an element for each time point.

We want to plot vectors containing the values of V, m, h, and n as a function of time so let's set up 4 subplots with appropriate labels:

```
figure(1)
subplot(4,1,1)
plot(t_vect,V_vect)
title('Hodgkin Huxley variables vs. time');
ylabel('Voltage in mV');
subplot(4,1,2)
plot(t_vect,m_vect)
ylabel('g_{Na} activation variable m');
subplot(4,1,3)
plot(t_vect,h_vect)
ylabel('g_{Na} inactivation variable h');
subplot(4,1,4)
plot(t_vect,n_vect)
xlabel('Time in ms');
ylabel('g_{K} activation variable n');
```

Now that we know what vectors we want to plot (i.e. t_vect, V_vect, m_vect, h_vect, n_vect), let's go back to the line after the parameters were defined and set up vectors of the correct size filled with zeros as placeholders (remember: this makes the code run faster, see Lab 2 for details):

```
%SET UP VECTORS TO BE PLOTTED
t_vect = 0:dt:t_end; %will hold vector of times
V_vect = zeros(1,length(t_vect)); %initialize the voltage vector
m_vect = zeros(1,length(t_vect)); %initialize the HH Na activation variable vector
h_vect = zeros(1,length(t_vect)); %initialize the HH Na inactivation variable vector
n_vect = zeros(1,length(t_vect)); %initialize the the HH K activation variable vector
```

Then, let's define the initial (first time point corresponding to t=0) values of these vectors:

```
%ASSIGN INITIAL VALUES OF VARIABLES
i = 1;   %index denoting which element of V is being assigned
V_vect(i)= -65;  %first element of V, i.e. value of V at t=0 [mV]
m_vect(i) = 0.0529;  %initially set m = m_inf(-65)
h_vect(i) = 0.5961;  %initially set h = h_inf(-65)
n_vect(i) = 0.3177;  %initially set n = n_inf(-65)
```

We initialize the voltage to -65 mV and then initialize the m, h, and n vectors to the steady-state values $m_\infty(-65)$, $h_\infty(-65)$, and $n_\infty(-65)$ that correspond to this voltage. You could have calculated these values in your code (or by hand on a calculator, or in MATLAB's command window). They are provided above to make our code a little shorter and simpler looking.

Now let's define the stimulus, the code of which we will borrow (with a few slight changes in variable names) from our last lab on the integrate-and-fire neuron, i.e. do a pulse of current from t_StimStart to t_StimEnd of magnitude I_0:

```
%DEFINE THE STIMULUS
%vector below will hold values of I_e/A over time;
I_0 = 200;  %magnitude of pulse of injected current [nA/mm^2]
I_e_vect = zeros(1,t_StimStart/dt);  %portion of I_e/A vector from t=0 to t=t_StimStart
I_e_vect = [I_e_vect I_0*ones(1,1+((t_StimEnd-t_StimStart)/dt))]; %add portion from
                                                        % t=t_StimStart to t=t_StimEnd
I_e_vect = [I_e_vect zeros(1,(t_end-t_StimEnd)/dt)]; %add portion from
                                                    % t=t_StimEnd to t=t_
```

Good! Now we're ready to integrate the equations. First you should convince yourself that the voltage equation is in the form of the "1 equation of neuroscience":

$$\tau_V \frac{dV}{dt} = -V + V_\infty$$

where

$$\tau_V = \frac{c}{\bar{g}_L + \bar{g}_K n^4 + \bar{g}_{Na} m^3 h}$$

$$V_\infty = \frac{\bar{g}_L E_L + \bar{g}_K n^4 E_K + \bar{g}_{Na} m^3 h E_{Na} + (I_e / A)}{\bar{g}_L + \bar{g}_K n^4 + \bar{g}_{Na} m^3 h}$$

(as a useful exercise, try deriving this now).

Now notice that the time constants for V,m,n, and h, and the steady-state variables V_∞, m_∞, h_∞, and n_∞ are all interdependent and will each change values at every time step. This means that we had better include them within the for loop over time. Let's layout the for loop systematically. First let's set up the for loop:

```
%INTEGRATE THE HH EQUATIONS
for t=dt:dt:t_end   %loop through values of t in steps of dt ms

end
```

Within this loop (please indent!), let's first assign all of the activation and inactivation variables, which first requires defining the α's and β's:

```
  %assign all of the alphas & betas
  alpha_m = .1*(V_vect(i)+40)/(1 - exp(-.1*(V_vect(i)+40)));
  beta_m = 4*exp(-.0556*(V_vect(i)+65));
  alpha_h = .07*exp(-.05*(V_vect(i)+65));
  beta_h = 1/(1 + exp(-.1*(V_vect(i)+35)));
  alpha_n = .01*(V_vect(i)+55)/(1 - exp(-.1*(V_vect(i)+55)));
  beta_n = 0.125*exp(-0.0125*(V_vect(i)+65));
  %from the alphas & betas above, assign the taus & x_inf's for m,h,n
  tau_m = 1/(alpha_m + beta_m);
  m_inf = alpha_m/(alpha_m + beta_m);
  tau_h = 1/(alpha_h + beta_h);
  h_inf = alpha_h/(alpha_h + beta_h);
```

```
tau_n = 1/(alpha_n + beta_n);
n_inf = alpha_n/(alpha_n + beta_n);
```

Notice that these are defined in terms of V_vect(i) which the first time through the loop will be V_vect(1).

While we are at it, let's define the time constant τ_V and steady-state value V_∞ for the voltage as well. Note that these depend on the values of the gating variables m_vect(i), h_vect(i), and n_vect(i) through the equation on the previous page. Also, looking at that equation, note that both τ_V and V_∞ are long expressions that each have the same denominator. Therefore, to make our code more readable (and slightly more efficient), we first define a variable to hold the denominator:

```
%assign tau_V and V_inf
V_denominator = gmax_L + gmax_K*(n_vect(i)^4) + gmax_Na*(m_vect(i)^3)*h_vect(i);
tau_V = c/V_denominator;
V_inf =(gmax_L*E_L + gmax_K*(n_vect(i)^4)*E_K + ...  %... let's you continue on next line
   gmax_Na*(m_vect(i)^3)*h_vect(i)*E_Na + I_e_vect(i))/V_denominator;
```

Notice that we have used a new MATLAB operation here: writing "…" lets you continue writing a single command on more than one line. This can be useful when you are writing very long lines if you don't like having to scroll to your right to read them.

Now we are ready to use our standard update rule to assign the next values of the gating variables m, h, and n and the voltage variable V:

```
%assign next elements of m,h,and n vectors using update rule
m_vect(i+1) = m_inf + (m_vect(i) - m_inf)*exp(-dt/tau_m);
h_vect(i+1) = h_inf + (h_vect(i) - h_inf)*exp(-dt/tau_h);
n_vect(i+1) = n_inf + (n_vect(i) - n_inf)*exp(-dt/tau_n);
%assign next element of V vector using update rule
V_vect(i+1) = V_inf + (V_vect(i) - V_inf)*exp(-dt/tau_V);
```

Finally, let's update i, so that the next time through the loop our code will be ready to assign the next elements of the m, h, n, and V vectors:

```
%add 1 to index, corresponding to moving forward 1 time step
i = i+1;
```

Try running this code (see below for code summary if you get errors). You should see 5 beautiful action potentials in the top row and you should see underneath it, respectively, the m, n, and h variables. Notice which one(s) change quickly and which only more slowly. Does this fit with what you know about the openings, closings, and inactivation of Na+ and K+ channels that underlie the action potential?

Next, to better be able to compare these variables, let's add a second figure with them all plotted on the same graph but in different colors. To do this, type:

```
figure(2)
```

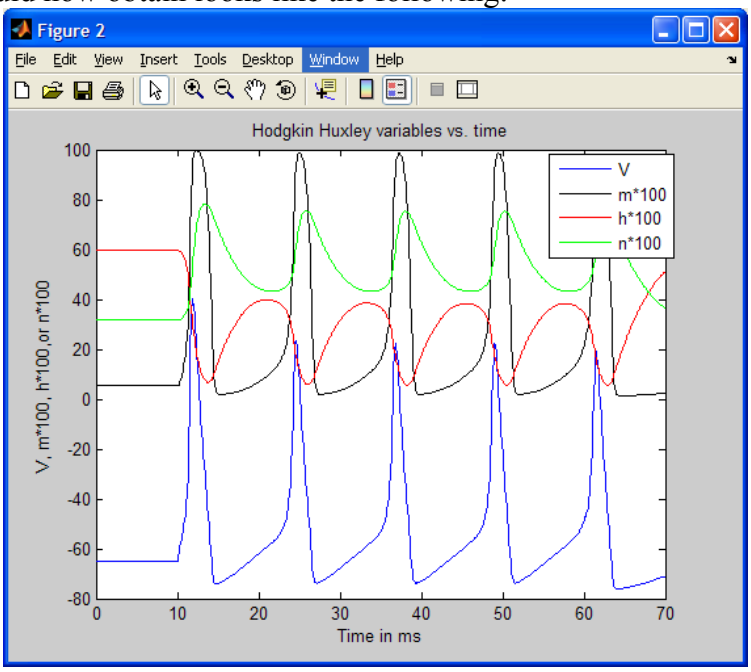
```
plot(t_vect,V_vect) %plot in blue
title('Hodgkin Huxley variables vs. time');
ylabel('V, m, h,or n');
xlabel('Time in ms');
hold on
plot(t_vect,m_vect,'k') %plot in black
plot(t_vect,h_vect,'r') %plot in red
plot(t_vect,n_vect,'g') %plot in green
legend('V', 'm', 'h','n')
```

You should see a beautiful trace of the voltage accompanied by a mess near a value of zero. What went wrong? (think about this before reading on…)

The problem is that the voltage goes over a range of nearly 100 mV whereas the gating variables go only from zero to 1. Therefore to make the plots more equal, let's multiply the gating variables by 100 within the plotting commands (this can be thought of as converting them from fractions of 1 to percentages where 100% = 1) by modifying the last five lines from above to read:

```
plot(t_vect,100*m_vect,'k') %plot in black
plot(t_vect,100*h_vect,'r') %plot in red
plot(t_vect,100*n_vect,'g') %plot in green
ylabel('V, m*100, h*100,or n*100');
xlabel('Time in ms');
legend('V', 'm*100', 'h*100','n*100')
```

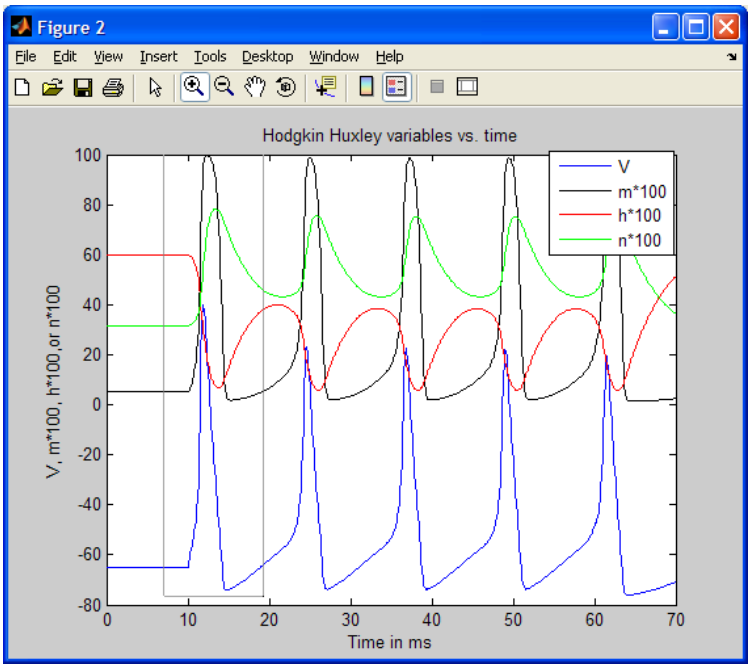
The figure you should now obtain looks like the following:



To really get a good idea of the timing of the various gating variables openings in relation to the voltage, you can zoom in on a smaller time window by clicking on the magnifying glass with the + sign in it in the Figure 2 toolbar and dragging the cursor (which should now look like the

magnifying glass) so as to make a box around the region of interest (as shown below, note gray box between 8 and 10 milliseconds). Take a moment now to study the timing of the different gating variables—which one rises to its peak first? Does this make sense? If you would like to return to the normal view, just double-click with the cursor in the window.

Finally, make a 3rd figure in which you plot the voltage V; total activations of Na^+ and K^+ channels, m^3 and n^4, respectively; total open probability of the Na^+ channel m^3h; and inactivation variable h. What do you notice about these more experimentally-relevant values compared to when you plotted only m and h? [Code for this is in the code summary below if you need to check your code.]



To summarize this section, your code should now read:

```
% Lab 3: Build a Hodgkin-Huxley model neuron; stimulate to produce AP's; plot V and
% activation and inactivation variables m, h, and n; and calculate firing rate

clear all;  %clear all variables
close all; %close any open matlab windows

%DEFINE PARAMETERS
dt = 0.1; %time step [ms]
t_end = 70; %total time of run [ms]
t_StimStart = 10;  %time to start injecting current [ms]
t_StimEnd = 60; %time to end injecting current [ms]
c = 10;  %capacitance per unit area [nF/mm^2]
gmax_L = 0.003e3; %leak maximal conductance per unit area [uS/mm^2]
E_L = -54.387; %leak conductance reversal potential [mV]
gmax_K = 0.36e3; %hodgkin-huxley maximal K conductance per unit area [uS/mm^2]
E_K = -77; %hodgkin-huxley K conductance reversal potential [mV]
gmax_Na = 1.2e3; %hodgkin-huxley maximal Na conductance per unit area [uS/mm^2]
```

```
E_Na = 50; %hodgkin-huxley Na conductance reversal potential [mV]

%SET UP VECTORS TO BE PLOTTED
t_vect = 0:dt:t_end; %will hold vector of times
V_vect = zeros(1,length(t_vect)); %initialize the voltage vector
m_vect = zeros(1,length(t_vect)); %initialize the HH Na activation variable vector
h_vect = zeros(1,length(t_vect)); %initialize the HH Na inactivation variable vector
n_vect = zeros(1,length(t_vect)); %initialize the the HH K activation variable vector

%ASSIGN INITIAL VALUES OF VARIABLES
i = 1;   %index denoting which element of V is being assigned
V_vect(i)= -65;  %first element of V, i.e. value of V at t=0 [mV]
m_vect(i) = 0.0529;  %initially set m = m_inf(-65)
h_vect(i) = 0.5961;  %initially set h = h_inf(-65)
n_vect(i) = 0.3177;  %initially set n = n_inf(-65)

%DEFINE THE STIMULUS
%vector below will hold values of I_e/A over time;
I_0 = 200;  %magnitude of pulse of injected current [nA/mm^2]
I_e_vect = zeros(1,t_StimStart/dt);  %portion of I_e/A vector from t=0 to t=t_StimStart
I_e_vect = [I_e_vect I_0*ones(1,1+((t_StimEnd-t_StimStart)/dt))]; %add portion from
                                                % t=t_StimStart to t=t_StimEnd
I_e_vect = [I_e_vect zeros(1,(t_end-t_StimEnd)/dt)]; %add portion from
                                                % t=t_StimEnd to t=t_end

for t=dt:dt:t_end   %loop through values of t in steps of dt ms
   %assign all of the alphas & betas
   alpha_m = .1*(V_vect(i)+40)/(1 - exp(-.1*(V_vect(i)+40)));
   beta_m = 4*exp(-.0556*(V_vect(i)+65));
   alpha_h = .07*exp(-.05*(V_vect(i)+65));
   beta_h = 1/(1 + exp(-.1*(V_vect(i)+35)));
   alpha_n = .01*(V_vect(i)+55)/(1 - exp(-.1*(V_vect(i)+55)));
   beta_n = 0.125*exp(-0.0125*(V_vect(i)+65));
   %from the alphas & betas above, assign the taus & x_inf's for m,h,n
   tau_m = 1/(alpha_m + beta_m);
   m_inf = alpha_m/(alpha_m + beta_m);
   tau_h = 1/(alpha_h + beta_h);
   h_inf = alpha_h/(alpha_h + beta_h);
   tau_n = 1/(alpha_n + beta_n);
   n_inf = alpha_n/(alpha_n + beta_n);
   %assign tau_V and V_inf
   V_denominator = gmax_L + gmax_K*(n_vect(i)^4) + gmax_Na*(m_vect(i)^3)*h_vect(i);
   tau_V = c/V_denominator;
   V_inf =(gmax_L*E_L + gmax_K*(n_vect(i)^4)*E_K + ...  %... let's you continue on next line
      gmax_Na*(m_vect(i)^3)*h_vect(i)*E_Na + I_e_vect(i))/V_denominator;
   %assign next elements of m,h,and n vectors using update rule
   m_vect(i+1) = m_inf + (m_vect(i) - m_inf)*exp(-dt/tau_m);
   h_vect(i+1) = h_inf + (h_vect(i) - h_inf)*exp(-dt/tau_h);
   n_vect(i+1) = n_inf + (n_vect(i) - n_inf)*exp(-dt/tau_n);
   %assign next element of V vector using update rule
   V_vect(i+1) = V_inf + (V_vect(i) - V_inf)*exp(-dt/tau_V);
   %add 1 to index, corresponding to moving forward 1 time step
   i = i+1;
end
```

```
figure(1)
subplot(4,1,1)
plot(t_vect,V_vect)
title('Hodgkin Huxley variables vs. time');
ylabel('Voltage in mV');
subplot(4,1,2)
plot(t_vect,m_vect)
ylabel('g_{Na} activation variable m');
subplot(4,1,3)
plot(t_vect,h_vect)
ylabel('g_{Na} inactivation variable h');
subplot(4,1,4)
plot(t_vect,n_vect)
xlabel('Time in ms');
ylabel('g_{K} activation variable n');

figure(2)
plot(t_vect,V_vect) %plot in blue
title('Hodgkin Huxley variables vs. time');
hold on
plot(t_vect,100*m_vect,'k') %plot in black
plot(t_vect,100*h_vect,'r') %plot in red
plot(t_vect,100*n_vect,'g') %plot in green
ylabel('V, m*100, h*100,or n*100');
xlabel('Time in ms');
legend('V', 'm*100', 'h*100','n*100')

figure(3)
plot(t_vect,V_vect) %plot in blue
title('Hodgkin Huxley variables vs. time');
hold on
plot(t_vect,100*m_vect.^3,'k') %plot in black
plot(t_vect,100*h_vect,'r') %plot in red
plot(t_vect,100*n_vect.^4,'g') %plot in green
plot(t_vect,100*m_vect.^3.*h_vect,'m:') %plot in dotted magenta
ylabel('V, m^3*100, h*100,or n^4*100');
xlabel('Time in ms');
legend('V', 'm^3*100', 'h*100','n^4*100','m^3h*100')
```

IV. Step 2: Calculate the firing rate

Save your previous work and then, before continuing on, re-save it as HHStep2.m. In this section we will compute the average firing rate of the Hodgkin-Huxley model neuron.

First, let's expand the amount of time we run the model for so that we can count more spikes. In the DEFINE PARAMETERS section change t_end, t_StimStart, and t_StimEnd to

```
t_end = 700; %total time of run [ms]
t_StimStart = 100;  %time to start injecting current [ms]
t_StimEnd = 600; %time to end injecting current [ms]
```

so that we get 500 ms of stimulation centered in a 700 ms window.

Now, how do we define when a spike occurred? Well, a spike occurs every time the voltage "goes sufficiently high". What voltage do we want to consider "high enough" to say that the cell spiked? Let's define a variable V_count_spike corresponding to the value we deem "high enough to count the cell as having spiked" and set V_count_spike equal to -10 mV, i.e. every time the cell passes -10 mV we'll say that it has spiked. Add to the parameter definitions:

```
V_count_spike = -10; % voltage we deem high enough to count as a spike [mV]
```

Now to be more precise, we really want to say that we count the cell as having spiked when the voltage *first* rises upward through V_count_spike = -10 mV. To do this, let's add an if statement within our for loop (put it just after the assignment of V_vect(i+1)):

```
%count this as a spike if just crossed up through V_count_spike
if ((V_vect(i) < V_count_spike) && (V_vect(i+1) > V_count_spike))
   NumSpikes = NumSpikes + 1
end
```

The && means "AND" so these statements say that we count a spike as occurring when the current value of the V vector element is greater than V_count_spike but the previous element is less than V_count_spike. This identifies when the voltage first passes upward through the spike-counting-threshold V_count_spike.

Now do you see something missing here? (You should…). We just added one to a new variable NumSpikes which represents the number of spikes that have occurred, but we never set this variable equal to zero at the beginning of our run. At the end of the section ASSIGN INITIAL VALUES OF VARIABLES, we should add:

```
NumSpikes = 0; %initializes variable holding number of spikes produced in run
```

Now finally, at the end of our calculation code (outside the 'for t' loop but before plotting) let's calculate the average firing rate as:

```
%CALCULATE AVERAGE FIRING RATE
AveRate = 1000*NumSpikes/(t_StimEnd - t_StimStart) %gives average firing rate in [#/sec = Hz]
```

Leave off the semicolon so that this value prints out to your command window. Try this for a few values of I_0. If you try I_0 = 62 nA/mm^2 (*try it!*), you'll find that there are a few spikes to start which then die out. (*Note: For the remainder of this laboratory, I recommend looking at Figure 1 because Figures 2 and 3 are rather crowded. You could even comment out the figure 2 and 3 code by selecting all of it, then right-clicking, then selecting "Comment". You can undo this, i.e. uncomment the code, by following the same procedure but selecting "Uncomment"*). We really don't want to count these into our calculation of the average (sustained) firing rate in response to a particular level of current injection, as they completely die out and for a very long interval of stimulation the number of these per unit time would approach zero (because the number of spikes would stay fixed but the interval would get very long). Therefore, we really should start counting our spikes a couple hundred ms after we start stimulating. Let's therefore make a new variable t_CountSpikeStart and set it to 200 ms after we start stimulating. Add to our parameter definitions, right after the line assigning t_StimStart:

```
t_CountSpikeStart = t_StimStart + 200; %when to start counting spikes for ave. rate [ms]
```

Finally, we now need to modify the if statement above for counting spikes. Since we only want to start counting spikes once the time t_CountSpikeStart is reached, we should really qualify it by saying "if the time is between t_CountSpikeStart and t_StimEnd." We can do this by adding a second if statement surrounding the first (we could also have added more &&'s but this would make the code harder to read):

```
    %count this as a spike if just crossed up through V_count_spike
    if ((t > t_CountSpikeStart) && (t <= t_StimEnd)) %if in spike counting interval
        if ((V_vect(i) < V_count_spike) && (V_vect(i+1) > V_count_spike))
            NumSpikes = NumSpikes + 1;
        end
    end
```

Note the indenting above. The second if statement only gets read by MATLAB if the first one is true, so in effect all of the above conditions must be true for NumSpikes to be increased by 1.

Finally, let's revise our AveRate assignment to reflect the new spike-counting interval:

```
AveRate = 1000*NumSpikes/(t_StimEnd - t_CountSpikeStart) %gives average firing rate in [#/sec = Hz]
```

Now if we again run our code with I_0 = 62, it correctly tells us that the average sustained rate (i.e. not including the few transient spikes when the stimulus first turns on) is zero. Try this with a few more values of input current to make sure that your code is working.

Congratulations! Your final code should be:

```
% Lab 3: Build a Hodgkin-Huxley model neuron; stimulate to produce AP's; plot V and
% activation and inactivation variables m, h, and n; and calculate firing rate

clear all;  %clear all variables
close all; %close any open matlab windows

%DEFINE PARAMETERS
dt = 0.1; %time step [ms]
t_end = 700; %total time of run [ms]
t_StimStart = 100;  %time to start injecting current [ms]
t_CountSpikeStart = t_StimStart + 200; %when to start counting spikes for ave. rate [ms]
t_StimEnd = 600; %time to end injecting current [ms]
c = 10;  %capacitance per unit area [nF/mm^2]
gmax_L = 0.003e3; %leak maximal conductance per unit area [uS/mm^2]
E_L = -54.387; %leak conductance reversal potential [mV]
gmax_K = 0.36e3; %hodgkin-huxley maximal K conductance per unit area [uS/mm^2]
E_K = -77; %hodgkin-huxley K conductance reversal potential [mV]
gmax_Na = 1.2e3; %hodgkin-huxley maximal Na conductance per unit area [uS/mm^2]
E_Na = 50; %hodgkin-huxley Na conductance reversal potential [mV]
V_count_spike = -10; % voltage we deem high enough to count as a spike [mV]

%SET UP VECTORS TO BE PLOTTED
```

```
t_vect = 0:dt:t_end; %will hold vector of times
V_vect = zeros(1,length(t_vect)); %initialize the voltage vector
m_vect = zeros(1,length(t_vect)); %initialize the HH Na activation variable vector
h_vect = zeros(1,length(t_vect)); %initialize the HH Na inactivation variable vector
n_vect = zeros(1,length(t_vect)); %initialize the the HH K activation variable vector

%ASSIGN INITIAL VALUES OF VARIABLES
i = 1;   %index denoting which element of V is being assigned
V_vect(i)= -65;  %first element of V, i.e. value of V at t=0 [mV]
m_vect(i) = 0.0529;  %initially set m = m_inf(-65)
h_vect(i) = 0.5961;  %initially set h = h_inf(-65)
n_vect(i) = 0.3177;  %initially set n = n_inf(-65)
NumSpikes = 0; %initializes variable holding number of spikes produced in run

%DEFINE THE STIMULUS
%vector below will hold values of I_e/A over time;
I_0 = 62;  %magnitude of pulse of injected current [nA/mm^2]
I_e_vect = zeros(1,t_StimStart/dt);  %portion of I_e/A vector from t=0 to t=t_StimStart
I_e_vect = [I_e_vect I_0*ones(1,1+((t_StimEnd-t_StimStart)/dt))]; %add portion from
                                        % t=t_StimStart to t=t_StimEnd
I_e_vect = [I_e_vect zeros(1,(t_end-t_StimEnd)/dt)]; %add portion from
                                        % t=t_StimEnd to t=t_end

for t=dt:dt:t_end   %loop through values of t in steps of dt ms
   %assign all of the alphas & betas
   alpha_m = .1*(V_vect(i)+40)/(1 - exp(-.1*(V_vect(i)+40)));
   beta_m = 4*exp(-.0556*(V_vect(i)+65));
   alpha_h = .07*exp(-.05*(V_vect(i)+65));
   beta_h = 1/(1 + exp(-.1*(V_vect(i)+35)));
   alpha_n = .01*(V_vect(i)+55)/(1 - exp(-.1*(V_vect(i)+55)));
   beta_n = 0.125*exp(-0.0125*(V_vect(i)+65));
   %from the alphas & betas above, assign the taus & x_inf's for m,h,n
   tau_m = 1/(alpha_m + beta_m);
   m_inf = alpha_m/(alpha_m + beta_m);
   tau_h = 1/(alpha_h + beta_h);
   h_inf = alpha_h/(alpha_h + beta_h);
   tau_n = 1/(alpha_n + beta_n);
   n_inf = alpha_n/(alpha_n + beta_n);
   %assign tau_V and V_inf
   V_denominator = gmax_L + gmax_K*(n_vect(i)^4) + gmax_Na*(m_vect(i)^3)*h_vect(i);
   tau_V = c/V_denominator;
   V_inf =(gmax_L*E_L + gmax_K*(n_vect(i)^4)*E_K + ...  %... let's you continue on next line
      gmax_Na*(m_vect(i)^3)*h_vect(i)*E_Na + I_e_vect(i))/V_denominator;
   %assign next elements of m,h,and n vectors using update rule
   m_vect(i+1) = m_inf + (m_vect(i) - m_inf)*exp(-dt/tau_m);
   h_vect(i+1) = h_inf + (h_vect(i) - h_inf)*exp(-dt/tau_h);
   n_vect(i+1) = n_inf + (n_vect(i) - n_inf)*exp(-dt/tau_n);
   %assign next element of V vector using update rule
   V_vect(i+1) = V_inf + (V_vect(i) - V_inf)*exp(-dt/tau_V);
   %count this as a spike if just crossed up through V_count_spike
   if ((t > t_CountSpikeStart) && (t <= t_StimEnd)) %if in spike counting interval
      if ((V_vect(i) < V_count_spike) && (V_vect(i+1) > V_count_spike))
         NumSpikes = NumSpikes + 1;
      end
   end
```

```
    %add 1 to index, corresponding to moving forward 1 time step
    i = i+1;
end

%CALCULATE AVERAGE FIRING RATE
AveRate = 1000*NumSpikes/(t_StimEnd - t_CountSpikeStart) %gives average firing rate in [#/sec = Hz]

figure(1)
subplot(4,1,1)
plot(t_vect,V_vect)
title('Hodgkin Huxley variables vs. time');
ylabel('Voltage in mV');
subplot(4,1,2)
plot(t_vect,m_vect)
ylabel('g_{Na} activation variable m');
subplot(4,1,3)
plot(t_vect,h_vect)
ylabel('g_{Na} inactivation variable h');
subplot(4,1,4)
plot(t_vect,n_vect)
xlabel('Time in ms');
ylabel('g_{K} activation variable n');

figure(2)
plot(t_vect,V_vect) %plot in blue
title('Hodgkin Huxley variables vs. time');
hold on
plot(t_vect,100*m_vect,'k') %plot in black
plot(t_vect,100*h_vect,'r') %plot in red
plot(t_vect,100*n_vect,'g') %plot in green
ylabel('V, m*100, h*100,or n*100');
xlabel('Time in ms');
legend('V', 'm*100', 'h*100','n*100')

figure(3)
plot(t_vect,V_vect) %plot in blue
title('Hodgkin Huxley variables vs. time');
hold on
plot(t_vect,100*m_vect.^3,'k') %plot in black
plot(t_vect,100*h_vect,'r') %plot in red
plot(t_vect,100*n_vect.^4,'g') %plot in green
plot(t_vect,100*m_vect.^3.*h_vect,'m:') %plot in dotted magenta
ylabel('V, m^3*100, h*100,or n^4*100');
xlabel('Time in ms');
legend('V', 'm^3*100', 'h*100','n^4*100','m^3h*100')
```