
Hodgkin Huxley LEMS Tutorial Documentation

Release 0.1

Joe Bowen

Dec 05, 2017

1	What is the Hodgkin-Huxley model?	3
1.1	Implementation of HH Model in Python and NeuroML 2	3
1.1.1	Installing the code	3
1.1.2	Running the model implementations	4
1.1.2.1	Membrane Capacitance	4
1.1.2.2	Sodium (Na) Ion Channel Variables	4
1.1.2.3	Potassium (K) Ion Channel Variables	5
1.1.2.4	Passive Leak Channel Variables	5
1.1.2.5	Time of Simulation	5
1.1.2.6	Input Current / Input Current Density	5
1.1.2.7	Channel Gating Kinetics for Sodium (Na) Channel m	6
1.1.2.8	Channel Gating Kinetics for Sodium (Na) Channel h	7
1.1.2.9	Channel Gating Kinetics for Potassium (K) channel n	7
1.1.2.10	Initial Values	8
1.1.2.11	Plots	8
1.1.2.12	Output of simulations	9
1.2	Biological/Electronic Equivalence	11
1.2.1	The Circuit	12
1.2.1.1	Modeling voltage-gating versus leak:	12
1.2.2	The Math	12
1.2.2.1	Lipid bilayer current	12
1.2.2.2	Ion channel current	12
1.2.2.3	Combining these currents	12
1.2.2.4	Adding in activation parameters	12
1.2.3	Plots	13
1.2.3.1	Description of each plot	13
1.2.3.2	Relationship between the plots	14
1.2.4	Terms	14
1.3	Current-voltage characteristic	14
1.3.1	The patch-clamp protocol	15
1.3.2	Making I/V plots	15
1.3.2.1	How does current even flow across the membrane?	18
1.3.3	Using code to produce these plots	18
1.4	Exercises	18
1.5	Hodgkin Huxley Sources	19
1.5.1	Hodgkin Huxley.py	19

1.5.2	LEMS_HH_Simulation.xml	22
1.5.3	HHCellNetwork.net.nml	23
1.5.4	hhcell.cell.nml	24
1.5.5	passiveChan.channel.nml	25
1.5.6	naChan.channel.nml	25
1.5.7	kChan.channel.nml	26
1.5.8	run.sh	27
1.5.9	run.bat	27

This tutorial gives an introduction to the [Hodgkin-Huxley model](#) by use of executable example implementations in Python and [NeuroML](#).

The aims of this tutorial are:

1. Provide a [guide](#) to implementing the Hodgkin-Huxley model using both [Python](#) and a [NeuroML2 implementation](#) of the same equations.
2. Give some background information on the [electrophysiology](#) underlying the Hodgkin-Huxley model.

This was originally created by [@joebowen](#) on behalf of the [OpenWorm project](#).

What is the Hodgkin-Huxley model?

From Wikipedia: The Hodgkin–Huxley model is a mathematical model that describes how action potentials in neurons are initiated and propagated.

The model describes represents the electrical properties of excitable membranes as typical electrical circuit components. For instance, the cell's membrane is modeled as a capacitor, and voltage-dependent conductances stand in for what are now known to be voltage-gated ion channels.

For a detailed run through of the Hodgkin-Huxley model's electronics, math and biology, take a look at the [Electrophysiology page](#).

After you understand the electronic model there, check out the [code walkthrough](#) to see an example implementation of the Hodgkin-Huxley model in Python, using a cell modeled in [NeuroML2](#).

You can look at the [current-voltage characteristic page](#) to get an understanding of another biological-electronic equivalence that is useful in describing ion channel and cell models.

There are also some [exercises](#) you can complete to get a feel for the model. These can be completed using either the Python or NeuroML versions.

Table of Contents:

1.1 Implementation of HH Model in Python and NeuroML 2

In this section, we make line-by-line comparisons of the contents of the [HodgkinHuxley.py](#) python script and the contents of the [NeuroML 2 files](#) and the related [LEMS_HH_Simulation.xml](#) LEMS file.

1.1.1 Installing the code

You can either clone a local copy of this repository from GitHub using:

```
git clone https://github.com/openworm/hodgkin_huxley_tutorial.git
cd hodgkin_huxley_tutorial/Tutorial/Source
```

or just download a [zip file](#) of the latest code, extract the contents and go to `hodgkin_huxley_tutorial-master/Tutorial/Source/`.

1.1.2 Running the model implementations

To run the Python version:

```
python HodgkinHuxley.py
```

To run the NeuroML 2 version on Linux/Mac:

```
./run.sh
```

or on Windows:

```
run.bat
```

These both use the [bundled jar file](#) generated from [jNeuroML](#). Alternatively, you can install the latest [jNeuroML](#) or [pyNeuroML](#) and use the corresponding command line utilities to run the model:

```
jnml LEMS_HH_Simulation.xml  
pynml LEMS_HH_Simulation.xml
```

1.1.2.1 Membrane Capacitance

This variable from `HodgkinHuxley.py`:

```
C_m = 1.0  
"""membrane capacitance, in uF/cm^2"""
```

Is used in this line in `hhcell.cell.nml`:

```
<specificCapacitance value="1.0 uF_per_cm2"/>
```

You can read more about the capacitance of a membrane.

1.1.2.2 Sodium (Na) Ion Channel Variables

These variables from `HodgkinHuxley.py`:

```
g_Na = 120.0  
"""Sodium (Na) maximum conductances, in mS/cm^2"""
```

```
E_Na = 50.0  
"""Sodium (Na) Nernst reversal potentials, in mV"""
```

Are used in this line in `hhcell.cell.nml`:

```
<channelDensity id="naChans" ionChannel="naChan" condDensity="120.0_  
↪mS_per_cm2" erev="50.0 mV" ion="na"/>
```

You can read more about the maximum conductance and reversal potential (zero-current potential) of an ion channel.

1.1.2.3 Potassium (K) Ion Channel Variables

These variables from `HodgkinHuxley.py`:

```
g_K = 36.0
"""Postassium (K) maximum conductances, in mS/cm^2"""
```

```
E_K = -77.0
"""Postassium (K) Nernst reversal potentials, in mV"""
```

Are used in this line in `hhcell.cell.nml`:

```
<channelDensity id="kChans" ionChannel="kChan" condDensity="36 mS_per_
↪cm2" erev="-77mV" ion="k"/>
```

You can read more about the maximum conductance and reversal potential (zero-current potential) of an ion channel.

1.1.2.4 Passive Leak Channel Variables

These variables from `HodgkinHuxley.py`:

```
g_L = 0.3
"""Leak maximum conductances, in mS/cm^2"""
```

```
E_L = -54.387
"""Leak Nernst reversal potentials, in mV"""
```

Are used in this line in `hhcell.cell.nml`:

```
<channelDensity id="leak" ionChannel="passiveChan" condDensity="0.3_
↪mS_per_cm2" erev="-54.387mV" ion="non_specific"/>
```

You can read more about the maximum conductance and reversal potential (zero-current potential) of an ion channel.

1.1.2.5 Time of Simulation

This variable from `HodgkinHuxley.py`:

```
t = sp.arange(0.0, 450.0, 0.01)
""" The time to integrate over """
```

Is used in this line in `LEMS_HH_Simulation.xml`:

```
<Simulation id="sim1" length="450ms" step="0.01ms" target="HHCellNetwork">
```

This specifies that the simulation should run for 450 milliseconds and use a step size for integration of 0.01 milliseconds.

1.1.2.6 Input Current / Input Current Density

The method from `HodgkinHuxley.py` takes the input in as a current density in the form of $\mu\text{A}/\text{cm}^2$. NeuroML/LEMS uses an input current in the form of nA, which requires a conversion in the input values.

This method from `HodgkinHuxley.py`:

```

1  def I_inj(self, t):
2      """
3      External Current
4
5      | :param t: time
6      | :return: step up to 10 uA/cm^2 at t>100
7      |           step down to 0 uA/cm^2 at t>200
8      |           step up to 35 uA/cm^2 at t>300
9      |           step down to 0 uA/cm^2 at t>400
10     """
11     return 10*(t>100) - 10*(t>200) + 35*(t>300) - 35*(t>400)

```

By using a given surface area of 1000.0 μm^2 in the cell, it makes the conversion from uA/cm^2 to nA easier.

$$\text{SurfaceArea} = 4 * \pi * (\text{radius})^2 = 4 * \pi * (\text{diameter}/2)^2 = 4 * \pi * (17.841242/2)^2 = 4 * \pi * (8.920621)^2 = 1000\mu\text{m}^2$$

```

<segment id="0" name="soma">
  <proximal x="0" y="0" z="0" diameter="17.841242"/> <!--Gives a
↪ convenient surface area of 1000.0 um^2-->
  <distal x="0" y="0" z="0" diameter="17.841242"/>
</segment>

```

Given a surface area of 1000.0 μm^2 in the cell the following equation is used to convert from X uA/cm^2 to Y nA:

$$(X \text{uA}/\text{cm}^2) * (1000.0 \mu\text{m}^2) * (1000 \text{nA}/\text{uA}) / (1 * 10^8 \mu\text{m}^2/\text{cm}^2) = Y \text{nA}$$

Line 11 can then be translated into the delay, duration and amplitude of the two pulseGenerator objects in HHCeIlNetwork.net.nml:

```
<network id="HHCeIlNetwork">
```

1.1.2.7 Channel Gating Kinetics for Sodium (Na) Channel m

m is the activation variable for the Sodium (Na) Channel.

The function that governs the activation of this channel is based on the overall membrane voltage, because the channel opens and closes based on detecting the membrane potential.

You can read more about these variables.

These methods from HodgkinHuxley.py:

```

1  def alpha_m(self, V):
2      """Channel gating kinetics. Functions of membrane voltage"""
3      return 0.1*(V+40.0)/(1.0 - sp.exp(-(V+40.0) / 10.0))

```

```

1  def beta_m(self, V):
2      """Channel gating kinetics. Functions of membrane voltage"""
3      return 4.0*sp.exp(-(V+65.0) / 18.0)

```

Are used in these lines in naChan.channel.nml:

```

<gateHHrates id="m" instances="3">
  <forwardRate type="HHExpLinearRate" rate="1per_ms" midpoint="-40mV" scale=
↪ "10mV"/>

```

```

    <reverseRate type="HHExpRate" rate="4per_ms" midpoint="-65mV" scale="-18mV
↪"/>
  </gateHHrates>

```

1.1.2.8 Channel Gating Kinetics for Sodium (Na) Channel h

h is the inactivation variable for the Sodium (Na) Channel. Inactivation is a different state than not being activated, which is called “deactivated”. You can [read more about how Sodium channel gating works](#).

The function that governs the activation of this channel is based on the overall membrane voltage, because the channel opens and closes based on detecting the membrane potential.

You can [read more about these variables](#).

These methods from `HodgkinHuxley.py`:

```

1  def alpha_h(self, V):
2      """Channel gating kinetics. Functions of membrane voltage"""
3      return 0.07*sp.exp(-(V+65.0) / 20.0)

```

```

1  def beta_h(self, V):
2      """Channel gating kinetics. Functions of membrane voltage"""
3      return 1.0/(1.0 + sp.exp(-(V+35.0) / 10.0))

```

Are used in these lines in `naChan.channel.nml`:

```

    <gateHHrates id="h" instances="1">
      <forwardRate type="HHExpRate" rate="0.07per_ms" midpoint="-65mV" scale="-
↪20mV"/>
      <reverseRate type="HHSigmoidRate" rate="1per_ms" midpoint="-35mV" scale=
↪"10mV"/>
    </gateHHrates>

```

1.1.2.9 Channel Gating Kinetics for Potassium (K) channel n

n is the activation variable for the Potassium (K) Channel. The potassium channel does not inactivate, so there is no inactivation variable.

The function that governs the activation of this channel is based on the overall membrane voltage, because the channel opens and closes based on detecting the membrane potential.

You can [read more about these variables](#).

These methods from `HodgkinHuxley.py`:

```

1  def alpha_n(self, V):
2      """Channel gating kinetics. Functions of membrane voltage"""
3      return 0.01*(V+55.0)/(1.0 - sp.exp(-(V+55.0) / 10.0))

```

```

1  def beta_n(self, V):
2      """Channel gating kinetics. Functions of membrane voltage"""
3      return 0.125*sp.exp(-(V+65) / 80.0)

```

Are used in these lines in `kChan.channel.nml`:

```
<gateHHrates id="n" instances="4">
  <forwardRate type="HHExpLinearRate" rate="0.1per_ms" midpoint="-55mV"
↪ scale="10mV"/>
  <reverseRate type="HHExpRate" rate="0.125per_ms" midpoint="-65mV" scale="-
↪ 80mV"/>
</gateHHrates>
```

1.1.2.10 Initial Values

This line from `HodgkinHuxley.py`:

```
X = odeint(self.dALLdt, [-65, 0.05, 0.6, 0.32], self.t, args=(self,))
```

Is used to define the initial values for the model in `hhcell.cell.nml`:

```
<initMembPotential value="-65mV"/>
```

The values for m , h , n at $t=0$ in LEMS/NML2 are worked out as the steady state values (inf) of each activation variable for the given initial membrane potential. See [here](#) for the NML2 implementation (see On Start).

You could refactor the script to do this too by introducing `tau_m()` and `inf_m()` and using `alpha_m` etc., change the expressions for `dmdt` etc. (e.g. $\text{dm}/\text{dt} = (\text{inf}_m - m) / \text{tau}_m$) etc. and:

```
V_init = -65
X = odeint(self.dALLdt, [V_init, m_inf(V_init), h_inf(V_init), n_inf(V_init)], self.t,
↪ args=(self,))
```

1.1.2.11 Plots

This line in `HodgkinHuxley.py`:

```
plt.subplot(4,1,1)
plt.title('Hodgkin-Huxley Neuron')
plt.plot(self.t, V, 'k')
plt.ylabel('V (mV)')
```

Is used in these lines in `LEMS_HH_Simulation.xml`:

```
<Display id="d1" title="Hodgkin-Huxley Neuron: V (mV)" timeScale="1ms" xmin="-
↪ 20" xmax="470" ymin="-90" ymax="50">
  <Line id="V" quantity="hhpop[0]/v" scale="1mV" color="#000000" timeScale=
↪ "1ms"/>
</Display>
```

This line in `HodgkinHuxley.py`:

```
plt.subplot(4,1,2)
plt.plot(self.t, ina, 'c', label='$I_{Na}$')
plt.plot(self.t, ik, 'y', label='$I_{K}$')
plt.plot(self.t, il, 'm', label='$I_{L}$')
plt.ylabel('Current')
plt.legend()
```

Is used in these lines in `LEMS_HH_Simulation.xml`:

```

<Display id="d3" title="Hodgkin-Huxley Neuron: Current" timeScale="1ms" xmin=
↪"-20" xmax="470" ymin="-10" ymax="10">
  <Line id="I_na" quantity="hhpop[0]/bioPhys1/membraneProperties/naChans/
↪iDensity" scale="1" color="#00ffff" timeScale="1ms"/>
  <Line id="I_k" quantity="hhpop[0]/bioPhys1/membraneProperties/kChans/
↪iDensity" scale="1" color="#ffff00" timeScale="1ms"/>
  <Line id="I_l" quantity="hhpop[0]/bioPhys1/membraneProperties/leak/
↪iDensity" scale="1" color="#ff00ff" timeScale="1ms"/>
</Display>

```

This line in `HodgkinHuxley.py`:

```

plt.subplot(4,1,3)
plt.plot(self.t, m, 'r', label='m')
plt.plot(self.t, h, 'g', label='h')
plt.plot(self.t, n, 'b', label='n')
plt.ylabel('Gating Value')
plt.legend()

```

Is used in these lines in `LEMS_HH_Simulation.xml`:

```

<Display id="d2" title="Hodgkin-Huxley Neuron: Gating Variables" timeScale=
↪"1ms" xmin="-20" xmax="470" ymin="-0.1" ymax="1.1">
  <Line id="m" quantity="hhpop[0]/bioPhys1/membraneProperties/naChans/
↪naChan/m/q" scale="1" color="#ff0000" timeScale="1ms"/>
  <Line id="h" quantity="hhpop[0]/bioPhys1/membraneProperties/naChans/
↪naChan/h/q" scale="1" color="#00dd00" timeScale="1ms"/>
  <Line id="n" quantity="hhpop[0]/bioPhys1/membraneProperties/kChans/kChan/
↪n/q" scale="1" color="#0000ff" timeScale="1ms"/>
</Display>

```

This line in `HodgkinHuxley.py`:

```

plt.subplot(4,1,4)
i_inj_values = [self.I_inj(t) for t in self.t]
plt.plot(self.t, i_inj_values, 'k')
plt.xlabel('t (ms)')
plt.ylabel('$I_{inj}$ ($\mu A/cm^2$)')

```

Is used in these lines in `LEMS_HH_Simulation.xml`:

```

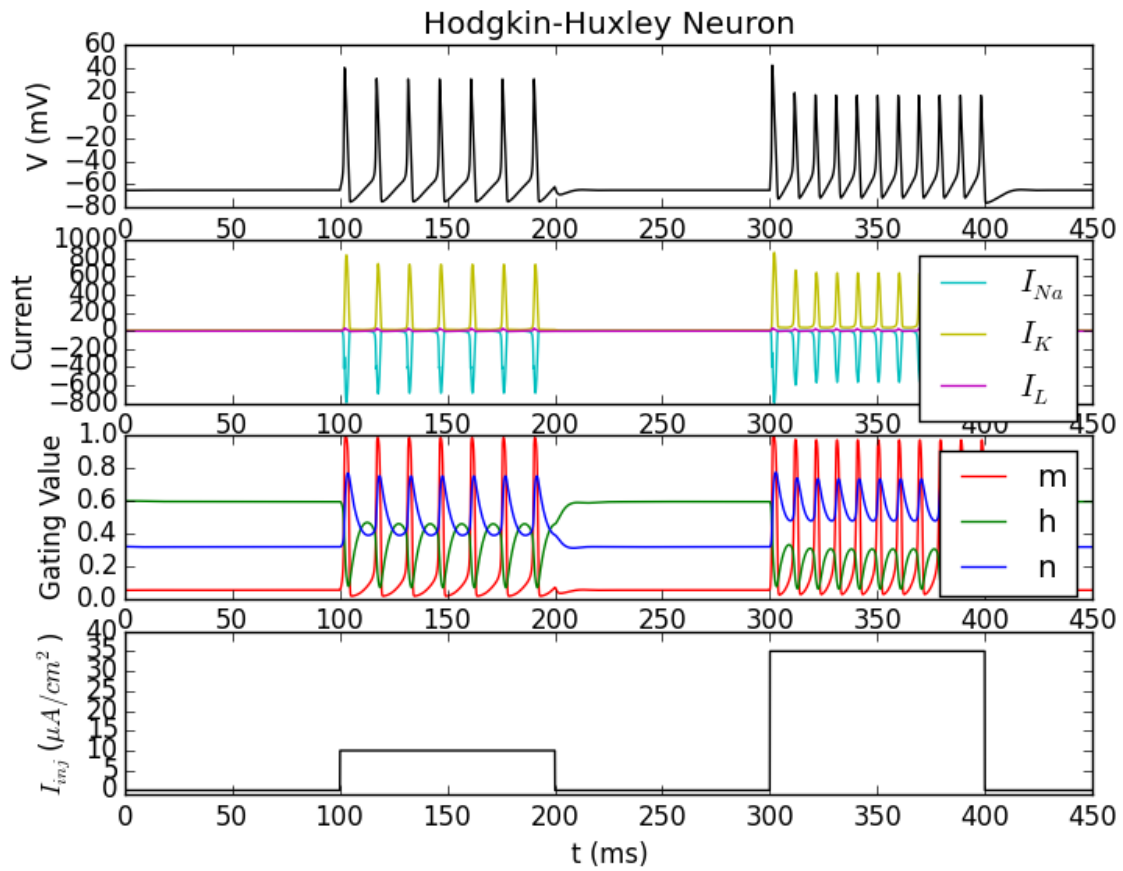
<Display id="d4" title="Hodgkin-Huxley Neuron: I_inj (nA)" timeScale="1ms"
↪xmin="-20" xmax="470" ymin="-0.01" ymax="0.4">
  <Line id="I_inj1" quantity="hhpop[0]/pulseGen1/i" scale="1nA" color="
↪#ffffff" timeScale="1ms"/>
  <Line id="I_inj2" quantity="hhpop[0]/pulseGen2/i" scale="1nA" color="
↪#000000" timeScale="1ms"/>
</Display>

```

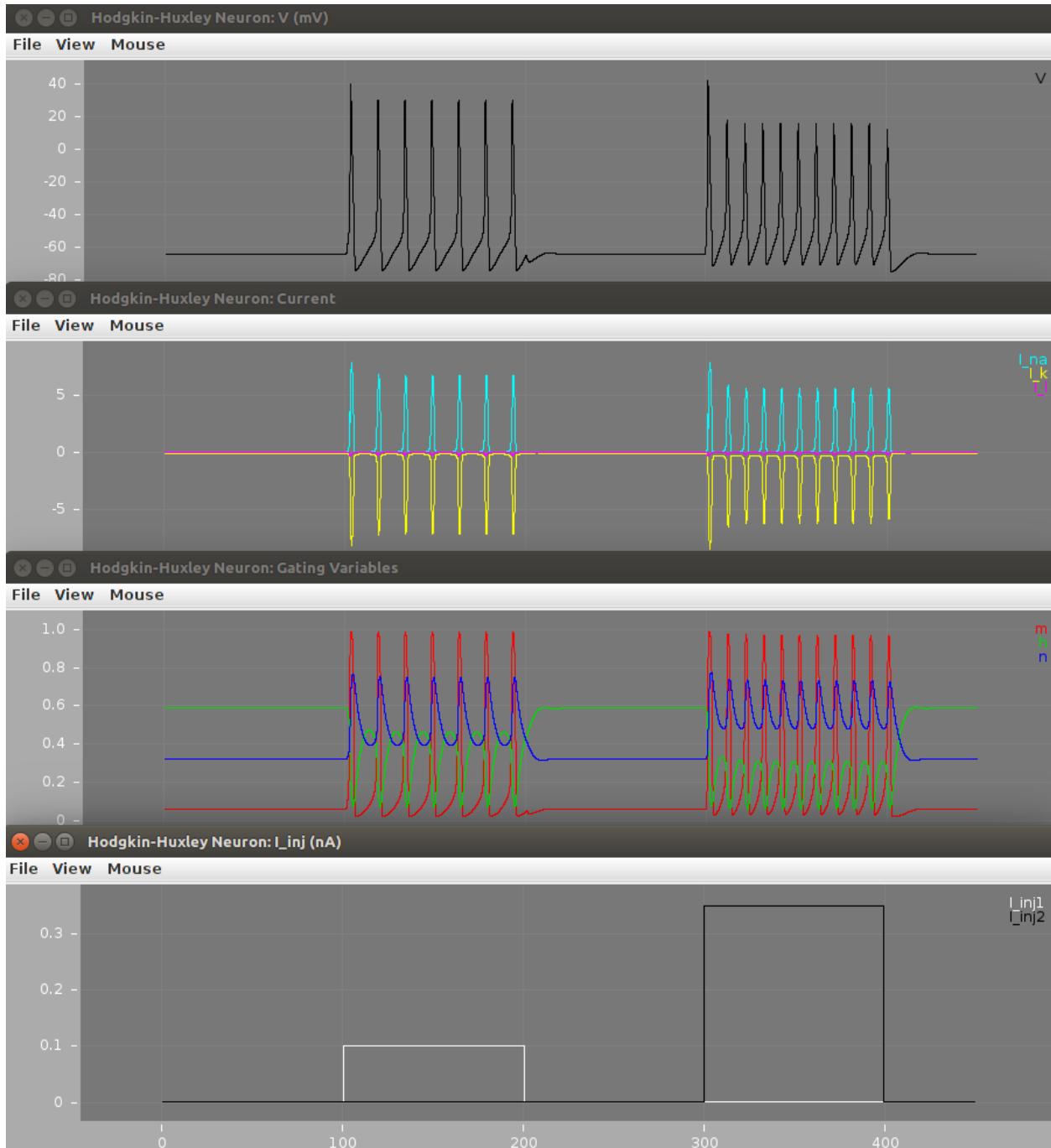
1.1.2.12 Output of simulations

After running the scripts the output figures should look like the ones below.

For: `python HodgkinHuxley.py`



For: *run.sh* (or *run.bat* on Windows)



Check out the [electrophysiology part of this tutorial](#) for an explanation of these plots.

1.2 Biological/Electronic Equivalence

This model relies on a basic equivalence between a biological membrane plus embedded ion channels, and an electronic circuit.

The circuit can be described by the diagram below, which is an electronic diagram representing a patch of cellular membrane.

1.2.1 The Circuit

The **membrane capacitance** (C_m) is taken to be a fixed property of the membrane.

Parallel to C_m are two “battery-capacitor” series; one for each of voltage-gated and leak ion channels.

Each of these ion pathways are modeled as the product of the ion’s **conductance** (g) and its driving electrochemical gradient (E), both of which may vary over time (except in the case of g_L ; see below).

I_p represents the active movement of ions provided by **ion transporters**.

The net result of all of this activity in the cell membrane is a current across the membrane (i.e. from intracellular medium to extracellular medium, or vice versa).

1.2.1.1 Modeling voltage-gating versus leak:

The conductances of voltage-gated and leak channels, g_n and g_L respectively, are modeled differently. Since the gating of voltage-gated ion channels depends on the membrane potential at a given moment, it is non-linear. In contrast, leak ion channels are always in the same state, so their conductance is modeled linearly.

1.2.2 The Math

1.2.2.1 Lipid bilayer current

The current across the cell’s lipid bilayer (I_c) is the product of the membrane’s capacitance (C_m) and the rate of change of membrane potential (V_m) with respect to time (t).

1.2.2.2 Ion channel current

The current through a given ion channel (I_i) is the product of that channel’s conductance (g_i) and the difference $V_m - V_i$

V_i is the ion species’ **reversal potential**. Notice that when V_m is equal to V_i the product becomes zero, and there is no net flow (I_i) for the ion, which is what defines reversal potential.

1.2.2.3 Combining these currents

If we sum the lipid bilayer current with ion channel currents for each ion species, we end up with a total current (I) for the patch of cellular membrane.

In the equation above, voltage-gated potassium (K) and sodium (Na) channels, as well as leak channels (L) are considered, leading to three instances of the ion channel current calculations.

1.2.2.4 Adding in activation parameters

Since the ion channels denoted in the equation above are in various states, some new variables must be added to the equation. Namely, activation and inactivation parameters are now included in each ion channel current calculation.

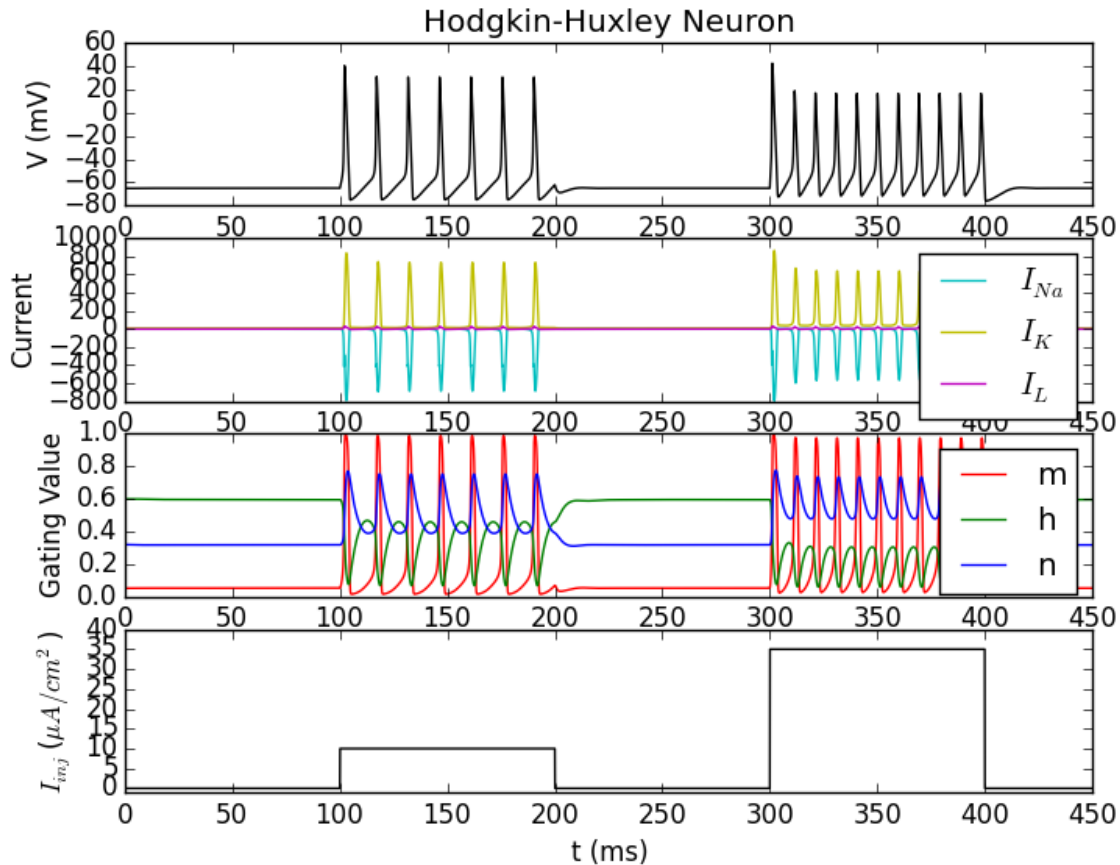
The new notation for each conductance variable (g) is the *maximal* conductance for that ion channel type. This, combined with the activation/inactivation parameters n , m and h , still represents the level of conductance for an ion channel, but with parameters that modify this conductance.

- m is the activation parameter for sodium (Na) channels
- h is the inactivation parameter for sodium channels

- n is the activation parameter for potassium (K) channels

1.2.3 Plots

The plots generated by running the `HodgkinHuxley.py` script bundled with this tutorial are show below.



1.2.3.1 Description of each plot

Starting from the bottom, the first (bottom-most) plot shows two currents injected into the cell membrane at times 100ms and 300ms. Notice that the second injected current is significantly larger in magnitude than the first.

The second plot from the bottom (let's call this the *gating plot*) shows the activation/inactivation parameters of the ion channels in the neuron. The precise meanings of the three lines labeled m , h and n are described above, but it is sufficient to say that these parameters are proportional to the “amount” of gating for their respective ion channels. In other words, the amount of influence of each parameter on internal dynamics is given proportional to its full possible influence. So a gating value of 1 is at its maximal influence, and zero is no influence at all.

The third plot from the bottom (the *current/time plot*) makes this more concrete, showing the influx (negative y-axis) and outflux (positive y-axis) of ions passing through each type of ion channel being modeled. Notice that, in the gating plot, at times when m is large and h is small for a moment (say, just after 100ms), the sodium current (I_{Na}) spikes *outward*. Notice also that the potassium current (I_K) spikes inward when its activation parameter n spikes in the gating plot.

Finally, consider the top plot, which shows neural membrane voltage activity. The spikes here are called “action potentials” and correspond directly to the current/time plot. Outflux of *Na* directly followed by influx of *K* causes the spiking activity observed in the topmost plot.

1.2.3.2 Relationship between the plots

Notice that the first set of action potentials (from about 100 to 200ms) is sparse compared to the second set (from 300 to 400ms). This is due to the increased current applied across the membrane in the second injection (see the bottom plot).

It is possible to see how intracellular and cell-patch dynamics are related through these four plots. Gating parameters affect ion channel conductance, which directly influences ion flow, which in turn controls electric potential across the membrane.

1.2.4 Terms

- **Ion channel**
 - Protein embedded in cellular membrane allowing *passive* flow of ions, depending on its configuration.
- **Ion channel conductance**
 - The rate of flow of ions through an ion channel. Directly affects membrane conductance, and changes with gating behaviour of an ion channel.
- **Ion transporter**
 - Protein embedded in cellular membrane that moves ions *actively*
- **Membrane capacitance**
- **Membrane conductance**
 - Total membrane conductance is the rate at which current (i.e. ions) can flow through the membrane, and is a result of the configuration of ion channels at a given moment.
- **Membrane potential**
 - The difference in electric potential between the exterior and interior of a cell.
- **Nernst potential**
 - See “Reversal potential”.
- **Reversal potential**
 - The membrane potential at which a given ion species has no overall flow across the membrane (i.e. the ion flow direction “reverses”).

1.3 Current-voltage characteristic

The current through an electronic component, with the corresponding potential difference (voltage) across it is called the **current-voltage characteristic**.

This relationship is usually represented with an *I/V* curve, which is just that: a plot of current versus voltage.

This property is one characteristic used to when examining the behaviour of electronic circuits. Given the electronic-biological equivalence [discussed earlier](#), it is easy to see how this property would also be useful in defining the behaviour of excitable membranes and their embedded ion channels.

1.3.1 The patch-clamp protocol

In a patch-clamp experiment, a piece of membrane is sealed off from its surrounding environment, such that there is almost no influence of external electrochemical process on what is happening in this small “patch” of membrane.

By applying a voltage through this membrane patch, almost perfect control of the membrane potential can be obtained. In this way, an experimenter can hold the membrane at various voltages (fig. 1) and observe the current response that occurs (fig. 2).

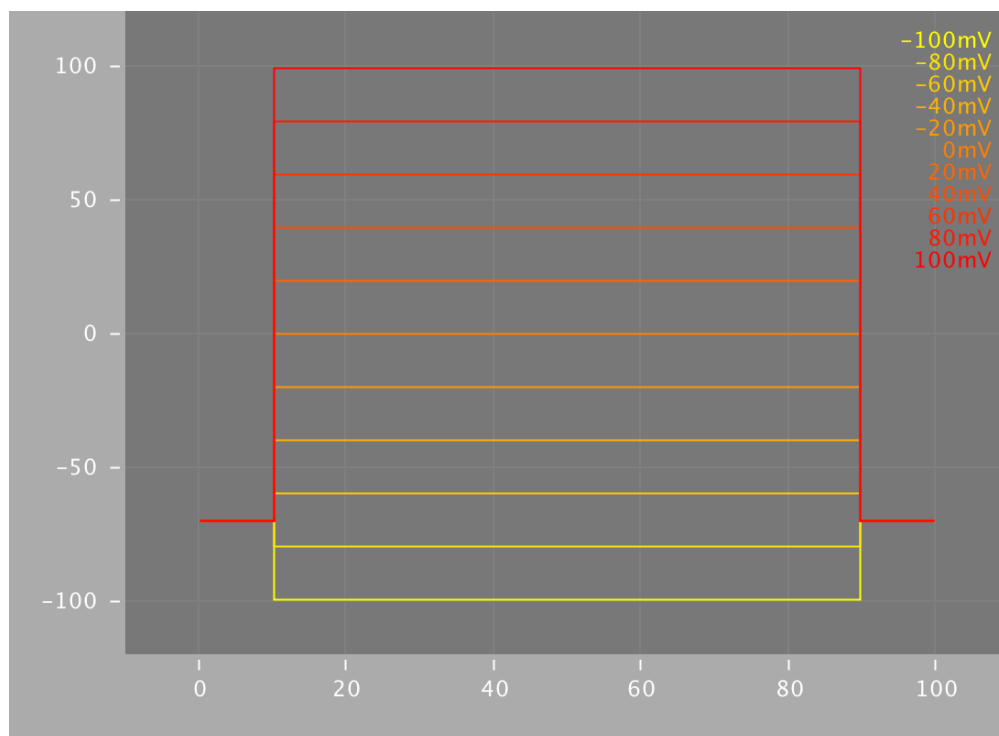


Fig. 1.1: Fig. 1

Voltage stepping in a patch-clamp protocol. Electrical potential applied across a patch of membrane, holding the membrane at that potential.

1.3.2 Making I/V plots

With this data, we can now plot a current-voltage relationship, to help us characterize and model the electrophysiological behaviour of the patch of membrane.

We will consider two types of I/V curves here. The first is the so-called “peak” I/V curve, where the largest current magnitude produced at each voltage step is plotted against the voltage that produced it. We can see an example of this in figure 3.

The second type of I/V curve is called a “steady-state” I/V curve, and is a representation of the somewhat leveled out current at the end of each voltage step, again plotted against the voltage step that produced it. See figure 4 for an example of this type of I/V plot.

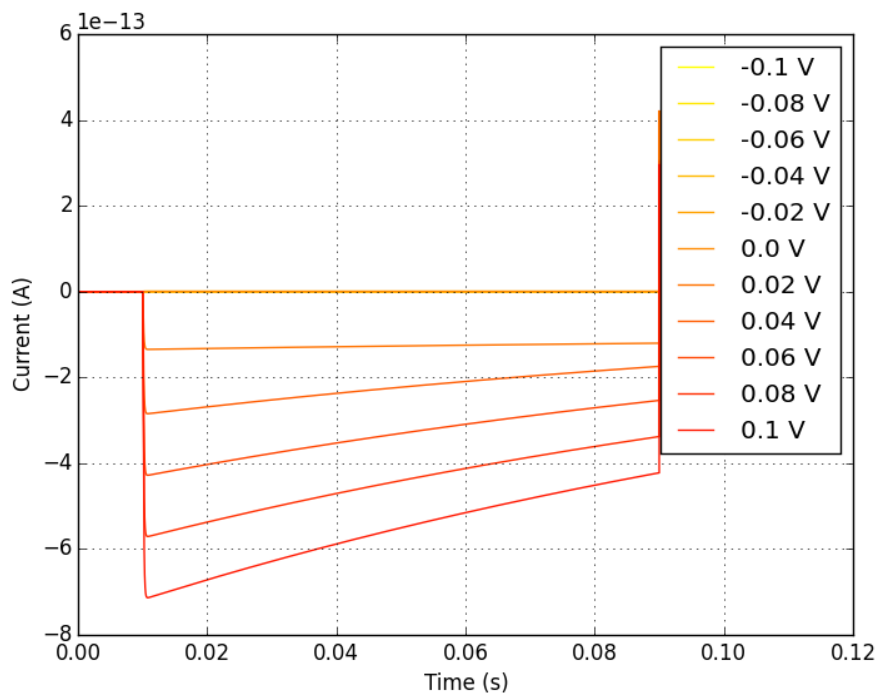


Fig. 1.2: Fig. 2

Current-time plot for a voltage-clamped membrane patch. This represents the current change over time in response to voltage clamping.

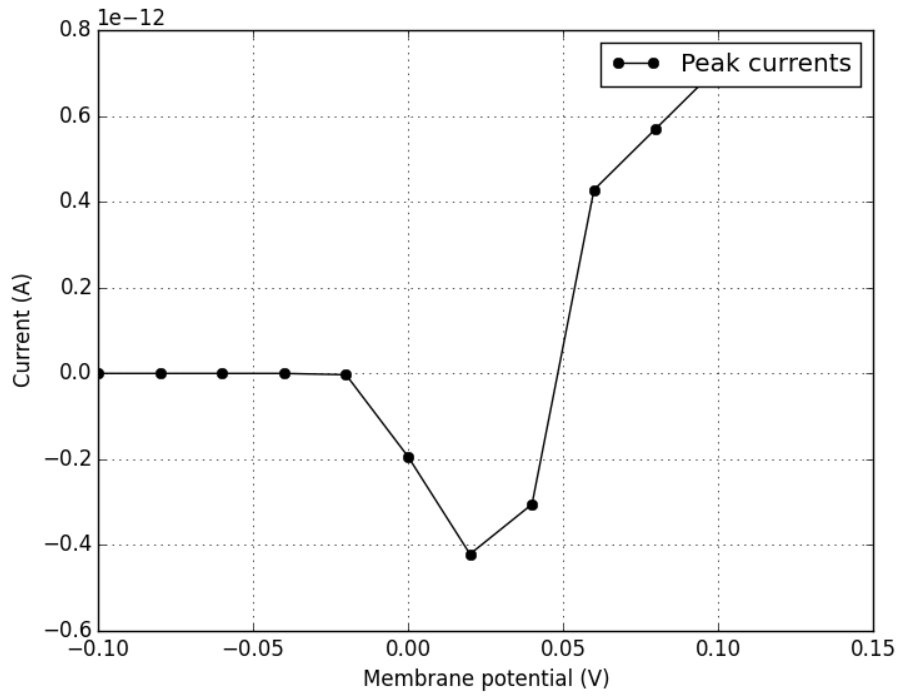


Fig. 1.3: Fig. 3
Peak I/V Curve. Plotting the maximum current at each voltage step produces a curve like this.

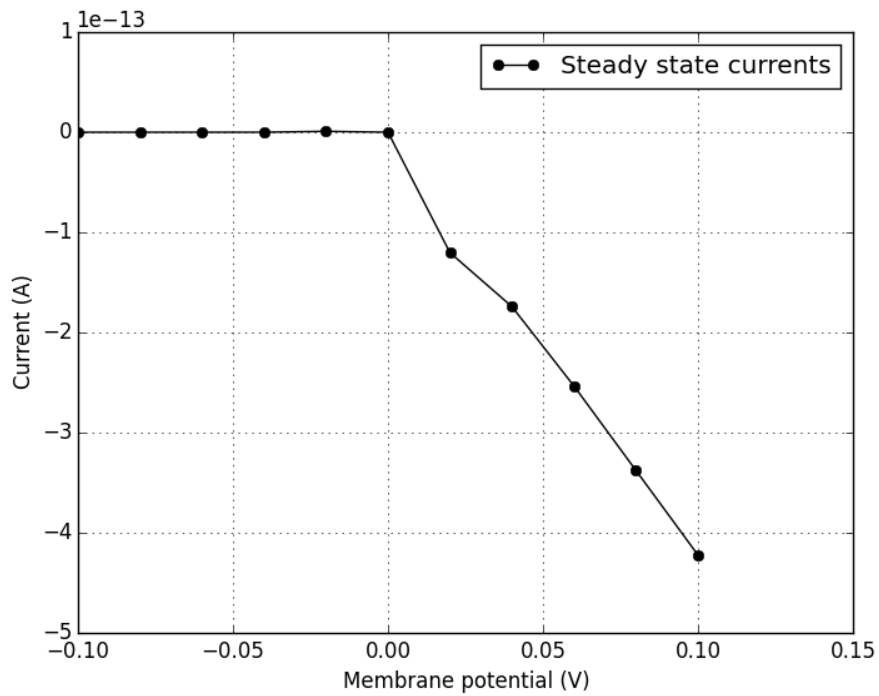


Fig. 1.4: Fig. 4
Steady-state I/V curve. Plotting the current at the end of each voltage step gives us a curve similar to this one.

1.3.2.1 How does current even flow across the membrane?

In the [electrophysiology](#) section we looked at how voltage-gated ion channels influence the kinetics of excitable cells. It is this behaviour that we are closely examining here, by holding the membrane potential at a particular level and observing what happens to ion flow (current) across the membrane.

If we patch-clamp a larger piece of membrane, there will be many ion channels exerting their effect. Technology now exists, however, that allows electrophysiologists to patch-clamp a *single* ion channel and perform the same experiments. In this way, it is possible to obtain data about individual ion channel types, and characterize their kinetics using I/V curves.

The above figures are all examples of this type of ion channel patch clamping.

1.3.3 Using code to produce these plots

At the risk of losing your trust, it must be admitted that the plots above were not actual biological recordings, but were instead generated by *simulating* a single ion channel patch-clamp experiment.

Using a [NeuroML2 model of an ion channel](#) and a suite of virtual electrophysiology tools ([pyNeuroML](#)), you can produce this set of curves, and a similar characterization for any number of ion channel models that exist.

First, make sure you have the latest version of [pyNeuroML](#) installed. Jump over to that project's [installation instructions](#) to get up and running.

Now, by doing the set of commands below in your shell, you should be presented with the same set of plots we have been using in this tutorial.

```
# grab a sample channel model
wget https://goo.gl/yrAfhn -O Cav1.channel.nml

# analyse it
pynml-channelanalysis -ivCurve Cav1.channel.nml
```

1.4 Exercises

Some exercises which can be carried out with the Python or NeuroML versions of the HH model. See [here](#) for installation instructions.

1) Adjust the input current to the cell to investigate the changes in membrane potential and Na, K current behaviour

To change the scripts for the HH model:

<i>Python</i>	adjust values in method <i>I_inj</i> in HodgkinHuxley.py
<i>NeuroML2</i>	adjust values for <i>amplitude</i> in <pulseGenerator> elements in HHCellNetwork.net.nml

- 1.1 What is the minimum current you can inject that will cause at least one spike?
- 1.2 Adjust the input current duration to stimulate the cell for the full duration of the simulation. What is the minimum current you need to inject to get the cell to fire for the full duration?
- 1.3 How much does a 10-fold increase in injected current from the repetitive firing current increase the firing rate?
- 1.4 What happens for a 100-fold increase? Why does this happen?
- 1.5 Is there any current you can inject to get a half height action potential?

1.6 Single action potentials can also be elicited by transient current pulses, even when the duration of the current pulse is shorter than the action potential. What is the effect of pulse duration on threshold current for eliciting a single action potential? Generate a plot of threshold current vs. pulse duration for pulse widths between 0.1 ms and 5 ms (You don't need to write code for this, you can just run the existing code several times to find the data points and then make a plot). Is there a simple relationship between pulse width and threshold current?

2) Adjusting properties of Na & K

<i>Python</i>	adjust values for <i>g_Na</i> , <i>g_K</i> , <i>E_Na</i> , etc. in HodgkinHuxley.py
<i>NeuroML2</i>	adjust values for <i>condDensity</i> , <i>erev</i> in <channelDensity> elements in hhcell.cell.nml

2.1 Return the input current injection to the original values. Reduce the conductance densities of Na and K. What is the impact on the AP?

2.2 Restore the original conductance densities. Reduce the reversal potential (e.g. to 20mV) of Na to simulate a decrease in the extracellular Na concentration (i.e. external [Na+] is closer to internal [Na+]). What is the impact on the height/waveform of the action potential? Can a change in the conductance density of Na compensate for this (i.e. increase the height of the AP)? If not, why not?

3) Hyperpolarizing current injections

All of the previously used current injection pulses have been depolarising (increasing the membrane potential from resting voltage). We now try some hyperpolarizing current injections (see 1) above for changing current amplitudes).

3.1 Set the pulse amplitude to -5 and the pulse duration to 5 ms. What happens for hyperpolarizing current injections?

3.2 What is the threshold, in terms of current magnitude and pulse duration, for eliciting this so-called *anode break excitation*?

3.3 What mechanisms in the model are responsible for this behaviour? Look at the time course of the activation and inactivation variables *n*, *m* and *h*.

1.5 Hodgkin Huxley Sources

1.5.1 Hodgkin Huxley.py

```
import scipy as sp
import pylab as plt
from scipy.integrate import odeint

class HodgkinHuxley():
    """Full Hodgkin-Huxley Model implemented in Python"""

    C_m = 1.0
    """membrane capacitance, in uF/cm^2"""

    g_Na = 120.0
    """Sodium (Na) maximum conductances, in mS/cm^2"""

    g_K = 36.0
    """Potassium (K) maximum conductances, in mS/cm^2"""

    g_L = 0.3
    """Leak maximum conductances, in mS/cm^2"""

    E_Na = 50.0
```

```

"""Sodium (Na) Nernst reversal potentials, in mV"""
E_K = -77.0
"""Potassium (K) Nernst reversal potentials, in mV"""
E_L = -54.387
"""Leak Nernst reversal potentials, in mV"""

t = sp.arange(0.0, 450.0, 0.01)
""" The time to integrate over """

def alpha_m(self, V):
    """Channel gating kinetics. Functions of membrane voltage"""
    return 0.1*(V+40.0)/(1.0 - sp.exp(-(V+40.0) / 10.0))

def beta_m(self, V):
    """Channel gating kinetics. Functions of membrane voltage"""
    return 4.0*sp.exp(-(V+65.0) / 18.0)

def alpha_h(self, V):
    """Channel gating kinetics. Functions of membrane voltage"""
    return 0.07*sp.exp(-(V+65.0) / 20.0)

def beta_h(self, V):
    """Channel gating kinetics. Functions of membrane voltage"""
    return 1.0/(1.0 + sp.exp(-(V+35.0) / 10.0))

def alpha_n(self, V):
    """Channel gating kinetics. Functions of membrane voltage"""
    return 0.01*(V+55.0)/(1.0 - sp.exp(-(V+55.0) / 10.0))

def beta_n(self, V):
    """Channel gating kinetics. Functions of membrane voltage"""
    return 0.125*sp.exp(-(V+65) / 80.0)

def I_Na(self, V, m, h):
    """
    Membrane current (in uA/cm^2)
    Sodium (Na = element name)

    | :param V:
    | :param m:
    | :param h:
    | :return:
    """
    return self.g_Na * m**3 * h * (V - self.E_Na)

def I_K(self, V, n):
    """
    Membrane current (in uA/cm^2)
    Potassium (K = element name)

    | :param V:
    | :param n:
    | :return:
    """
    return self.g_K * n**4 * (V - self.E_K)

# Leak

```



```

def I_L(self, V):
    """
    Membrane current (in uA/cm^2)
    Leak

    | :param V:
    | :param h:
    | :return:
    """
    return self.g_L * (V - self.E_L)

def I_inj(self, t):
    """
    External Current

    | :param t: time
    | :return: step up to 10 uA/cm^2 at t>100
    |           step down to 0 uA/cm^2 at t>200
    |           step up to 35 uA/cm^2 at t>300
    |           step down to 0 uA/cm^2 at t>400
    """
    return 10*(t>100) - 10*(t>200) + 35*(t>300) - 35*(t>400)

@staticmethod
def dALLdt(X, t, self):
    """
    Integrate

    | :param X:
    | :param t:
    | :return: calculate membrane potential & activation variables
    """
    V, m, h, n = X

    dVdt = (self.I_inj(t) - self.I_Na(V, m, h) - self.I_K(V, n) - self.I_L(V)) / ↵
↵self.C_m
    dmdt = self.alpha_m(V)*(1.0-m) - self.beta_m(V)*m
    dhdt = self.alpha_h(V)*(1.0-h) - self.beta_h(V)*h
    dndt = self.alpha_n(V)*(1.0-n) - self.beta_n(V)*n
    return dVdt, dmdt, dhdt, dndt

def Main(self):
    """
    Main demo for the Hodgkin Huxley neuron model
    """

    X = odeint(self.dALLdt, [-65, 0.05, 0.6, 0.32], self.t, args=(self,))
    V = X[:,0]
    m = X[:,1]
    h = X[:,2]
    n = X[:,3]
    ina = self.I_Na(V, m, h)
    ik = self.I_K(V, n)
    il = self.I_L(V)

    plt.figure()

    plt.subplot(4,1,1)

```

```

plt.title('Hodgkin-Huxley Neuron')
plt.plot(self.t, V, 'k')
plt.ylabel('V (mV)')

plt.subplot(4,1,2)
plt.plot(self.t, ina, 'c', label='$I_{Na}$')
plt.plot(self.t, ik, 'y', label='$I_{K}$')
plt.plot(self.t, il, 'm', label='$I_{L}$')
plt.ylabel('Current')
plt.legend()

plt.subplot(4,1,3)
plt.plot(self.t, m, 'r', label='m')
plt.plot(self.t, h, 'g', label='h')
plt.plot(self.t, n, 'b', label='n')
plt.ylabel('Gating Value')
plt.legend()

plt.subplot(4,1,4)
i_inj_values = [self.I_inj(t) for t in self.t]
plt.plot(self.t, i_inj_values, 'k')
plt.xlabel('t (ms)')
plt.ylabel('$I_{inj}$ ($\mu A/cm^2$)')
plt.ylim(-1, 40)

plt.show()

if __name__ == '__main__':
    runner = HodgkinHuxley()
    runner.Main()

```

1.5.2 LEMS_HH_Simulation.xml

```

<Lems>

  <!-- Example with Simple Hodgkin-Huxley cell specifying segment details-->

  <!-- This is a file which can be read and executed by the LEMS Interpreter.
  It imports the LEMS definitions of the core NeuroML 2 Components,
  imports in "pure" NeuroML 2 and contains some LEMS elements for running
  a simulation -->

  <Target component="sim1"/>

  <Include file="Cells.xml"/>
  <Include file="Networks.xml"/>
  <Include file="Simulation.xml"/>

  <Include file="HHCellNetwork.net.nml"/>
  <Include file="hhcell.cell.nml"/>
  <Include file="passiveChan.channel.nml"/>
  <Include file="naChan.channel.nml"/>
  <Include file="kChan.channel.nml"/>

```

```

<Simulation id="sim1" length="450ms" step="0.01ms" target="HHCellNetwork">

  <Display id="d1" title="Hodgkin-Huxley Neuron: V (mV)" timeScale="1ms" xmin="-
↪20" xmax="470" ymin="-90" ymax="50">
    <Line id="v" quantity="hhpop[0]/v" scale="1mV" color="#000000" timeScale=
↪"1ms"/>
  </Display>

  <Display id="d2" title="Hodgkin-Huxley Neuron: Gating Variables" timeScale=
↪"1ms" xmin="-20" xmax="470" ymin="-0.1" ymax="1.1">
    <Line id="m" quantity="hhpop[0]/bioPhys1/membraneProperties/naChans/
↪naChan/m/q" scale="1" color="#ff0000" timeScale="1ms"/>
    <Line id="h" quantity="hhpop[0]/bioPhys1/membraneProperties/naChans/
↪naChan/h/q" scale="1" color="#00dd00" timeScale="1ms"/>
    <Line id="n" quantity="hhpop[0]/bioPhys1/membraneProperties/kChans/kChan/
↪n/q" scale="1" color="#0000ff" timeScale="1ms"/>
  </Display>

  <Display id="d3" title="Hodgkin-Huxley Neuron: Current" timeScale="1ms" xmin=
↪"-20" xmax="470" ymin="-10" ymax="10">
    <Line id="I_na" quantity="hhpop[0]/bioPhys1/membraneProperties/naChans/
↪iDensity" scale="1" color="#00ffff" timeScale="1ms"/>
    <Line id="I_k" quantity="hhpop[0]/bioPhys1/membraneProperties/kChans/
↪iDensity" scale="1" color="#ffff00" timeScale="1ms"/>
    <Line id="I_l" quantity="hhpop[0]/bioPhys1/membraneProperties/leak/
↪iDensity" scale="1" color="#ff00ff" timeScale="1ms"/>
  </Display>

  <Display id="d4" title="Hodgkin-Huxley Neuron: I_inj (nA)" timeScale="1ms"
↪xmin="-20" xmax="470" ymin="-0.01" ymax="0.4">
    <Line id="I_inj1" quantity="hhpop[0]/pulseGen1/i" scale="1nA" color="
↪#ffffff" timeScale="1ms"/>
    <Line id="I_inj2" quantity="hhpop[0]/pulseGen2/i" scale="1nA" color="
↪#000000" timeScale="1ms"/>
  </Display>

  <!-- Saved the membrane potential to file: hh_v.dat -->
  <OutputFile id="of0" fileName="hh_v.dat">
    <OutputColumn id="v" quantity="hhpop[0]/v"/>
  </OutputFile>

</Simulation>

</Lems>

```

1.5.3 HHCellNetwork.net.nml

```

<?xml version="1.0" encoding="UTF-8"?>

<neuroml xmlns="http://www.neuroml.org/schema/neuroml2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.neuroml.org/schema/neuroml2 https://raw.
↪githubusercontent.com/NeuroML/NeuroML2/master/Schemas/NeuroML2/NeuroML_v2beta3.xsd"

```

```

    id="HHCellNetwork">

    <include href="hhcell.cell.nml"/> <!-- Include the cell definition -->

    <!-- Short small current pulse input & short larger current pulse input -->
    <pulseGenerator id="pulseGen1" delay="100ms" duration="100ms" amplitude="0.10nA"/>
    <pulseGenerator id="pulseGen2" delay="300ms" duration="100ms" amplitude="0.35nA"/>

    <network id="HHCellNetwork">

        <notes>Network with a single cell based on the Hodgkin Huxley model with 2_
↳input currents</notes>

        <population id="hhpop" component="hhcell" size="1"/>
        <explicitInput target="hhpop[0]" input="pulseGen1"/>
        <explicitInput target="hhpop[0]" input="pulseGen2"/>
    </network>

</neuroml>

```

1.5.4 hhcell.cell.nml

```

<?xml version="1.0" encoding="UTF-8"?>

<neuroml xmlns="http://www.neuroml.org/schema/neuroml2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.neuroml.org/schema/neuroml2 https://raw.
↳githubusercontent.com/NeuroML/NeuroML2/master/Schemas/NeuroML2/NeuroML_v2beta3.xsd"
  id="hhcell">

    <include href="passiveChan.channel.nml"/> <!-- Include the channel definitions -->
    <include href="naChan.channel.nml"/>
    <include href="kChan.channel.nml"/>

    <cell id="hhcell">

        <notes>Conductance based cell model NeuroML2 format: standard Hodgkin Huxley_
↳model cell with Na, K and passive conductances</notes>

        <morphology id="morphology">
            <segment id="0" name="soma">
                <proximal x="0" y="0" z="0" diameter="17.841242"/> <!--Gives a_
↳convenient surface area of 1000.0 um^2-->
                <distal x="0" y="0" z="0" diameter="17.841242"/>
            </segment>

            <segmentGroup id="soma_group">
                <member segment="0"/>
            </segmentGroup>

        </morphology>

        <biophysicalProperties id="bioPhys1">

            <membraneProperties>

```

```

        <channelDensity id="leak" ionChannel="passiveChan" condDensity="0.3_
↪mS_per_cm2" erev="-54.387mV" ion="non_specific"/>
        <channelDensity id="naChans" ionChannel="naChan" condDensity="120.0_
↪mS_per_cm2" erev="50.0 mV" ion="na"/>
        <channelDensity id="kChans" ionChannel="kChan" condDensity="36 mS_per_
↪cm2" erev="-77mV" ion="k"/>

        <spikeThresh value="-20mV"/>
        <specificCapacitance value="1.0 uF_per_cm2"/>
        <initMembPotential value="-65mV"/>

    </membraneProperties>

    <intracellularProperties>
        <resistivity value="0.03 kohm_cm"/>    <!-- Note: not used in single_
↪compartment simulations -->
    </intracellularProperties>

    </biophysicalProperties>

</cell>

</neuroml>

```

1.5.5 passiveChan.channel.nml

```

<?xml version="1.0" encoding="UTF-8"?>

<neuroml xmlns="http://www.neuroml.org/schema/neuroml2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.neuroml.org/schema/neuroml2 https://raw.
↪githubusercontent.com/NeuroML/NeuroML2/master/Schemas/NeuroML2/NeuroML_v2beta3.xsd"
  id="passiveChan">

    <ionChannelHH id="passiveChan" conductance="10pS" type="ionChannelPassive">

        <notes>Single ion channel in NeuroML2 format: passive channel providing a_
↪leak conductance </notes>

    </ionChannelHH>

</neuroml>

```

1.5.6 naChan.channel.nml

```

<?xml version="1.0" encoding="UTF-8"?>

<neuroml xmlns="http://www.neuroml.org/schema/neuroml2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.neuroml.org/schema/neuroml2 https://raw.
↪githubusercontent.com/NeuroML/NeuroML2/master/Schemas/NeuroML2/NeuroML_v2beta3.xsd"
  id="naChan">

```

```

<ionChannelHH id="naChan" conductance="10pS" species="na">
  <notes>Single ion channel in NeuroML2 format: standard Sodium channel from
↳the Hodgkin Huxley model</notes>
  <gateHHrates id="m" instances="3">
    <forwardRate type="HHExpLinearRate" rate="1per_ms" midpoint="-40mV" scale=
↳"10mV"/>
    <reverseRate type="HHExpRate" rate="4per_ms" midpoint="-65mV" scale="-18mV
↳"/>
  </gateHHrates>
  <gateHHrates id="h" instances="1">
    <forwardRate type="HHExpRate" rate="0.07per_ms" midpoint="-65mV" scale="-
↳20mV"/>
    <reverseRate type="HHSigmoidRate" rate="1per_ms" midpoint="-35mV" scale=
↳"10mV"/>
  </gateHHrates>
</ionChannelHH>
</neuroml>

```

1.5.7 kChan.channel.nml

```

<?xml version="1.0" encoding="UTF-8"?>
<neuroml xmlns="http://www.neuroml.org/schema/neuroml2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.neuroml.org/schema/neuroml2 https://raw.
↳githubusercontent.com/NeuroML/NeuroML2/master/Schemas/NeuroML2/NeuroML_v2beta3.xsd"
  id="kChan">
  <ionChannelHH id="kChan" conductance="10pS" species="k">
    <notes>Single ion channel in NeuroML2 format: standard Potassium channel from
↳the Hodgkin Huxley model</notes>
    <gateHHrates id="n" instances="4">
      <forwardRate type="HHExpLinearRate" rate="0.1per_ms" midpoint="-55mV"
↳scale="10mV"/>
      <reverseRate type="HHExpRate" rate="0.125per_ms" midpoint="-65mV" scale="-
↳80mV"/>
    </gateHHrates>
  </ionChannelHH>
</neuroml>

```

1.5.8 run.sh

```
#!/bin/bash

#####
# LEMS Hodgkin Huxley Neuron Model
#
# Command to run LEMS_HH_Simulation.xml script on Linux/Mac
#
# Usage: ./run.sh
#
#####

set -e
java -jar jNeuroML-0.7.2-jar-with-dependencies.jar LEMS_HH_Simulation.xml
```

1.5.9 run.bat

```
@echo off

REM #####
REM # LEMS Hodgkin Huxley Neuron Model
REM #
REM # Command to run LEMS_HH_Simulation.xml script on Windows
REM #
REM # Usage: run.bat
REM #
REM #####

java -jar jNeuroML-0.7.2-jar-with-dependencies.jar LEMS_HH_Simulation.xml
```